제17회 임베디드SW경진대회
개발완료보고서[주니어 임베디드SW챌린저]

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

1. 팀 정보

팀명	GGL	팀장	조민성
팀원	김예건, 김태정, 이한결	팀원	

2. 개발완료보고서

1. 개요

1.1. 작품 개요

제안하는 작품에 대한 개요를 자세히 기술한다.

1.2. 개발 목표

개발 목표를 명확하게 제시한다.

2. 개발 환경 설명 (최대한 자세하게 기술)

2.1. Software 구성

2.2. Software 설계도(흐름도 및 클래스 다이어그램 등 (개발언어에 따라 선택))

2.3. Software 기능 (알고리즘 설명 포함)

2.4. 프로그램 사용법 (Interface)

2.5. 개발환경 (언어, Tool, 사용시스템 등)

3. 개발 프로그램 설명 (최대한 자세하게 기술)

3.1. 파일 구성

3.2. 함수별 기능

3.3. 주요 함수의 흐름도

3.4. 기술적 차별성

4. 개발 중 장애요인과 해결방안

개발 과정에서 나타났던 모든 장애 요인(Risk)들을 나열하고 이러한 장애요인들이 발생했던 경우 어떻게 해결했는지 구체적으로 제시한다.

5. 개발결과물의 차별성

개발한 결과물에 대해 타 팀과의 차별성 혹은 우수성을 서술한다.

6. 단계별 개발계획 및 실제 참여인원 및 업무 분장

참여 인원의 역할을 구체적으로 제시하고 개발 일정 계획상에서 각 참여 인력이 어떤 역할을 수행했는지, 어떤 부분을 협업하였고, 어떤 개발 절차로 개발은 진행했는지에 대해 자세히 명시한다.

1. 개요

1.1. 작품 개요

직접 개발한 소프트웨어가 탑재된 로봇을 이용하여 Mobility 서비스를 제공하는 자율 주행 시스템을 개발하는 미션을 수행하는 작품이다.

1.2. 개발 목표

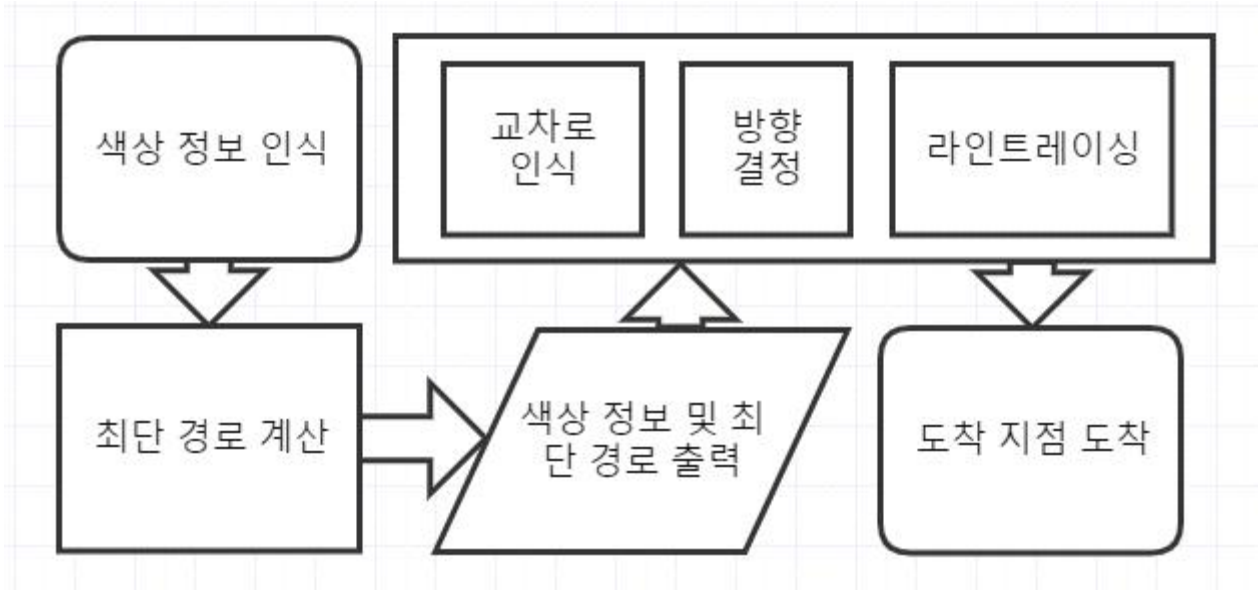
제작한 HW는 경기장에 표시된 각 교차로의 색상 정보를 통해 교통 상황 정보(원활, 서행, 정체)를 파악하고, 교통 상황을 반영한 최적의 경로를 따라 최단 시간 내에 도착 구역에 도착한다.

