제17회 임베디드SW경진대회
개발완료보고서[주니어 임베디드SW챌린저]

1. 팀 정보

팀명	TOT	팀장	신승엽
팀원	이승우, 이서진, 황서정, 김민권	팀원	

0. 작성 시 주의사항

※아래의 작성 양식(제출분량, 폰트, 크기, 줄 간격 등)을 미준수 시 서류 평가의 감점요인됨

※ 제출 분량 : A4 용지 상세내용 포함 20 page 이내
※ 작성 양식 (폰트 : 맑은 고딕 / 폰트 크기 : 10pt / 자간 : 0% / 장평 : 100% / 줄 간격 : 130%)
※ 제출 포맷 : pdf

2. 개발완료보고서

1. 개요

1.1. 작품 개요

3개의 컬러센서를 이용해 컬러센서 한 개는 도로교통사항을 알려주는 색상측정을 하고 나머지 두 개는 라인트레이싱에 사용되어 다익스트라 알고리즘을 통해 최단경로를 검색하여 주행에 접목시켜 도착지점까지 주행함. 장애물을 만났을 경우 우회하여 다른 최단경로를 검색하여 최대한 빠른 시간 안에 도착지점까지 주행하는 로봇이다.

1.2. 개발 목표

4차 산업혁명의 시대를 맞이하여 IT에 대한 중요성이 대두대고 있다. Mobility 분야도 중요 키워드 중하나이다. 무인 자동차도 나날이 발전해가고 있다 이미 미국의 대부분의 주는 무인 자동차가 실용단계에 들어서고 우리나라도 실용 단계를 검토하고 있다. 이에 TOT팀은 무인 자동차에 관심을 갖고 연구하게 되었고 추후에 지속적인 관심과 연구로 어떤 상황에서도 잘 대처할 수 있는 시스템을 만들어 차 안에 타는 사람의 안전을 100%보장할 수 있는 무인 자동차를 만들어 보고자 한다.

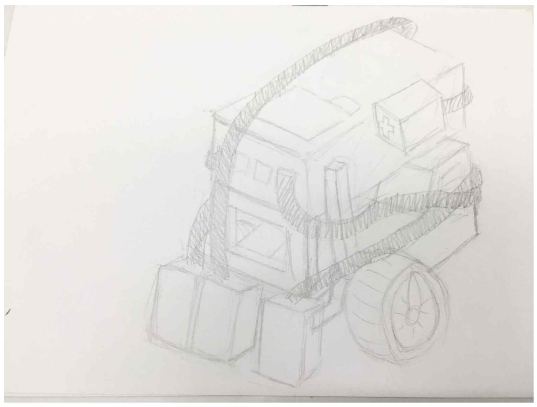
2. 개발 환경 설명

2.1 하드웨어(Hardware)

● 구성

● 설계

□ 로봇 하드웨어 디자인

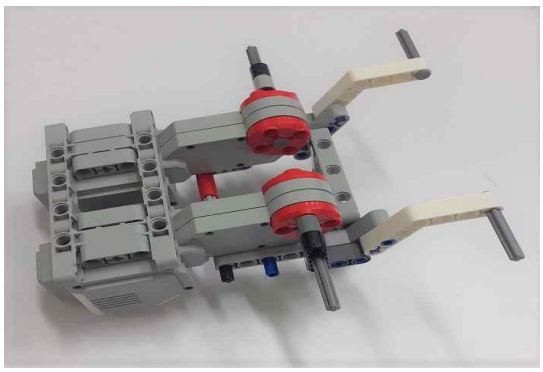


● 디자인한 하드웨어 제작

□ 본체 접합 전 기본 차체

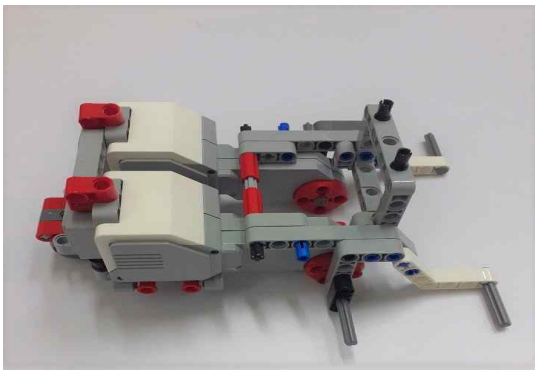
1) 아직 본체인 ev3를 접합시키기 전의 모습이며 팀의 모든 하드웨어 디바이스의 기초가 된 골격 부분

2) ev3의 기본 베이스봇의 기본 골격을 이용하였음



□ 본체 접합 전 차체

1) 본체인 p-brick을 접합시키기 위해 부품을 추가시켰음



□ 본체 접합 완료 차체

1) 본체인 p-brick을 접합 시킨 상태

2) 본체가 정중앙에 위치시켜 센서와 모터연결을 위한 거리를 확보 하였음

□ 주행 가능 차체

1) 본체접합 완료 차체에 바퀴를 연결한 상태이며 이때부터 기본적인 주행이 가능함

2) 아직 센서를 사용하기 전 이며 센서를 차체의 여러 부분에 쉽게 부착시킬 수 있음

□ 보조 주행 기능 탑재 차체

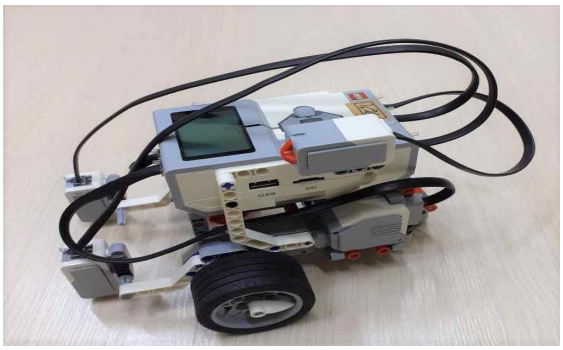
1) 컬러센서와 터치센서를 각각 하나씩 부착시킨 상태이며 컬러센서를 이용해 라인트레이싱을 할 수 있음

