FBD 2020 Project

최적화된 엘리베이터 운행 알고리즘 개발 및 분석과 엘리베이터 시뮬레이팅 프로그램 개발

Optimized algorithm & Simulator for two button-sharing elevators

팀 FBD

동국대학교 기계로봇에너지공학과 강재원

한국항공대학교 항공전자정보공학부 정보통신공학전공 김경민

건국대학교 컴퓨터공학과 이경호

목차

I. 서론

1. 프로젝트의 목적

프로젝트를 시작하기에 앞서 구상했던 계획은 실생활이나 주변에서 일어나는 문제점, 불편 등을 최소화하기 위한 방향으로 잡았고, 평소 사람들이 많이 이용하는 학교에서 문제점을 찾아보기로 하였다. 그중 건국대학교 새천년관에 주목하게 되었는데, 건국대학교 새천년관은 다수의 사람이 이용하기도 하고 평소 대학생활과도 밀접한 관련이 있기에 자세히 알아보기로 하였다. 새천년관에서는 학생, 교수, 직원 등 많은 사람이 각자의 목적에 따라 이동하기 위해 엘리베이터를 이용한다. 현재 새천년관에는 엘리베이터가 두 대가 있으며, 이 두 대를 이용해 B2 층부터 14층까지 이동할 수 있다. 많은 사람이 이용함에 따라 각자의 불편함이 생길 수밖에 없기에 어느 부분에서 이 불편이 생기는지 관찰하였다. 처음에는 엘리베이터가 두 대이기 때문에 어느 정도 원활한 이동이 가능할 것으로 생각했지만, 특정 부분에서 문제가 발생하였다. 가까이 있는 엘리베이터가 아닌 더 먼 쪽에 있는 엘리베이터를 기다릴 때도 있고, 이 와중에 다른 엘리베이터가 자신의 층을 통과하는 경우도 있었다. 평소 학생들이 이런 경우를 인지하고 계단을 많이 이용하기도 한다고 했다. 흔히 일상 속에 일어날 수 있는 간단한 현상이라 생각하고 넘어갈 수도 있지만, 이러한 문제점을 해결하고 더 나아가 여러 요소를 고려해 효율적으로 엘리베이터를 운행하기 위해 이 프로젝트를 시작하게 되었다. 새천년관 엘리베이터를 계기로 시작하였지만 이뿐만 아니라 곳곳에 많은 엘리베이터가 존재하기에 이를 통해 더 폭넓게 적용할 수 있을 것으로 전망한다.

2. 프로젝트의 진행과정

가. 간트차트

프로젝트 설계를 계획하는 시점부터, 해야 하는 작업의 리스트를 기반으로 간트차트를 작성하였다. 세부적인 작업 내용은 생략하고, 주제별로 작업과 일정을 정리하였다. 주차별로 작업의 진행 계획 일정과 진행 상황 일정을 표시하여, 프로젝트의 진행을 한 눈에 보도록 함으로써 작업에 도움이 되도록 하였다. 실제 프로젝트에서 작성한 간트차트는 아래와 같다.

프로젝트 시작일 : 12/30	1주차 1/6	2주차 1/13	3주차 1/20	4주차 1/27	5주차 2/3	6주차 2/10	7주차 2/17	8주차 2/24	9주차 2/29	10주차 3/7	11주차 3/14
현장답사											
자료조사											
보드/펌웨어 조사											
PCB 설계											
펌웨어 코딩											
Button Board 모델링											
GUI 구현											
알고리즘 기본 포맷 설계											
내부 메인 알고리즘 코딩											
알고리즘 평가 및 분석											
최종 시연 및 보고서 작성											

: 진행계획 : 진행상황

나. Notion

Notion이란 메모 정리를 도와주는 note app로서, 많은 곳에서 팀 단위 프로젝트를 진행하는 데 있어 협업 tool로 사용하는 애플리케이션이다. 웹 서비스와 앱 서비스 모두 지원하며, evernote, slack, google drive 등 다른 서비스와의 연동도 가능하다는 장점이 있다. 우리는 프로젝트를 진행하며 서로의 작업 내용, 작업 진행 상황, 의견 등을 공유하기 위해 notion을 아래와 같이 사용하였다.

- 공지사항

  프로젝트에서 공통으로 진행하는 작업에 대한 규칙을 공지한 페이지

- Works

  실제 작업 리스트를 정리한 페이지

  작업별로 세부적인 작업 내용, 작업 상태, 작업 일정, 작업자, 태그 등을 표시.

- 알고리즘 리스트

  프로젝트 코딩의 주요 부분인 엘리베이터 운행 알고리즘에 관한 내용을 따로 정리한 페이지

  알고리즘별로 작동 원리, 작업자, 작업 상태를 표시

- 온라인 회의 페이지

  오프라인이 아닌 온라인으로 회의를 진행하고자 할 때, 의견을 공유하기 위한 페이지

  회의 주제, 회의 일정, 결정 마감일 등을 표시

  실시간 음성 회의 혹은 투표를 통해 회의를 진행

- 부품 구매 내역

  프로젝트를 진행하며 구매한 부품에 대한 정리 페이지

  구매 부품, 구매 상태, 가격, 주무자, 관련 작업 페이지, 구매 링크 등을 공유

다. Git

git이란 프로그램, 파일 등의 소스 코드 관리를 위한 분산 버전 관리 시스템이다. 이는 여러 개발자가 하나의 프로젝트 및 하나의 프로그램의 소스코드에 대해 협업하여 진행해야 하는 경우, 분산된 버전에 대해서 각자 진행을 하고 이를 병합하는 작업까지 지원하는 시스템을 의미한다. 협업해야 하는 개발자들에게 편하고 신속한 개발 환경을 지원한다는 장점이 있으며, 이전의 수정 내역과 당시에 작성한 코멘트 등 모두 저장되어 나중에 확인할 수 있는 특징도 있다. 우리는 git 서비스 중 github을 선택하여 사용하였다. github에 private repository를 만든 뒤, collaborator로 프로젝트 참여자를 초대하여 프로그래밍을 진행하였다.

II. 이론적 배경

1. 엘리베이터의 작동원리

가. 엘리베이터의 작동원리

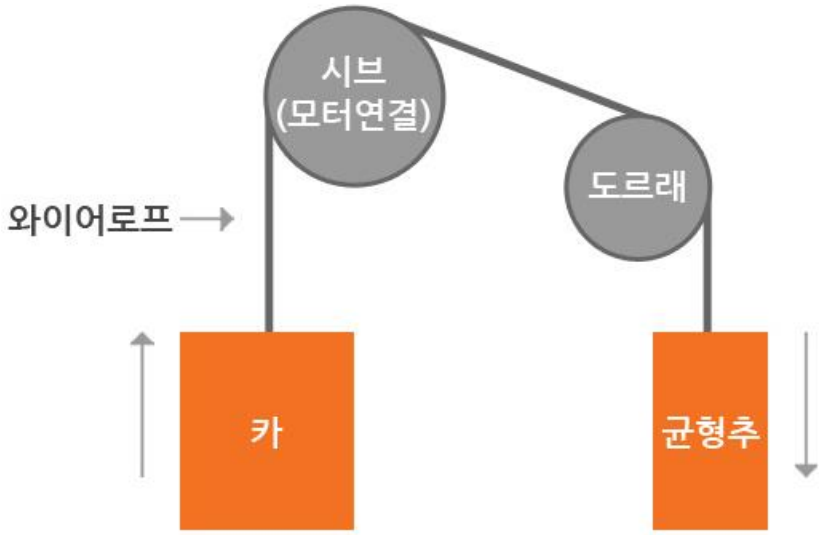


그림 1 권상식 엘리베이터의 원리

엘리베이터의 작동원리는 크게 5가지로 나누어지는데, 로프식, 유압식, 리니어 모터식, 스크류식, 그리고 랙&피니언식이 바로 그것이다. 이 중 대한민국의 엘리베이터 대다수를 차지하고 있는 것은 로프식으로, 로프를 이용하여 구동부가 엘리베이터를 운용하는 방식을 의미한다. 로프식 엘리베이터는 다시 권상식(트랙션식)과 권동식으로 나누어진다. 권상식 구동방식이란, 위의 그림과 같이 권상기(전동기 출력 축과 감속기 입력 축을 커플링으로 연결한 구조)의 출력 축에 시브가 연결되고 간격 유지를 위해 적당한 위치에 도르래를 연결한 후, 와이어 로프를 시브와 도르래에 걸어 한쪽에는 카를 매달고 반대쪽에는 균형추를 매달아 전동기의 회전에 의해 감속기 축에 있는 시브를 정회전/역회전시켜 카를 운행하는 구동방식을 말한다. 일반적인 엘리베이터들은 대부분 권상식 구동방식을 채택하고 있으며, 새천년관 엘리베이터에 사용되고 있는 방식 또한 권상식이다. 반면, 권동식 엘리베이터는 균형 추 없이 카를 바로 구동부에 연결하는 방식을 말한다.

권상식과 권동식의 차이를 그림을 통해 조금 더 자세하게 알아보도록 하자. 다음의 그림들은 권동식 엘리베이터와 권상식 엘리베이터 각각의 구동시브가 받는 힘을 나타낸 Free Body Diagram이다. 시브와 도르래의 질량은 무시하였다.

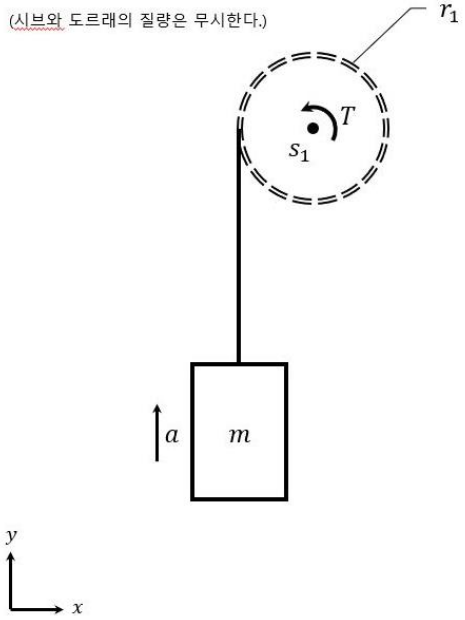
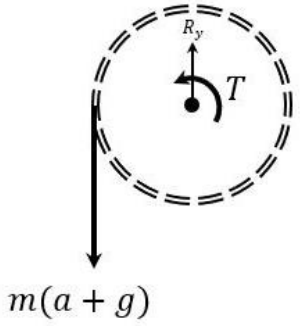


그림 2 권동식 엘리베이터 모델



$$R_y = m(a+g)$$
$$T = -mr_1(a+g)$$

그림 3 권동식 엘리베이터의 구동시브 FBD

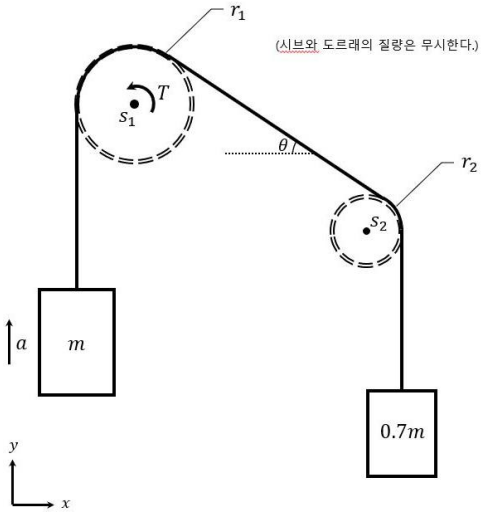
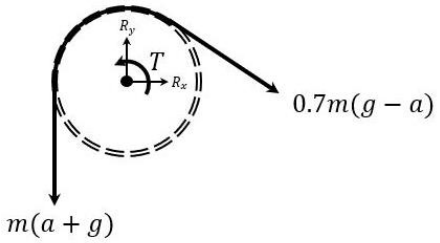


그림 4 권상식 엘리베이터 모델



$$R_x = -0.7m(g-a)\cos\theta$$
$$R_y = m\{a(1-0.7\sin\theta) + g(1+0.7\sin\theta)\}$$
$$T = -mr_1(1.7a - 0.7g)$$

그림 5 권상식 엘리베이터의 구동시브 FBD

그림에서 볼 수 있듯이, 카를 모터에 직접 연결하지 않고 균형 추를 사용함으로써 모터 shaft가 받는 전단응력은 더 커지지만, 모터가 필요로 하는 토크를 크게 줄일 수 있다.

다. 새천년관 엘리베이터의 Specification

12/30에 진행하였던 건국대학교 새천년관 답사에서, 본 팀은 승강기의 고유번호를 비롯한 세부사항을 파악하였다. 이를 통해, 국가승강기정보센터에서 승강기의 세부정보를 열람할 수 있었다. 그 내용은 아래와 같다.

승강기정보 열람

승강기번호	0005-831	검사이력	고장이력	사고이력	자체점검이력

건물명	건국대학교		
소재지	서울특별시 광진구 능동로 120 (화양동) 위치보기		
제조업체	오티스엘리베이터(유)서울	모델명	GR1
종류	장애/전망용 (엘리베이터)	상태	운행중
설치일/최초설치일	2017-03-30 / 2000-03-31	최종검사일	2019-05-28
검사유효기간	2019-03-30 ~ 2020-03-29		

유지관리업체명	오티스엘리베이터(유)서울	유지관리업체 연락처	02-2007-5800
하도급/공동수급 업체명	주식회사 광성엘리베이터	하도급/공동수급 연락처	02-452-0982
최종검사기관	한국승강기안전공단 서울강동지사		

그림 6 새천년관 엘리베이터의 세부정보

OTIS에서 제공하는 승강기에 대한 정보를 토대로, 건국대학교 새천년관의 두 개의 승강기가 OTIS의 Gen2® Life(승객용 중저속 승강기) 분류에 속한 기계실이 없는 모델(MRL)중 GR1-PA20 모델이라는 것을 파악하였다. OTIS는 제품 브로셔에서 GR1-PA20 모델의 규격을 제공한다. 이 값들을 통해, 새천년관 엘리베이터 모델을 보다 현실적으로 구현할 수 있을 것으로 기대된다.

2. 시리얼 통신

프로젝트의 하드웨어 개발 부분에 속한 버튼보드에서는 버튼 입력에 대한 데이터를 전달하기 위해서 시리얼 통신을 사용한다. 시리얼 통신이란 전자 장비 간의 데이터 통신을 지원하는 통신 방법으로서, 하나 또는 두 개의 전송 라인을 사용하여 데이터를 송수신하며 UART 통신, I2C 통신 SPI 통신 등이 있다. 이 중 프로젝트에서는 다음 2가지의 통신 방법을 사용한다.

가. UART 통신

UART 통신은 시리얼 통신에서 가장 많이 쓰이는 방식으로, PC, 아두이노, 라즈베리파이 등 대부분의 전자장비에서 지원한다. UART 통신은 VCC, Rx, Tx, GND로 구성된 4개의 line을 연결해야 한다. 각각의 장비에 전원이 입력되고 있다면 VCC 연결은 생략할 수 있으며, Rx와 Tx는 토글하여 연결해야 한다. UART 통신을 지원하는 하드웨어 규격 또한 다양하다. RS232, RS485, TTL 등이 있으며 서로 다른 규격을 연결하는 경우, 중간에 convertor module을 추가로 연결해야 한다. USB port로 UART 통신을 지원하는 장비의 경우, 대부분 USB driver 및 convertor module이 내장되어 있어 USB 케이블을 직접 연결하여 통신에 사용할 수 있다.

나. I2C 통신

I2C 통신은 대부분의 센서 모듈에서 데이터 통신을 할 때 지원하는 방식이다. I2C 통신은 VCC, SDA(data), SCL(clock), GND로 구성된 4개의 line을 연결해야 하며, 각각의 장비에 전원이 입력되고 있다면 VCC 연결은 생략할 수 있다. I2C 통신의 특징 중 하나는 디바이스 별로 마스터 모드 혹은 슬레이브 모드 중 하나를 선택하여 해당 모드로 작동한다는 것이다. 또한 하나의 회로에서 1개의 마스터 디바이스 + n개의 슬레이브 디바이스로 구성이 가능하다는 점에서 여러 디바이스를 동시에 통신하고자 할 때 유용하다. 마스터 모드 디바이스는 I2C 회로를 전반적으로 제어하는 기능을 한다. 특정 슬레이브 모드 디바이스에게 데이터를 송신하거나, 특정 슬레이브 모드 디바이스가 데이터를 송신할 것을 지시하는 등의 기능을 한다. 슬레이브 모드 디바이스는 각각 고유의 주소값으로 구분되며 마스터 모드 디바이스의 제어 하에 데이터 송수신을 진행한다.

III. 버튼보드 설계

1. PCB 보드

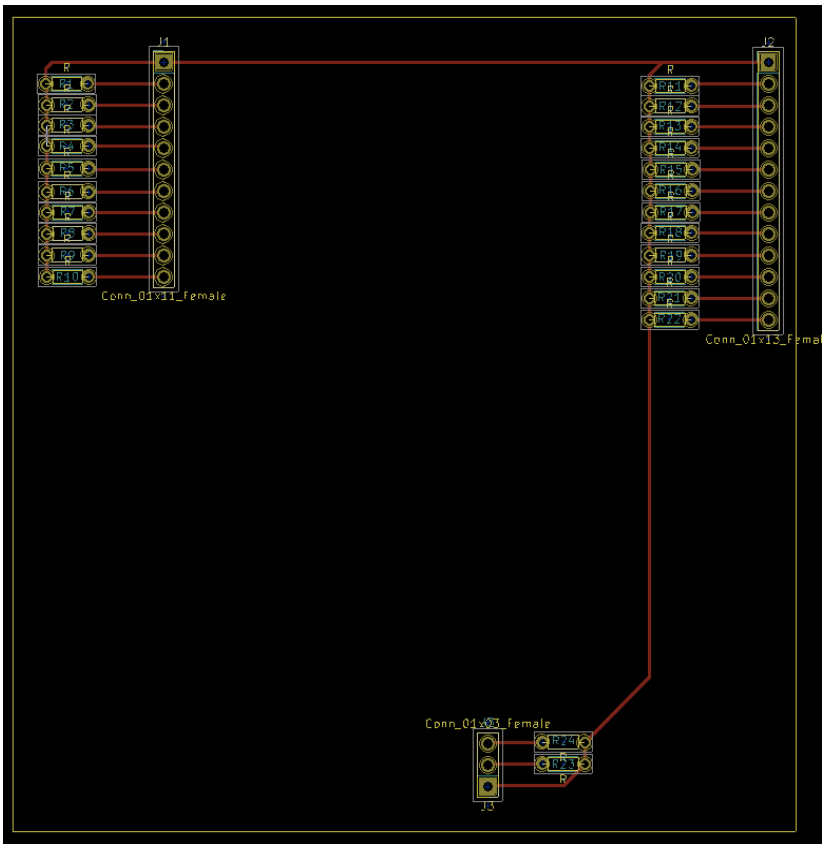
가. PCB란

PCB란 Printed Circuit Board의 준말로, 인쇄회로기판을 의미한다. 절연판 위와 그 내부에 회로를 형성하고 이와 연결된 부품끼리 전기적으로 연결하여 전자장비를 동작시키는 기능을 한다. 대부분의 전자 장비는 PCB를 통해 전자부품들이 연결되어 있으며, 여러 층으로 구성된 구조 등 다양한 방식으로 설계되어있다. bread board와 같은 보드 위에 점퍼선으로 전자 장비들을 연결하던 것을 간편하게 하나의 단면 기판 위에 회로를 구현할 수 있다는 장점이 있다. PCB 설계의 경우, CAD tool을 통해 진행하며 schematic 설계, footprint 설계, 보드 제작, 부품 납땜 등의 작업으로 구성된다.

나. 설계 내용

버튼보드의 PCB는 2 layer board 2개의 복층 구조로 구성된다. 2개의 보드는 pin header로 고정 및 연결이 되며, footprint는 아래와 같다.

- 1 층 board

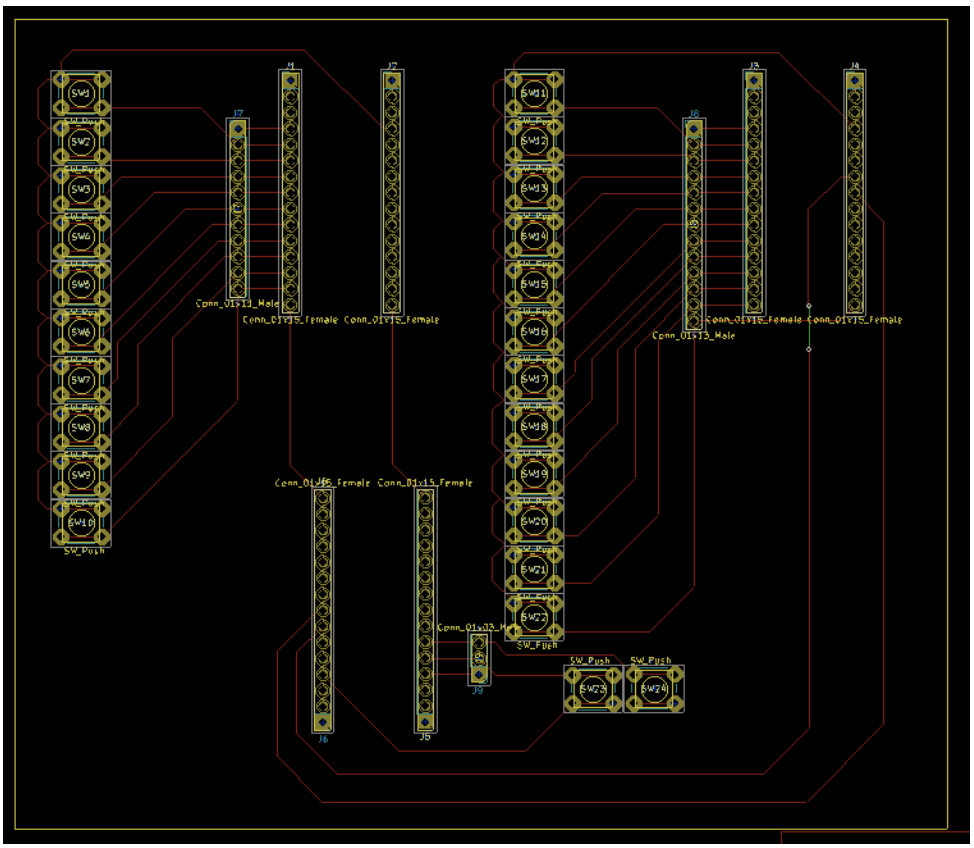


■ J1, J2, J3 은 각각 1x11, 1x13, 1x3 2.54mm female pin header 가 들어간다. J1, J2, J3 은 2 층 board 와 고정을 위해 사용되는 동시에 아두이노 나노의 digital pin 에 저항과 ground 를 잇는 기능을 한다. 각각의

1 번 pin 은 모두 하나의 node 로 연결되어 있는데 이로써 PCB 회로의 ground 를 통일시킨다.

■ R1~R24 는 1kohm 의 저항이 납땜 되며 버튼과 ground 를 잇는 기능을 한다.

– 2 층 board



■ J1, J2, J3, J4, J5, J6 에는 1x15 2.54mm female pin header 가 들어간다. 각각 2 개씩 짝지어 아두이노 나노 3 대의 연결 및 고정을 하는 기능을 한다.