2. 개발 환경 설명

2.1. Software 구성

로봇C 프로그램을 사용하여 센서값 인식, 정보 표시, 주행(전진, 회전, 라인트레이싱) 등을 구현할 수 있게 구성되어있다.

2.2. Software 설계도



2.3. Software 기능

인식한 색상 정보를 통해 각 교차로의 교통 상황 정보를 인식하고, 인식한 정보를 통해 각 정점(교차로의 검은 지점)과 그 간선에 대한 유향 가중치 그래프를 인접 행렬로 표현한다. 이 정보를 통해 다익스트라 알고리즘을 이용하여 최단 경로를 계산한 후, 길을 찾는 다양한 함수를 이용하여 라인 트레이싱 방법을 통해 도착지에 도착할 수 있으며, 주행 중 사고 지점을 발견하였을 때, 그 간선을 제외한 후 인접 행렬을 재구성하여 새로운 최단 경로를 찾아 주행하는 기능을 갖추고 있다.

이 소프트웨어에서 사용한, 다익스트라 알고리즘에 대한 설명은 다음과 같다. 다익스트라 알고리즘은 도로 교통망 같은 곳에서 나타날 수 있는 그래프에서 꼭짓점 간의 최단 경로를 찾는 알고리즘이다. 이 알고리즘은 컴퓨터 과학자 에츠허르 다익스트라가 1956년에 고안했다. 그래프에서 주어진 꼭짓점에 대

해서, 다익스트라 알고리즘은 그 노드와 다른 모든 꼭지점 간의 가장 짧은 경로를 찾는다. 이 알고리즘은 어떤 한 꼭짓점에서 다른 한 도착점까지 가는 경로를 찾을 때, 그 도착점까지 가는 가장 짧은 경로가 결정되면 멈추는 식으로 사용할 수 있다. 예컨대 어떤 그래프에서, 꼭짓점들이 각각 도시를 나타내고, 변들이 도시 사이를 연결하는 도로의 길이를 나타낸다면, 다익스트라 알고리즘을 통하여 두 도시 사이의 최단 경로를 찾을 수 있다.

2.4. 프로그램 사용법

EV3 로봇의 전원을 눌러 작동시키면, 탑재된 소프트웨어를 통해 교통 정보를 인식하고 최단 경로를 찾아 주행한다.

2.5. 개발환경

개발 컴퓨터의 경우 윈도우 환경에서 로봇 C 프로그램을 사용하였고, EV3 로봇의 경우 리눅스 기반의 로봇 C Firmware를 사용하였다.

3. 개발 프로그램 설명

3.1. 파일 구성

다음의 모든 기능을 작동시키는 함수들로 구성된 c파일이다.

3.2. 함수별 기능

InputColor()
: main함수 실행 시 가장 먼저 실행되는 함수로 1단계를 담당한다. 실행 시 전진하며 색깔 테이프의 색을 인식하며 인식한 색을 'colorChar' 배열 안에 순차적으로 넣는다.

MakeMap()
: InputColor()에서 입력받은 교차로의 색상정보를 바탕으로 정점 사이에 가중치를 할당하여 인접행렬을 완성한다.