2) 이 모듈을 사용해 과제를 수행할 수도 있지만 프로그램이 복잡해지고 주행이 힘들어져서 하드웨어의 의존도를 높이기 위해 추가 센서를 탑재한 모델을 개발

□ 멀티 주행기능 탑재 차체

1) 컬러센서를 다중으로 탑재하여 과제를 보다 수월하게 수행할 수 있다.

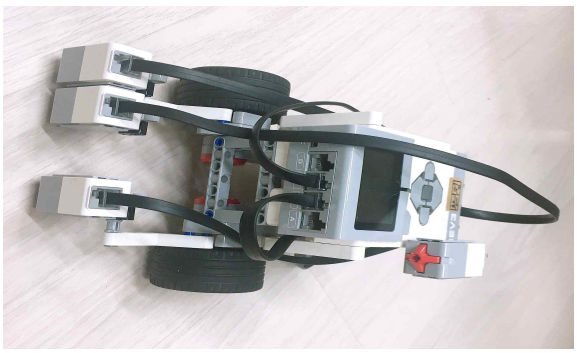
2) 센서를 다중으로 탑재함으로써 프로그래밍 의존도를 낮춤



□ 완성품

1) 센서를 3개 부착해 라인트레이싱과 색상측정이 가능하여 과제수행을 효율적으로 수행 가능하다.

2) 센서를 연결할 때 쓰는 연결선이 너무 팽팽하여 센서값이 오차가 나기도 하였으나 연결선에 대한 길이를 확보함으로써 문제를 해결



● 하드웨어 테스트

□ 1차 테스트

테스트 : 칼라센서를 한 개 장착 후 테스트, 미션 맵 주행

오 류 : 컬러센서 한 개를 장착할 경우 교통상황을 파악하기 위한 테이프를 읽고 나서 홈으로 귀환한 후 해당 컬러센서로 노드를 따라가기 위해서는 수동으로 차체의 위치를 변경하거나 소프트웨어로 위치를 강제로 바꾸어 줘야하는 불편한 상황이 발생

수 정 : 컬러센서를 두 개 장착, 미션맵 주행 시 컬러센서 A는 노드들의 교통상황을 선 파악하는 색 테이프 인식용으로 사용하고, 컬러센서 B는 노드선을 따라가기 위한 용도로 사용

□ 2차 테스트

테스트 : 칼라센서를 두 개 장착 후 테스트, 미션 맵 주행

오 류 : 미션맵 주행 시 1차 테스트의 오류는 수정을 하였으나 원형 노드, 직선노드 연결 하는 부분에서 제대로 인식하지 못하고 일정 오차값이 발생

