제17회 임베디드SW경진대회
개발완료보고서[주니어 임베디드SW챌린저]

1. 팀 정보

팀명	YE0W01_3.θ	팀장	남궁찬
팀원	안재원	팀원	김준민
팀원	오준혁	팀원	김성찬

2. 개발완료보고서

1. 개요

1.1. 작품 개요

이제는 자율주행자동차가 실제로도 만들어지고 시험주행되고도 있으며, 많은 회사들이 이 사업에 참여하고 있다. 이러한 시점에서 기계들을 다루는 엔지니어나 소프트웨어를 다루는 프로그래머라는 직업이 주목을 받고 있다.

우리는 이러한 기술들에 관심이 있었다. 그렇게 우리가 이런 것을 직접 해볼 수 있는 것이 있는지 찾아보다가 이 대회에 대해 알게 되었고 이 대회에 참가하게 되었다. 그리고 이대회는 우리에게 컴퓨팅 사고력과 협동력을 기르고 더 다양한 경험을 줄 수 있는 좋은 기회가 될 것이라 생각한다.

1.2. 개발 목표

이 대회의 문제를 해결하기 위해서는 로봇의 구조 즉 하드웨어의 안정성과 효율성도 필요하지만 결국에는 그 로봇을 작동시키고 문제 상황을 해결할 소프트웨어, 알고리즘이 가장 중요한 것이라 생각했다. 그래서 이 대회의 문제를 해결할 알고리즘을 직접 구상해보고 만들어보고 싶었다.이렇게 함으로써 우리는 서로의 아이디어를 모아보고 상의하면서 협동력을 키울 수 있을 것이라 생각하고 로봇이 자동으로 최단경로를 찾아내는 알고리즘을 만들어 봄으로써 컴퓨팅 사고력을 키우고 알고리즘에 대해 더 잘 이해할 수 있을 것이라 생각한다.

2. 개발 환경 설명 (최대한 자세하게 기술)

2.1. Software 구성

로봇에는 ev3dev 운영체제를 설치했고 ev3 python을 이용하여 소프트웨어를 설계했다.

2.5. 개발환경 (언어, Tool, 사용시스템 등)

개발언어 : Python3

개발 Tool : Visual Studio Code

사용시스템 : Windows 10

3. 개발 프로그램 설명 (최대한 자세하게 기술)

3.1. 파일 구성

main.py

3.2. 함수별 기능

r_1 : 1단계 미션을 해결하는 함수로 센서 2개를 사용해 라인트레이싱을 하면서 나머지 1개의 센서를 활용하여 교차로의 교통정보를 읽은 뒤 그 교통정보를 배열에 저장한다.

safe : x좌표와 y좌표가 입력되는데 이 좌표가 배열밖을 벗어났는지 아닌지 확인한다.

f_des : 로봇이 읽은 교통정보를 바탕으로 로봇이 3차원 배열에서 로봇이 가상으로 주행하여 그 시간값을 교차로의 부분 부분마다 배열에 저장하여 최종적으로 모든 교차로의 위치에 대한 시간값이 나오는 배열을 만드는 함수이다.

f_pass : f_des 에서 저장한 시간값이 저장된 배열을 사용하는데 도착지점부터 로봇이 출발지점으로 역으로 올라가는 방식이다. 그래서 도착지점의 시간값에서 그전의 시간값이 무엇인지 판단하여 이 로봇이 전에 있던 위치가 어디인지 파악하게 된다. 이런 행동을 반복하여 최종적으로 로봇이 출발지점에 도착하게 되고 최단경로를 구할 수 있는 것이다.

f_route : f_pass 에서 탐색한 경로를 이용하여 로봇이 현재 있는 위치와 다음으로 갈 위치를 구하고 로봇이 주행해야 할 길의 방향과 교차로 회전 함수의 반복 횟수를 찾아낸다. 이를 바탕으로 주행과 회전을 실행한다.