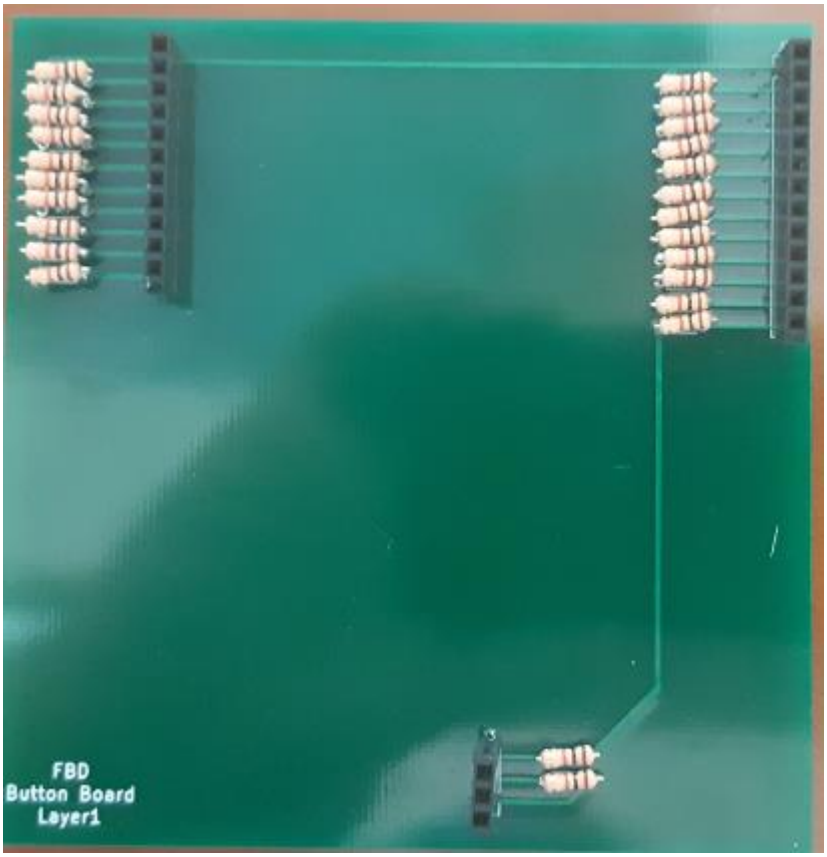
■ J7, J8, J9 은 각각 1x11, 1x13, 1x3 2.54mm male pin header 가 들어간다. 앞서 말한 1 층 board 의 J1, J2, J3 과 연결되어 고정을 위해 사용된다. 또한 버튼과 아두이노의 digital pin 에 잇는 기능을 한다.

■ SW1~SW24 은 버튼이 납땜 되어 5V 와 digital pin 을 잇는 회로를 구성한다.

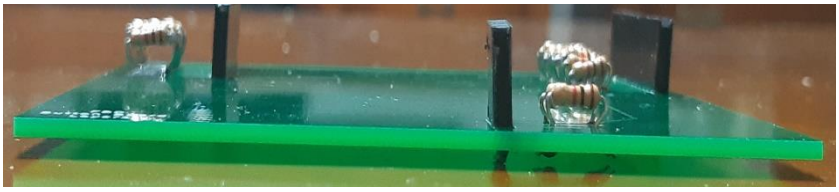
■ 아두이노 나노간의 데이터 통신을 위하여 softwareSerial 을 위한 pin 과 I2C 를 위한 pin 의 연결을 위해 각각 도선을 연결해주었다.

다. 실제 보드 사진

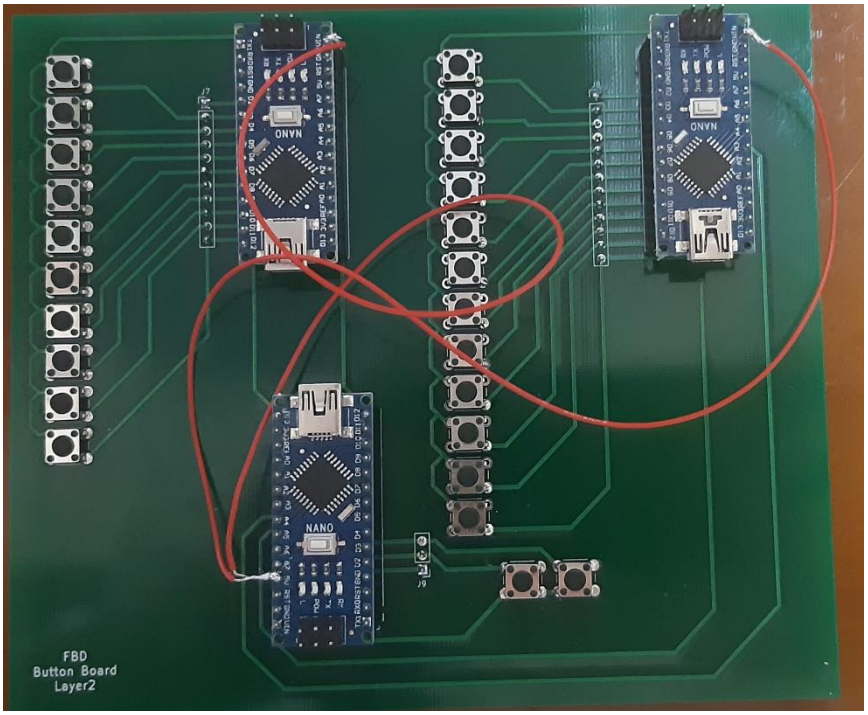
- 버튼보드 1 층 전면



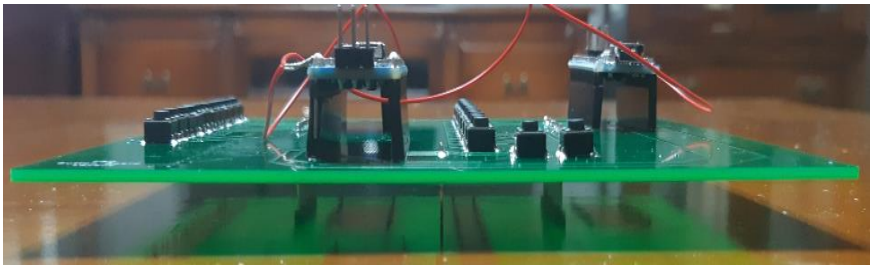
- 버튼보드 1 층 측면



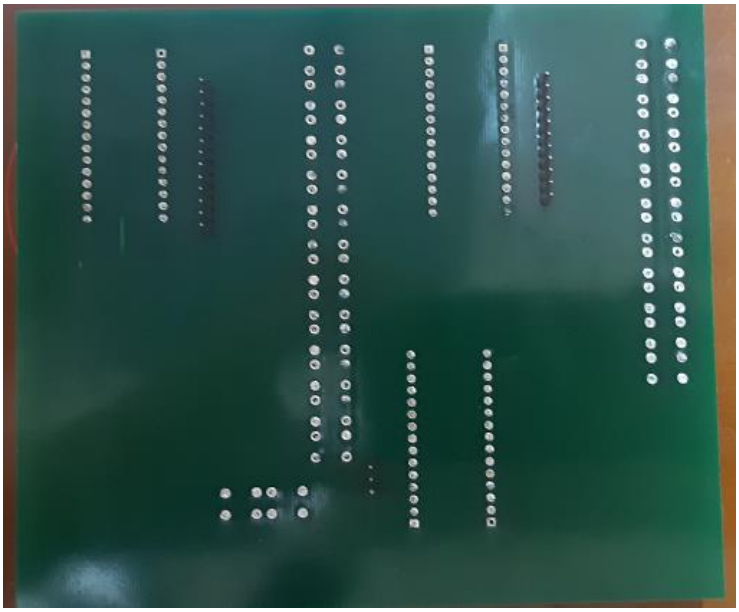
- 버튼보드 2층 전면



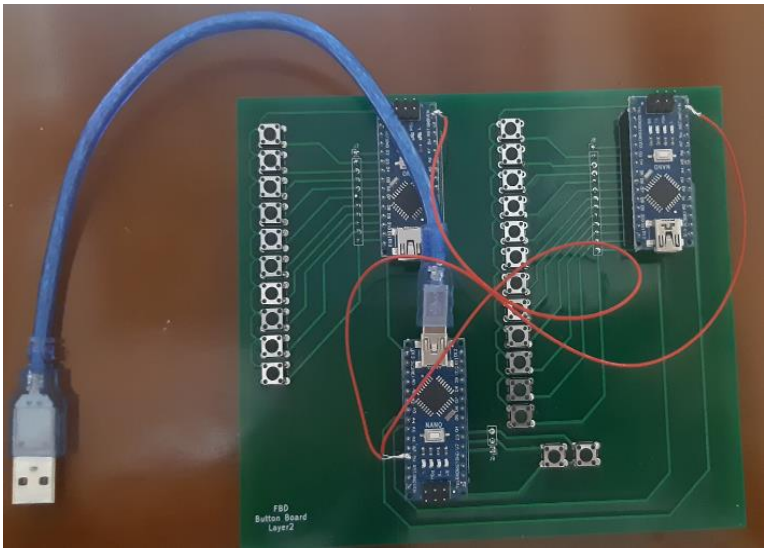
- 버튼보드 2층 측면



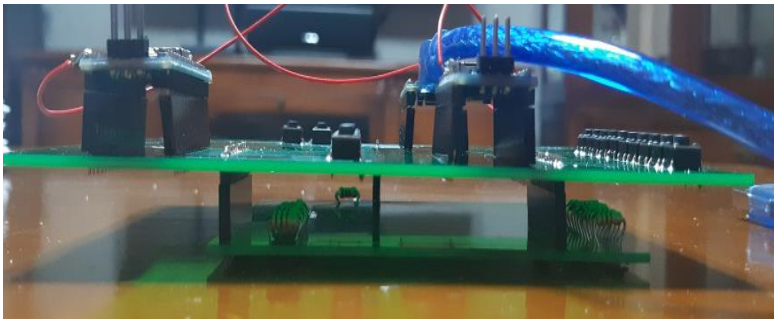
- 버튼보드 2층 후면



– 연결된 버튼보드 전면



– 연결된 버튼보드 측면



2. 버튼보드 데이터 처리장치

가. 아두이노

버튼보드의 메인 데이터 처리장치로는 아두이노를 선택하였다. 아두이노란 개발자가 설계한 프로그램을 쉽게 임베디드시켜 실행할 수 있도록 디자인된 마이크로 컨트롤러다. 여러 통신 방식을 통해 센서와 모터, PC 등의 하드웨어와의 연결을 지원함으로써 다양한 전자 장비를 이용한 프로젝트를 구현하게 해준다. 아두이노는 오픈소스를 제공한다는 점에서 큰 장점이 있고, 여러 모델이 존재하며 개발자가 스펙을 확인한 뒤 사용하기에 적합한 모델을 선택하여 개발을 진행한다.

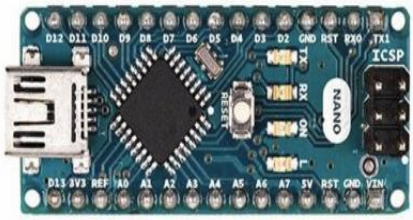


그림 7 아두이노 나노 보드

프로젝트에서 선택한 모델인 아두이노 나노는 가장 보편적으로 사용되는 아두이노 우노와 동일한 ATMega328 MCU chip을 사용한다. mini USB port를 통해 PC와의 연결을 지원하며 digital pin, analog pin, 전원 pin 등 필요한 pin을 최소화하여 디자인된 보드다. 아두이노 나노의 큰 특징으로는 pin socket이 male형 pin header로 보드에 연결되어 있어 다른 보드와 연결할 때, 점퍼선이 필요 없어 간편하다는 장점이 있다.

나. 구성

버튼 보드에는 총 3개의 아두이노 나노가 들어간다. 버튼의 입력을 통해 이미 정의한 통신 프로토콜에 따른 1byte의 데이터를 주고받으며 최종적으로 PC에 해당 데이터를 시리얼로 전달하는 방식이다.

- 1 번 아두이노

  총 10 개의 버튼과 연결되어 car call 에 대한 입력을 받는다. 버튼 입력 시, serial 통신을 통해 3 번 아두이노에게 데이터를 전달한다.

- 2 번 아두이노

  총 12 개의 버튼과 연결되어 landing call 에 대한 입력을 받는다. 버튼 입력 시, I2C 통신을 통해 3 번 아두이노에게 데이터를 전달한다.

- 3 번 아두이노

  총 2 개의 버튼과 연결되어 엘리베이터의 열림 버튼에 대한 입력을 받는다. PC 와 USB 케이블을 통해 연결되어 있으며, 1 번 아두이노와 2 번 아두이노로부터 전달받은 데이터, 열림 버튼에 대한 데이터를 PC 에 전달한다.

3. 버튼보드 케이스

PCB 보드를 보호하기 위해서, PCB 보드를 전체적으로 덮는 외형 케이스를 만들고자 하였다. 3D 모델링 도구 중 하나인 Creo를 사용하여 실제 크기의 PCB 보드와 호환되는 케이스를 디자인하고, 3D 프린터를 사용해 인쇄하도록 계획하였다.

먼저, 실제 보드와의 호환성을 위하여 실제 크기의 PCB 보드를 모델링하였다(그림 8). 이후, 보드 위의 버튼을 케이스 밖에서 조작할 수 있도록 케이스의 버튼을 모델링하였다(그림 9). 다음으로, 직육면체 모양으로 케이스의 외형을 모델링했으며, PCB 보드와의 조립이 쉽도록 상단의 덮개와 하단의 받침대로 나누어 주었다(그림 10, 11). 또한, 덮개와 받침대가 쉽게 분리되지 않도록 케이스의 후면에 고정용 키를 추가해주었다.

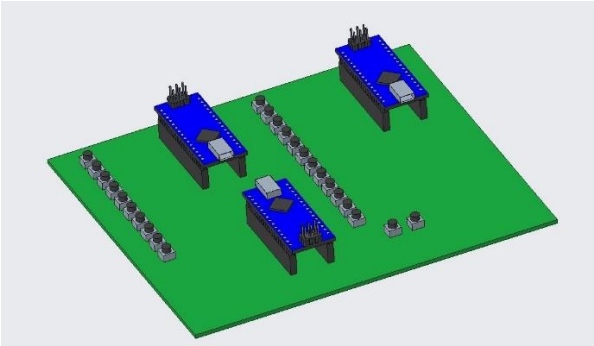


그림 8 PCB보드 모델

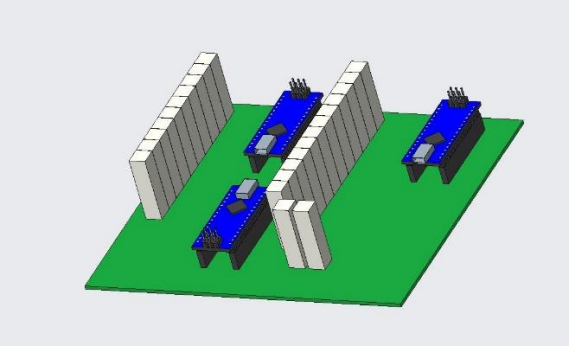


그림 9 케이스의 버튼 모델

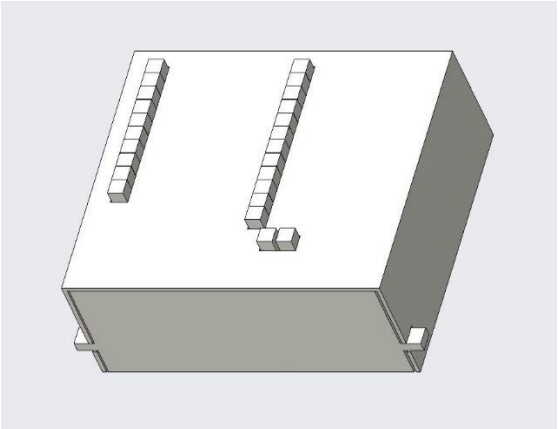


그림 10 버튼보드 케이스 모델(정면)

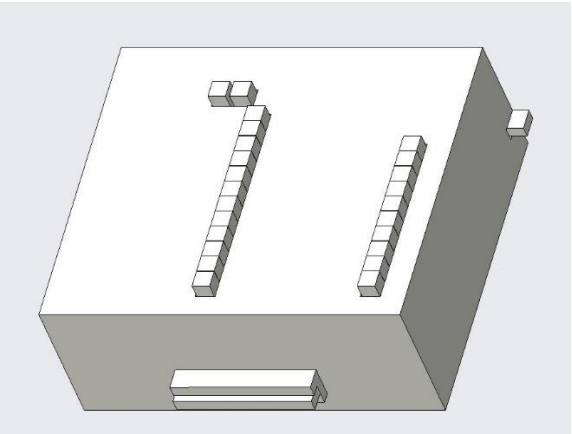


그림 11 버튼보드 케이스 모델(후면)

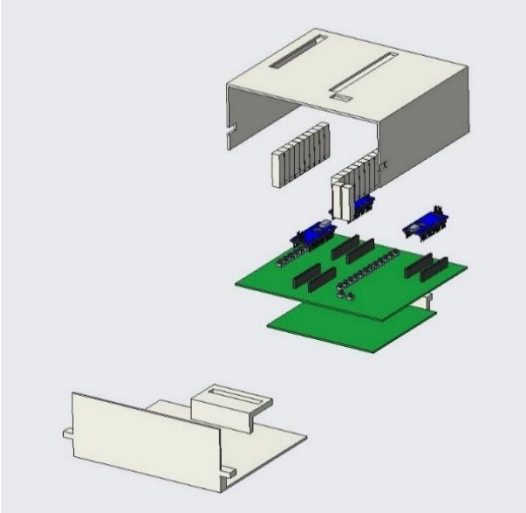


그림 12 버튼보드 케이스 분해도(정면)

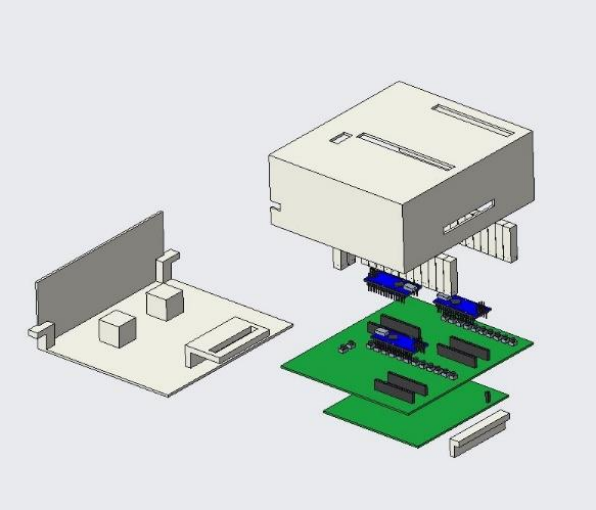


그림 13 버튼보드 케이스 분해도(후면)

IV. 엘리베이터 구현 소프트웨어 설계

1. 엘리베이터 거동 구현 소프트웨어

본 팀이 알고리즘들을 설계하기 전에, 먼저 엘리베이터의 거동을 구현하는 소프트웨어를 코딩하여야 했다. 엘리베이터 구현 소프트웨어는 크게 다음과 같이 이루어져 있다.

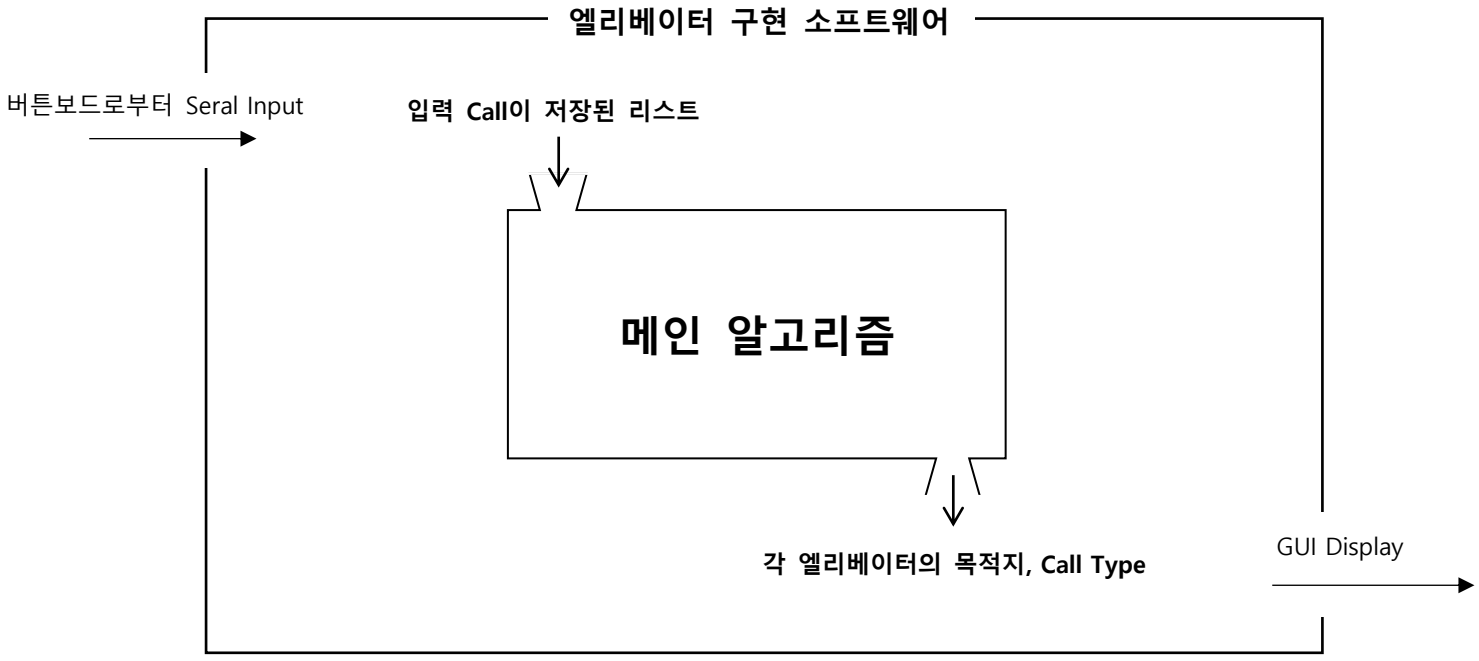


그림 14 엘리베이터 구현 소프트웨어의 기본 틀

소프트웨어 기본 틀에서 볼 수 있듯이, 메인 알고리즘의 설계가 달라져도 시리얼 인풋을 받는 코드나, GUI를 생성하고 디스플레이하는 코드, 그리고 각 엘리베이터의 위치변수를 바꿔주는 코드는 변하지 않는다. 따라서 메인 알고리즘의 설계 이전에 이 공통되는 소프트웨어의 틀을 설계할 필요가 있었다. 가장 먼저, 메인 알고리즘을 제외한 틀을 설계하기 위해서는 메인 알고리즘의 입력변수와 출력변수를 정의해야 하였다. 따라서 본 팀은 메인 알고리즘의 입력변수로 버튼보드로부터 Serial 통신을 통해 받은 landing call과 car call을 저장한 리스트를 사용하기로 하였고, 출력변수로는 각 엘리베이터의 목적지와 그 목적지가 어떤 종류의 call인지를 담은 command 리스트를 사용하기로 하였다. 이제, GUI를 제외한 나머지 코드들을 먼저 살펴보자.

먼저, 버튼보드의 아두이노 나노 보드와의 시리얼 통신은 pyserial이라는 모듈을 사용해 진행하였다. 이때, 통신의 프로토콜을 정의하여 이 모듈을 통해 받은 1바이트 데이터를 유의미한 데이터로 해석해서 사용하였다. 이렇게 받은 call들을 리스트에 저장한 후 메인 알고리즘에 넘겨주게 되고, 메인 알고리즘은 두 대의 엘리베이터의 목적지와 call type을 반환한다. 소프트웨어는 이제 각 엘리베이터의 현재 위치와 목적지, 문이 얼마나 열려 있는지 등을 고려하여 다음 loop에 엘리베이터의 상승이나 하강 등의 거동을 결정

하게 된다. 이를 간편하게 코딩하기 위하여 본 팀은 Elevator 클래스를 정의하였다. 다음의 그림 15처럼, Elevator 클래스는 그 인스턴스로 Elevator1, Elevator2를 가진다.

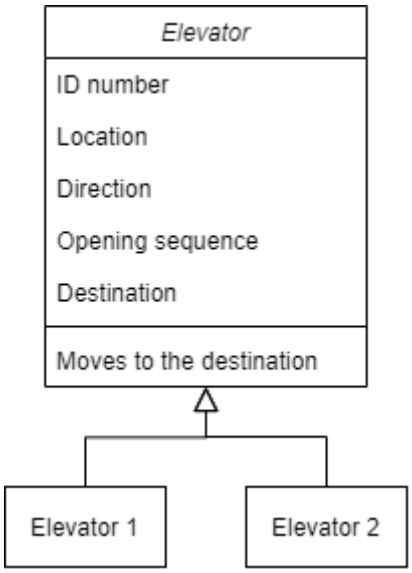


그림 15 Elevator 클래스와 그 인스턴스

Elevator 클래스의 변수들은 각 엘리베이터의 번호, 위치, 진행방향, 문이 열려있는 정도, 목적지 등을 나타낸다. 메인 알고리즘으로부터 엘리베이터의 목적지가 반환되면, "moves to the destination" 메서드를 사용하여 인스턴스에게 각각의 목적지를 입력해주고 자신의 위치와 목적지의 위치를 비교하여 엘리베이터는 다음 loop의 거동을 결정하게 된다. 또한, 만약 엘리베이터가 목적지에 도착하였다면 "door open" 메서드를 사용하여 엘리베이터의 문이 열린 정도(opening sequence)를 최대로 설정한다. 그리고 loop의 끝에선 "door close" 메서드를 사용하여 opening sequence가 0 이상이라면 그 값을 감소시켜 엘리베이터의 문이 서서히 닫히도록 하였다.