Dijkstra()
: 다익스트라 알고리즘을 이용해서 MakeMap() 함수에서 만든 인접 행렬을 기준으로 도착점까지의 최단 경로를 계산한다.

MakePath()
: Dijkstra() 함수를 거치면서 만들어진 path 배열(경로를 역순으로 받을 수 있는 배열)을 통해 Reversewholepath 배열을 채우고, 이것을 다시 wholepath 배열에 넣어 시작지점부터 도착지점까지의 경로를 배열에 저장한다.

Print_Result()
: 위의 함수들을 거치면서 입력받은 색, 찾은 경로를 EV3 LCD에 출력한다.

FindBlack()
: 검은 지점을 인식하지 못한 동안 실행되는 코드로 라인 트레이싱을 담당한다. 좌우 센서값의 차가 5보다 크다면 우회전을 -5보다 작다면 좌회전을 그 사이에 있다면 직진을 하도록 라인 트레이싱을 설정했다. 만약 센서가 검은 지점을 인식한다면 함수를 벗어나고 Drive 함수가 실행된다.

Drive()

: 로봇이 경기장을 주행할 때 사용하는 함수로, 정점에 도착했을 때 직진할지 우회전할지를 결정하며 라인 트레이싱 코드가 들어있다. 라인 트레이싱 도중 사고 지점 (빨간색 테이프)를 만났을 시에 경로를 다시 계산하는 코드까지 들어있다.

ResetList()

: Drive() 함수 도중 사고 지점 (빨간색 테이프)를 만났을 때 사고 지점을 지나지 않는 새로운 경로를 찾아야 하는데, 이때 다익스트라 알고리즘을 다시 사용하기 위해서 앞서 사용했던 배열들을 초기화한다.

NewDrive()

: 사고 지점 (빨간색 테이프)를 만난 후 우회경로까지 계산한 뒤 Drive() 함수와 같이 도착지점까지 운행하는 역할을 한다. 역시 라인 트레이싱 코드가 들어있다.

goStraight()

: 검은색 지점을 만났을 때 해야 하는 동작이 직진(방향 전환 X)일 때 계속해서 라인 트레이싱을 하게 하는 함수.