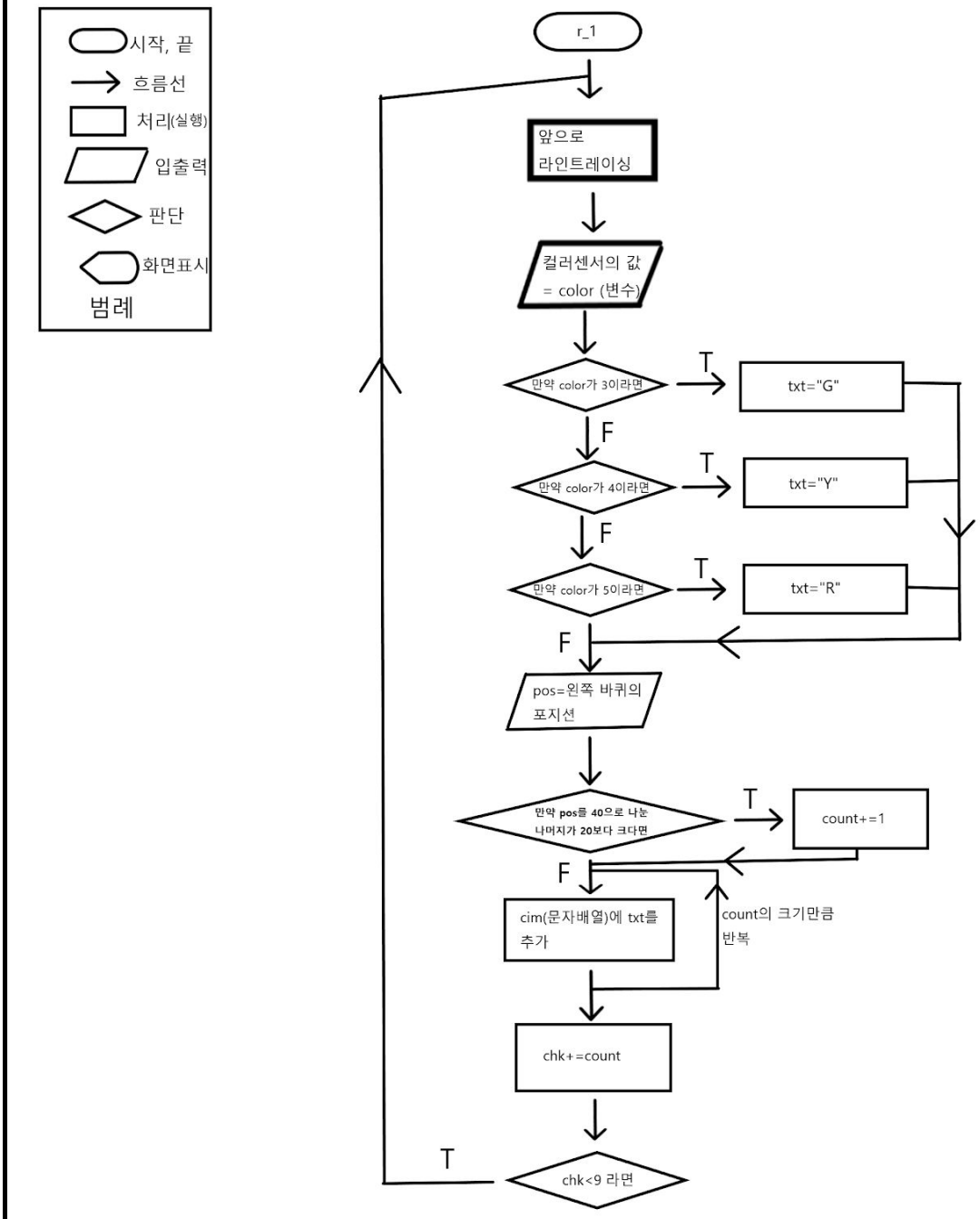
start : 3단계를 시작할 때 라인트레이싱을 하면서 첫번째 교차로에 도착하게 하는 함수이다.

go : 교차로에서 로봇이 라인트레이싱을 이용하여 교차로의 1/4만큼을 회전하게 하는 함수이다.

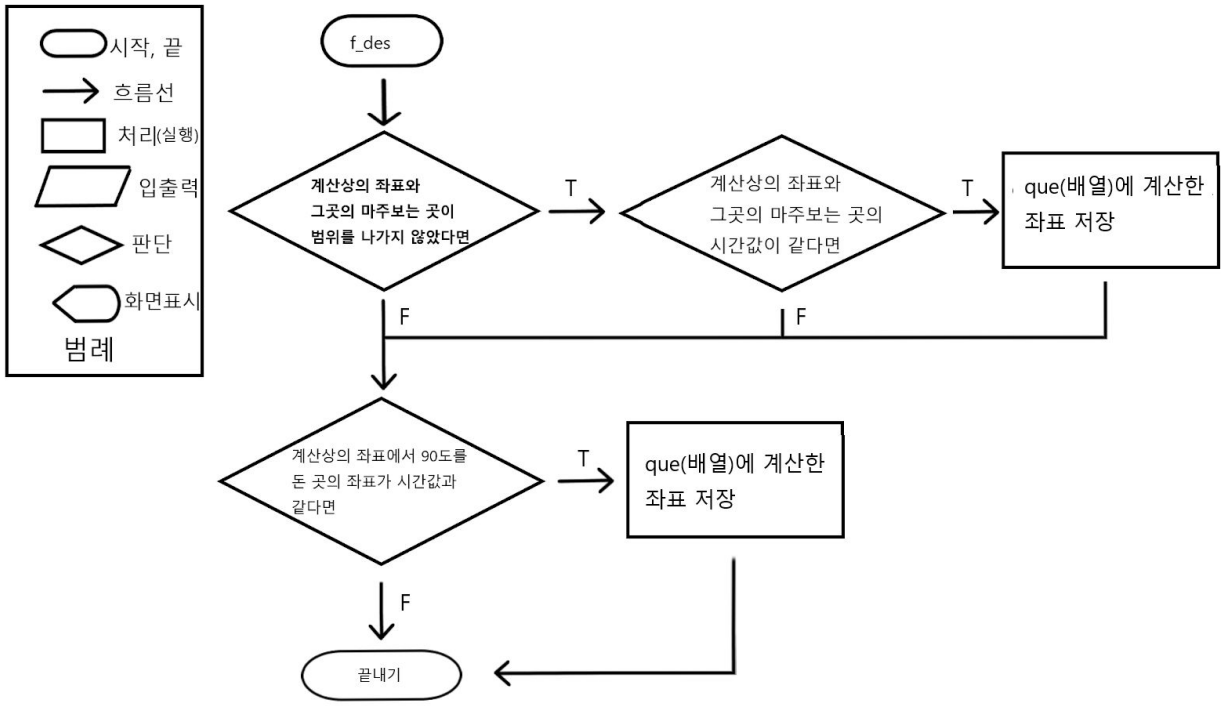
turn : 로봇이 교차로와 교차로 사이를 이동하게 하는 함수 교차로 이동 도중 빨간색(돌발상황)을 만나게 되면 다시 원래 있었던 자리로 돌아가게 되고 이후 f_route 안에서 다시 최단경로를 찾기 위한 작업이 시작되게 한다.