엘리베이터 구현 소프트웨어를 코딩하며, 본 팀은 이 프로젝트 전체를 관통하는 몇 가지의 전제를 가정하였다. 이는 다음과 같다.

- 하나의 Call 은 한 명의 승객과 일대일 대응한다.

  버튼 입력만을 가지고는 한 명의 승객이 특정 경로로 이동하는 경우와 여러 명의 승객이 동일한 경로로 이동하는 경우를 구분할 수 없기 때문에, 하나의 call에는 한 명의 승객만이 존재한다고 가정하였다.

- 엘리베이터는 등속도로 이동한다.

  시간과 위치계산의 편의를 위하여, 그리고 GUI 구현의 편의를 위하여 엘리베이터는 평균속도인 1m/s로 등속운동 한다고 가정하였다. 하지만 이는 소비전력에는 크게 영향을 주므로, 소비전력을 계산할 때에는 엘리베이터의 실제 거동에 맞춰서 계산하였다.

엘리베이터 거동 구현 소프트웨어의 전체적인 블록 다이어그램은 다음과 같다. 연두색 점선은 하나의 함수를, 분홍색 글씨는 클래스의 메서드를 의미한다.

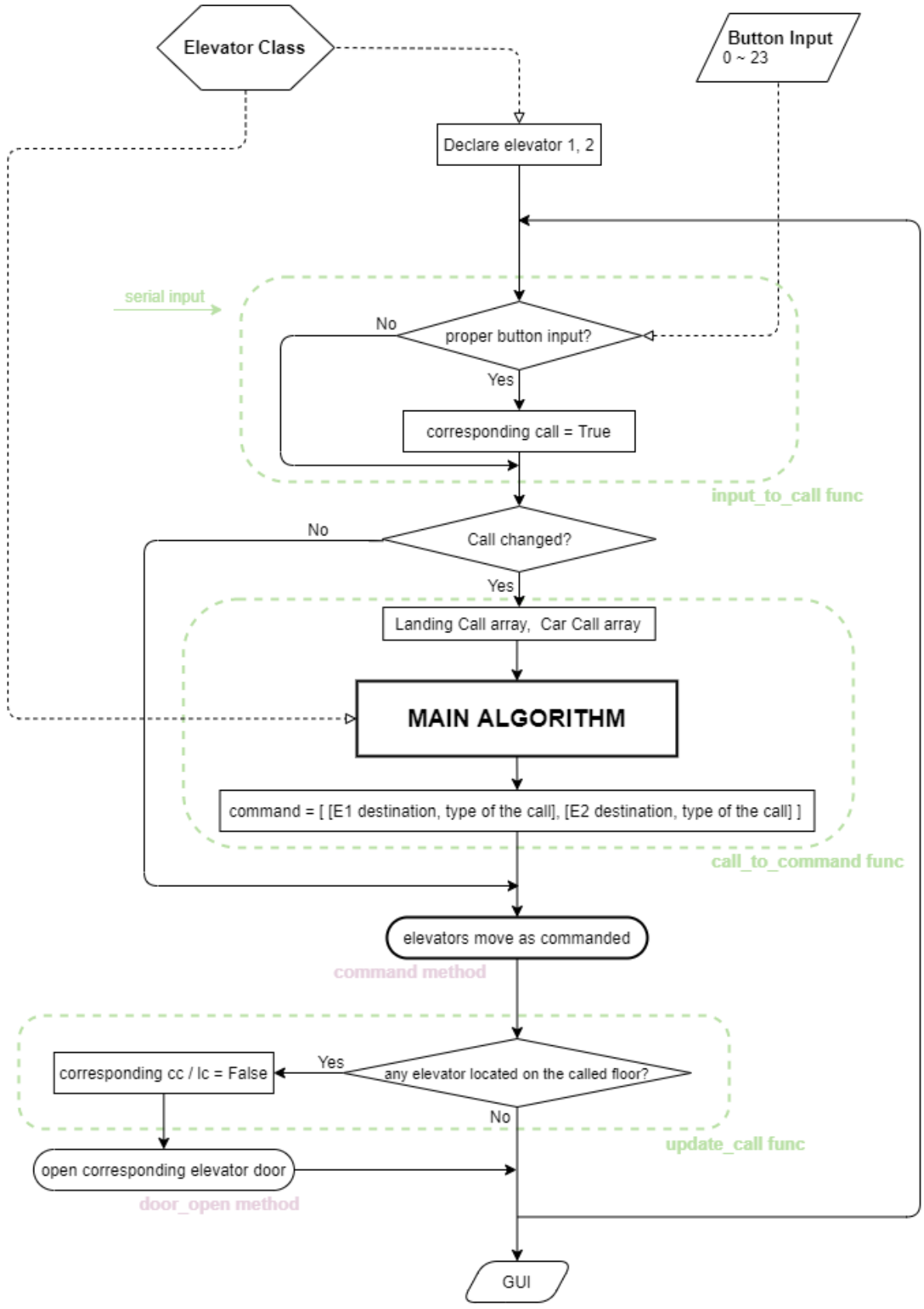


그림 16 엘리베이터 구현 소프트웨어의 블록 다이어그램

2. 엘리베이터 GUI

가. pygame

pygame이란 python으로 작성한 프로그램에 대한 SDL 기반의 멀티미디어 표현을 위한 라이브러리다. 오픈 소스로 제공되며, 다양한 OS를 지원하고, 여러 형식의 입력, 사운드 재생 등 다양한 기능을 탑재한 GUI 모듈이다.

나. 구성

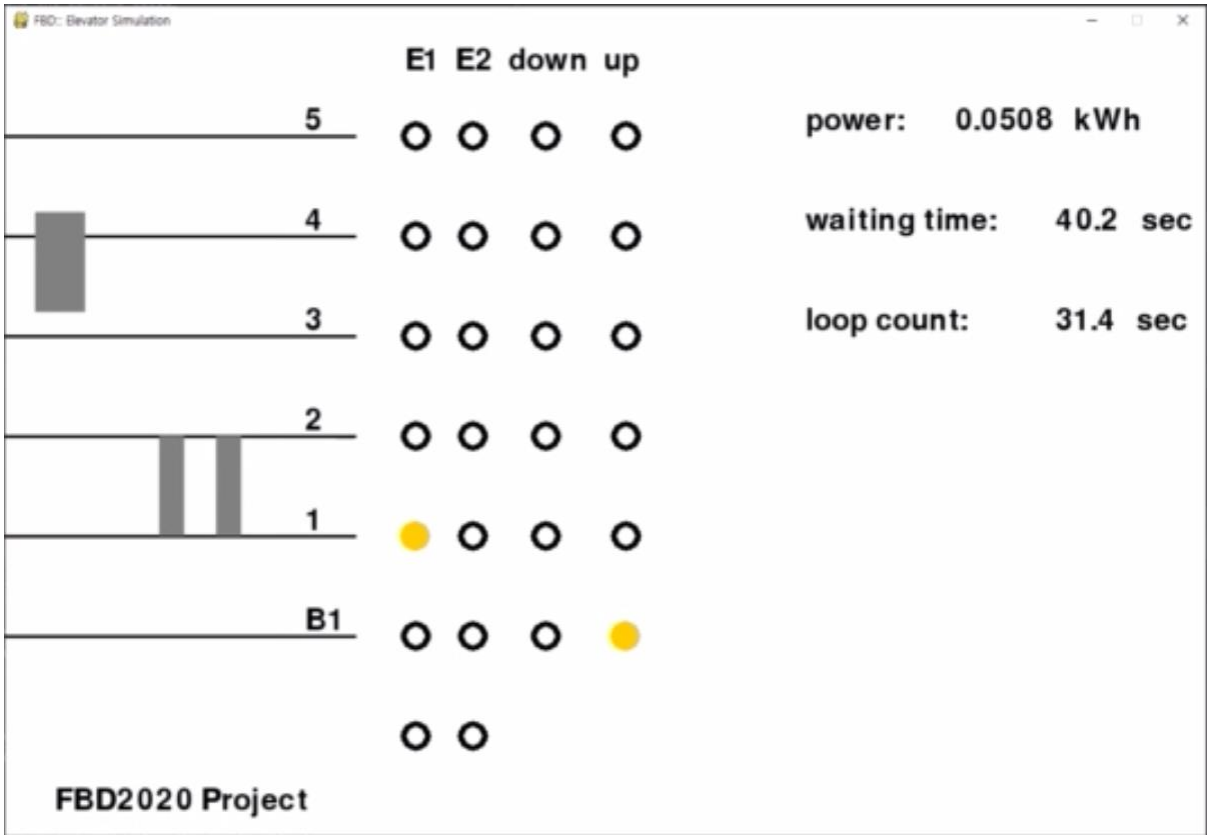


그림 17 엘리베이터 구현 소프트웨어의 GUI

- 배경

  지하 1 층부터 5 층까지 구성된 건물을 묘사하였다.

- 입출력

  실시간으로 입력받고 처리하는 call 의 상태와 엘리베이터의 이동을 표시하였다. 또한, 엘리베이터의 문 사이의 간격을 opening sequence 와 비례하도록 하여 엘리베이터 문의 여닫힘을 표현하였다.

- 평가 지표

  실시간으로 실행시간과 소비 전력, 대기시간을 표시하였다.

V. 운행 알고리즘 개발 및 분석

1. 엘리베이터의 거동 평가지표

가. 대기시간

먼저, 엘리베이터의 버튼을 누른 모든 승객의 대기시간을 첫 번째 평가지표로 사용하기로 하였다. 여기서 모든 승객이란 엘리베이터를 기다리는 승객과 엘리베이터 안에서 목표층으로 이동 중인 승객 모두를 말하며 이들의 시간(대기 시간, 이동 시간)의 총합을 계산하여 데이터로 활용한다. 승객이 엘리베이터 이용을 위해 버튼을 누르게 되면 버튼의 상태가 바뀌는 점을 이용하여 시간을 산출하였다. 매 루프마다 모든 버튼(CC, LC)의 값을 검사하여 활성화되어 있는 버튼의 수를 구한다. 이후 이 값만큼의 시간을 누적하여 총 대기시간을 구해 평가지표로 이용하였다. 따라서 n번째 승객의 대기시간을 T_n이라 할 때, 대기시간 평가지표 E_{time}은 다음과 같이 계산된다.

$$E_{time} = \sum_{n=1} T_n \qquad (1)$$

나. 소비전력

두 번째 평가지표로는, 엘리베이터를 움직이는 데 필요한 총 소비전력을 선정하였다. 이를 계산하고 평가지표로 사용하기 위해서, 엘리베이터의 거동에 따라 전력의 소모량이 어떻게 달라지는지에 대해 조사를 진행하였다. 새천년관의 엘리베이터의 경우 정격하중이 1350kg이었지만, 정격하중이 1350kg인 엘리베이터의 소비전력 그래프를 찾을 수 없어, 1800kg인 엘리베이터의 소비전력 그래프[1]를 참고하였다. 이 그래프를 공식화에 용이하도록 간소화한 뒤에, Matlab을 이용하여 그래프로 출력하였다(그림 18). 그 후, 승객의 총 무게에 따른 등속도 구간의 소비전력이 무게에 대해 선형으로 분포한다고 가정하여, 그림 19와 같이 엘리베이터 구현 소프트웨어에 적용될 수 있도록 수정하였다.

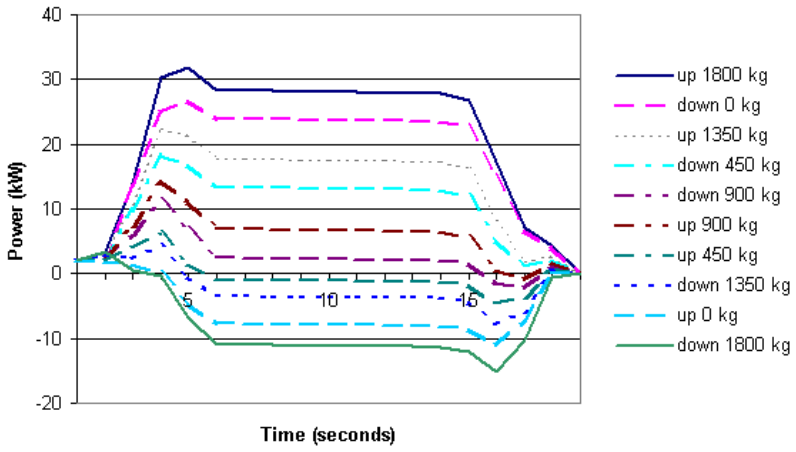


그림 18 정격하중이 1800kg인 엘리베이터의 소비전력

[1] Lutfi Al-Sharif, Elevator Energy Simulation Model, 2004

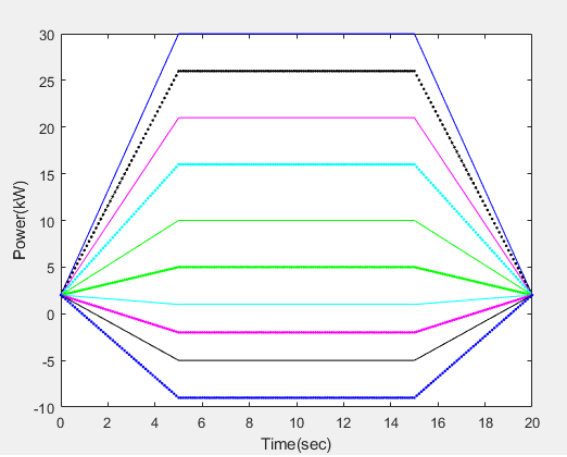


그림 19 간소화한 소비전력 그래프

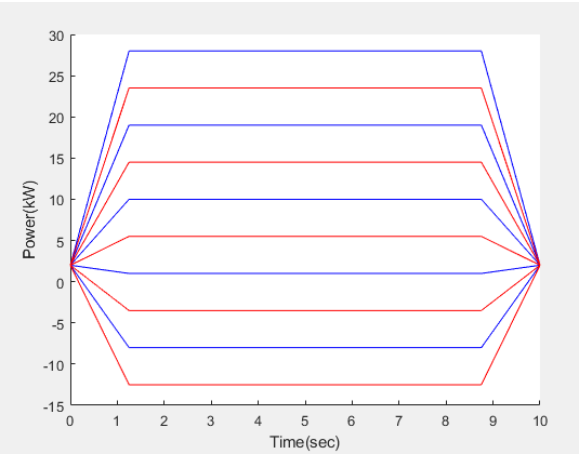


그림 20 엘리베이터 구현 소프트웨어에 적용될 수 있도록 선형화한 그래프

그림 19에서, 엘리베이터의 변위가 s이고 승객의 무게가 w일 때, 시간 t에 대한 엘리베이터의 소비전력함수를 $p(t,s,w)$라고 하자. 이때, 엘리베이터가 한 번 이동할 때 소모되는 전력의 총합은 $P(s,w)$로, 다음과 같이 계산된다.

$$P(s,w) = \int_0^{\frac{s}{v}} p(t,s,w)dt \qquad (2)$$

따라서, i번째 엘리베이터의 k번째 이동에서의 변위를 $s_{i,k}$, 승객의 무게를 $w_{i,k}$라 하면, 두 엘리베이터의 소비전력을 더한 소비전력 평가지표 E_{time}은 다음과 같이 계산된다. 모든 계산에서, 승객 1명의 무게는 70kg으로 계산하였다.

$$E_{power} = \sum_{k=1} P(s_{1,k}, w_{1,k}) + \sum_{k=1} P(s_{2,k}, w_{2,k}) \qquad (3)$$

2. 공통 평가상황 정의

설계한 운행 알고리즘의 성능을 평가하기 위해 공통 평가상황을 아래와 같이 정의하였다. 총 4가지의 시뮬레이션 시나리오를 설계하였으며, 알고리즘에 따라 시나리오 구현이 불가한 경우, 일부 수정을 거친 뒤 진행하도록 하였다.

① 출근 시간

대부분 1 층에서 위층으로 이동하는 승객이 많은 시간대이다. 따라서 1 층에서 3~5 층으로 이동하는 승객 4 명을 배치하였다. 또한 예외의 상황을 주기 위해, 5 층에서 1 층으로 이동하는 승객과 B1 층에서 2 층으로 이동하는 승객을 추가하였다.

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ
호출 시각(초)	10	10	15	25	26	30
탑승 위치	1	1	1	5	1	B1
목적 위치	4	5	3	1	3	2

② 퇴근 시간

대부분 위쪽에서 저층으로 이동하는 승객이 많은 시간대이다. 따라서 4~5 층에서 1~B1 층으로 이동하는 승객 5 명을 배치하였다, 또한 예외의 상황을 주기 위해, 1 층에서 5 층으로 이동하는 승객과 3 층에서 5 층으로 이동하는 승객을 추가하였다.

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅅ
호출 시각(초)	10	10	12	20	24	30	30
탑승 위치	5	5	4	1	3	5	5
목적 위치	1	B1	1	5	5	1	B1

③ 점심 시간

올라가는 승객과 내려가는 승객이 골고루 분포하는 상황이다. 따라서 1 층으로 내려가는 승객 3 명과 1 층에서 올라가는 승객 3 명을 배치하였다. 또한 예외의 상황을 주기 위해, 4 층에서 B1 층으로 이동하는 승객을 추가하였다.

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅅ
호출 시각(초)	10	10	13	18	18	24	30
탑승 위치	5	2	1	3	1	1	4
목적 위치	1	1	4	1	5	4	B1

④ 한산한 시간

엘리베이터를 이용하는 승객이 드문 시간대에서의 상황이다. 승객의 탑승 위치와 목적 위치도 임의로 부여함으로써 운행 알고리즘의 성능을 평가한다.

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ
호출 시각(초)	10	40	70	100	130
탑승 위치	5	1	2	3	1
목적 위치	1	4	B1	1	B1

3. 운행 알고리즘 평가결과 및 분석

가. Binary Allocation Algorithm

1) 알고리즘의 작동원리

본 알고리즘은 이중 할당에 중점을 둔 알고리즘이다. 여기서 이중 할당이란, car call이 어느 엘리베이터에 종속되지 않고, 실시간으로 가장 빨리 처리할 수 있는 엘리베이터에게 배정되는 방식을 의미한다. 알고리즘의 작동 방식을 아래의 상자에 나타냈고, 다음 장에 알고리즘의 전체적인 블록 다이어그램을 첨부하였다.

1. 이중 리스트 calls[[],[]]에 각각 e1과 e2에 배정할 수 있는 call(cc1, cc0, lc)을 저장

2. e1과 e2가 모든 call을 해결한 상태에서 들어온 call인지 확인 후, 그렇다면 둘 중 가까운 엘리베이터 배정

3-1) 중복 할당을 피하기 위해 e1에 배정할 수 있는 call list인 calls[0]에서 이전에 e2가 실행하던 call을 삭제

3-2) calls[0]이 비어 있다면 e1은 현상태 유지

3-3) e1이 어느 목적지에 도착하여 문이 열려 있는 상태라면 현 상태 유지

3-4) 이전에 움직이던 방향을 참고하여 다음에 움직일 방향 선택

3-4.a) 이전에 올라가고 있었다면, 현재 층보다 위에 call이 있으면 올라가야 하고 아니면 내려가야 한다.

3-4.b) 이전에 내려가고 있었다면, 현재 층보다 아래에 call이 있으면 내려가야 하고 아니면 올라가야 한다.

3-5) 이동해야 하는 방향에서 가장 먼저 해결할 call 배정

3-5.a) 올라가는 경우

➔ 1순위: 가장 가까운 cc1 or lc

➔ 2순위: 가장 멀리 있는 cc0

3-5.b) 내려가는 경우

➔ 1순위: 가장 가까운 cc0 or lc

➔ 2순위: 가장 멀리 있는 cc1

4-1) 중복 할당을 피하기 위해 e2에 배정할 수 있는 call list인 calls[1]에서 e1에 배정한 call을 삭제

4-2) e2에 대해 3-2)와 동일

4-3) e2에 대해 3-3)과 동일

4-4) e2에 대해 3-4)와 동일

4-5) e2에 대해 3-5)와 동일

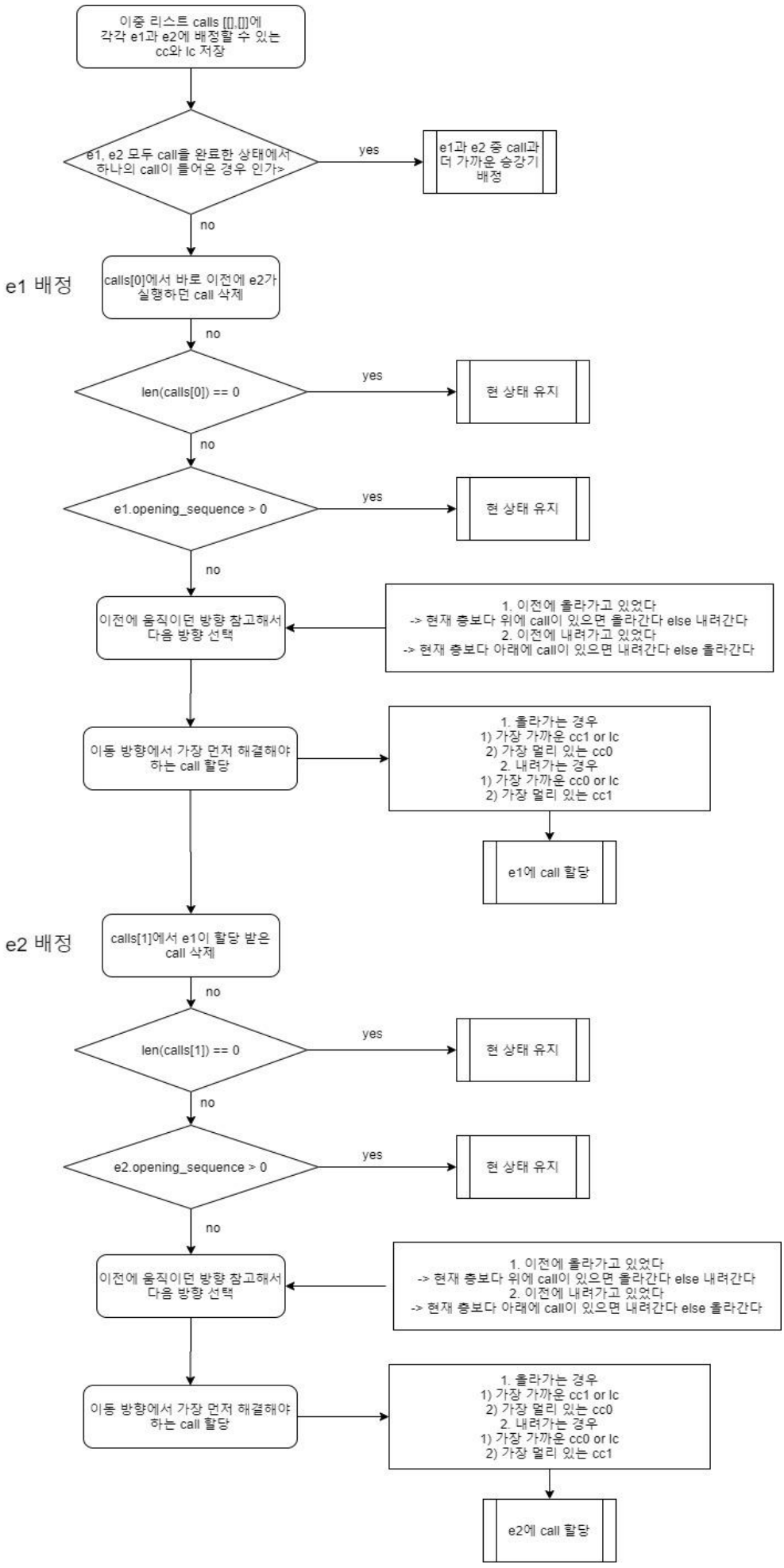
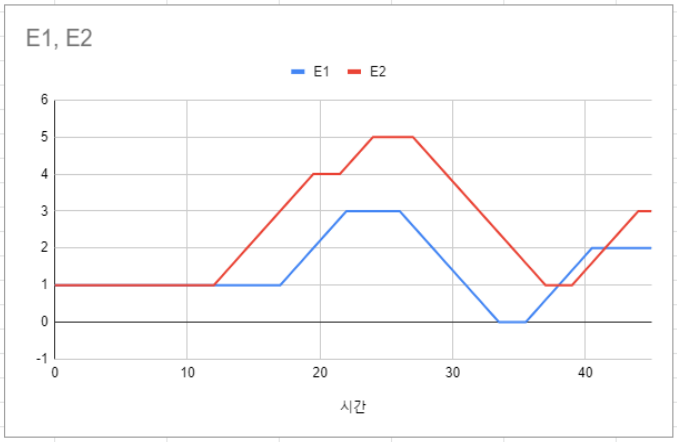


그림 21 Binary Allocation Algorithm의 블록 다이어그램

2) 평가상황에서 엘리베이터의 거동

① 출근 시간

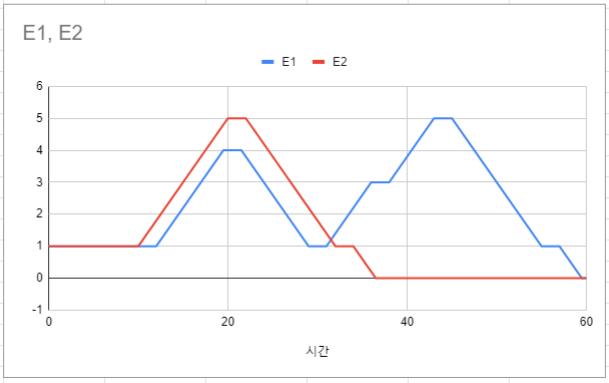
승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ
탑승 위치	1	1	1	5	1	B1
목적 위치	4	5	3	1	3	2
호출 시각	10	10	15	25	26	30
탑승 시각	10	10	15	25	37	33.5
도착 시각	19.5	24	22	37	44	40.5
사용 승강기	E2	E2	E1	E2	E2	E1
소비전력	0.095kWh		대기시간	71		



대부분의 시간 동안 E1과 E2가 동일한 방향으로 움직이는 모습을 볼 수 있다. 출근 시간임을 고려하면 동일한 방향으로 이동하는 승객이 많은 상황에서 승객이 기다리는 시간을 단축했음을 알 수 있다.