수 정 : 컬러센서를 세 개 장착, 컬러센서 B, C의 경우 노드 라인트레이싱에 대한 오차값을 줄임

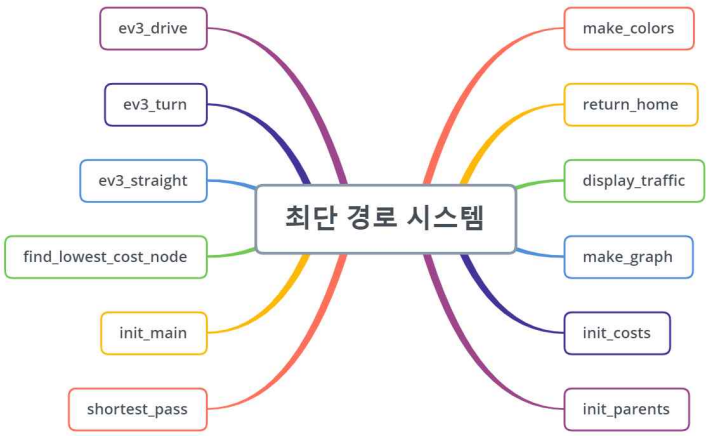
□ 3차 테스트

테스트 : 컬러센서 세 개 장착 후 테스트, 미션 맵 주행

수 정 : 2차 테스트의 오류를 수정함을 확인함

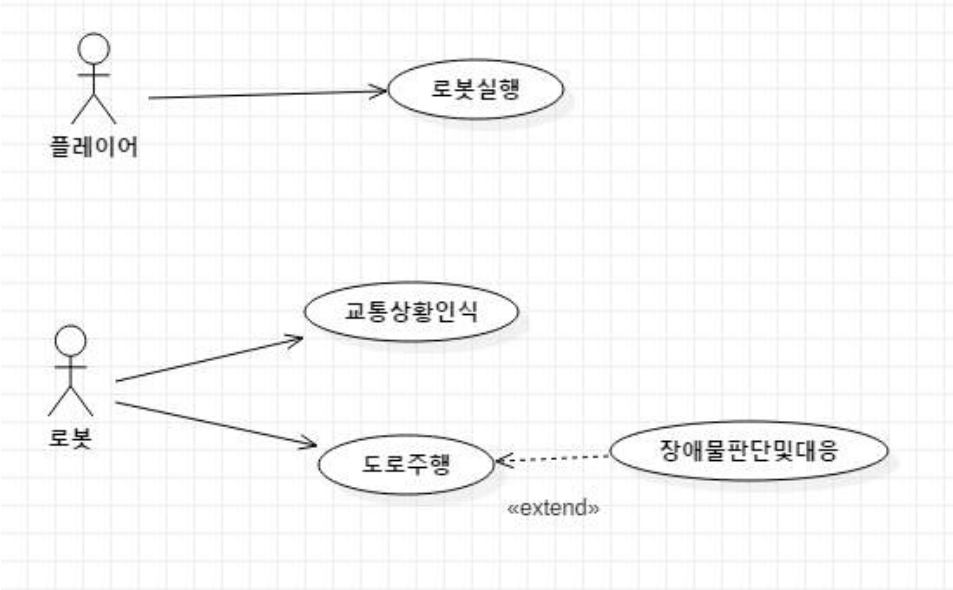
2.2 소프트웨어(Software)