reset : 저장된 모든 시간정보와 way(최단경로)의 모든 값을 초기화한다.

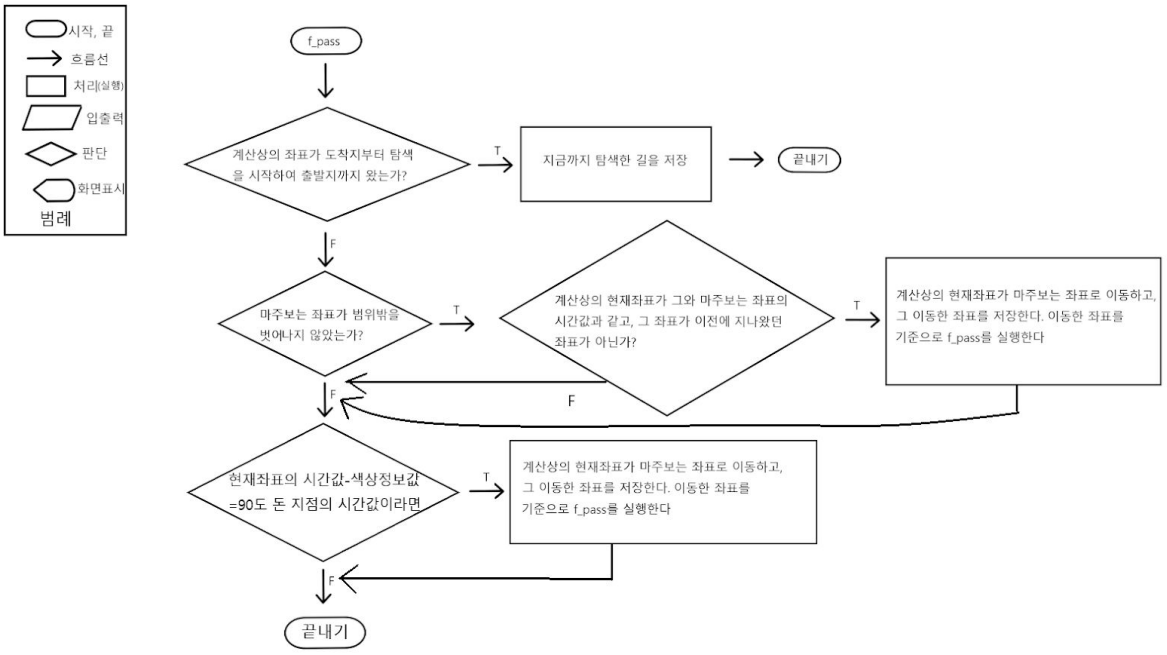
3.3. 주요 함수의 흐름도


시작, 끝
흐름선
처리(실행)
입출력
판단
화면표시
범례
r_1
앞으로
라인트레이싱
컬러센서의 값
= color (변수)
만약 color가 3이라면
T
txt="G"
F
만약 color가 4이라면
T
txt="Y"
F
만약 color가 5이라면
T
txt="R"
F
pos=왼쪽 바퀴의
포지션
만약 pos를 40으로 나눈
나머지가 20보다 크다면
T
count+=1
F
cim(문자배열)에 txt를
추가
count의 크기만큼
반복
chk+=count
T
chk<9 라면