② 퇴근 시간

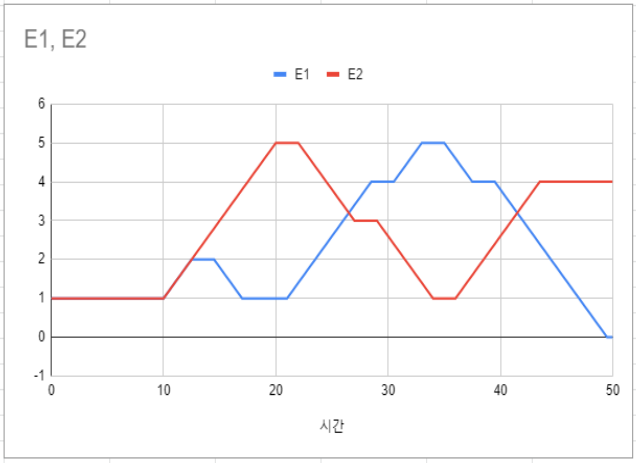
승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅅ
탑승 위치	5	5	4	1	3	5	5
목적 위치	1	B1	1	5	5	1	B1
호출 시각	10	10	12	20	24	30	30
탑승 시각	20	20	19.5	29	36	43	43
도착 시각	32	36.5	29	43	43	55	59.5
사용 승강기	E2	E2	E1	E1	E1	E1	E1
소비전력	0.125kWh		대기시간	162			



10초에서 30초 사이에는 E1과 E2가 동일한 방향으로 움직이는 모습을 볼 수 있다. 퇴근 시간임을 고려하면 동일한 방향으로 이동하는 승객이 많은 상황에서 승객이 기다리는 시간을 단축했음을 알 수 있다. 또한 30초 이후에는 E1만 고층으로 이동하여 call을 한 번에 처리함으로써 엘리베이터의 소비전력을 감소시켰다.

③ 점심 시간

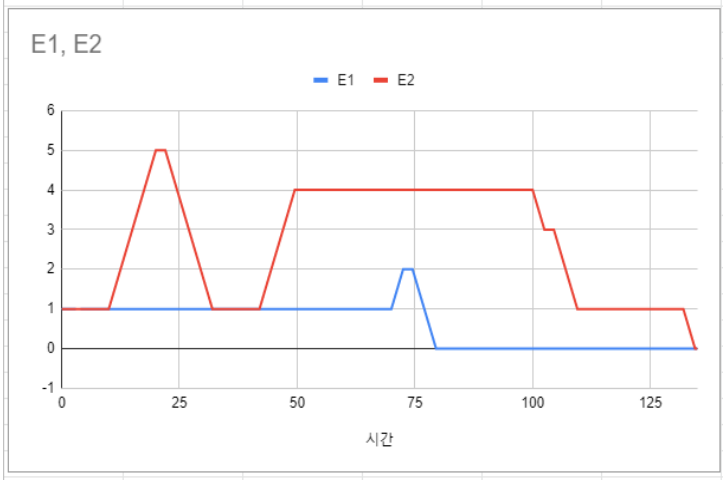
승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅅ
탑승 위치	5	2	1	3	1	1	4
목적 위치	1	1	4	1	5	4	B1
호출 시각	10	10	13	18	18	24	30
탑승 시각	20	12.5	17	27	19	34	37.5
도착 시각	34	17	28.5	34	33	43.5	49.5
사용 승강기	E2	E1	E1	E2	E1	E2	E1
소비전력	0.118kWh		대기시간	116.5			



점심 시간은 내려오는 승객과 올라가는 승객이 골고루 분포하는 시간대임을 고려하면 엘리베이터의 이동 그래프는 이상적이다. E1과 E2가 서로 다른 방향으로 이동하며 승객의 요청을 효율적으로 처리하고자 한다.

④ 한산한 시간

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ
탑승 위치	5	1	2	3	1
목적 위치	1	4	B1	1	B1
호출 시각	10	40	70	100	130
탑승 시각	20	40	72.5	102.5	130
도착 시각	32	49.5	79.5	109.5	134.5
사용 승강기	E2	E2	E1	E2	E2
소비전력	0.199kWh		대기시간	55	



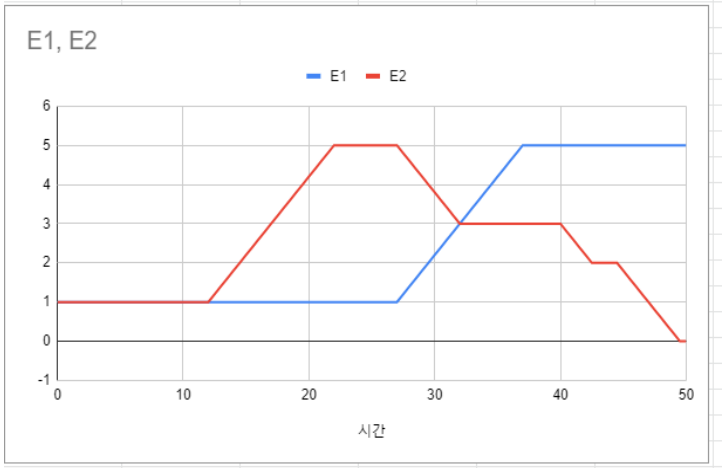
한산한 시간에는 대부분의 경우, 두 엘리베이터가 멈춰 있을 때, 새로운 call이 들어온다. 따라서 call에 가까운 엘리베이터를 배정함으로써 효율적으로 운행했음을 알 수 있다.

3) 평가결과 분석

승객이 많은 상황에서는 두 개의 엘리베이터를 동시에 효율적으로 운행하는 것을 볼 수 있다. 방향이 같은 승객이 많은 출근/퇴근 시간에는 두 엘리베이터가 동시에 같은 방향으로 이동하고, 방향이 다른 승객이 많은 점심시간에는 두 엘리베이터가 동시에 다른 방향으로 이동한다. 이로써 승객이 엘리베이터를 기다리는 시간과 목적지에 도착하는 시간을 단축해 대기 시간에서 이득을 보인다. 따라서 다른 알고리즘에 비해 들어온 call을 빠르게 처리하고 다음 call을 기다리는 상태가 된다. 해당 상태에서 새로운 call을 받으면 가까운 엘리베이터를 배정하기 때문에 소비전력에서의 이득도 최대화가 된다.

4) 특정상황 정의 및 결과분석

승객	ㄱ	ㄴ	ㄷ	ㄹ
탑승 위치	1	1	5	2
목적 위치	5	5	3	B1
호출 시각	10	25	25	40
탑승 시각	10	25	25	42.5
도착 시각	22	37	32	49.5
사용 승강기	E2	E2	E1	E2
소비전력	0.048kWh		대기시간	40.5



승객 ㄴ과 ㄷ이 동시에 엘리베이터를 이용하는 25초 이후를 살펴보면, 서로 다른 방향으로 이동해야 하는 상황을 잘 처리하고 있다. 그리고 두 엘리베이터가 모두 멈춘 40초에 2층에서 들어온 call을 해결하기 위해 더 가까이에 있는 E2가 이동한다. 이로써 대기시간과 소비전력의 측면에서 모두 이득을 보는 해당 알고리즘의 장점을 잘 보여준다

나. Binary Allocation Algorithm + Default floor

1) 알고리즘의 작동원리

본 알고리즘은 Binary Allocation Algorithm과 동일하게 작동하며, default floor 배정 기능이 추가되었다. default floor 배정 기능이란 두 엘리베이터가 15초 동안 이동이 없는 경우, default floor인 1층으로 자동으로 이동하는 것을 의미한다. 매 루프마다 default_count 함수를 통해 두 엘리베이터가 정지하고 있는 시간을 저장하는 변수인 default_time의 값을 조절했고, 이 변수를 조건 변수로 하여 엘리베이터의 움직임을 제어했다. 아래에는 Binary Allocation Algorithm에서 추가된 코드의 블록 다이어그램을 나타냈다.

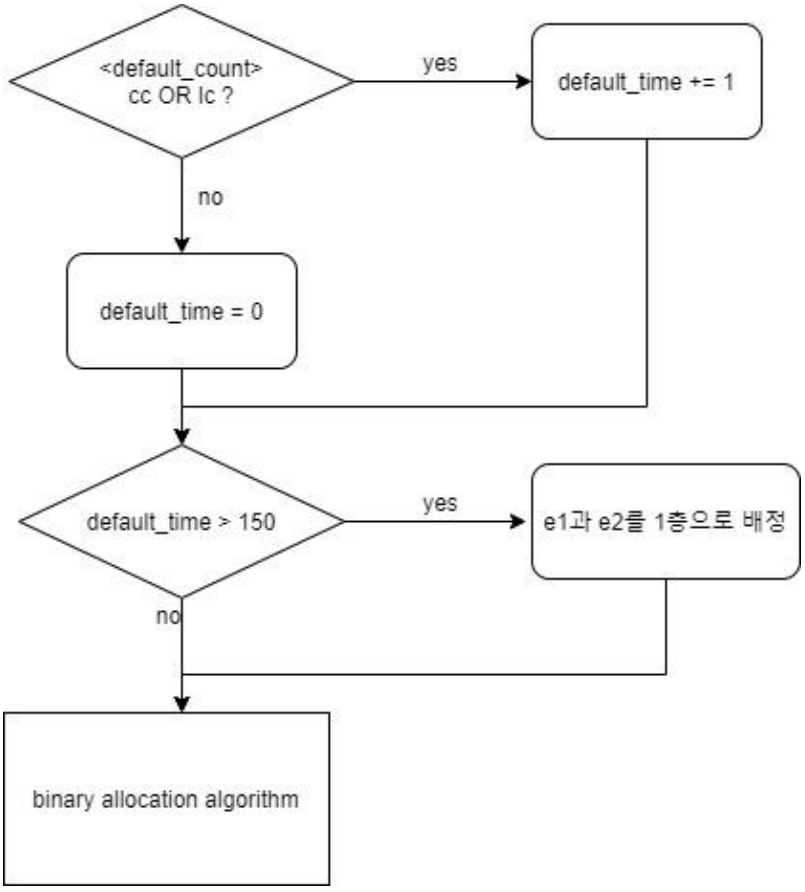


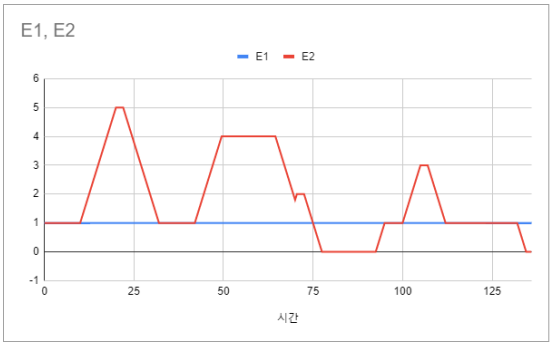
그림 22 Binary Allocation Algorithm에서 추가된 코드의 블록 다이어그램

2) 평가상황 및 특정상황 정의 및 결과분석

출근 시간, 퇴근 시간, 그리고 점심 시간에 대해서는 binary allocation algorithm과 동일하기에 생략한다.

① 한산시간

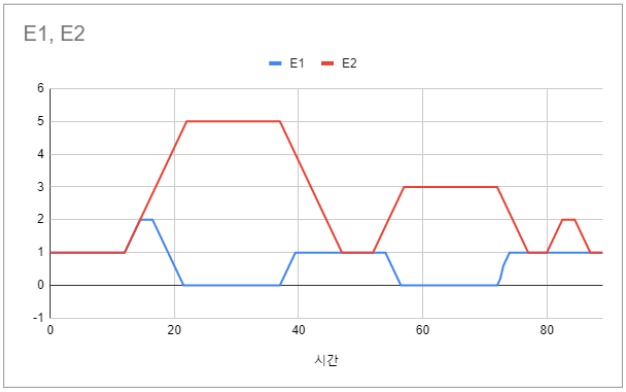
승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ
탑승 위치	5	1	2	3	1
목적 위치	1	4	B1	1	B1
호출 시각	10	40	70	100	130
탑승 시각	20	40	70.5	105	130
도착 시각	32	49.5	77.5	112	134.5
사용 승강기	E2	E2	E2	E2	E2
소비전력	0.205kWh		대기시간	55.5초	



Binary Allocation Algorithm과는 다르게, 한산한 시간에는 default 기능이 작동하여 계속해서 E2가 1층으로 이동하게 된다. 이 과정에서 가까운 층에 승객이 있는 경우 바로 그 call을 배정받고 이동하는 것을 볼 수 있다. 1층으로 이동 후 call이 들어오기 때문에 E2가 계속 배정받고 E1이 계속해서 1층에 머물러 있는 현상을 볼 수 있다.

② 특정상황

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ
탑승 위치	1	2	1	1	2
목적 위치	5	B1	3	B1	1
호출 시각	10	12	50	52	80
탑승 시각	10	14.5	50	52	82.5
도착 시각	22	21.5	57	56.5	87
사용 승강기	E2	E1	E2	E1	E2
소비전력	0.1591kWh		대기시간	40.0초	



Default 기능을 최대한 활용하기 위한 상황으로 대부분의 이동이 1층에서 이루어지도록 하였다. 이로 인해 미리 1층으로 이동한 엘리베이터를 바로 이용하였기에 대기시간이 단축된 것을 볼 수 있다.

다. High-Low Split Algorithm

1) 알고리즘의 작동원리

본 알고리즘은 고층과 저층을 나누어서 운행하는 알고리즘이다. 베이스는 binary allocation algorithm을 따르며, E1은 B1, 1, 4, 5층을 운행, E2는 B1, 1, 2, 3층을 운행한다. 알고리즘의 작동 방식을 아래의 상자에 나타냈고, 다음 장에 알고리즘의 전체적인 블록 다이어그램을 첨부하였다.

1. 이중 리스트 calls[[],[]]에 각각 e1과 e2에 배정할 수 있는 call(cc1, cc0, lc)을 저장.

(고층과 저층을 따로 운행하는 점 고려)

2. e1과 e2가 모든 call을 해결한 상태에서 들어온 call인지 확인 후, 그렇다면 둘 중 가까운 엘리베이터 배정

3-1) 중복 할당을 피하고자 e1에 배정할 수 있는 call list인 calls[0]에서 이전에 e2가 실행하던 call을 삭제

3-2) calls[0]이 비어 있다면 e1은 현 상태 유지

3-3) e1이 어느 목적지에 도착하여 문이 열려 있는 상태라면 현 상태 유지

3-4) 이전에 움직이던 방향을 참고하여 다음에 움직일 방향 선택

3-4.a) 이전에 올라가고 있었다면, 현재 층보다 위에 call이 있으면 올라가야 하고 아니면 내려가야 한다.

3-4.b) 이전에 내려가고 있었다면, 현재 층보다 아래에 call이 있으면 내려가야 하고 아니면 올라가야 한다.

3-5) 이동해야 하는 방향에서 가장 먼저 해결할 call 배정

3-5.a) 올라가는 경우

➔ 1순위: 가장 가까운 cc1 or lc

➔ 2순위: 가장 멀리 있는 cc0

3-5.b) 내려가는 경우

➔ 1순위: 가장 가까운 cc0 or lc

➔ 2순위: 가장 멀리 있는 cc1

4-1) 중복 할당을 피하고자 e2에 배정할 수 있는 call list인 calls[1]에서 e1에 배정한 call을 삭제

4-2) e2에 대해 3-2)와 동일

4-3) e2에 대해 3-3)과 동일

4-4) e2에 대해 3-4)와 동일

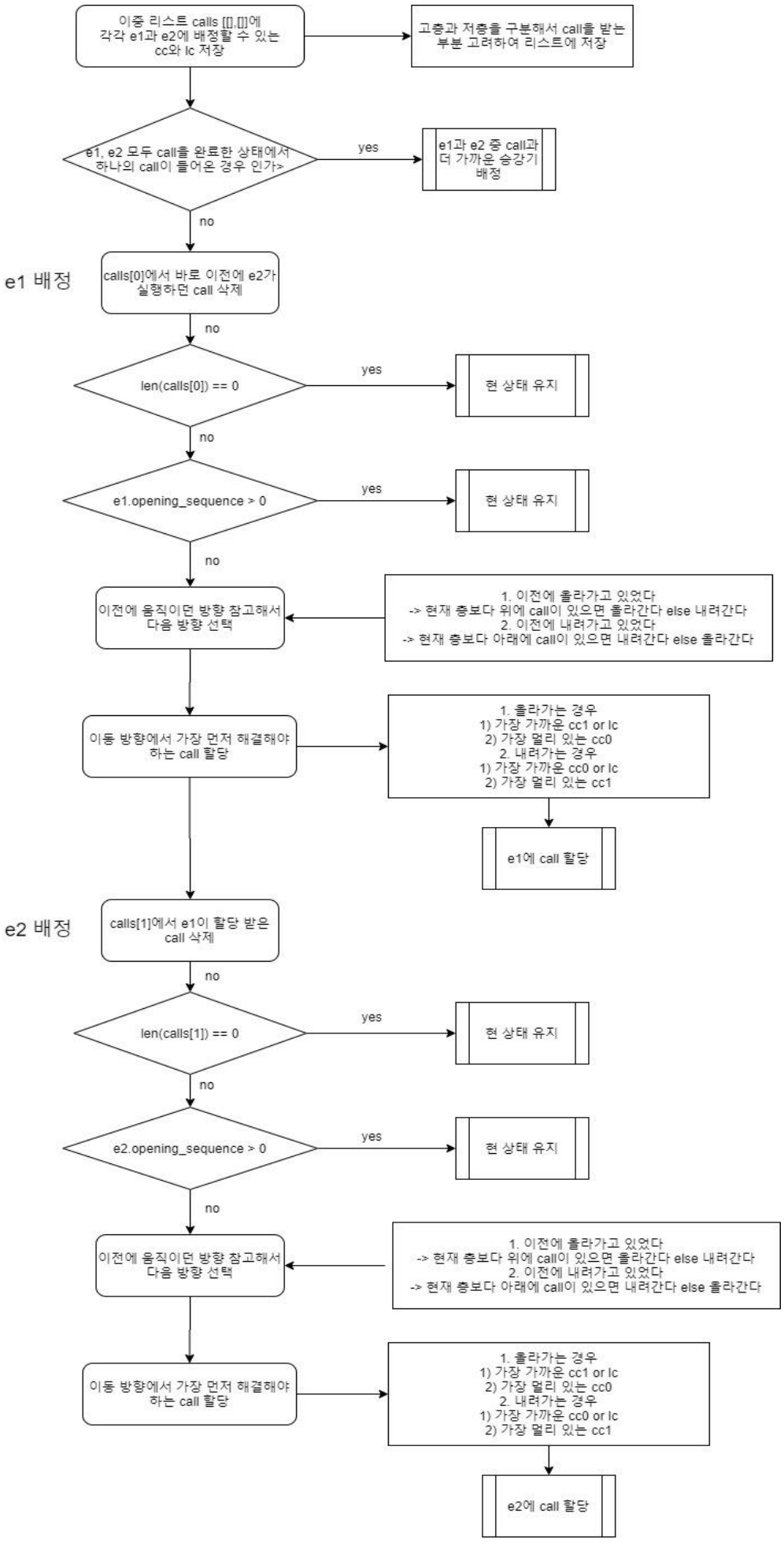
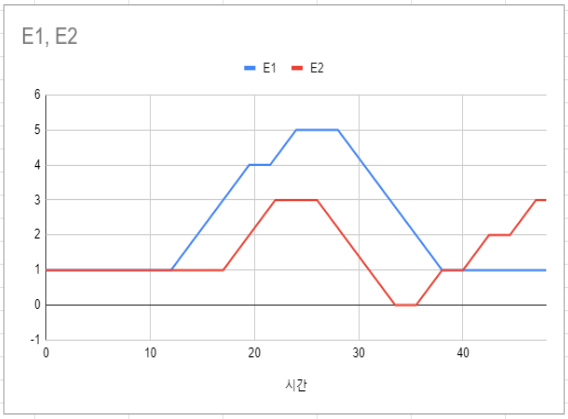


그림 23 High-Low Split Algorithm의 블록 다이어그램

2) 평가상황에서 엘리베이터의 거동

① 출근 시간

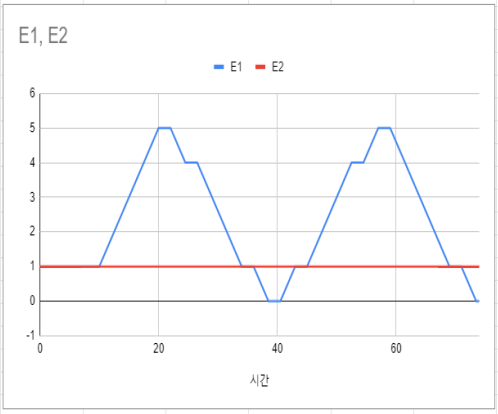
승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ
탑승 위치	1	1	1	5	1	B1
목적 위치	4	5	3	1	3	2
호출 시각	10	10	15	25	26	30
탑승 시각	10	10	15	26	38	33.5
도착 시각	19.5	24	22	38	47	42.5
사용 승강기	E1	E1	E2	E1	E2	E2
소비전력	0.117kWh		대기시간	77		



대부분의 시간 동안 E1과 E2가 동일한 방향으로 움직이는 모습을 볼 수 있다. 출근 시간임을 고려하면 동일한 방향으로 이동하는 승객이 많은 상황에서 승객이 기다리는 시간을 단축했음을 알 수 있다.