● Software 구성

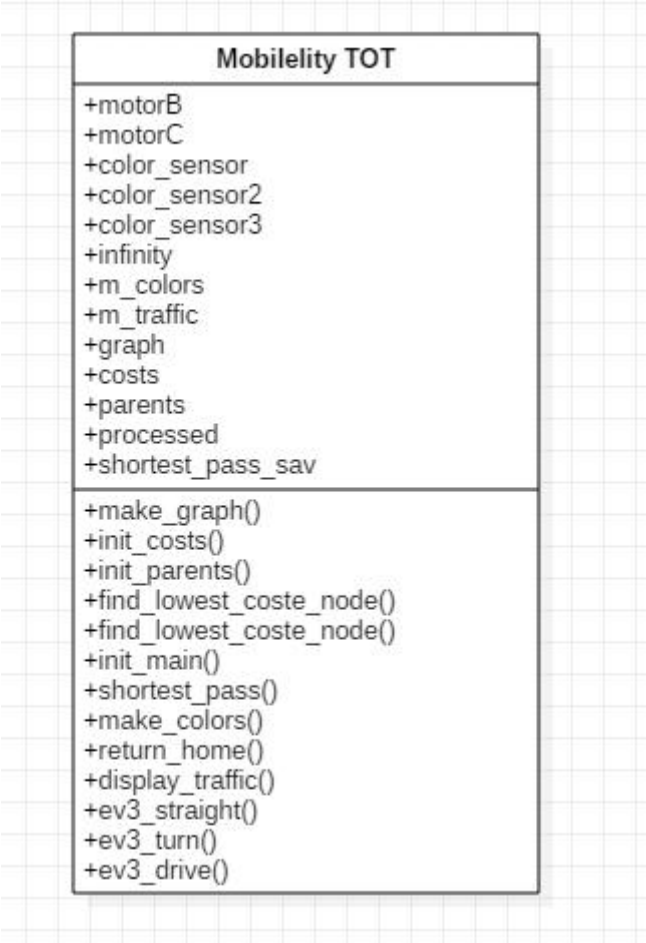


● Software 설계도

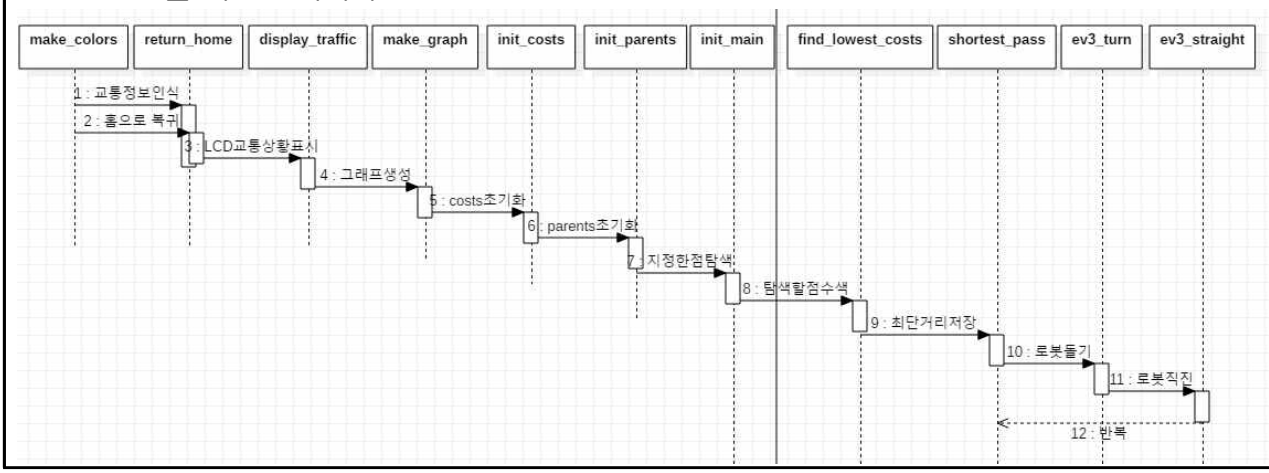
□ 유즈케이스 다이어그램



□ 클래스 다이어그램

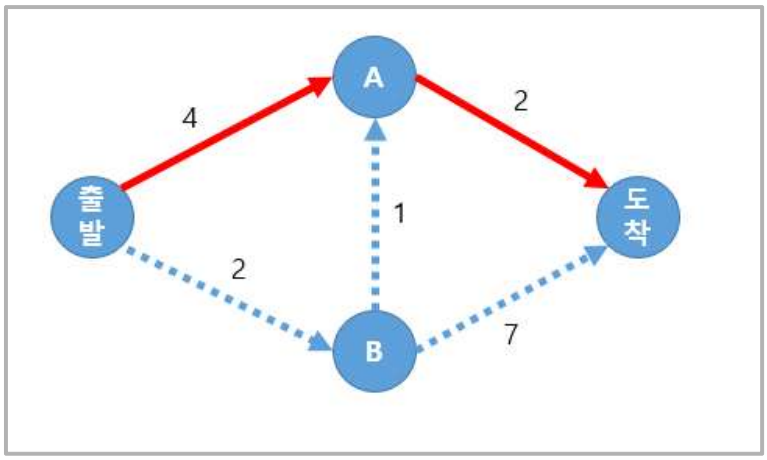


□ 시퀀스 다이어그램



● Software 기능(알고리즘 설명 포함)

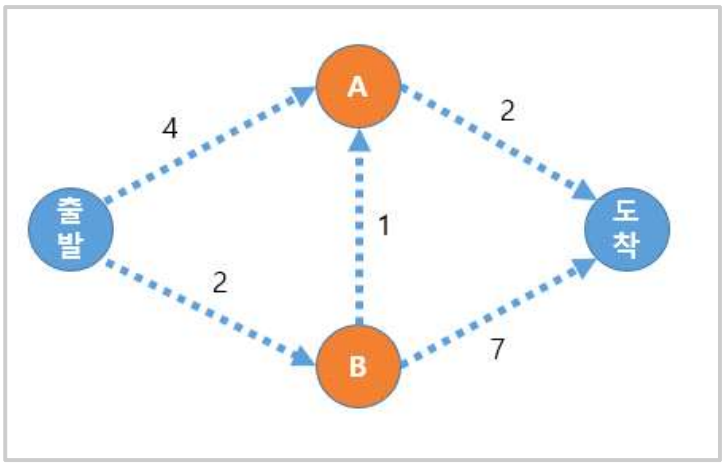
◈ 다익스트라 알고리즘



가중치가 있는 경로의 경우에 너비우선 탐색으로 구한 경로는 시간이 6분이 걸림
경로시스템의 경우 최단거리가 아닌 최단시간 알고리즘을 구하므로 다익스트라 알고리즘을 사용하여 최단시간 노선을 구함

[다익스트라 알고리즘을 구하는 방법]
- 현재 노드에서 가장 시간이 적게 걸리는 정점을 구함
- 현재 노드에서 이웃노드까지 가는데 걸리는 시간을 구함
- 모든 정점에 대해 반복
- 최종 경로를 계산

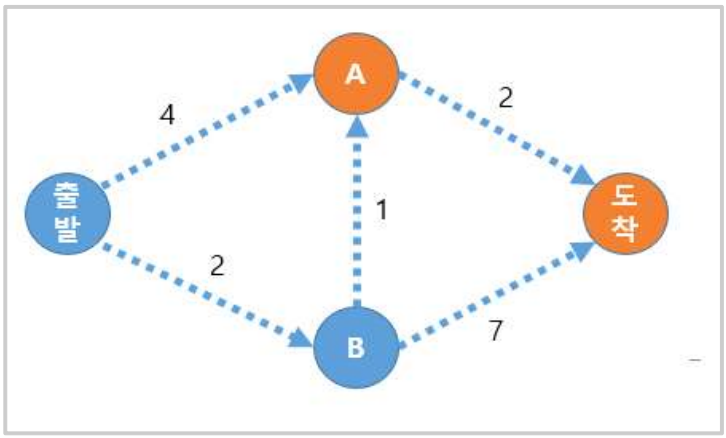
[출발에서 이웃한 정점들까지 가는데 걸리는 시간 구하기]



A	4
B	2
도착점	∝

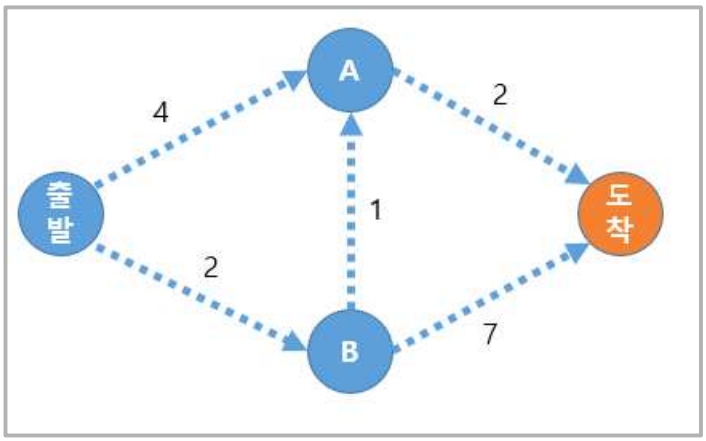
[정점 B의 이웃 정점들에 대해 시간 계산하기]