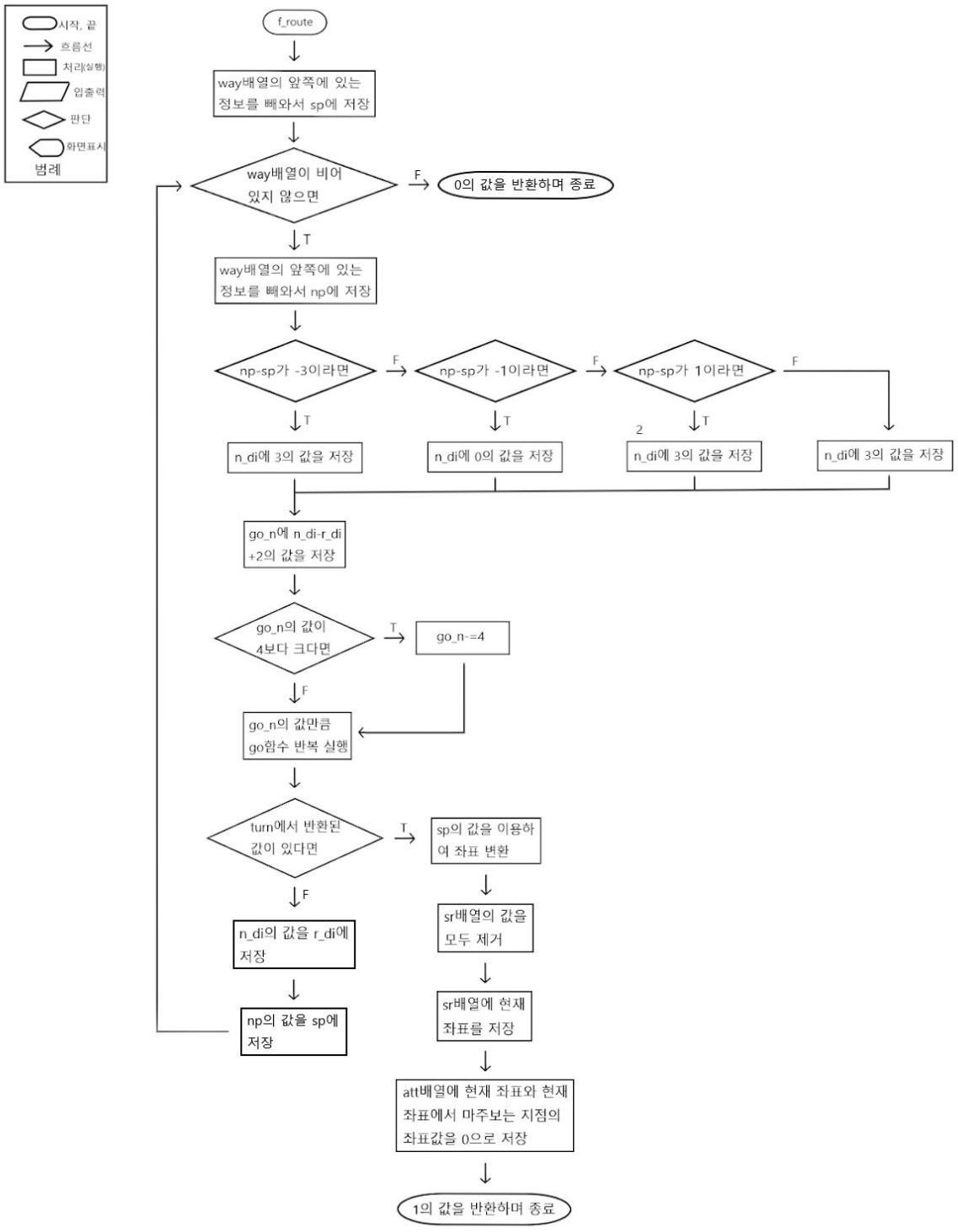
r_1


시작, 끝
흐름선
처리(실행)
입출력
판단
화면표시
범례
f_des
계산상의 좌표와
그곳의 마주보는 곳이
범위를 나가지 않았다면
T
계산상의 좌표와
그곳의 마주보는 곳의
시간값이 같다면
T
que(배열)에 계산한
좌표 저장
F
F
계산상의 좌표에서 90도를
돈 곳의 좌표가 시간값과
같다면
T
que(배열)에 계산한
좌표 저장
F
끝내기