② 퇴근시간

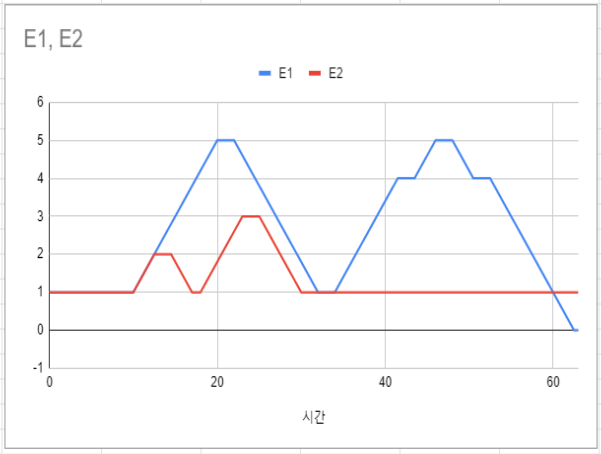
승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅅ
탑승 위치	5	5	4	1	4	5	5
목적 위치	1	B1	1	5	5	1	B1
호출 시각	10	10	12	20	24	30	30
탑승 시각	20	20	24.5	43	52.5	57	57
도착 시각	34	38.5	34	57	57	69	73.5
사용 승강기	E1	E1	E1	E1	E1	E1	E1
소비전력	0.129kWh		대기시간	227			



퇴근 시간 상황에서는 모두 4층과 5층을 이용하는 승객만 있었으므로, E1만 사용된 것을 볼 수 있다. 따라서 소비전력과 대기시간에 있어 비효율적인 상황을 유발한다. 또한 승객 ㅁ의 경우, 원래의 시나리오상에서는 3층에서 5층으로 이동하는 승객으로 설정했다. 그러나 해당 알고리즘에서는 3층에서 5층으로 바로 갈 수 없으므로 3층에서 4층으로 계단으로 올라간 뒤, 4층에서 5층으로 엘리베이터를 타고 올라간 것으로 변경을 줘야 했다.

③ 점심시간

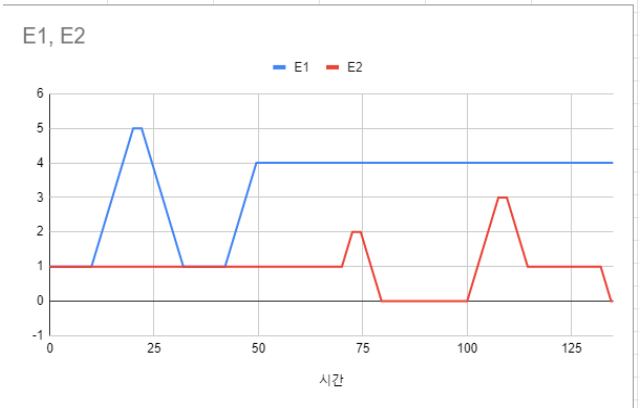
승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅅ
탑승 위치	5	2	1	3	1	1	4
목적 위치	1	1	4	1	5	4	B1
호출 시각	10	10	13	18	18	24	30
탑승 시각	20	12.5	32	23	32	32	50.5
도착 시각	32	17	41.5	30	46	41.5	62.5
사용 승강기	E1	E2	E1	E2	E1	E1	E1
소비전력	0.154kWh		대기시간	147.5			



올라가고 내려가는 승객이 모두 많은 상황인 점심 시간에는 유의미한 이동 모습을 보인다. 30초 이후로는 고층 승객만 있으므로, E1만 운행함으로써 승객 입장에서는 불편을 느낄 수 있다.

④ 한산한시간

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ
탑승 위치	5	1	2	3	1
목적 위치	1	4	B1	1	B1
호출 시각	10	40	70	100	130
탑승 시각	20	40	72.5	107.5	130
도착 시각	32	49.5	79.5	114.5	134.5
사용 승강기	E1	E1	E2	E2	E2
소비전력	0.197kWh		대기시간	55.5	



한산한 시간에는 대부분의 경우, 두 엘리베이터가 멈춰 있을 때, 새로운 call이 들어온다. 따라서 call에 가까운 엘리베이터를 배정하는 것이 이상적이나, 고층/저층을 분리함으로써 오는 엘리베이터는 정해져 있다. 따라서 승객 ㅁ은 가까이 있는 E1을 두고 멀리 있는 E2를 기다리는 불편을 겪게 된다.

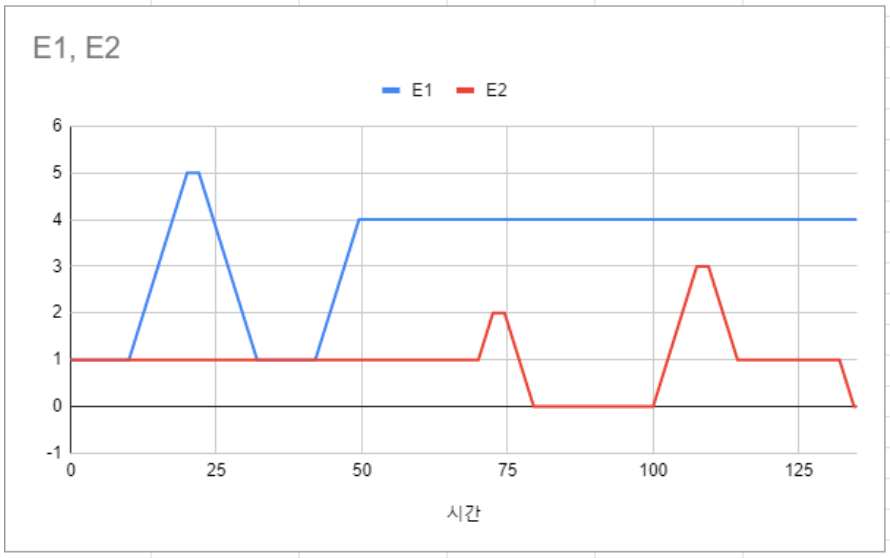
3) 평가결과 분석

공통 평가 상황 중 퇴근 시간, 점심 시간, 한산한 시간에서의 엘리베이터 이동 그래프를 보면 상당 시간 동안 하나의 엘리베이터만 사용되는 상황이 발생한다. 고층으로 이동하는 승객 혹은 고층에서 이동하는 승객이 몰리는 경우에는 효율적으로 운행할 수 있으나, 그 외 대부분의 상황에서는 엘리베이터가 운행되는 것보다 승객의 입장에서 비효율적으로 느껴지게 될 것이다.

다른 문제점으로는 위에서 언급했듯이 고층/저층을 구분함에 따라 원하는 층으로 이동하지 못하는 승객이 발생하게 된다는 점이 있다.

4) 특정상황 정의 및 결과분석

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ
탑승 위치	5	1	2	3	1
목적 위치	1	4	B1	1	B1
호출 시각	10	40	70	100	130
탑승 시각	20	40	72.5	107.5	130
도착 시각	32	49.5	79.5	114.5	134.5
사용 승강기	E1	E1	E2	E2	E2
소비전력	0.197kWh		대기시간	60	



해당 알고리즘의 특징을 보여주기 위해 극단적인 출근 시간 상황을 연출하였다. 모든 승객이 1층에서 엘리베이터를 탑승하고 각자의 목적 층에 하차한다. 고층으로 이동하는 승객을 한 번에 태워서 간다는 점에서 소비전력을 절약할 수는 있지만, 경우에 따라 대기 시간에서 막대한 손해를 본다. 예를 들어 승객 ㄷ은 E1을 약 3초 차이로 놓쳤다는 이유로 1층에서 20초 동안 엘리베이터를 기다리게 되었다.

라. Odd-Even Split Algorithm

1) 알고리즘의 작동원리

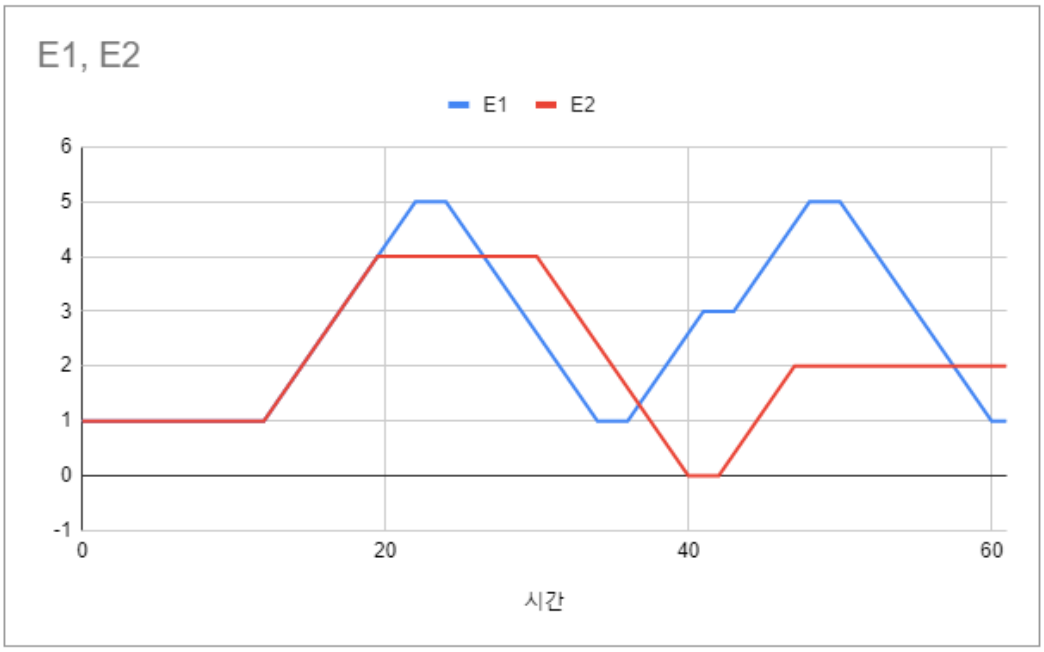
본 알고리즘은 홀수 층과 짝수 층을 나누어서 운행하는 알고리즘이다. 베이스는 binary allocation algorithm을 따르며, E1은 B1, 1, 3, 5층을 운행, E2는 B1, 1, 2, 4층을 운행한다. 알고리즘의 작동 방식과 블록 다이어그램은 High-Low Split Algorithm과 call을 배정해줄 층만 다를 뿐, 매우 유사하므로 생략하였다.

2) 평가상황에서 엘리베이터의 거동

① 출근 시간

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ
탑승 위치	1	1	1	5	1	B1
목적 위치	4	5	3	1	3	2
호출 시각	10	10	15	25	26	30
탑승 시각	10	10	34	48	34	40
도착 시각	19.5	22	41	60	41	47
사용 승강기	E2	E1	E1	E1	E1	E2

소비전력	0.1536kWh	대기시간	114.5초

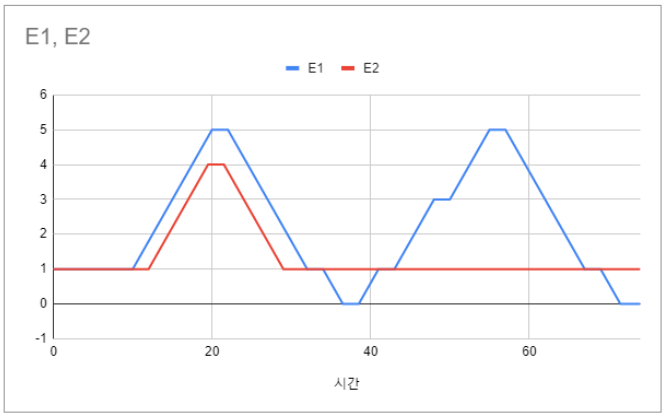


E1, E2 모두 출근 시간임에 따라 위 방향으로 움직이는 경향을 보였다. 평가상황에 홀수 층이 많음에 따라 E1이 E2에 비해 많은 움직임을 보였으며 짝수 층을 이용하는 승객이 상대적으로 E2를 이용해 적은 대기시간으로 이용할 수 있었다.

② 퇴근 시간

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅅ
탑승 위치	5	5	4	1	3	5	5
목적 위치	1	B1	1	5	5	1	B1
호출 시각	10	10	12	20	24	30	30
탑승 시각	20	20	19.5	41	48	55	55
도착 시각	32	36.5	29	55	55	67	71.5
사용 승강기	E1	E1	E2	E1	E1	E1	E1

소비전력	0.1774kWh	대기시간	210.0초

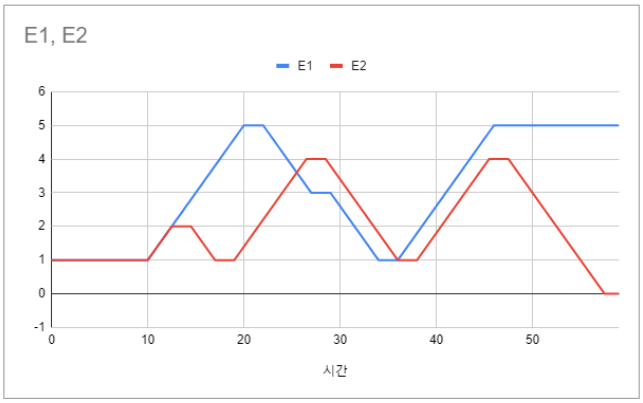


평가 상황에 대부분이 홀수 층으로 구성되어 있어 E1의 사용빈도가 높았다. 승객 ㄷ을 제외한 모든 승객이 E1을 기다려야 했기에 대기시간의 차이를 크게 보였다. 홀수 층, 짝수 층이 골고루 분배되지 않는 경우 한쪽 엘리베이터가 상대적으로 움직임의 여유가 생기는 것을 볼 수 있다.

③ 점심 시간

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅅ
탑승 위치	5	2	1	3	1	1	4
목적 위치	1	1	4	1	5	4	B1
호출 시각	10	10	13	18	18	24	30
탑승 시각	20	12.5	17	27	34	36	45.5
도착 시각	34	17	26.5	34	46	45.5	57.5
사용 승강기	E1	E2	E2	E1	E1	E2	E2

소비전력	0.125kWh	대기시간	137.5초

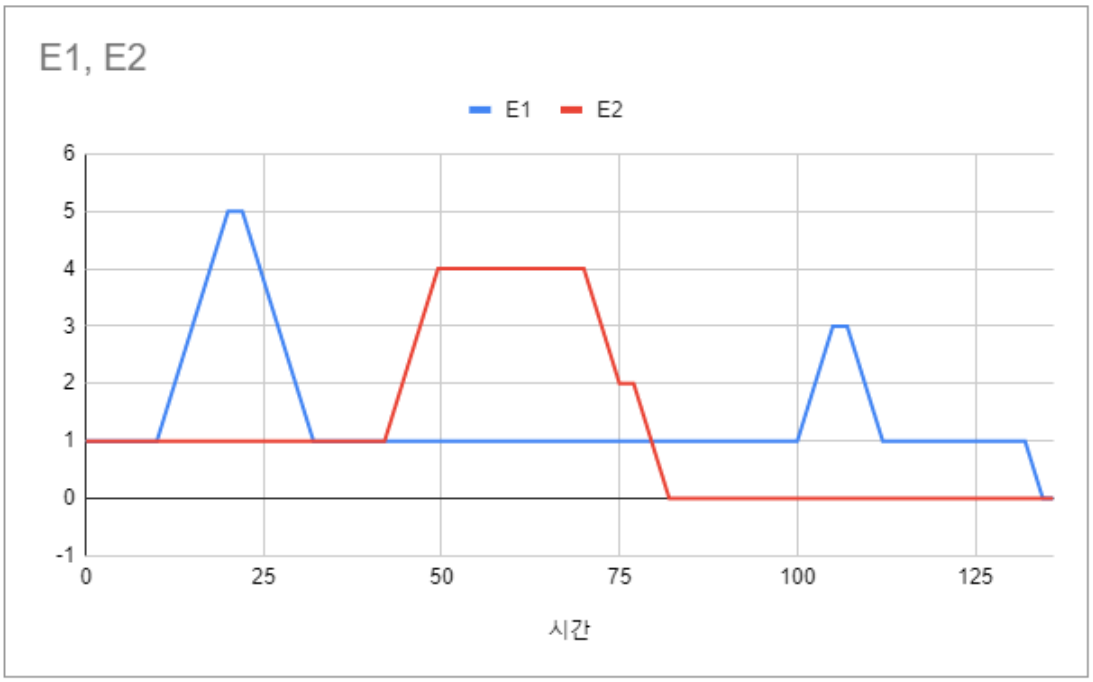


홀수, 짝수 층에 골고루 승객이 있기에 각자 엘리베이터가 유동적인 움직임을 보여주고 있다. 큰 특이점을 보이지 않고 있으며 승객들의 불편 또한 다른 시간에 비해 적은 편이다.

④ 한산한 시간

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ
탑승 위치	5	1	2	3	1
목적 위치	1	4	B1	1	B1
호출 시각	10	40	70	100	130
탑승 시각	20	40	75	105	130
도착 시각	32	49.5	82	112	134.5
사용 승강기	E1	E2	E2	E1	E1

소비전력	0.2264kWh	대기시간	60.0초



대부분 멈춰진 상태에서 call을 받기에 가까이 있는 엘리베이터가 배정되었다. 홀수/짝수 층 분리에 따라 조금의 차이는 발생하였으나 대기시간의 큰 영향은 없었다.

3) 평가결과 분석

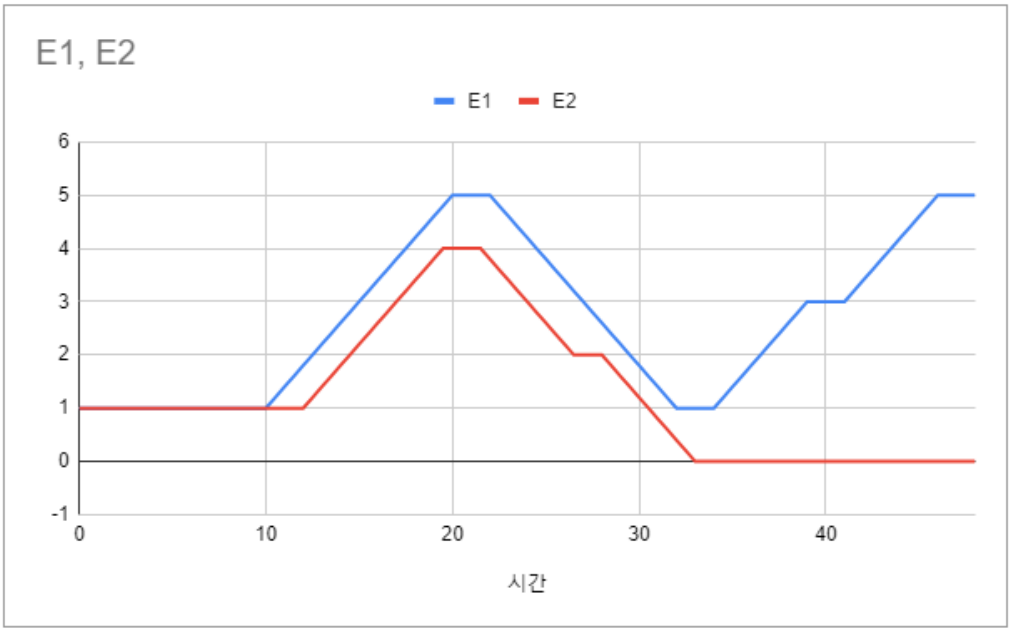
홀수, 짝수 층이 분리됨에 따라 상황 구성에 따라 유불리가 발생함을 볼 수 있다. 출근, 퇴근 시간 같은 경우 홀수 층 구성이 많아 E1을 많이 이용하게 되었다. 이에 따라 승객들의 불편함이 커지고 상대적으로 짝수 층은 수월하게 이용하였다. 점심시간 같은 경우 골고루 분배되어 있어 엘리베이터가 유동적인 움직임을 보여주었다.

엘리베이터 두 개의 운행을 특정 층으로 제한하는 경우와 다수의 승객이 여러 층에서 이용할 경우에서 효율적인 움직임을 보여주지만, 특정 층에 인원이 몰리면 대기 시간이 오히려 길어지는 현상이 발생하였다.

4) 특정상황 정의 및 결과분석

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ
탑승 위치	5	3	4	2	1	1
목적 위치	1	1	B1	B1	3	5
호출 시각	10	22	12	26	30	30
탑승 시각	20	27	19.5	26.5	32	32
도착 시각	32	32	33	33	39	46
사용 승강기	E1	E1	E2	E2	E1	E1

소비전력	0.0807kWh	대기시간	85.0초



이 알고리즘의 특징은 같은 홀수/짝수 층 방향이면 같이 탈 수 있어 시간을 절약할 수 있다는 점이다. 이 점을 살려 5층에서 3층을 지나 1층으로 이동 혹은 4층에서 2층을 지나 B1층으로 이동하는 상황을 두어 승객의 대기시간을 줄일 수 있었다.

결과적으로 두 엘리베이터가 이동 중에 새로운 승객을 태울 수 있어 승객들의 대기시간을 최소화할 수 있게 되었다.

마. Cost Comparing Algorithm

1) 알고리즘의 작동원리

Cost Comparing Algorithm은 엘리베이터가 가장 효율적인 경로를 찾는 문제를 선형 최적화(linear optimization) 문제로 가정하고, 엘리베이터 운행의 평가지표들의 합을 목적함수로 두어 최소의 목적함수 값을 갖는 경로를 선택한다. 다시 말해, 본 알고리즘은 모든 입력 값들에 대해 두 대의 엘리베이터가 이동할 수 있는 모든 경로를 탐색하고, 경로마다 비용을 계산하여 가장 낮은 비용의 경로를 선택한다. 각 경로의 총 비용을 계산할 때에는 대기시간, 소비전력, 그리고 이동방향의 변화 이렇게 3가지의 비용을 고려한다. 각 비용의 계산에 사용된 변수들은 아래에 나타냈다.

v : 엘리베이터의 속도

h_i : i번째 엘리베이터의 현재 높이

m_i : i번째 엘리베이터가 배정받은 call의 총 개수

$f_{i,k}$: i번째 엘리베이터의 k번째 목적지의 높이

$w_{i,k}$: i번째 엘리베이터가 k번째 call을 처리하기 위해 이동할 때, 승객들의 총 무게

$P(s,w)$: 승객의 총 무게가 w일 때, 엘리베이터가 s만큼 이동하는 데 소모되는 전력

① 승객들의 총 대기시간

경로의 첫 번째 비용은 승객들의 총 대기시간이다. 앞서 언급된 것처럼, 승객의 이동시간을 포함한 시간으로, 승객이 출발 층에서 버튼을 누른 순간부터 목적 층에 도착하는 순간까지의 시간을 의미한다. 하나의 call에는 한 명의 승객만이 탑승한다고 가정했으므로, 한 대의 엘리베이터가 m개의 목적지를 배당받았을 때, 현재 위치로부터 첫 번째 목적지까지 이동하는 데에 걸리는 시간은 m명의 승객이 기다려야 한다. 마찬가지로, k번째 목적지에서 다음 목적지로 이동하는 데에 걸리는 시간은 $(m-k)$명의 사람이 기다려야 한다. 따라서 i번째 엘리베이터의 시간비용 $C_t(i)$는 다음과 같이 계산된다.

$$C_t(i) = m_i \frac{(f_{i,1} - h_i)}{v} + \sum_{k=1}^{m_i-1} (m_i - k) \frac{(f_{i,k+1} - f_{i,k})}{v} \qquad (4)$$

② 엘리베이터의 총 소비전력

경로의 두 번째 비용은 엘리베이터의 이동에 소모되는 전력이다. 앞서 언급된 것처럼, 엘리베이터의 소비전력 값은 승객의 무게에 큰 영향을 받는다. 알고리즘 평가지표 계산 코드와 동일하게, 승객 한 명의 무게를 70kg으로 가정하여 계산하였으며, 평가지표 계산에서 사용한 소비전력 계산함수 $P(s,w)$함수를 사용하였다. i번째 엘리베이터의 전력비용 $C_p(i)$는 다음과 같이 계산된다.

$$C_p(i) = P(f_{i,1} - h_i, w_{i,k}) + \sum_{k=1}^{m_i-1} P(f_{i,k+1} - f_{i,k}, w_{i,k+1}) \qquad (5)$$

③ 다음 순간 엘리베이터 이동방향의 변화

경로의 세 번째 비용은 엘리베이터 이동방향의 변화이다. 대기시간과 소비전력이 가장 작은 경로를 선택하다 보면, 엘리베이터의 목적지가 변경되어 상승하던 엘리베이터가 목적지에 도달하지 않았음에도 방향을 바꾸어 하강하는 상황이 발생할 수 있다. 실제 엘리베이터에서 이런 상황이 발생하면 승객의 안전에 위협이 될 수 있으므로, 엘리베이터의 이동방향은 특정 층에 도달하기 전에는 바뀌어서는 안 된다. 따라서 엘리베이터의 이동방향을 바꾸는 경로에 방향변화비용을 부과하고, 그 가중치로 매우 큰 값을 설정하여 알고리즘이 그 경로를 선택하지 않도록 하였다.