이전에 계산된 출발에서 가장 가까운 노드가 B이므로 B노드를 기준으로 이웃한 노드들의 시간을 계산하여 짧은 시간을 계산
정점 A로 가는 거리를 B를 통해서 가면 더 빠르므로 4에서 3으로 변경하고,
도착점으로 가는 거리는 B를 통해서 가면 9분이면 가므로 ∝(무한대)에서 9로 변경



A	3
B	2
도착점	7

[정점 A의 이웃 정점들에 대해 시간 계산하기]



A	3
B	2
도착점	5

\

◈ 데이터 구조

3개의 해시테이블을 만듬. 해시테이블이란 컴퓨터가 데이터를 관리하기 위해 키와 값을 쌍으로 관리하는 구조

출발점	A	4
	B	2
A	도착점	2
B	A	1
	도착점	7
도착점	-------	

그래프

A	4
B	2
도착점	∝

가격

A	출발점
B	출발점
도착점	-------

부모

◈ 소스

```
233  def find_lowest_cost_node(costs):
234      lowest_cost = float("inf")
235      lowest_cost_node = None
236
237      for node in costs:
238          cost = costs[node]
239          if cost < lowest_cost and node not in processed:
240              lowest_cost = cost
241              lowest_cost_node = node
242      return lowest_cost_node
243
244  def init_main():
245      node = find_lowest_cost_node(costs)
246
247      while node is not None:
248          cost = costs[node]
249          neighbors = graph[node]
250          for n in neighbors.keys():
251              new_cost = cost + neighbors[n]
252              if costs[n] > new_cost:
253                  costs[n] = new_cost
254                  parents[n] = node
255
256          processed.append(node)
257          node = find_lowest_cost_node(costs)
258
```

```
259  def shortest_pass():
260      stack="NO-9R"
261      samsteak=9
262      i=0
263      shortest_pass_save.append("NO-9R")
264
265      while parents[stack] !='start':
266          shortest_pass_save.append(parents[stack])
267          stack=parents[stack]
268
269          if samsteak!=stack[3]:
270              samsteak=stack[3]
271              i=i+1
272              brick.display.text(stack[3], (50-10*i,
273              shortest_pass=samsteak
274
275      wait(7000)
276
277  def ev3_straight():
278      motorB.run(100)
279      motorC.run(-20)
280
281
282      wait(2100)
283      while True:
284          colorss = color_sensor3.color()
285          reflection = color_sensor2.reflection()
286          if colorss == 1:
287              brick.sound.beep()
288              break
289          elif colorss == 5:
290              return 0
```

```
        else:
            if reflection <= 54:
                motorB.run(200)
                motorC.run(130)
            elif reflection >= 55:
                motorB.run(130)
                motorC.run(200)
    motorB.run(100)
    motorC.run(20)

    wait(1800)

def ev3_turn():
    while True:
        colorss = color_sensor3.color()
        reflection = color_sensor2.reflection()
        if colorss == 1:
            brick.sound.beep()
            break
        else:
            if reflection <= 54:
                motorB.run(200)
                motorC.run(90)
            elif reflection >= 55:
                motorB.run(30)
                motorC.run(200)
    motorB.run(-15)
    motorC.run(110)

    wait(740)
```

```
    while True:
        if shortest_pass_save[count] == "NO-9R":
            motorB.run(100)
            motorC.run(-20)

            wait(1250)

            motorB.run(90)
            wait(1400)

            motorB.run(200)
            motorC.run(200)

            wait(4400)

            break
```

```
        if graph[shortest_pass_save[count]][shortest_pass_s
            if_color_red=ev3_straight()


        else:
            b=shortest_pass_save[count]
            b=int(b[3])
            turnCount = graph[shortest_pass_save[count]][sh
            if turnCount == 1/4:
                ev3_turn()
            elif turnCount == 1/2:
                ev3_turn()
                ev3_turn()
            elif turnCount == 3/4:
                ev3_turn()
                ev3_turn()
                ev3_turn()
        count = count + 1
```

```
make_colors()
return_home()
display_traffic()
make_graph()
init_costs()
init_parents()
init_main()
shortest_pass()
ev3_drive()
```