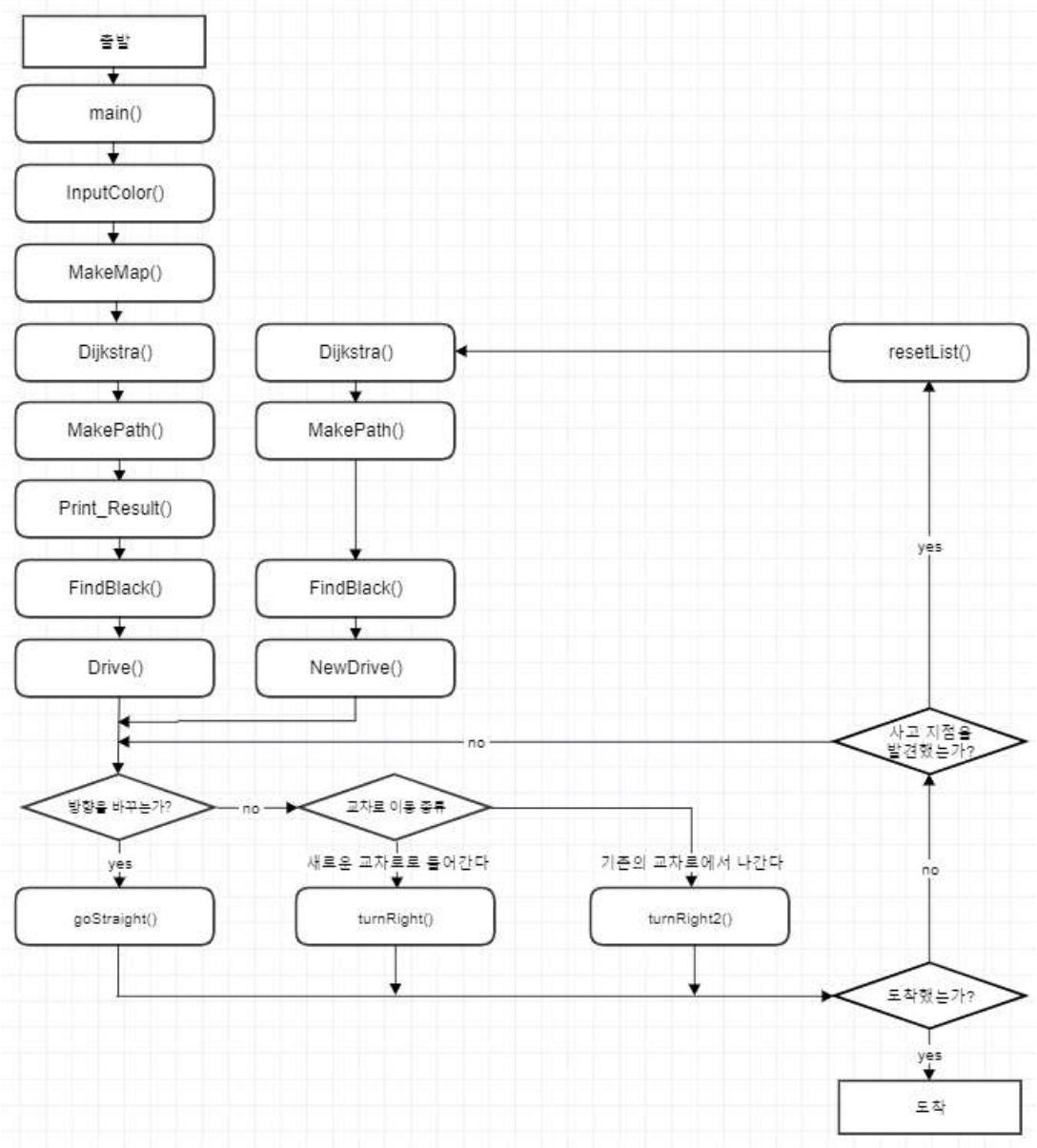
turnright()

: 검은색 지점을 만났을 때 해야 하는 동작이 우회전하여 원래의 교차로를 벗어나는 동작일 때 일정 각도만큼 회전하여 포인트 턴 하는 함수.

turnright2()

: 검은색 지점을 만났을 때 해야 하는 동작이 우회전하여 새로운 교차로로 들어가는 동작일 때 일정 각도만큼 회전하여 포인트 턴 하는 함수.

3.3. 주요 함수의 흐름도



3.4. 기술적 차별성

최단 경로를 추적하는 알고리즘을 만들 때 다익스트라 알고리즘을 사용하였다. 다익스트라 알고리즘은 첫 정점으로부터 다른 정점들까지의 최단경로를 구하는 알고리즘인데, 다른 알고리즘보다 시간복잡도가 작고 실행시간이 짧기 때문에, 운행 초반이나 사고를 만나 다시 운행을 시작할 때, 최단 경로를 구하는 데에 있어서 로봇을 개발하는 동안 문제점이 생기지 않았다.

4. 개발 중 장애요인과 해결방안

첫 번째로 주행 1단계에서 색상 정보를 인식할 때부터 어려움을 겪었다. 색을 인식하기 위해서는 정확한 간격을 이동하며 컬러센서로 색상 정보를 읽어야 하는데, 그 정확한 간격을 맞추기 위해 설정해야 하는 속도와 시간이 로봇의 충전상태, 로봇의 바퀴나 바닥 상태에 따라 바뀌기 때문이다. 이러한 문제점을 해결하기 위해서 로봇을 항시 충전 상태로 유지하였고, 일정한 거리를 이동한 후 정지하여 색을 읽고 다시 이동하는 방식으로 색상 정보를 인식하도록 수정하여 문제를 해결하였다.