엘리베이터의 방향변화비용에는 상승/하강하던 엘리베이터가 하강/상승하게 될 경우, 1을 부여하고, 정지 후 하강, 상승 후 정지 등의 다른 상황에는 0을 부여하였다. 따라서, i번째 엘리베이터의 방향변화비용 $C_c(i)$는 다음과 같이 계산된다.

$$C_c(i) = \begin{cases} 1, & \frac{dh_i}{dt} \cdot (f_{i,1} - h_i) < 0 \\ 0, & \frac{dh_i}{dt} \cdot (f_{i,1} - h_i) \geq 0 \end{cases} \qquad (6)$$

세 가지의 비용은 각각의 가중치를 가지며, 엘리베이터 한 대의 총 비용 $C_{total}(i)$을 계산할 때 자신의 가중치만큼 곱해져서 더해진다. 경로의 총 비용, 즉 목적함수 C_{total}은 각 엘리베이터의 총 비용 $C_{total}(i)$을 모두 더한 값으로, 다음과 같이 계산된다.

$$C_{total} = \sum_{i=1} \{ C_t(i) \cdot W_t + C_p(i) \cdot W_p + C_c(i) \cdot W_c \} \qquad (7)$$

본 알고리즘의 전체적인 블록 다이어그램은 다음과 같다.

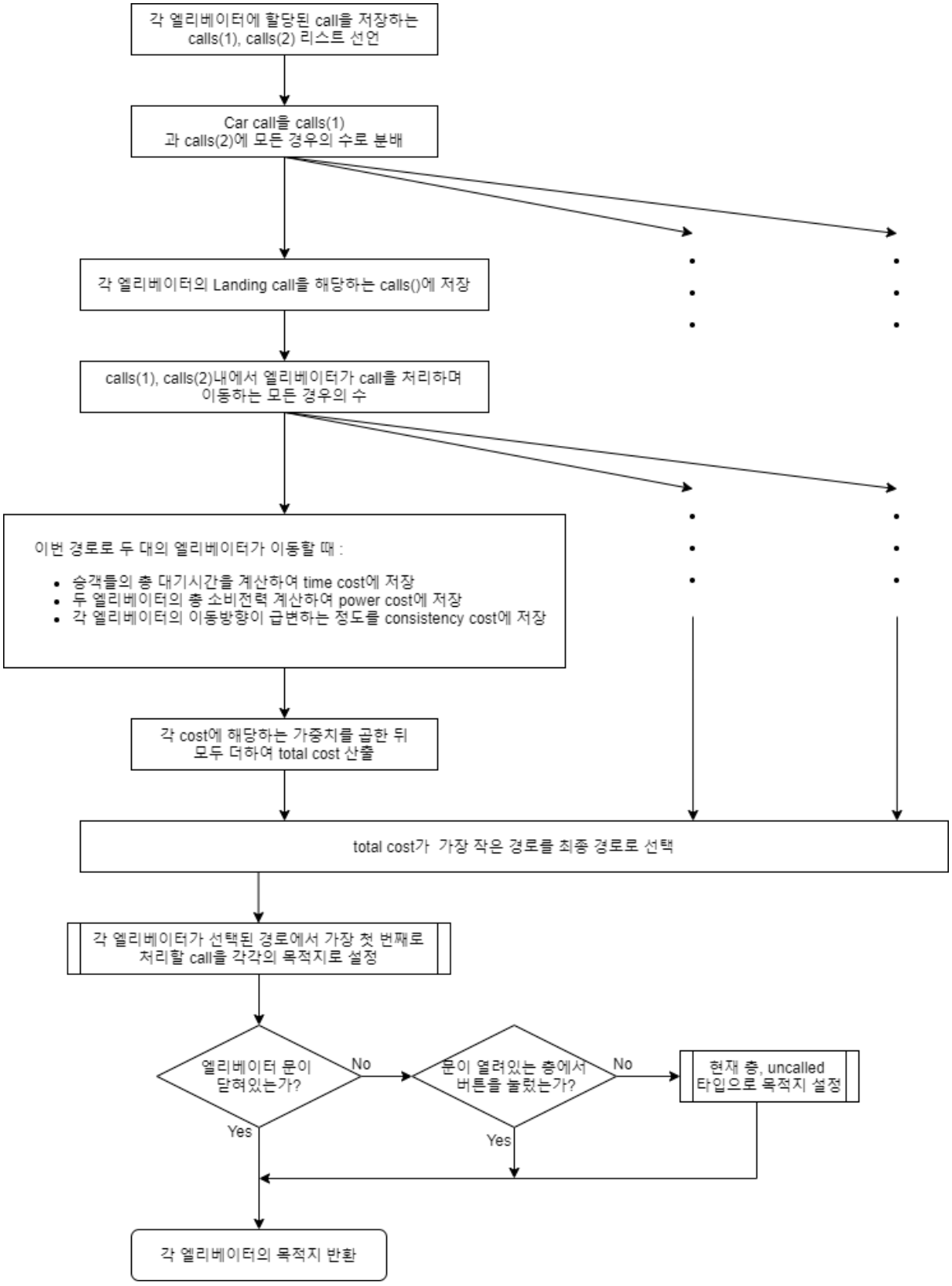


그림 24 Cost Comparing Algorithm의 블록 다이어그램

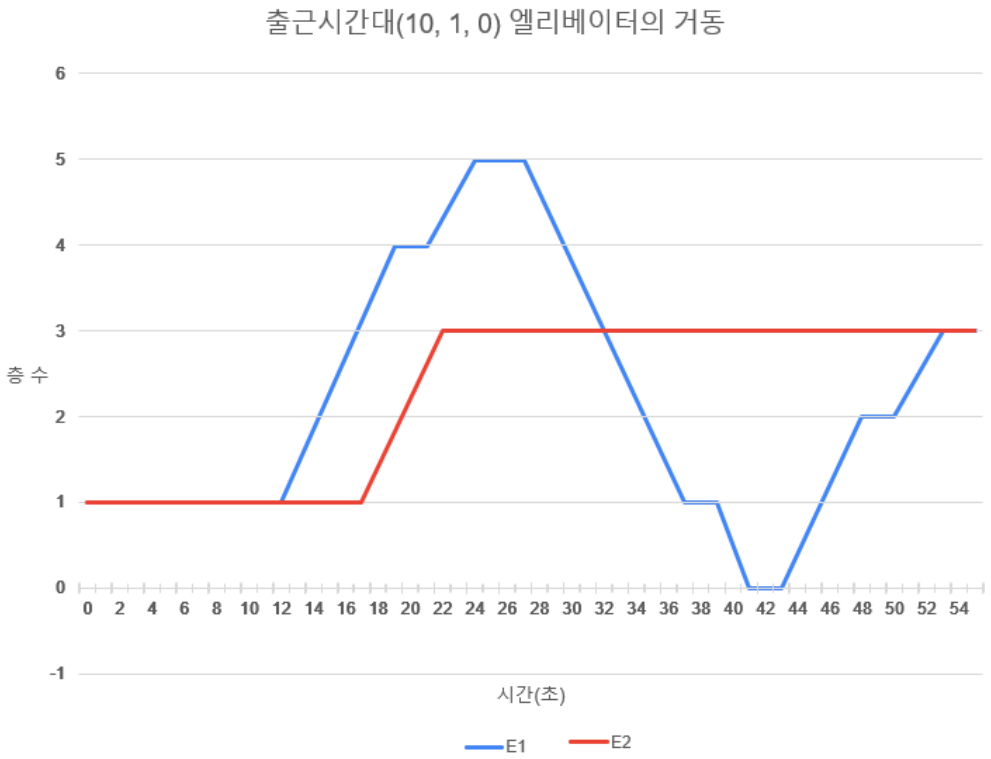
2) 평가상황에서 엘리베이터의 거동

네 가지의 공통평가상황 아래에서, 세 가중치의 값을 조금씩 바꾸어 가며 시연을 진행하고 각 시연마다 엘리베이터가 어떻게 이동했는지를 $s-t$그래프와 도표로 나타냈다. 엘리베이터가 이미 승객이 있는 층에 도착해 있지만 문이 열리지 않고 다른 엘리베이터에 call을 배정하는 등 실제 엘리베이터에 적용될 수 없는 경로의 경우 나타내지 않았다.

① 출근시간대

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ
탑승 위치	1	1	1	5	1	B1
목적 위치	4	5	3	1	3	2
호출 시각	10	10	15	25	26	30
탑승 시각	10	10	15	25	31	35.5
도착 시각	19.5	24	22	37	47	42.5
사용 승강기	E1	E1	E2	E1	E2	E2

대기시간	76.0초
소비전력	0.1002kWh



승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ
탑승 위치	1	1	1	5	1	B1
목적 위치	4	5	3	1	3	2
호출 시각	10	10	15	25	26	30
탑승 시각	10	10	15	25	37	41.5
도착 시각	19.5	24	22	37	53	48.5
사용 승강기	E1	E1	E2	E1	E1	E1

대기시간	88.0초
소비전력	0.0910kWh

첫 번째 그래프와 도표의 경우, 시간비용에만 가중치를 부여하여 알고리즘이 가장 빠른 경로를 선택하도록 하였다. 두 번째 그래프와 도표의 경우, 전력비용에도 일정량의 가중치를 부여하여 시연을 진행하였다. 그 결과, 대기시간은 12초 증가하였으나, 0.0092kWh의 소비전력을 절약할 수 있었다.

다양한 가중치 조합에 대한 시연을 효율적으로 진행하기 위하여, 전력비용 가중치를 1로 고정하고 시간비용 가중치를 계속 감소시키며 시연을 진행하였다. 전력비용 가중치에 대한 시간비용 가중치의 비가 12 이상일 때에는 시간비용에만 가중치를 부여한 알고리즘과 동일하게 거동하였으며, 6 이상 11 이하에서는 위의 두 번째 그래프처럼 거동하였다. 비가 5 이하로 감소하는 순간부터, 첫 번째 엘리베이터가 5층에 도달한 뒤에는 승객의 call을 무시하고 더 이상 움직이지 않았으며 이는 실제 엘리베이터에 적용될 수 없으므로 생략하였다.

② 퇴근시간대

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅅ
탑승 위치	5	5	4	1	3	5	5
목적 위치	1	B1	1	5	5	1	B1
호출 시각	10	10	12	20	24	30	30
탑승 시각	22	22	17.5	27	29	39	39
도착 시각	36	40.5	27	39	52.5	51	55.5
사용 승강기	E2	E2	E1	E1	E2	E1	E1

대기시간	165.5초
소비전력	0.1311kWh

위의 그래프와 도표는 시간비용에만 가중치를 부여했을 때의 엘리베이터의 거동을 나타낸다. 전력비용 가중치에 대한 시간비용 가중치의 비가 2 이상일 때에는 위의 그래프와 동일하게 거동하였으며, 1 이하로 떨어졌을 때부터는 엘리베이터가 자신의 층의 call을 무시하였다.

두 번째 엘리베이터의 이동 경로에서, 본 알고리즘이 지닌 문제점이 드러난다. 24초부터 29초까지를 주목해보자. 엘리베이터는 1층과 B1층이 목적지인 승객 ㄱ,ㄴ을 태운 상태로 5층에서 내려오는 도중, 3층에서 상위의 층으로 올라가려던 승객 ㅁ을 태우고 다시 하강한다. 승객 ㅁ의 입장에서는 시간의 손해를 보지 않았지만, 승객 ㄱ,ㄴ은 기다릴 필요가 없었던 2초의 opening sequence를 기다려야 했다. 즉, 전체 대기시간에서 4초의 손해를 보게 된 것이다. 이는 본 알고리즘이 경로를 선택할 때, car call의 층수만을 고려하고 그 방향을 고려하지 않았기 때문에 발생한 문제이다.

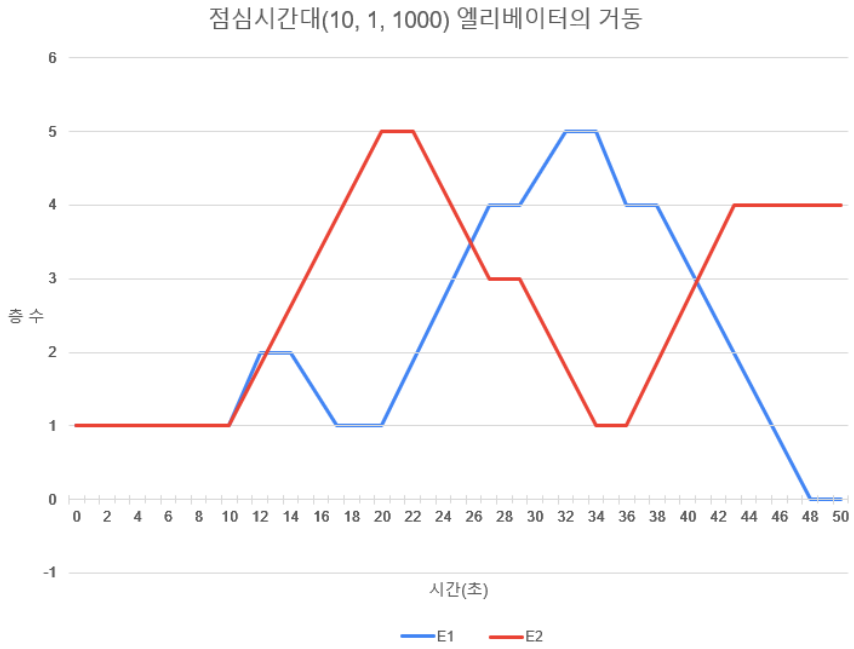
③ 점심시간대

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅅ
탑승 위치	5	2	1	3	1	1	4
목적 위치	1	1	4	1	5	4	B1
호출 시각	10	10	13	18	18	24	30
탑승 시각	20	12.5	17	25	18	32	30
도착 시각	32	17	29.5	46.5	34.5	41.5	46.5
사용 승강기	E2	E1	E1	E1	E1	E2	E1

대기시간	124.5초
소비전력	0.1247kWh

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅅ
탑승 위치	5	2	1	3	1	1	4
목적 위치	1	1	4	1	5	4	B1
호출 시각	10	10	13	18	18	24	30
탑승 시각	20	12.5	17	27	18	34	30.5
도착 시각	34	17	27.5	34	35	43.5	49.5
사용 승강기	E2	E1	E1	E2	E1	E2	E1

대기시간	117.5초
소비전력	0.1305kWh



승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅅ
탑승 위치	5	2	1	3	1	1	4
목적 위치	1	1	4	1	5	4	B1
호출 시각	10	10	13	18	18	24	30
탑승 시각	20	12.5	17	27	18	34	36.5
도착 시각	34	17	27.5	34	32	43.5	48.5
사용 승강기	E2	E1	E1	E2	E1	E2	E1

대기시간	113.5초
소비전력	0.1224kWh

위의 그래프와 도표의 경우 순서대로 시간비용에만 가중치를 부여했을 때, 전력비용에 일정량의 가중치를 부여했을 때, 그리고 전력비용에 일정량의 가중치와 방향변화비용에 큰 가중치를 부여했을 때를 나타낸다.

여기서 한 가지 흥미로운 점을 발견할 수 있는데, 전력비용 가중치에 대한 시간비용 가중치의 비를 17 이상에서 16 이하로 감소시키면, 대기시간과 소비전력 두 평가지표 모두가 감소한다는 점이다. 이는 전력비용 가중치가 상대적으로 증가하며 다른 경로를 선택하게 되면서, 우연히 시간비용에 대한 국부최소문제를 해결한 것이다. 국부최소문제에 대한 자세한 이야기는 뒤의 평가결과 분석에서 서술한다.

두 번째와 세 번째 가중치 조합의 경우, 전력비용 가중치에 대한 시간비용 가중치의 비는 10으로 동일하지만, 세 번째 가중치 조합에는 방향변화비용 가중치에 1000이라는 큰 값을 부여하였다. 방향변화비용 가중치에 큰 값을 부여함으로써, 두 번째 그래프와는 달리 엘리베이터가 이동방향을 갑자기 바꾸는 경로가 아닌 다른 경로를 선택하여 이동한 것을 볼 수 있다. 또한, 이를 통해 대기시간과 소비전력 모두에서 이익을 본 것을 볼 수 있다. 이처럼 엘리베이터의 방향이 갑자기 바뀌는 경우, 승객에 위협이 될 수 있으므로 실제 엘리베이터에서는 적용될 수 없는 상황이나, 방향 비용가중치의 필요성을 위하여 그래프와 도표를 따로 삽입하였다.

④ 한산한 시간대

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ
탑승 위치	5	1	2	3	1
목적 위치	1	4	B1	1	B1
호출 시각	10	40	70	100	130
탑승 시각	20	40	72.5	102.5	130
도착 시각	32	49.5	79.5	109.5	134.5
사용 승강기	E1	E1	E2	E1	E1

대기시간	55.0초
소비전력	0.2046kWh

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ
탑승 위치	5	1	2	3	1
목적 위치	1	4	B1	1	B1
호출 시각	10	40	70	100	130
탑승 시각	20	40	72.5	107.5	130
도착 시각	32	49.5	79.5	114.5	134.5
사용 승강기	E1	E1	E2	E2	E2

대기시간	60.0초
소비전력	0.1747kWh

한산한 시간대에서는, 가중치에 따른 경로들 사이의 차이가 가장 직관적으로 나타났다. 전력비용 가중치에 대한 시간비용 가중치의 비가 16 이상일 경우 엘리베이터는 첫 번째 그래프처럼 거동하였고, 15 이하일 경우 두 번째 그래프와 동일하게 거동하였다. 시간비용 가중치가 증가하면 알고리즘은 대기시간을 5초 줄일 수 있었으며, 반대로 시간비용 가중치가 감소하면 소비전력을 0.0299kWh만큼 절약할 수 있었다.

3) 평가결과 분석

각 가중치에 따른 엘리베이터 거동의 평가결과를 도표로 나타내면 아래와 같다.

<table>
<tr><th>C_t</th><th>C_p</th><th>C_c</th><th>출근시간</th><th>퇴근시간</th><th>점심시간</th><th>한산한 시간</th></tr>
<tr><td>18</td><td rowspan="14">1</td><td rowspan="14">1000</td><td rowspan="7">76.0초
0.1002kWh</td><td rowspan="14">165.5초
1311kWh</td><td>124.5초
0.1247kWh</td><td rowspan="3">55.0초
0.2046kWh</td></tr>
<tr><td>17</td></tr>
<tr><td>16</td><td rowspan="13">113.5초
0.1224kWh</td></tr>
<tr><td>15</td><td rowspan="12">60.0초
0.1747kWh</td></tr>
<tr><td>14</td></tr>
<tr><td>13</td></tr>
<tr><td>12</td></tr>
<tr><td>11</td><td rowspan="6">88.0초
0.0910kWh</td></tr>
<tr><td>10</td></tr>
<tr><td>9</td></tr>
<tr><td>8</td></tr>
<tr><td>7</td></tr>
<tr><td>6</td></tr>
<tr><td>5</td><td>–</td></tr>
</table>

공통 평가상황들을 통해 얻을 결과로 미루어 볼 때, 사용자가 추구하는 바에 따라서 다음 세 가중치 조합 중 하나를 선택하여 사용하는 것이 적절하다고 생각된다.

<table>
<tr><th></th><th>C_t</th><th>C_p</th><th>C_c</th></tr>
<tr><td>Time-Oriented</td><td>20</td><td rowspan="3">1</td><td rowspan="3">1000</td></tr>
<tr><td>Balance-Oriented</td><td>14</td></tr>
<tr><td>Power-Oriented</td><td>10</td></tr>
</table>

본 알고리즘은 엘리베이터가 이동할 수 있는 모든 경로를 탐색하고, 그 경로 중에서 가장 비용이 낮은 경로를 찾아주므로, 많은 상황에서 최적의 경로를 찾아준다는 장점을 갖는다. 또한, 사용자가 임의로 가중치를 조절하며 필요에 따라 엘리베이터가 다르게 이동하도록 조절할 수 있다는 점도 본 알고리즘만의 강점이다.

더하여, 본 프로젝트에서는 두 대의 엘리베이터를 모델로 시연을 진행하였지만, 이 알고리즘은 엘리베이터의 개수에 영향을 받지 않기에, 여러 대의 엘리베이터에도 적용될 수 있다는 장점을 가지고 있다.

그러나 시연을 진행할 때, 많은 상황에서 국부최소문제(local minima)가 발생하였다. 다시 말해, 엘리베이터는 매 순간 가장 최적의 경로로 이동하지만, 결과적으로는 그 경로가 최적의 경로가 아닐 수도 있다는 것이다. 본 알고리즘에서 엘리베이터는 앞으로 입력될 call 에 대하여 알 수 없기 때문에, 현재의 call 들에 의해서만 경로를 선택한다. 즉, 경로를 선택하는 데에서 전역경로계획 없이 지역경로계획만을 사용하는데, 이 때문에 국부최소문제가 발생한다. 국부최소문제를 최소화하기 위해선 전역경로계획과 지역경로계획을 함께 진행해야 하는데, 최종 목적지가 없고, 미리 알 수 없는 승객들의 call 에 의해서만 움직이는 엘리베이터의 특성상 전역경로계획은 불가능하다. 따라서 지역경로계획을 진행할 때, 이후의 승객들의 call 을 예측하며 다음 경로를 선택하는 것이 매우 중요하다.

또한, 퇴근시간대 평가상황에서 나타났던 것처럼, car call 의 방향을 고려하지 않아 이로 인해 대기시간 및 소비전력에서 손해를 볼 수 있다는 점도 큰 문제점으로 나타났다. 이를 해결하기 위해서, 다음의 알고리즘을 고안하였다.

바. Adapted Cost Comparing Algorithm

1) 알고리즘의 작동원리

Adapted Cost Comparing Algorithm은 Cost Comparing Algorithm의 단점을 보완하고자 고안한 알고리즘이다. 본 알고리즘은 Cost Comparing Algorithm과 동일하게 작동하지만, 승객의 목적지를 예측하여 경로 탐색에 사용한다. 즉, car call이 입력되었을 때 call의 위치와 층수를 고려하여 가상의 landing call을 생성해주고, 경로를 탐색할 때 이 가상의 call을 포함하여 탐색하였다. 이때, car call이 대응하는 가상의 landing call보다 앞 순서가 아닐 경우에는 모순이 생기므로 비용을 계산하지 않고 다음 경우의 수로 넘어가도록 하였다. 가상의 landing call은 승객이 한 층 이동한다는 가정 하에 생성해주었다. 예를 들어, 4층에서 승객의 아래 방향 car call이 입력되면, 그 승객의 목적지는 한 층 아래인 3층이라고 가정하고 3층으로 향하는 가상의 landing call을 생성하였다.

본 알고리즘이 각 경로의 비용을 계산하는 방식은 Cost Comparing Algorithm과 동일하게 대기시간, 소비전력, 그리고 이동방향의 변화 3가지의 비용을 고려하는 방식이다. 전력비용과 방향변화비용의 계산은 Cost Comparing Algorithm과 동일하지만, 시간비용을 계산하는 식에서 미세한 차이가 있다.

Cost Comparing Algorithm의 경우, 남아있는 call의 총 개수가 현재 엘리베이터가 도착하길 기다리는 인원과 일치했다. 그러나 본 알고리즘의 경우 가상의 landing call을 추가해 주었기 때문에, 한 명의 탑승하고자 하는 승객에게 두 개의 call이 대응된다. 따라서 $l_{i,k}$을 i번째 엘리베이터의 k번째 call부터 마지막 call까지, landing call(가상의 call 포함)의 개수라고 할 때, i번째 엘리베이터의 시간비용 $C_t(i)$는 식(4)에서 다음과 같이 변형된다.

$$C_t(i) = l_{i,1}\frac{(f_{i,1} - h_i)}{v} + \sum_{k=1}^{m_i-1}(l_{i,k+1})\frac{(f_{i,k+1} - f_{i,k})}{v} \qquad (8)$$

전력비용과 방향변화비용의 계산 식은 각각 Cost Comparing Algorithm의 식(5), 식(6)과 동일하므로 생략한다. 본 알고리즘의 전체적인 블록 다이어그램은 다음과 같다.