◈ 노드 그래프 구성부

```
def make_graph():
    graph["NO-1L"] = {}
    graph["NO-1L"]["NO-1D"] = m_traffic[0]*(1/4)

    graph["NO-1D"] = {}
    graph["NO-1D"]["NO-1R"] = m_traffic[0]*(1/4)
    graph["NO-1D"]["NO-4U"] = 0

    graph["NO-1R"] = {}
    graph["NO-1R"]["NO-2L"] = 0

    graph["NO-2L"] = {}
    graph["NO-2L"]["NO-2D"] = m_traffic[1]*(1/4)

    graph["NO-2D"] = {}
    graph["NO-2D"]["NO-2R"] = m_traffic[1]*(1/4)
    graph["NO-2D"]["NO-5U"] = 0

    graph["NO-2R"] = {}
    graph["NO-2R"]["NO-3L"] = 0

    graph["NO-3L"] = {}
    graph["NO-3L"]["NO-3D"] = m_traffic[2]*(1/4)

    graph["NO-3D"] = {}
    graph["NO-3D"]["NO-6U"] = 0

    graph["NO-4U"] = {}
    graph["NO-4U"]["NO-4D"] = m_traffic[3]*(1/2)

    graph["NO-4D"] = {}
    graph["NO-4D"]["NO-4R"] = m_traffic[3]*(1/4)
    graph["NO-4D"]["NO-7U"] = 0

    graph["NO-4R"] = {}
    graph["NO-4R"]["NO-5L"] = 0
```

```
    graph["NO-5U"] = {}
    graph["NO-5U"]["NO-5L"] = m_traffic[4]*(1/4)

    graph["NO-5L"] = {}
    graph["NO-5L"]["NO-5D"] = m_traffic[4]*(1/4)

    graph["NO-5D"] = {}
    graph["NO-5D"]["NO-5R"] = m_traffic[4]*(1/4)
    graph["NO-5D"]["NO-8U"] = 0

    graph["NO-5R"] = {}
    graph["NO-5R"]["NO-6L"] = 0

    graph["NO-6U"] = {}
    graph["NO-6U"]["NO-6L"] = m_traffic[5]*(1/4)

    graph["NO-6L"] = {}
    graph["NO-6L"]["NO-6D"] = m_traffic[5]*(1/4)

    graph["NO-6D"] = {}
    graph["NO-6D"]["NO-9U"] = 0

    graph["NO-7U"] = {}
    graph["NO-7U"]["NO-7R"] = m_traffic[6]*(3/4)

    graph["NO-7R"] = {}
    graph["NO-7R"]["NO-8L"] = 0

    graph["NO-8L"] = {}
    graph["NO-8L"]["NO-8R"] = m_traffic[7]*(1/2)
```

```
    graph["NO-8R"] = {}
    graph["NO-8R"]["NO-9L"] = 0

    graph["NO-8U"] = {}
    graph["NO-8U"]["NO-8L"] = m_traffic[7]*(1/4)

    graph["NO-9U"] = {}
    graph["NO-9U"]["NO-9L"] = m_traffic[8]*(1/4)

    graph["NO-9L"] = {}
    graph["NO-9L"]["NO-9R"] = m_traffic[8]*(1/2)
    graph["NO-9R"] = {}

def init_costs():
    costs["NO-1L"] = 0
    costs["NO-1D"] = infinity
    costs["NO-1R"] = infinity

    costs["NO-2L"] = infinity
    costs["NO-2D"] = infinity
    costs["NO-2R"] = infinity

    costs["NO-3L"] = infinity
    costs["NO-3D"] = infinity

    costs["NO-4U"] = infinity
    costs["NO-4R"] = infinity
    costs["NO-4D"] = infinity

    costs["NO-5U"] = infinity
    costs["NO-5R"] = infinity
    costs["NO-5D"] = infinity
    costs["NO-5L"] = infinity
```

```
    parents["NO-4U"] = None
    parents["NO-4R"] = None
    parents["NO-4D"] = None

    parents["NO-5U"] = None
    parents["NO-5L"] = None
    parents["NO-5R"] = None
    parents["NO-5D"] = None

    parents["NO-6U"] = None
    parents["NO-6D"] = None
    parents["NO-6L"] = None

    parents["NO-7U"] = None
    parents["NO-7R"] = None
    parents["NO-4D"] = None

    parents["NO-8U"] = None
    parents["NO-8R"] = None
    parents["NO-8L"] = None

    parents["NO-9U"] = None
    parents["NO-9R"] = None
    parents["NO-9L"] = None
```

```
    costs["NO-6U"] = infinity
    costs["NO-6D"] = infinity
    costs["NO-6L"] = infinity

    costs["NO-7U"] = infinity
    costs["NO-7R"] = infinity

    costs["NO-8U"] = infinity
    costs["NO-8R"] = infinity
    costs["NO-8L"] = infinity

    costs["NO-9U"] = infinity
    costs["NO-9R"] = infinity
    costs["NO-9L"] = infinity

def init_parents():
    parents["NO-1L"] = "start"
    parents["NO-1D"] = None
    parents["NO-1R"] = None

    parents["NO-2L"] = None
    parents["NO-2D"] = None
    parents["NO-2R"] = None

    parents["NO-3L"] = None
    parents["NO-3D"] = None
```