두 번째로 라인 트레이싱 중 검은 정점을 인식하는데 오류가 발생하였다. 오류가 발생한 대부분의 이유는 한 센서로 검은 정점을 인식 후, 다음 행동을 진행할 때 여전히 그 센서가 검은 정점의 일부 위에 위치하였기 때문에 오류가 발생하였다. 따라서 검은 정점 인식 후 완전히 그 위치를 벗어나도록 수정하여 문제를 해결하였다.

세 번째로 라인트레이싱 과정에서 센서에서 반사값을 받아들이는 데 오류가 생겨 라인을 이탈하는 문제가 발생하였다. 맵의 배경이 단색으로 이루어져있지 않고 다양한 색의 그림으로 이루어져 있기 때문이다. 이는 수많은 경험과 측정 시도를 통해 라인트레이싱에 사용되는 정확한 반사값의 기준을 설정하여 해결할 수 있었다.

네 번째로 회전 시 문제를 겪었다. 개발 초기에는 교차로의 정점을 만나 방향을 꺾을 때 스윙 턴을 이용하였는데, 라인에 제대로 안착하도록 하는 데 어려움이 있었다. 따라서 본선 대회 이후 회전방법을 포인트 턴을 이용하여 회전하도록 코드를 수정하였다. 검은 정점을 인식한 후, 로봇을 앞으로 전진시켜 두 바퀴 사이에 정점이 오게 한 후, 그곳을 중심으로 포인트 턴을 하여 회전한다.

5. 개발결과물의 차별성

우리 팀은 개발 결과물의 차별성을 길을 따라가는 라인 트레이싱과 코너링, 방향 전환에 중점적으로 두었다. 우선 라인 트레이싱을 할 때 센서 2개를 이용하였는데, 두 센서 사이의 간격을 도로 폭과 거의 비슷하게 하여 로봇이 도로 중앙에 있을 때 센서 2개가 도로 양끝에 걸치도록 위치시켰다. 이때 두 센서값이 같기 때문에 그 차이는 0에 가깝다. 하지만 로봇이 도로에서 이탈하면 두 센서값에 차이가 발생하고, 우리는 로봇이 두 센서값의 차를 0으로 유지하는 방향으로 움직이도록 코드를 설계하였다. 또한, 교차로 안에서는 반시계방향으로 곡선 도로를 따라가야 한다는 점을 고려하여 모터 속도를 조절해 항상 살짝 왼쪽으로 회전하도록 하였다.

방향 전환의 방식은 길을 따라가다가 방향 전환이 필요하면 조금 직진 후 오른쪽으로 포인트 턴 하여 다시 라인 트레이싱을 하는 것이다. 하지만 교차로를 들어갈 때 필요한 회전과 교차로에서 나갈 때 필요한 회전의 각도가 조금 다르고, 회전 각도가 다른 두 종류의 방향 전환 함수를 만들어 각각 맞는 상황에 사용하도록 설계하였다. 이처럼 라인 트레이싱과 방향 전환에 집중했기 때문에 로봇의 부드러운 운행이 가능하였다.

6. 단계별 개발계획 및 실제 참여인원 및 업무 분장

1단계
1) 색상 정보 인식 방법
- 인식 속도 - 김예건
- 인식 센서 - 김예건
2) 색상 정보 저장 방법
- 저장 방법 - 김예건

2단계
1) 인식 후 제자리로 돌아오기
- 인식 후 제자리로 돌아오는 방법 - 김태정
2) 색상정보 출력하기
- 색상정보 출력 방법 - 김태정
3) 최단경로 출력하기
- 최단경로 계산 알고리즘 - 이한결, 조민성
- 최단경로 출력 방법 - 김태정

3단계

1) 주행 (라인트레이싱)

- 센서 - 김태정
    1. 위치
    2. 개수
- 도로 정보 인식 - 이한결
- 라인 트레이싱 - 이한결, 조민성
- 정점 인식하기 - 이한결
- 회전하기 - 조민성
- 직진하기 - 조민성

2) 사고 발생

- 사고 발생 지점 인식하기 - 조민성
- 사고 발생시 대처 조민성 - 이한결