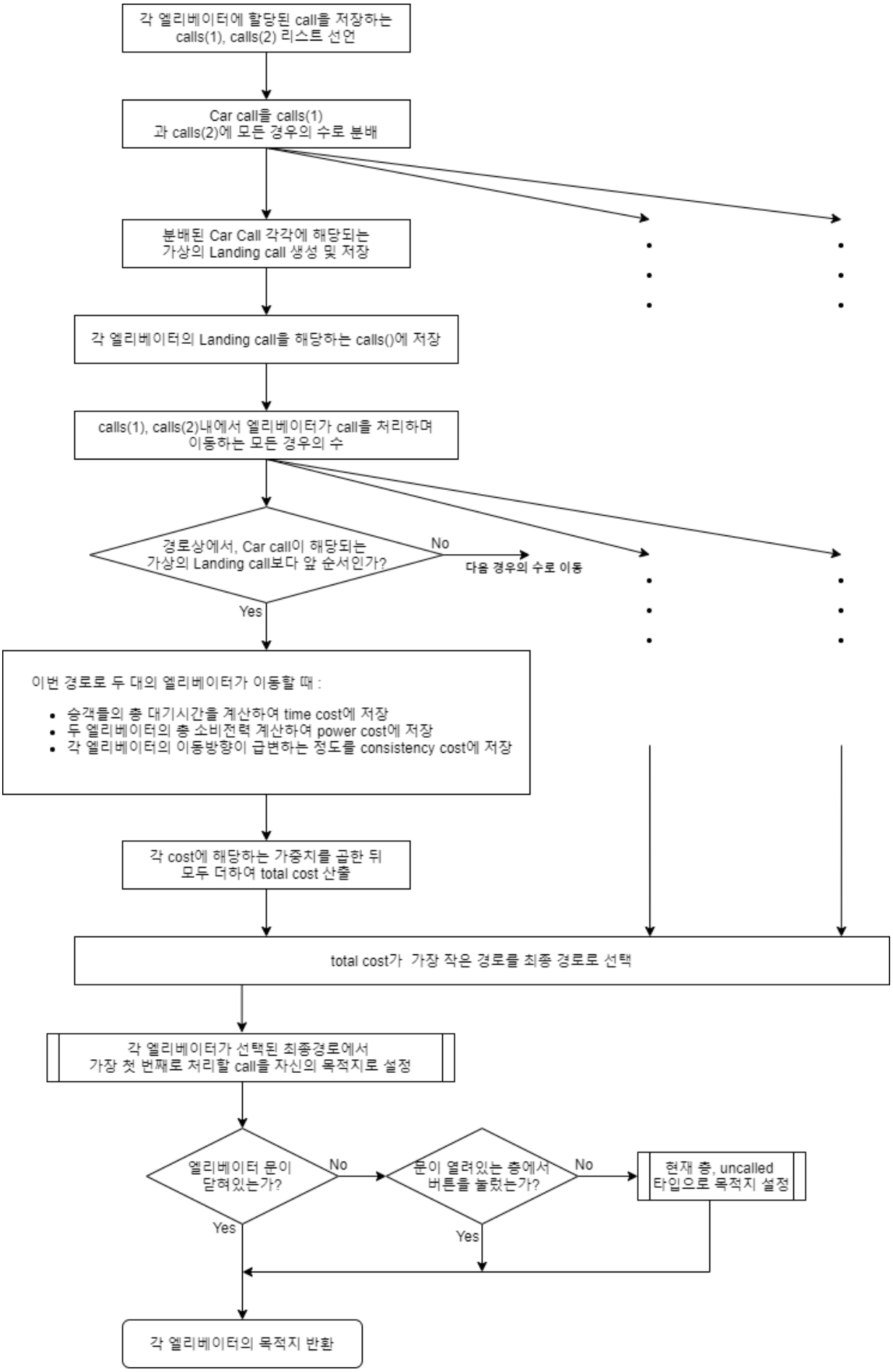


그림 25 Adapted Cost Comparing Algorithm의 블록 다이어그램

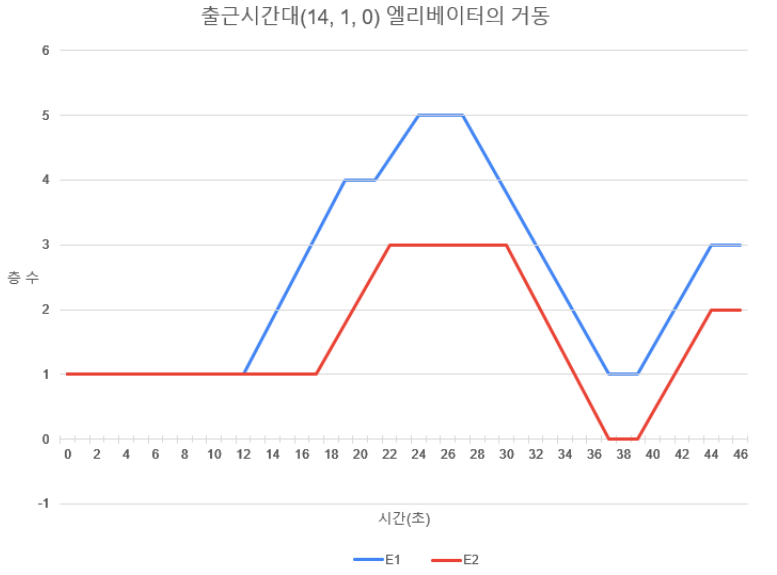
2) 평가상황에서 엘리베이터의 거동

앞의 Cost Comparing Algorithm과 동일한 방법으로 서술하였다. 가중치의 조합을 변경해가며 네 가지의 공통평가상황 하에서 엘리베이터의 거동을 나타냈으며, Cost Comparing Algorithm과의 차이를 중점적으로 서술하였다.

① 출근시간대

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ
탑승 위치	1	1	1	5	1	B1
목적 위치	4	5	3	1	3	2
호출 시각	10	10	15	25	26	30
탑승 시각	10	10	15	25	37	33.5
도착 시각	19.5	24	22	37	44	40.5
사용 승강기	E1	E1	E2	E1	E1	E2

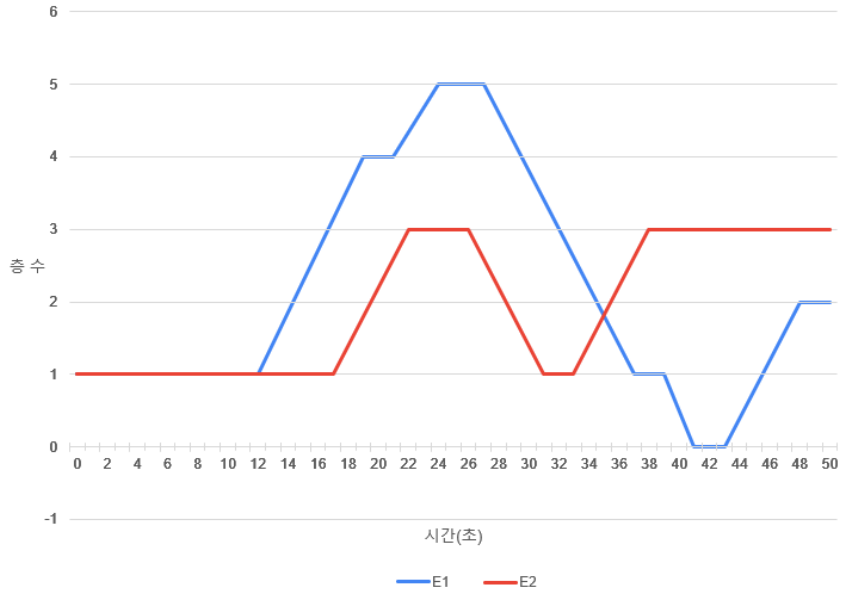
대기시간	71.0초
소비전력	0.1009kWh



승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ
탑승 위치	1	1	1	5	1	B1
목적 위치	4	5	3	1	3	2
호출 시각	10	10	15	25	26	30
탑승 시각	10	10	15	25	37	37.5
도착 시각	19.5	24	22	37	44	44.5
사용 승강기	E1	E1	E2	E1	E1	E2

대기시간	75.0초
소비전력	0.0989kWh

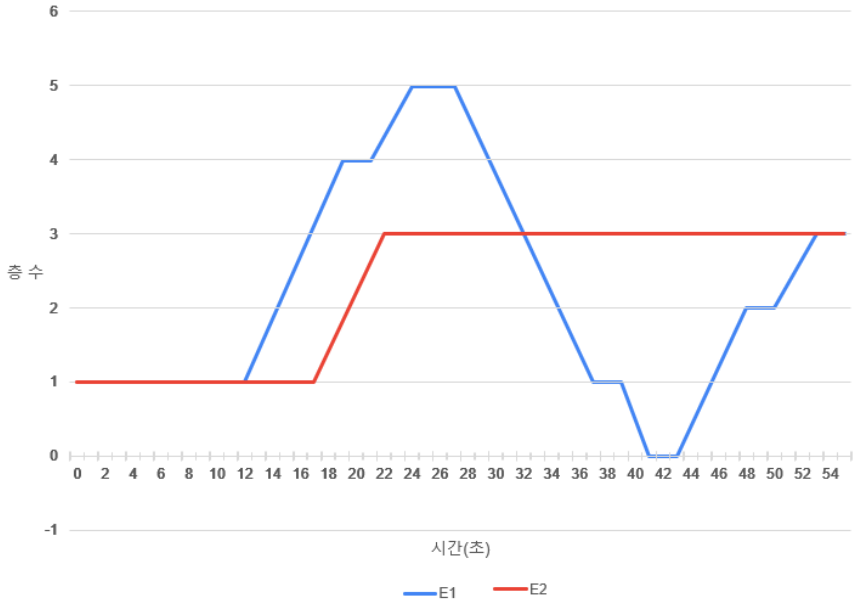
출근시간대(10, 1, 0) 엘리베이터의 거동



승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ
탑승 위치	1	1	1	5	1	B1
목적 위치	4	5	3	1	3	2
호출 시각	10	10	15	25	26	30
탑승 시각	10	10	15	25	31	41.5
도착 시각	19.5	24	22	37	38	48.5
사용 승강기	E1	E1	E2	E1	E2	E1

대기시간	73.0초
소비전력	0.1034kWh

출근시간대(8, 1, 0) 엘리베이터의 거동



승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ
탑승 위치	1	1	1	5	1	B1
목적 위치	4	5	3	1	3	2
호출 시각	10	10	15	25	26	30
탑승 시각	10	10	15	25	37	41.5
도착 시각	19.5	24	22	37	53	48.5
사용 승강기	E1	E1	E2	E1	E1	E1

대기시간	88.0초
소비전력	0.0910kWh

위의 그래프와 도표는 공통평가상황 중 출근시간대에, 전력비용 가중치에 대한 시간비용 가중치의 비를 감소시키며 진행한 시연결과이다. 엘리베이터는 전력비용 가중치에 대한 시간비용 가중치의 비가 15 이상일 경우 첫 번째 그래프처럼, 14일 경우 두 번째 그래프처럼, 13 이하 9 이상일 경우 세 번째 그래프처럼, 8 이하 6 이상일 경우 마지막 그래프처럼 거동했으며, 5 이하일 경우 승객의 call을 무시하며 이동하였다.

그래프들을 보면, Cost Comparing Algorithm과 비교했을 때 거동의 변화가 더 다양한 것을 볼 수 있다. 또한, 시간비용 가중치에만 값을 부여했을 때, 대기시간이 5초 절약되었다. 이는 운행이 시작된 후 30초가 지났을 때, 알고리즘이 1층의 승객 ㅁ이 위로 올라가려 한다는 것을 인지하고, 승객 ㅁ의 call을 가까운 두 번째 엘리베이터가 아닌 첫 번째 엘리베이터에게 넘겨줌으로써 절약된 시간이다.

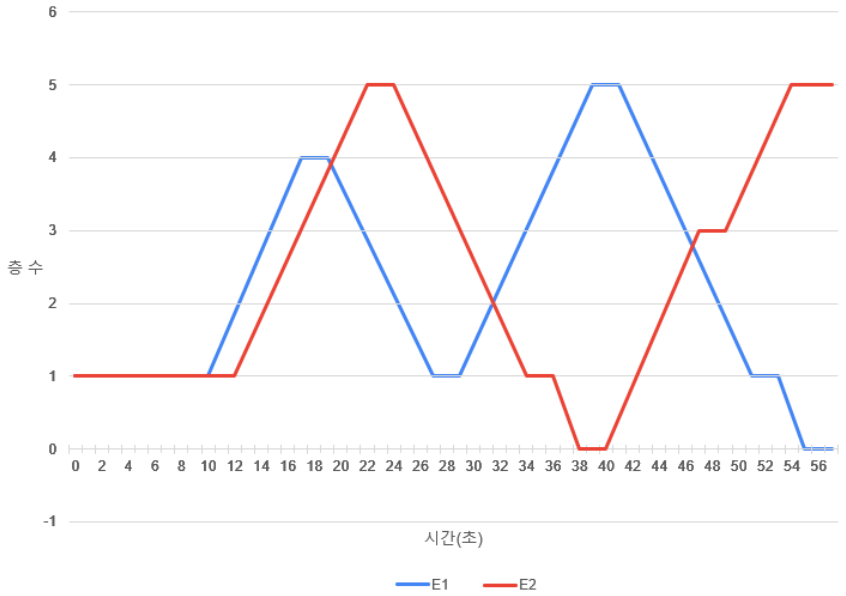
흥미로운 점은 전력비용 가중치에 대한 시간비용 가중치의 비가 14일 때 나타난다. 비가 13일 때와 비교했을 때 시간비용 가중치가 상대적으로 더 컸지만, 승객들의 대기시간은 오히려 2초 증가하였다. 이는 특정 상황에 발생한 특이점으로 보이는데, 알고리즘은 승객 ㅁ의 call을 첫 번째 엘리베이터에 배정하고 두 번째 엘리베이터는 이동할 필요가 없다고 판단했으나, B1층에서 승객 ㅂ의 call이 들어오며 두 번째 엘리베이터를 이동시킴으로써 발생하였다. 이 특이점은 엘리베이터가 승객의 call이 언제 어디서 입력될지 알 수 없다는 점으로부터 발생한 국부최소문제의 단적인 예를 보여주며, 이에 대해서는 평가결과 분석에서 자세히 서술하도록 하겠다.

② 퇴근시간대

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅅ
탑승 위치	5	5	4	1	3	5	5
목적 위치	1	B1	1	5	5	1	B1
호출 시각	10	10	12	20	24	30	30
탑승 시각	22	22	17.5	27	34	41	39
도착 시각	34	38.5	27	41	41	53	57.5
사용 승강기	E2	E2	E1	E1	E1	E1	E1

대기시간	156.0초
소비전력	0.1340kWh

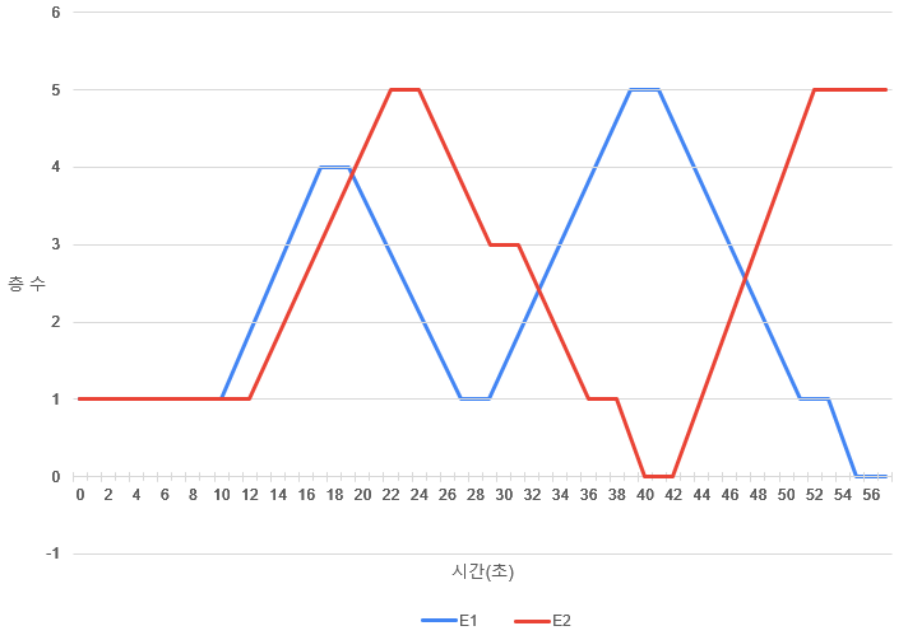
퇴근시간대(10, 1, 0) 엘리베이터의 거동



승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅅ
탑승 위치	5	5	4	1	3	5	5
목적 위치	1	B1	1	5	5	1	B1
호출 시각	10	10	12	20	24	30	30
탑승 시각	22	22	17.5	27	47.5	39	39
도착 시각	34	38.5	27	39	54.5	51	55.5
사용 승강기	E2	E2	E1	E1	E2	E1	E1

대기시간	163.5초
소비전력	0.1386kWh

퇴근시간대(3, 1, 0) 엘리베이터의 거동



승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅅ
탑승 위치	5	5	4	1	3	5	5
목적 위치	1	B1	1	5	5	1	B1
호출 시각	10	10	12	20	24	30	30
탑승 시각	22	22	17.5	27	29	39	39
도착 시각	36	40.5	27	39	52.5	51	55.5
사용 승강기	E2	E2	E1	E1	E2	E1	E1

대기시간	165.5초
소비전력	0.1311kWh

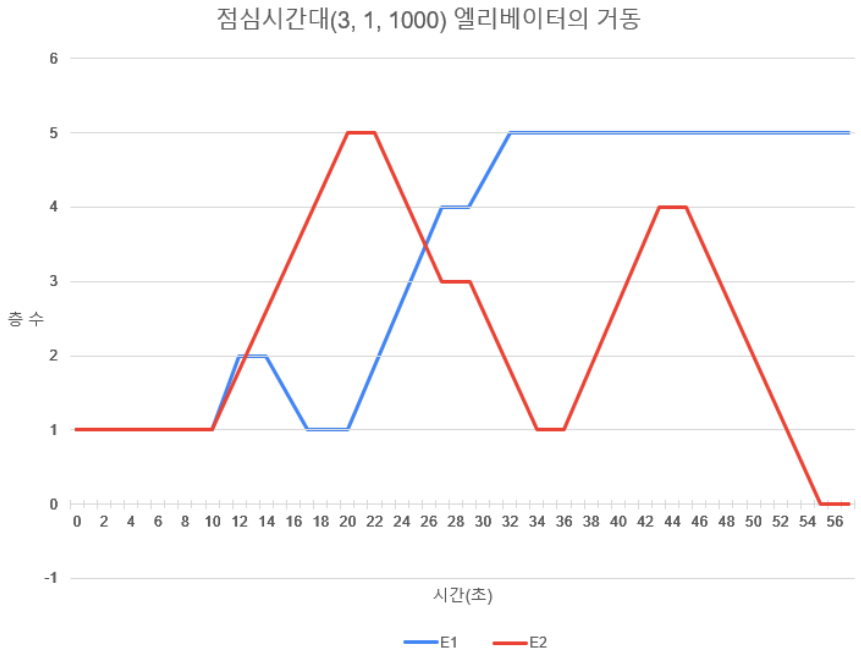
위의 그래프와 도표는 공통평가상황 중 퇴근시간대에, 전력비용 가중치에 대한 시간비용 가중치의 비를 감소시키며 진행한 시연결과이다. 엘리베이터는 전력비용 가중치에 대한 시간비용 가중치의 비가 13 이상일 경우 첫 번째 그래프처럼, 12 이하 8 이상일 경우 두 번째 그래프처럼, 7 이하 3 이상일 경우 마지막 그래프처럼 거동했으며, 2 이하일 경우 승객의 call을 무시하며 이동하였다.

먼저, 시간비용 가중치에만 값을 부여했을 때, 대기시간이 9.5초라는 큰 폭으로 떨어진 것을 볼 수 있다. 이는 승객의 car call의 방향을 고려하며, 두 번째 엘리베이터가 승객 ㅁ을 태우지 않음으로써 절약된 시간이다. 또한, 전력비용 가중치에 대한 시간비용 가중치의 비가 7 이하 3 이상일 때에는 두 번째 엘리베이터가 승객 ㅁ을 태우고 내려갔다가 다시 올라오면서, 승객의 행선지를 고려하지 않았던 Cost Comparing Algorithm과 동일하게 거동하는 것을 볼 수 있다. 이는 전력비용 가중치가 상대적으로 증가하면서, 승객 ㄱ, ㄴ이 손해를 보는 대기시간보다 승객 ㅁ을 태움으로써 엘리베이터 하강 시 절약되는 소비전력이 더 가치 있다고 알고리즘이 판단한 것으로, Cost Comparing Algorithm과 엘리베이터의 운행경로는 동일하지만 그 의도는 다르다.

③ 점심시간대

승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅅ
탑승 위치	5	2	1	3	1	1	4
목적 위치	1	1	4	1	5	4	B1
호출 시각	10	10	13	18	18	24	30
탑승 시각	20	12.5	17	27	18	34	36.5
도착 시각	34	17	27.5	34	32	43.5	48.5
사용 승강기	E2	E1	E1	E2	E1	E2	E1

대기시간	113.5초
소비전력	0.1224kWh



승객	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅅ
탑승 위치	5	2	1	3	1	1	4
목적 위치	1	1	4	1	5	4	B1
호출 시각	10	10	13	18	18	24	30
탑승 시각	20	12.5	17	27	18	34	43.5
도착 시각	34	17	27.5	34	32	43.5	55.5
사용 승강기	E2	E1	E1	E2	E1	E2	E2

대기시간	120.5초
소비전력	0.0950kWh

위의 그래프와 도표는 공통평가상황 중 점심시간대에, 전력비용 가중치에 대한 시간비용 가중치의 비를 감소시키며 진행한 시연결과이다. 엘리베이터는 전력비용 가중치에 대한 시간비용 가중치의 비가 4이상일 경우 첫 번째 그래프처럼, 3이하일 경우 두 번째 그래프처럼 거동했다.

첫 번째 그래프처럼 시간비용 가중치에만 값을 부여했을 때, 대기시간이 11초만큼 크게 감소한 것을 볼 수 있는데, 이 거동은 Cost Comparing Algorithm에 어느 정도의 전력비용 가중치를 부여했을 때 나타났던 거동과 동일하다. Cost Comparing Algorithm이 소비전력을 낮추기 위한 경로를 탐색하다 우연히 해결했던 대기시간에 대한 국부최소문제를, 시간비용에 대해서만 가중치를 부여했을 때 본 알고리즘이 의도적으로 해결했다는 점은 굉장히 긍정적인 결과이다. 이는 본 알고리즘이 Cost Comparing Algorithm이 해결하지 못했던 국부최소문제를 완화했음을 보여주는 단적인 예이다.

또한, 전력비용 가중치에 대한 시간비용 가중치의 비를 3 이하로 감소시켰을 시, 대기시간은 7초 증가했지만, 소비전력은 0.0274kWh 절약되었다. 0.0274kWh는 의미 있는 수치로서, 가중치가 그 역할을 제대로 하고 있음을 알 수 있다.

④ 한산한 시간대

한산한 시간대의 경우, 승객들의 call이 서로 겹치지 않기 때문에 국부최소문제가 발생하지 않으며, call의 방향도 고려할 필요가 없다. 따라서 Cost Comparing Algorithm과 정확히 동일하게 거동하므로, 생략한다.

3) 평가결과 분석

각 가중치에 따른 엘리베이터 거동의 평가결과를 도표로 나타내면 아래와 같다.

<table>
<tr><th>C_t</th><th>C_p</th><th>C_c</th><th>출근시간</th><th>퇴근시간</th><th>점심시간</th><th>한산한 시간</th></tr>
<tr><td>17</td><td rowspan="15">1</td><td rowspan="15">1000</td><td rowspan="3">71.0초
0.1009kWh</td><td rowspan="5">156.0초
0.1340kWh</td><td rowspan="13">113.5초
0.1224kWh</td><td rowspan="3">55.0초
0.2046kWh</td></tr>
<tr><td>16</td></tr>
<tr><td>15</td><td rowspan="13">60.0초
0.1747kWh</td></tr>
<tr><td>14</td><td>75.0초
0.0989kWh</td></tr>
<tr><td>13</td><td rowspan="5">73.0초
0.1034kWh</td></tr>
<tr><td>12</td><td rowspan="5">163.0초
0.1386kWh</td></tr>
<tr><td>11</td></tr>
<tr><td>10</td></tr>
<tr><td>9</td></tr>
<tr><td>8</td><td rowspan="3">88.0초
0.0910kWh</td></tr>
<tr><td>7</td><td rowspan="5">165.5초
0.1311kWh</td></tr>
<tr><td>6</td></tr>
<tr><td>5</td><td rowspan="3">–</td></tr>
<tr><td>4</td></tr>
<tr><td>3</td><td rowspan="2">120.5초
0.950kWh</td></tr>
<tr><td>2</td><td>–</td></tr>
</table>

공통 평가상황들을 통해 얻을 결과로 미루어 볼 때, 사용자가 추구하는 바에 따라서 다음 세 가중치 조합 중 하나를 선택하여 사용하는 것이 적절하다고 생각된다. Balance-Oriented 가중치 조합의 경우, 조합(1)은 조합(2)보다 대기시간을 절약하는 경로를 지향하고, 조합(2)는 조합(1)보다 소비전력을 절약하는 경로를 지향한다.