◈ 로봇 주행부

```python
#!/usr/bin/env pybricks-micropython

from pybricks import ev3brick as brick
from pybricks.ev3devices import (Motor, TouchSensor, ColorSensor,
                                 InfraredSensor, UltrasonicSensor, Gyrc
from pybricks.parameters import (Port, Stop, Direction, Button, Color,
                                 SoundFile, ImageFile, Align)
from pybricks.tools import print, wait, StopWatch
from pybricks.robotics import DriveBase

motorB = Motor(Port.B)
motorC = Motor(Port.C)
color_sensor = ColorSensor(Port.S1)
color_sensor2 = ColorSensor(Port.S3)
color_sensor3 = ColorSensor(Port.S2)

infinity = float("inf")
m_colors=[8,8,8,8,8,8,8,8,8]
m_traffic=[8,8,8,8,8,8,8,8,8]
graph = {}
costs = {}
parents = {}
processed = []
shortest_pass_save=[]
i=0

def make_colors():

    while True:
        motorB.run(100)
        motorC.run(100)

        color = color_sensor.color()
        if color == 3:
            break
        elif color == 4:
            break
        elif color == 5:
            break
    motorB.run(-100)
    motorC.run(-100)

    wait(90)

    brick.display.clear()

    for i in range(0,9):
        motorB.run(100)
        motorC.run(100)
        wait(350)
        color = color_sensor.color()

        if color == 3:
            m_colors[i] = 'G'
            m_traffic[i] = 4
        elif color == 4:
            m_colors[i] = 'Y'
            m_traffic[i] = 8
        elif color == 5:
            m_colors[i] = 'R'
            m_traffic[i] = 12

def return_home():
    motorB.run(-100)
    motorC.run(-100)

    wait(5000)

    motorB.run(0)
    motorC.run(0)

def display_traffic():
    for i in range(0,9):
        brick.display.text(m_colors[i], (10*i, 50))
    wait(7000)
```

● 프로그램 사용법(Interface)

1) 물리적인 로봇의 위치와 각도를 제대로 세팅

컬러센서 A는 색 절연 테이프 뒤에 위치, 컬러센서 B,C는 회색 노드에 위치하도록 세팅

(x)

(o)

(x)

2) P-brick에서 프로그램을 실행

● 개발환경(언어, Tool, 사용시스템 등)

언어 : microPython

Tool : Visual Studio Code

사용시스템 : OS/windows10 64 bit, RAM/ 32GB

3. 개발 프로그램 설명

3.1 파일 구성

3.2 함수별 기능

● 노드데이터 생성부

◈ make_graph() : 각 노드와 그 노드와 이웃하는 노드의 거리를 m_traffic[x](x번째 도로 색상정보 에서 한 바퀴를 돌 때 걸리는 시간)을 이용하여 나타내는 함수

◈ Init_costs() : 자신이 출발할 지점에서 어떤 노드까지 가는데 걸리는 치소 시간을 숫자로 표현, (출발(NO-1L):0 그 외 : infinity)

◈ init_parents() : 어떤 노드를 가기 위해 전에 거쳐야하는 노드를 정의 하는 함수 (출발(NO1L):start 그 외: None)

● 다익스트라 알고리즘

◈ find_lowest_cost_node() : 다익스트라 알고리즘에서 로봇이 처음에 탐색할 노드를 찾는 함수 탐색하지 않은 노드 중에 가장 출발에서 노드까지 가는 시간이 적은 노드를 탐색.

◈ init_main() : find_lowest_cost_node의 함수에서 찾은 탐색 할 노드를 탐색한다(탐색할 노드의 이웃한 점을 탐색)

◈ shortest_pass() : 디스플레이에 우리가 가야하는 경로를 표시하고 shortest_pass_save에 가야할 길을 저장(ex NO-1L NO-1D NO-4U NO-4D...)

● 로봇 제어부

◈ make_colors() : 색깔을 인지해 m_color에 저장하는 함수 그리고 m_traffic에 노드별로 한 바퀴를 돌았을 때의 시간을 m_traffic에 저장.(GREEN:4 ,YELLOW:8 ,RED:12)

◈ return_home() : 로봇이 make_colors()를 이용해 색깔 인지를 완료하면 다시 출발지점으로 돌아가고 로봇은 5초 동안 정지

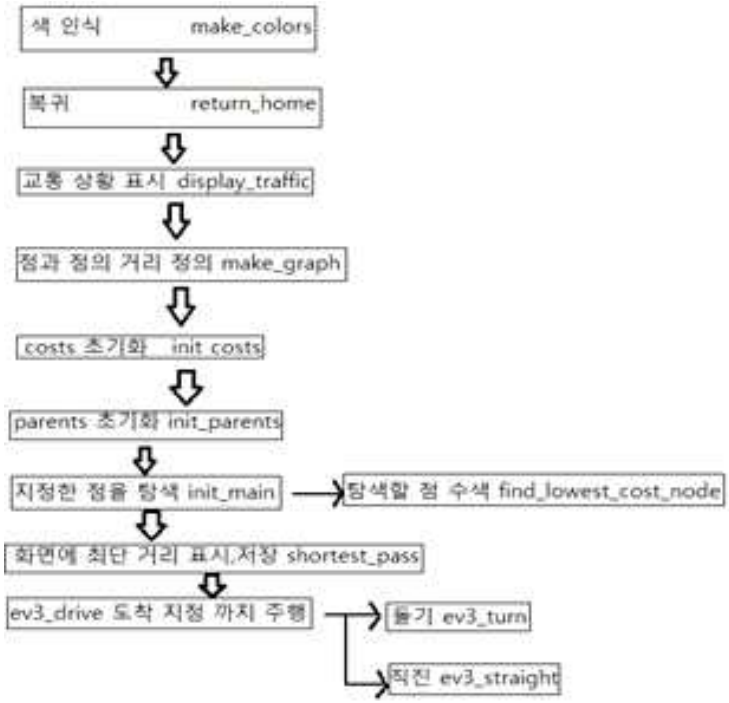
◈ display_traffic() 로봇이 make_colors()를 이용해 입력 받은 색상 정보를 7초 동안 출력해주는 함수(동시에 7초를 기다려 준다) 색상 정보 출력형식은 입력 받은 색상 정보를 RGB(red,green,blue)이 새 가지로 나눠 출력하는 함수이