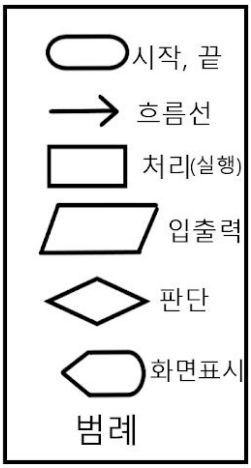
f_des

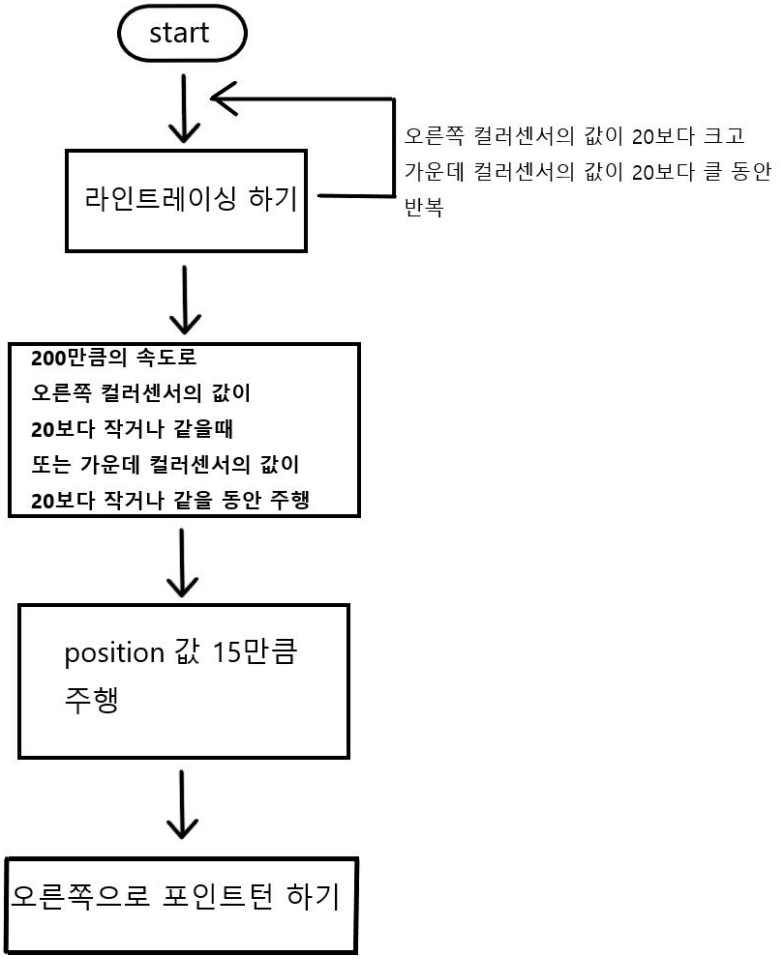

시작, 끝
흐름선
처리(실행)
입출력
판단
화면표시
범례
f_pass
계산상의 좌표가 도착지부터 탐색
을 시작하여 출발지까지 왔는가?
T
지금까지 탐색한 길을 저장
끝내기
F
마주보는 좌표가 범위밖을
벗아나지 않았는가?
T
계산상의 현재좌표가 그와 마주보는 좌표의
시간값과 같고, 그 좌표가 이전에 지나왔던
좌표가 아닌가?
T
계산상의 현재좌표가 마주보는 좌표로 이동하고,
그 이동한 좌표를 저장한다. 이동한 좌표를
기준으로 f_pass를 실행한다
F
F
현재좌표의 시간값-색상정보값
=90도 돈 지점의 시간값이라면
T
계산상의 현재좌표가 마주보는 좌표로 이동하고,
그 이동한 좌표를 저장한다. 이동한 좌표를
기준으로 f_pass를 실행한다
F
끝내기

f_pass

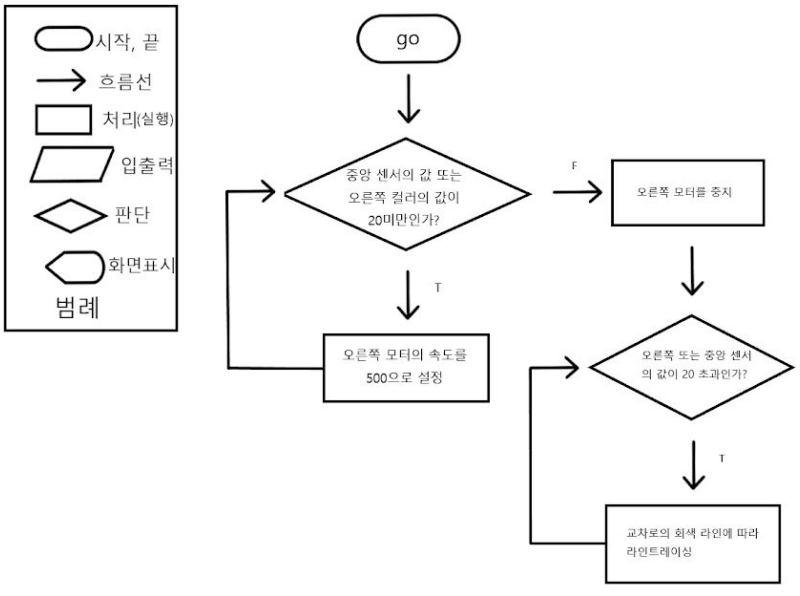

범례
시작, 끝
흐름선
처리(실행)
입출력
판단
화면표시
f_route
way배열의 앞쪽에 있는 정보를 빼와서 sp에 저장
way배열이 비어 있지 않으면
F
0의 값을 반환하며 종료
T
way배열의 앞쪽에 있는 정보를 빼와서 np에 저장
np-sp가 -3이라면
F
np-sp가 -1이라면
F
np-sp가 1이라면
F
T
T
2
T
n_di에 3의 값을 저장
n_di에 0의 값을 저장
n_di에 3의 값을 저장
n_di에 3의 값을 저장
go_n에 n_di-r_di +2의 값을 저장
go_n의 값이 4보다 크다면
T
go_n-=4
F
go_n의 값만큼 go함수 반복 실행
turn에서 반환된 값이 있다면
T
sp의 값을 이용하여 좌표 변환
F
n_di의 값을 r_di에 저장
sr배열의 값을 모두 제거
sr배열에 현재 좌표를 저장
np의 값을 sp에 저장
att배열에 현재 좌표와 현재 좌표에서 마주보는 지점의 좌표값을 0으로 저장
1의 값을 반환하며 종료