	C_t	C_p	C_c
Time-Oriented	20	1	1000
Balance-Oriented(1)	14		
Balance-Oriented(2)	11		
Power-Oriented	7		

시연결과, 승객의 행선지를 예상하고 가상의 landing call 을 생성하는 코드를 추가해 줌으로써 한산한 시간대를 제외한 모든 상황에서 엘리베이터의 거동이 크게 달라졌으며, Comparing Algorithm 보다 더 좋은 결과를 나타냈다. 소비전력 평가지표 부분에서도 좋은 결과가 나타났지만, 특히 대기시간 평가지표에서 두각을 나타냈다. Time-Oriented 가중치 조합($C_t : 20,\ C_p : 1,\ C_t : 1000$)을 예로 들어보면, 공통평가상황의 출근시간대, 퇴근시간대, 그리고 점심시간대에서 각각 5 초, 9.5 초, 11 초의 대기시간을 절약한 것을 볼 수 있다. 이는 국부최소문제가 어느 정도 해소되며 나타난 것으로, 방향이 엇갈리는 call 들이 많아 국부최소문제가 발생하기 쉬운 퇴근시간대와 점심시간대에서 특히 더 많은 대기시간을 절약한 것을 볼 수 있다.

국부최소문제가 해소됨으로 인하여, 각 비용의 가중치도 제 역할을 할 수 있게 되었다. Cost Comparing Algorithm 의 경우, 공통평가상황 중 하나인 점심시간대에서 시간비용 가중치를 증가시켰음에도, 국부최소문제로 인해 대기시간이 오히려 증가하는 상황이 발생하기도 하였다. 하지만 본 알고리즘의 경우, 몇몇 상황에서 특이점이 발생한 것을 제외하고, 시간비용 가중치를 증가시키면 대기시간이 감소하고, 전력비용 가중치를 상대적으로 증가시키면 소비전력이 감소하는 것을 위의 도표에서 확인할 수 있다. 또한, 가중치의 변화에 따른 엘리베이터 거동의 변화가 더 다양하기 때문에 사용자는 더욱 넓은 선택의 폭을 가질 수 있다.

출근시간대에서 전력비용 가중치에 대한 시간비용 가중치의 비가 14 일 때 볼 수 있는 상황처럼, 국부최소문제를 완전히 해소할 수는 없었다. 그러나 이는 앞서 언급한 것처럼 엘리베이터의 특성상 불가능하며, 승객들의 call 을 예측하며 지역경로계획을 진행하는 것이 국부최소문제를 완화하는 유일한 방법이다. 본 알고리즘에서 더 나아가 국부최소문제를 더욱 완화하기 위해서는 두 가지의 방법이 있다고 생각된다. 먼저 첫 번째로, 건물의 엘리베이터 이동에 대한 빅데이터를 이용해, 승객의 이동에 대한 확률을 계산하는 것이다. 이는 현대엘리베이터에서 사용 중인 기술인 '인공지능 군 관리 시스템'으로, 딥러닝 AI 를 사용해 엘리베이터 교통량에 대한 인공지능의 분석으로 여러 대수의 엘리베이터 운행을 효율적으로 통제 및 제어한다. 두 번째 방법은, 목적 층 예약시스템을 도입하는 것이다. 이는 최근 유동인구가 많은 건물이 도입하고 있는 방법으로, 고층의 회사건물이나 대형백화점에서 사용한다. 이 시스템은 승객이 엘리베이터에 call 을 부여할 때 목적 층을 입력하도록 하여 엘리베이터의 불필요한 이동을 최소화한다. 본 알고리즘에서는 승객의 행선지를 승객의 이동방향으로 한 층 이동한 층으로 가정하여 경로를 설정하였지만, 이 시스템을 사용하면 승객의 목적지를 정확하게 알 수 있어 더 효율적인 운행이 가능하다.

결과적으로 본 알고리즘은 Cost Comparing Algorithm 의 장점은 그대로 가져오고, 한계점이었던 국부최소문제와 car call 의 방향 문제를 해결하여 최적의 경로로 승객들을 이동시킬 수 있었다.

VI. 결론

1. 알고리즘 비교 분석

공통 평가상황에서의 대기시간에 대한 알고리즘의 순위는 아래의 도표와 같다.

<table>
<tr><th></th><th>출근시간</th><th>퇴근시간</th><th>점심시간</th><th>한산한 시간</th></tr>
<tr><td>1st</td><td rowspan="3"><71.0초>
Binary Allocation

Binary Allocation + Default

Adapted Cost Comparing: Time-Oriented</td><td><156.0초>
Adapted Cost Comparing: Time-Oriented

Adapted Cost Comparing: Balance-Oriented(1)</td><td rowspan="4"><113.5초>
Cost Comparing: Balance-Oriented

Cost Comparing: Power-Oriented

Adapted Cost Comparing: Time-Oriented

Adapted Cost Comparing: Balance-Oriented(1)

Adapted Cost Comparing: Balance-Oriented(2)

Adapted Cost Comparing: Power-Oriented</td><td rowspan="2"><55.0초>
Binary Allocation

Cost Comparing: Time-Oriented

Adapted Cost Comparing: Time-Oriented</td></tr>
<tr><td>2nd</td></tr>
<tr><td>3rd</td><td rowspan="2"><162.0초>
Binary Allocation

Binary Allocation + Default</td></tr>
<tr><td>4th</td><td><73.0초>
Adapted Cost Comparing: Balance-Oriented(2)</td><td><55.5초>
Binary Allocation + Default</td></tr>
<tr><td>5th</td><td><75.0초>
Adapted Cost Comparing: Balance-Oriented(1)</td><td><163.0초>
Adapted Cost Comparing: Balance-Oriented(2)</td><td rowspan="6"><60.0초>
High-Low Split

Odd-Even Split

Cost Comparing: Balance-Oriented

Cost Comparing: Power-Oriented

Adapted Cost Comparing: Balance-Oriented(1)

Adapted Cost Comparing: Balance-Oriented(2)

Adapted Cost Comparing: Power-Oriented</td></tr>
<tr><td>6th</td><td rowspan="2"><76.0초>
Cost Comparing: Time-Oriented

Cost Comparing: Balance-Oriented</td><td rowspan="4"><165.5초>
Cost Comparing: Time-Oriented

Cost Comparing: Balance-Oriented

Cost Comparing: Power-Oriented

Adapted Cost Comparing: Power-Oriented</td></tr>
<tr><td>7th</td><td rowspan="2"><116.5초>
Binary Allocation

Binary Allocation + Default</td></tr>
<tr><td>8th</td><td><77.0초>
High-Low Split</td></tr>
<tr><td>9th</td><td><88.0초>
Cost Comparing: Power-Oriented</td><td><124.5초>
Cost Comparing: Time-Oriented</td></tr>
<tr><td>10th</td><td>Adapted Cost Comparing: Power-Oriented</td><td><210.0초>
Odd-Even Split</td><td><137.5초>
Odd-Even Split</td></tr>
<tr><td>11th</td><td><114.5초>
Odd-Even Split</td><td><227.0초>
High-Low Split</td><td><147.5초>
High-Low Split</td></tr>
</table>

Adapted Cost Comparing Algorithm: Time-Oriented가 모든 상황에서 1위를 차지했다. 이 밖에, Binary Allocation Algorithm은 세 가지 상황에서 공동 1위와 공동 3위를 차지했으며, Binary Allocation Algorithm + Default는 세 가지 상황에서 각각 공동 1위, 공동 3위, 공동 4위를 차지했고, Adapted Cost Comparing Algorithm: Balance-Oriented는 꾸준하게 중상위권을 차지하고 있다.

공통 평가상황에서의 소비전력에 대한 알고리즘의 순위는 아래의 도표와 같다. 단, 각 알고리즘에 대해 시뮬레이션 시연자가 달라, 엘리베이터가 동일하게 움직인 상황임에도 측정한 소비전력 데이터가 상이한 경우가 있었다. 따라서 이 경우에는 상이한 데이터 간의 중간값을 사용하였다.

<table>
<tr><th></th><th>출근시간</th><th>퇴근시간</th><th>점심시간</th><th>한산한 시간</th></tr>
<tr><td>1st</td><td rowspan="2"><0.0910kWh>
Cost Comparing:
Power-Oriented

Adapted Cost
Comparing:
Power-Oriented</td><td rowspan="2"><0.1250kWh>
Binary Allocation

Binary Allocation +
Default</td><td rowspan="2"><0.1180kWh>
Binary Allocation

Binary Allocation +
Default</td><td rowspan="5"><0.1858kWh>
High-Low Split

Cost Comparing:
Balance-Oriented

Cost Comparing:
Power-Oriented

Adapted Cost
Comparing:
Balance-Oriented(1)

Adapted Cost
Comparing:
Balance-Oriented(2)

Adapted Cost
Comparing:
Power-Oriented</td></tr>
<tr><td>2nd</td></tr>
<tr><td>3rd</td><td rowspan="3"><0.0975kWh>
Binary Allocation

Binary Allocation +
Default

Adapted Cost
Comparing:
Time-Oriented</td><td><0.1290kWh>
High-Low Split</td><td rowspan="5"><0.1224kWh>
Cost Comparing:
Balance-Oriented

Cost Comparing:
Power-Oriented

Adapted Cost
Comparing:
Time-Oriented

Adapted Cost
Comparing:
Balance-Oriented(1)

Adapted Cost
Comparing:
Balance-Oriented(2)

Adapted Cost
Comparing:
Power-Oriented</td></tr>
<tr><td>4th</td><td rowspan="3"><0.1311kWh>
Cost Comparing:
Time-Oriented

Cost Comparing:
Balance-Oriented

Cost Comparing:
Power-Oriented

Adapted Cost
Comparing:
Power-Oriented</td></tr>
<tr><td>5th</td></tr>
<tr><td>6th</td><td><0.0989kWh>
Adapted Cost
Comparing:
Balance-Oriented(1)</td></tr>
<tr><td>7th</td><td rowspan="2"><0.1002kWh>
Cost Comparing:
Time-Oriented

Cost Comparing:
Balance-Oriented</td><td rowspan="3"><0.1340kWh>
Adapted Cost
Comparing:
Time-Oriented

Adapted Cost
Comparing:
Balance-Oriented(1)</td><td rowspan="3"><0.2018kWh>
Binary Allocation

Cost Comparing:
Time-Oriented

Adapted Cost
Comparing:
Time-Oriented</td></tr>
<tr><td>8th</td></tr>
<tr><td>9th</td><td><0.1034kWh>
Adapted Cost
Comparing:
Balance-Oriented(2)</td><td><0.1247kWh>
Cost Comparing:
Time-Oriented</td></tr>
<tr><td>10th</td><td><0.1170kWh>
High-Low Split</td><td><0.1386kWh>
Adapted Cost
Comparing:
Balance-Oriented(2)</td><td><0.1250kWh>
Odd-Even Split</td><td><0.2050kWh>
Binary Allocation +
Default</td></tr>
<tr><td>11th</td><td><0.1536kWh>
Odd-Even Split</td><td><0.1774kWh>
Odd-Even Split</td><td><0.1640kWh>
High-Low Split</td><td><0.2264kWh>
Odd-Even Split</td></tr>
</table>

Cost Comparing Algorithm: Power-Oriented와 Adapted Cost Comparing Algorithm: Power-Oriented는 모든 상황에서 공동 1위, 공동 3위를 차지했다. Binary Allocation Algorithm과 Binary Allocation Algorithm + Default는 세 가지 상황에서 공동 1위와 공동 3위를 차지했다

먼저, Odd-Even Split 알고리즘과 High-Low Split 알고리즘을 살펴보자. 두 알고리즘 모두 효율적인 거동을 위하여 각 엘리베이터가 갈 수 있는 구역을 제한하는 알고리즘으로, 이러한 방법을 Elevator Zoning이라고 한다. 그러나 결과 도표에서 드러나듯이, 두 알고리즘 모두 대기시간과 소비전력 어느 방면으로도 좋은 성능을 보여주지 못하였다. 각 승강기가 갈 수 있는 구역이 정해져 있기에 유동적으로 승객을 배정할 수 없는 단점 때문에 이런 현상이 나타났다.

그 다음으로 Binary allocation 알고리즘과 Binary allocation + default 알고리즘을 살펴보자. 두 알고리즘은 한산한 시간의 상황을 제외하고 모두 같은 결과를 나타냈다. 성능 측면에서도 대부분의 상황에서 상위권을 차지하고 있다. 들어오는 call과 엘리베이터의 움직임에 대해 실시간으로 분석하여 움직이던 경로 위에서 최적의 목적지를 배정하기 때문이다. 추가로 살펴볼 내용으로, 한산한 시간에서 default 층으로 움직이는 알고리즘이 오히려 더 좋지 않은 성능을 보였다. 엘리베이터가 오랜 시간 동안 움직이지 않아 1층으로 이동하는 중에 새로운 call이 들어왔고 이에 대응하는 과정에서 더 큰 전력 소모와 대기 시간을 일으켰다.

마지막으로 Cost Comparing Algorithm과 Adapted Cost Comparing Algorithm은 철저하게 선정한 가중치에 의존한 결과를 나타냈다. Adapted Cost Comparing: Time-Oriented는 모든 상황의 대기 시간 지표에서 1위를 차지했고, Power-Oriented는 모든 상황의 소비 전력 지표에서 1위와 3위를 차지했으며, Balance-Oriented는 대부분의 상황에서 모든 지표에 대해 중위권을 차지했다.

모든 상황에 대해 모든 지표가 우수한 알고리즘은 없었으며, 상황 별, 시간대 별로 최적의 알고리즘을 복합적으로 사용하면 될 것으로 생각한다.

2. 프로젝트의 한계점과 발전 가능성

가. 대기시간 평가지표

주요한 평가 지표인 승객의 대기시간을 계산하는 방식에서 본 프로젝트의 한계점이 나타났다. 우선, 프로그램상에서 해당 부분을 구현한 코드에 문제점이 있었다. 대기 시간은 매 loop마다 car call과 landing call을 검사하여 True인 call의 수 만큼을 대기 시간에 더해주는 방식이다. 실제 상황에서, 승객이 엘리베이터 안에서 도착 층을 누르는 버튼에 해당하는 landing call이 들어오는 시점은 일정하지 않다. 누군가는 엘리베이터에 타자마자 버튼을 누르고, 누군가는 엘리베이터 문이 닫히는 순간에 버튼을 누를 수 있다. 그에 따라 landing call이 True가 되는 순간도 랜덤하게 되고 이는 대기 시간을 신뢰할 수 없게 만드는 이유가 된다. 따라서 위에서 알고리즘을 분석할 때에는 프로그램에서 계산된 대기 시간이 아닌 호출 시각과 목적지 도착 시각의 차이를 이용하였다.

또한, 대기시간을 계산하는 방식에서도 한계점을 찾을 수 있었다. 우선 현재 계산하는 대기시간은 선형적인 계산 방식으로, 기다린 만큼의 시간이 정확하게 더해지는 방식이다. 하지만 실제 상황에서는 꼭 그렇지 않다. 승객의 엘리베이터 이용 시각, 탑승 위치, 목적 위치, 기다리는 시간의 가중치 등을 고려할 필요가 있다. 예를 들어, 1층에서 엘리베이터를 호출한 승객과 5층에서 엘리베이터를 호출한 승객이 있다. 전자의 경우 엘리베이터를 이용하는 사람이 많은 층이기 때문에 더 빨리 오기를 기대할 수 있고, 후자의 경우는 반대의 이유로 인해 심리적으로 더 오랜 시간을 기다릴 수 있다. 출근 시간에서 기다리는 시간 동안 느껴지는 불편함과 퇴근 시간에서 기다리는 시간 동안 느껴지는 불편함의 정도 역시 다를 수 있으며, 엘리베이터를 오래 기다릴수록 불편함의 정도는 선형함수가 아닌 지수함수의 그래프처럼 증가할 수도 있다. 이처럼 우리가 지금 대기 시간을 계산하는 방식은 실제 승객이 기다림에 따라 느끼는 불편함을 고려하지 못했다는 한계점이 있다.

나. 비교군인 새천년관의 데이터 부재

두 번째 한계점은 이 프로젝트의 계기이자 시작인 새천년관의 데이터를 구할 수 없다는 것이다. 앞서 서론에 언급했듯이 프로젝트의 출발점은 새천년관의 두 엘리베이터에 있다. 여러 평가상황을 두고 프로그램을 통해 엘리베이터를 운영하고 데이터를 얻는 것은 성공하였으나 비교를 위한 데이터 즉, 실제 새천년관 엘리베이터의 데이터를 구하는 것은 현실적으로 많은 어려움이 있다고 판단하였다. 많은 사람이 이용하기에 엘리베이터를 독점적으로 운행할 수도 없었고 시간을 정확히 측정하기에는 역부족이라 생각했기 때문이다. 이로 인해 비교군 데이터를 구하지 못해 실질적인 프로그램 성능을 측정하지 못했다는 한계점이 발생하였다.

다. 버튼보드 케이스

다음 한계점은 사전에 계획했던 버튼 보드의 케이스를 제작하지 못했다는 점이다. 앞서 언급했던 것처럼, Creo를 이용해 만든 버튼 보드 케이스를 3D 프린터를 사용해 인쇄하려 했으나, 비용과 시간의 문제로 인하여 실패하였다. 버튼보드 케이스의 부재로 인하여 프로그램에 버튼 입력을 넣어줄 때, PCB 보드에 납땜 된 버튼을 직접 눌러야 했다. 이는 버튼 보드 파손의 위험성을 증가시켰으며, 어떤 버튼이 어떤 입력을 넣어주는지 구분하기 어렵다는 문제를 낳았다.

라. 하나의 Call에 대응하는 승객의 수

마지막 한계점은 엘리베이터 구현 소프트웨어를 만들 때 설정했던 "하나의 Call은 한 명의 승객과 일대일 대응한다." 라는 가정에서 비롯된다. 물론, 어떤 상황에서 이 가정은 틀리지 않는다. 그러나 다수의 승객이 동일한 층에서 출발하여, 동일한 층으로 이동하는 경우도 적지 않으며, 이때 소프트웨어는 하나의 call만을 인식하므로 한 명의 승객만을 인식하게 된다. 엘리베이터 안에 탑승하고 있는 승객의 인원을 인지하는 것은 매우 중요한데, 이는 승객의 인원이 대기시간과 소비전력에 큰 영향을 주기 때문이다. 대기시간의 경우, 모든 승객의 대기시간을 모든 합친 수치이기 때문에, 대기 승객의 인원수가 달라지는 것은 대기시간의 큰 변화를 가져온다. 또한 소비전력의 경우, 엘리베이터 내의 승객의 총 무게와 이동방향에 따라 그 값이 달라진다. 즉, 하나의 Call에 대응하는 승객의 수를 한 명으로 가정함으로써, 여러 명의 승객이 동일한 경로로 움직일 경우 대기시간과 소비전력의 평가지표를 제대로 계산할 수 없다는 한계점이 발생하였다.

3. 프로젝트의 의의

근 몇 십년 동안, 건물에 있어 높이는 중요한 요소 중 하나가 되었다. 제한된 부지에서 더 많은 사람을 수용해야 한다는 수요와 더불어, 관련된 기술력이 발전하자 서울 한복판에 지상 123 층의 신축 건물도 생겼다. 그에 따라, 고층 건물에서 사람을 운송하는 엘리베이터의 운행 방식에서도 같이 발전이 있어야 했지만, 그렇지 않음을 우리는 곳곳에서 실감하고 있다. 따라서 우리는 여러 대의 엘리베이터를 효율적으로 운행하는 알고리즘에 대한 설계 및 시뮬레이션을 주제로 프로젝트를 진행하였다. 이로써, 실생활에서 불편을 느끼는 상황에 대해 문제를 정의하고, 의미 있는 설계를 진행했다는 점에서 프로젝트의 의의가 있다. 다음으로 각 팀원이 본 프로젝트를 통해 느낀 점들을 서술하였다.

강재원

Notion과 Github를 처음으로 제대로 사용해보며 그 편리성을 깨달았고, 익숙하게 사용할 수 있게 되었다. 엘리베이터 구현 소프트웨어의 포맷을 설계하면서, 여러 사람의 코드와 호환될 수 있는 코드를 짜는 것이 매우 어렵고, 그만큼 중요한 일이라는 것을 실감하였으며, 객체 지향 프로그래밍의 강점이 여기서 드러난다는 생각을 하게 되었다. 또한, 매주 회의를 진행하며 프로젝트의 진행에 있어서 주기적인 회의와 문서화의 중요성을 깨달았다. 또한 다른 분야의 팀원들과 프로젝트를 진행하면서, 익숙하지 않았던 도구나 새로운 코딩방식에 친숙해질 수 있었다.

마지막으로, 자율주행 자동차에 대해 공부할 때 알게 된 최적화 문제가 엘리베이터의 운행에도 적용될 수 있다는 사실에 놀랐으며, 내가 알고 있던 하나의 문제 해결의 방식이 여러 분야에 걸쳐서 적용될 수 있다는 것을 다시 한 번 알게 되었다.

김경민

하나의 프로젝트에 대해 각자 담당하는 부분을 정하고 이를 조율하고 협업하는 과정에서 역시 커뮤니케이션이 중요하다는 것을 느꼈다. 그런 점에서 SNS 으로 연락하는 것뿐만 아니라 Notion 이라는 협업 애플리케이션을 이용하여 서로의 작업 내용 등을 공유하며 프로젝트를 진행한 것이 도움되었다. 또한 가능한 자주 회의를 진행하여 서로의 진행 상황 및 프로젝트의 진행 방향에 대해 논의하고자 노력했고, 이를 회의록으로 기록한 것도 프로젝트에 큰 도움이 되었다.

하드웨어와 소프트웨어를 융합하여 프로젝트를 진행했다는 점에서도 의의가 있다. 특히 개인적으로는 하드웨어 파트 중 PCB 설계에서 새로운 구성 방식을 시도했는데, 결과가 괜찮아서 만족스러웠다. 또한 객체지향 프로그래밍을 처음 해보았는데 좋은 경험이 되었으며, 여러 명이 하나의 문제를 해결하기 위해 각자가 구상한 알고리즘을 각자의 스타일로 구현한 코드를 공유하니 사고의 폭도 넓어졌다.
프로젝트를 진행하며 아쉬운 점도 있었다. 앞서 말한 PCB 설계 중 더 효율적인 부품 배치가 가능했는데 여러 제약 조건으로 인해 그렇게 설계하지 못한 부분이 있다. 또한 시간 관계상 코드의 최적화를 하지 못한 것과 좀 더 큰 스케일의 건물에 대해 설계를 진행하지 못한 것이 아쉽다.
추가로 엘리베이터의 알고리즘을 조사하다 보니 직접 코드로 구현해보고 싶은 알고리즘이 있었는데, 이는 기존 시스템의 일부 수정이 필요한 부분이 있어서 이후에 시도해볼 필요가 있겠다.

이경호

이전에 프로젝트를 진행할 때 Github는 사용해보았다. 이번에 Notion을 처음 접해봤는데 팀 프로젝트에 맞춘 좋은 tool이란 생각이 들었다. 프로그래밍을 같이하며 서로의 코드가 하나로 모이는 과정에서 많은 시행착오가 있었으며 이를 극복하기 위해 많은 의사소통과 회의가 필요했다. 실제로 만나서 할 때도 있고 상황이 여의치 않은 경우 온라인으로 진행해 불편을 최소화한 것이 프로젝트 진행에 많은 도움이 된 것 같다. 이 모든 상황을 기록으로 남겨 Notion에 유지하였으며 이 또한 프로젝트에 많은 도움이 된 것 같다. 또한 평소 학교에서는 소프트웨어 중심으로 배워왔는데 이번 기회에 다른 친구들이 배운 하드웨어 분야를 접할 수 있게 되어 좋은 경험이 된 것 같다. 아직 프로그래밍이나 프로젝트 부분에 미흡한 부분이 많은 것 같다. 앞으로 이 분야에 종사하려면 많은 프로그래밍과 팀 프로젝트를 겪어야 하기에 더 많은 노력이 필요할 것 같다.

VII. 참고문헌

1) Lutfi Al-Sharif, Elevator Energy Simulation Model, 2004