◈ ev3_straight() 제자리에서 오른쪽으로 조금 돌고 라인트래이서를 사용하여 직진모양으로 컬러 샌서(color_sensor3)를 이용하여 검정점이 인지 될 때까지 직진을 실행하는 함수

◈ ev3_turn() 검정색 점(1/4)이 인지 될 때 까지 반사광을 이용한 라인트래이서를 하며 회전

◈ ev3_drive() 위에 함수와 변수들을 토대로 실질적(물리적)으로 조사한 최단 경로를 토대로 다음 점을 가기 위해서는 어떤 함수를 이용해서 가야하는지 (straight(직진), turn(90도), turn x2(180도), turn x3(270도))를 판단해서 결국 끝점에 도달하게 됨.

3.3 주요 함수의 흐름도



색 인식 -> 복귀-> 교통상황 표시-> 점과 점의 거리 정의-> costs 초기화->parents 초기화
->지정한 점을 탐색(탐색을 안 한 노드가 없을 때 까지 탐색할 노드를 찾는다)
->화면에 최단 거리 표시, 저장->도착 지점 까지 주행
(필요시 ev3_turn, ev3_straight 호출)

3.4 기술적 차별성

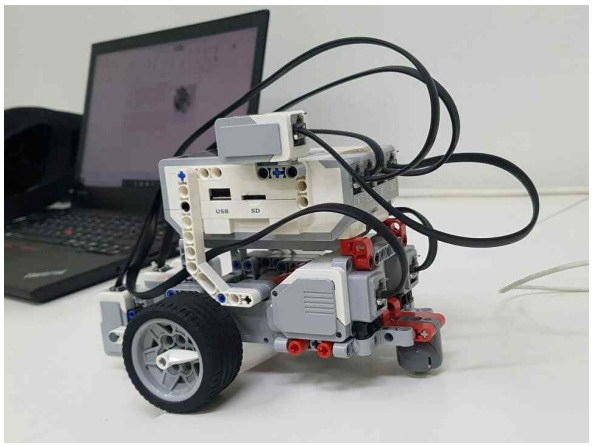
하드웨어적으로는 컬러센서를 3개 사용하여 한 개는 노드정보가 담겨져 있는 컬러테이프를 통해 교통복잡도 정보를 읽어오고 두 개는 노드를 따라가기 위한 라인트레이싱을 위해 사용함으로서 전체적인 시스템에 안정감을 줌

소프트웨어적으로는 최단거리 알고리즘이 아니라 최단시간 알고리즘인 다익스트라 알고리즘을 사용하여 시스템의 정확도를 올리고 있음

4. 개발 중 장애요인과 해결방안

2번 항목의 개발 환경 설명 -> [하드웨어 테스트], [소프트웨어 테스트] 부분에 명시

5. 개발결과물의 차별성



● 심플한 디자인

심플한 디자인으로 로봇을 보다 더 세련되고 가볍게 제작 그리고 모터와 모터 사이를 핀으로 단단히 고정하므로서 내구성을 좋게 만듬

● 오차 해결

처음 새팅(2-4)에 오차가 있다고 해도 조금에 오차는 로봇에서 자채적으로 오차를 해결해 줌(라인 트래이싱 부분)

● 컬러센서 3개 사용

컬러센서 역할을 분장하기 위해 컬러센서를 3개를 사용. 노드 교통정보 인식 용도와 검정색 점 인식 라인트래이서 용도로 나눔.

6. 단계별 개발계획 및 실제 참여인원 및 업무 분장

개발타입	프로세서	업무 분장		개발예정일정	담당
규정	분석			07/10~07/15	팀원전체
H/W	분석	모빌리티 로봇들 벤치마킹		07/10~07/15	
	설계	로봇 하드웨어 디자인		07/12~07/17	신승엽, 김민권
	제작 및 테스트	기본차체 제작		07/17~07/17	
		본체탑제 제작		07/17~07/17	
		보조주행 기능 탑재 차체		07/17~07/17	
		보조주행 기능 탑재 차체 소프트웨어 파일럿 테스트		07/17~07/17	
		멀티 주행기능 탑재 차체		07/18~07/20	
		멀티 주행기능 탑재 차체 소프트웨어 파일럿 테스트		07/18~07/20	
SW	분석	유즈케이스 분석		07/10~07/12	이승우, 이서진, 황서정
	설계	클래스 다이어그램	다익스트라 알고리즘	07/10~07/15	이서진
			노드그래프 구성부	07/10~07/15	이승우
			로봇 주행부	07/10~07/15	황서정
		시퀀스 다이어 그램	다익스트라 알고리즘	07/16~07/20	이서진
			노드그래프 구성부	07/16~07/20	이승우
			로봇 주행부	07/16~07/20	황서정
	구현	다익스트라 알고리즘 구현 및 테스트		07/16~07/29	이서진
		노드그래프 구성부 구현 및 테스트		07/16~07/29	이승우
		로봇 주행부 구현 및 테스트		07/16~07/29	황서정
	테스트	모빌리티 시스템 테스트		07/29~08/01	팀원전체