f_route


시작, 끝
흐름선
처리(실행)
입출력
판단
화면표시
범례


start
라인트레이싱 하기
오른쪽 컬러센서의 값이 20보다 크고
가운데 컬러센서의 값이 20보다 클 동안
반복
200만큼의 속도로
오른쪽 컬러센서의 값이
20보다 작거나 같을때
또는 가운데 컬러센서의 값이
20보다 작거나 같을 동안 주행
position 값 15만큼
주행
오른쪽으로 포인트턴 하기

start


시작, 끝
흐름선
처리(실행)
입출력
판단
화면표시
범례
go
중앙 센서의 값 또는
오른쪽 컬러의 값이
20미만인가?
F
오른쪽 모터를 중지
T
오른쪽 모터의 속도를
500으로 설정
오른쪽 또는 중앙 센서
의 값이 20 초과인가?
T
교차로의 회색 라인에 따라
라인트레이싱

go

3.4. 기술적 차별성

1단계 미션 시 색상정보(교통정보)를 읽을 때 바퀴의 포지션값을 이용하여 어느정도의 오차범위 내에서도 정상적으로 색상정보를 읽고 출력할 수 있다. 따라서 1단계 색상정보를 읽을 때 보다 빠른 주행속도를 이용하여 시간 단축을 할 수 있다는 것이다.

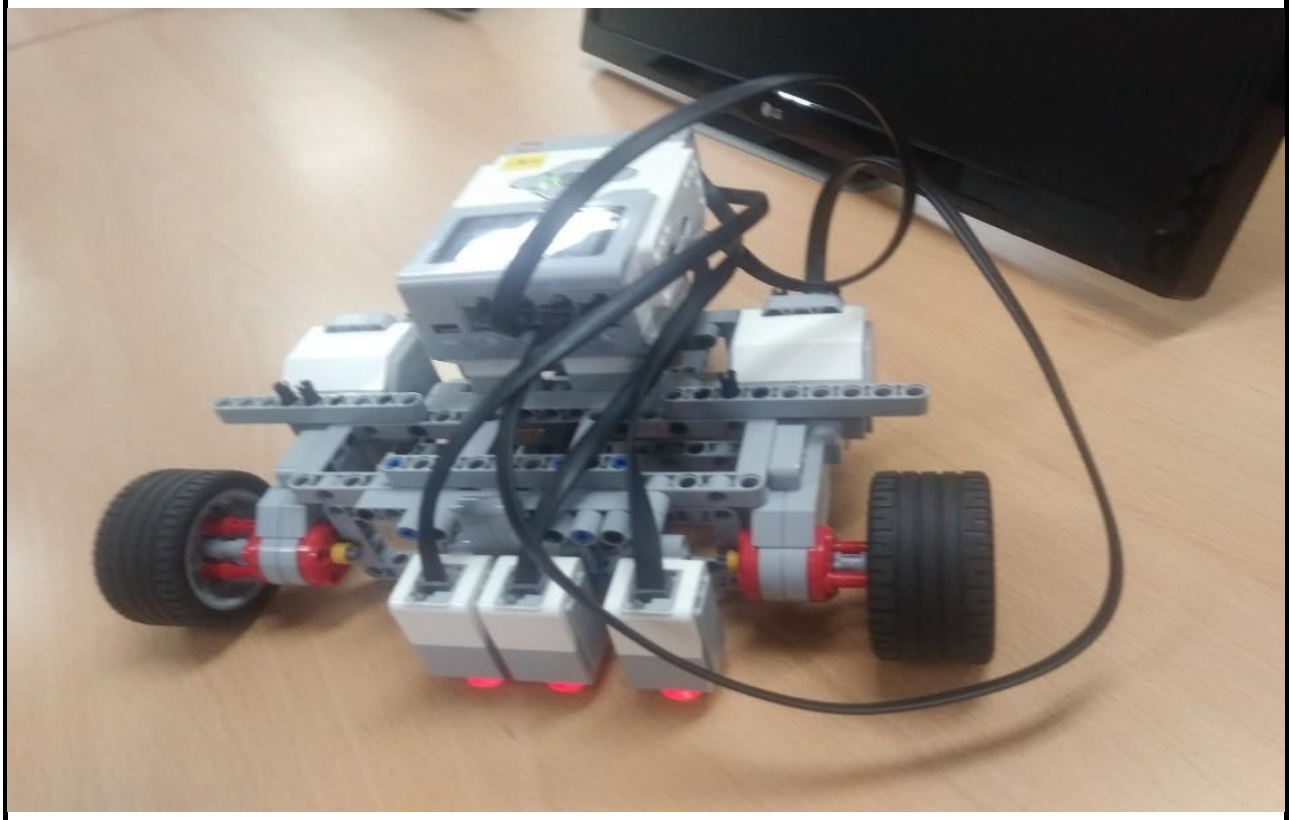
4. 개발 중 장애요인과 해결방안

문제사항 1 : 로봇이 교차로를 라인트레이싱하면서 회전할 때 회전이 불안정했던 문제

문제사항 2 : 로봇이 돌발상황을 만나고 최단경로를 찾지 못하는 문제

문제사항 3 : 로봇이 미션수행은 성공적이나 미션 완료시간이 오래 걸리는 문제

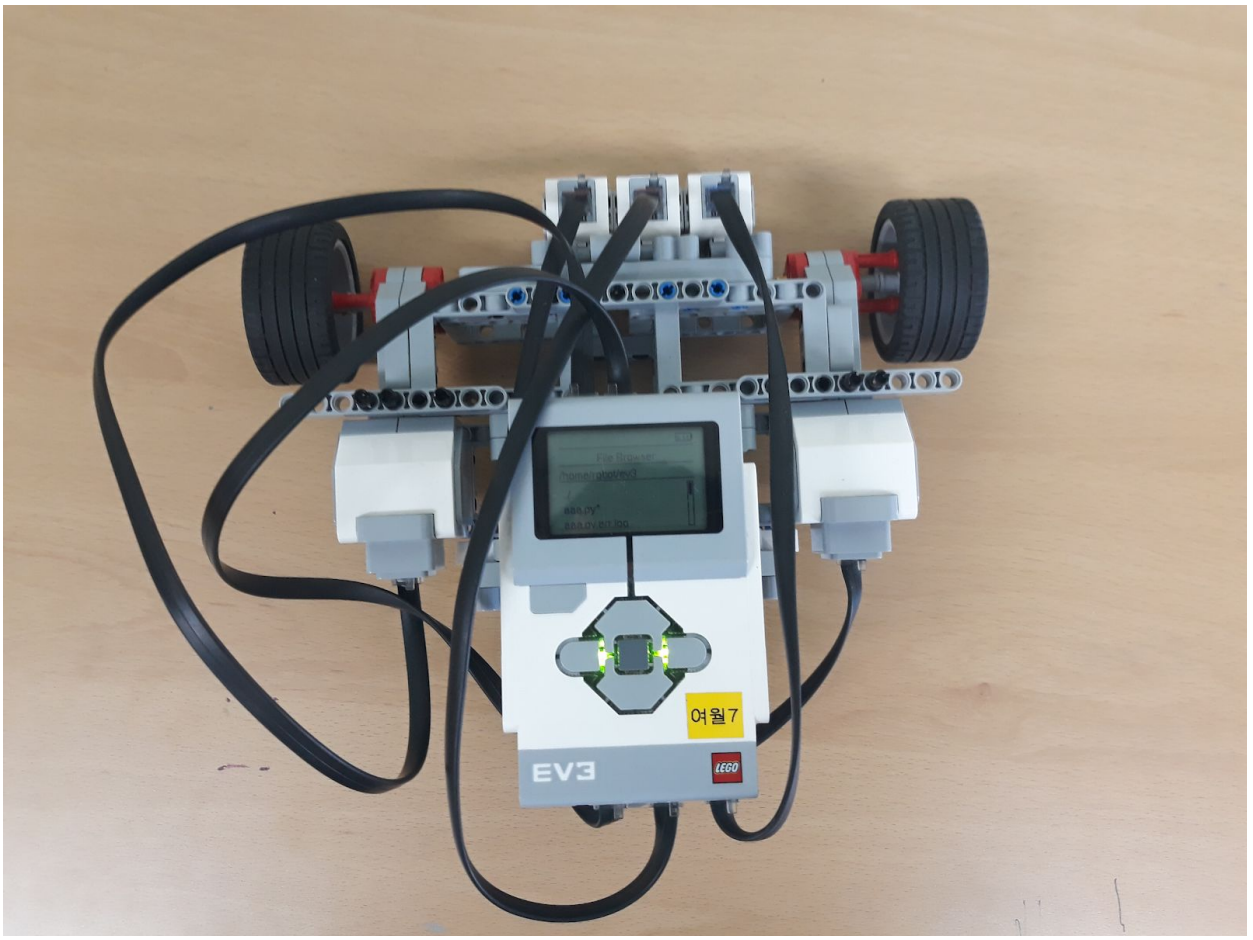
문제사항 1 해결방안 : 로봇이 교차로를 라인트레이싱 하면서 회전할 때 불안정한 것은 생각외로 고치기 간단했다. 바로 로봇의 바퀴 간격을 늘려 교차로를 회전할 때 회전의 안정성을 보안했다.



위의 사진이 문제상황 1을 해결하기 만든 모델이다.(바퀴와 바퀴 사이의 간격이 23cm로 로봇이 센서를 활용하여 주행할 때 안정성을 유지한 채 주행할 수 있게된다.)

문제사항 2 해결방안 : 기존의 알고리즘 교차로의 일정한 부분을 기준으로 상하좌우 좌표를 분석해 최단경로를 탐색하는 방식이었는데 이 방식은 돌발상황을 만난 뒤에는 기준점이 달라져 사용을 할 수 없게 된다는 문제점이 발생했다. 따라서 알고리즘을 변화시킬 필요성을 느꼈고 도착지점부터 역으로 걸리는 시간에 따라 되돌아가는 식으로 최단경로를 찾는 식으로 변화시켰다. 그리고 돌발상황을 만났을 때 돌발상황이 걸린 부분을 배열에 저장하여 최단경로를 탐색할 때 해당경로를 탐색하지 않고 다른 경로를 찾게 만들었다.

문제상황 3 해결방안 : 이제 알고리즘적으로 문제사항은 없으나 미션 완료까지 2분 10초정도 소비되었는데 이는 꽤나 느린 해결속도라 생각이 들었다. 그래서 기존의 모델에서 1단계 이후 사용되지 않던 센서를 활용해 3개의 센서로 라인트레이싱을 하기로 하였다.



위의 사진이 주행속도 증가를 위해 수정한 모델이다. 왼쪽 부터 오른쪽 순으로 1,2,3번째 센서라고 할 때 1번과 2번 센서는 교차로와 교차로 사이를 주행할때 사용하고 2,3번째 센서는 교차로에서 회전 할 때 사용된다. 이렇게 2,3번 센서를 활용해 주행을 하면 왼쪽 바퀴가 거의 교차로의 중심에 있게 되어 주행을 매우 안정적으로 할 수 있게 된다. 그런데, 1단계 주행 시에는 2,3번째 센서를 활용하여 주행한다. 일반 주행에서 2,3번째 센서를 활용하면 바퀴의 중심과 맞지 않아서 불안정할 것으로 생각하였는데, 생각보다 주행이 잘 되어 사용하게 되었다. 이렇게 안정적인 주행이 보장된 상태에서 기존의 주행 속도를 2~3배 정도 증폭시켜 문제 해결 속도를 동일 조건으로 2분 10초에서 58초로 대폭 단축되었다.

5. 개발결과물의 차별성

우리의 개발결과물이 다른 작품과 다른점은 1단계 미션 해결 속도가 비교적 빠르다는 점이다. 또한 3단계에서 교차로 회전 시 왼쪽 바퀴가 교차로의 중심점에 근접하여 빠른 속도에도 안정성을 추구할 수 있다. 그리고 센서를 활용한 라인트레이싱을 할 때 직진과 교차로 회전을 할 경우의 사용 센서를 다르게 하여 보다 안정적인 회전과 직진을 확보할 수 있었다.

6. 단계별 개발계획 및 실제 참여인원 및 업무 분장

하드웨어 부분 : 해커톤 당시 우리가 사용하던 로봇은 교차로를 회전할 때 매우 불안하며 직진의 안정성도 확식하지 않다는 느낌을 받았다.그래서 우리는 로봇이 교차로를 주행할 때 불안정하게 하는 요인에 대해 생각해보았고 최종적으로 바퀴간 간격이 너무 좁아서 생기는 문제점이라 판단했다. 그래서 우리는 해커톤 이후 로봇의 바퀴간격을 최대치로 수정하기로 계획했다.

소프트웨어 부분 : 우리가 해커톤에 참여할 때, 우리는 돌발상황을 만나서 새로운 최단경로를 찾아야 하는 미션을 해결하지 못했다. 또한 처음 로봇이 최단경로를 찾을 때 특별한 경우에는 경로를 찾지 못한다는 문제점이 발견되어 이를 해결하면서 이 최단경로 탐색 함수를 돌발상황에서 활용할 수 있게 만들 계획이다.

안재원 : 전반적인 소프트웨어 제작 담당이었으며, 팀원들이 구상한 알고리즘들을 코드로 나타내는 역할을 해주었다.

남궁찬 : 전반적인 하드웨어 제작 담당이었으며, 팀원들이 문제상황에 대해 서로 논의하고 토의해 볼 수 있도록 해주는 역할이었다

김준민 : 하드웨어 제작 보조이었으며, 문제상황을 해결할만할 알고리즘의 뼈대를 구상하는 역할이며,보고서 작성 단계에서 함수 흐름도를 작성하는 역할이었다.

오준혁 : 소프트웨어 제작 보조이었으며, 미션 해결 시연 동영상을 촬영하고 개발일지를 일부 작성하였다.

김성찬 : 주로 개발일지 작성 담당이었으며, 문제를 해결하기 위한 방법을 제시하는 역할이었다.