Embedded System Design- Team Project

10조 김현수, 최진명

1. Explanation of your path-finding algorithms

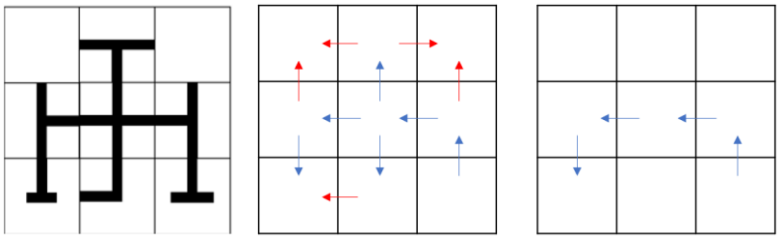
처음 마주치는 교차로의 좌표를 (25, 25)로 해서, 이동중에 마주치는 교차로를 모두 Map 2차원 배열에 기록했습니다. 각 Map은 해당 교차로에서 회전을 몇 번 했는지, 좌회전/우회전/직진시 도착하는 Map의 주소와 해당 Map을 탐색했는지, 부모 Map은 무엇인지 정보가 적혀 있습니다. 로봇의 좌표를 적절히 계산하고 totalMap에 기록해 같은 교차로가 여러 번 그래프에 그려지는 것을 막고, 로봇에 사이클에 빠지지 않도록 했습니다.

로봇이 이동하며 교차로를 마주치면 좌회전/우회전/직진이 존재하는 지 확인 후 map에 기록합니다. 그 이후 직진/좌회전/우회전 우선순위로 로봇을 회전시킵니다. 로봇의 회전에 시간이 많이 소요되기 때문에 직진을 우선시하고, 보통 시작선이 맵의 오른쪽에 위치하기 때문에 우선순위를 그렇게 정했습니다. 이 알고리즘은 Detect_And_Turn_Intersection에 구현되어 있습니다.

길을 가다 빈 칸이 감지되면 map의 parent에 적힌 이전 교차로로 되돌아갑니다. 교차로에 도착한 이후에는 아까 기록했던 leftTurnCount, rightTurnCount에 맞게 반대로 회전해, 해당 교차로에 처음 들어왔을 때와 똑같이 로봇을 정렬시킵니다. 도착한 교차로에서 더 갈 길이 남아있으면 그곳으로 이동하고, 없으면 다시 이전 교차로로 돌아가는 것을 반복합니다. 이 알고리즘은 Find_Next_Load에 구현되어 있습니다.

맵 탐색이 완료되고 나면 로봇의 LED가 점멸하고, 다시 시작 라인에 올려놓으면 탐색 단계에서 기록한 Map 그래프를 Backtracking 하며 스택에 넣고, 이를 하나씩 Pop 하며 길을 찾습니다. 이 알고리즘은 Optimal_Go에 구현되어 있습니다.

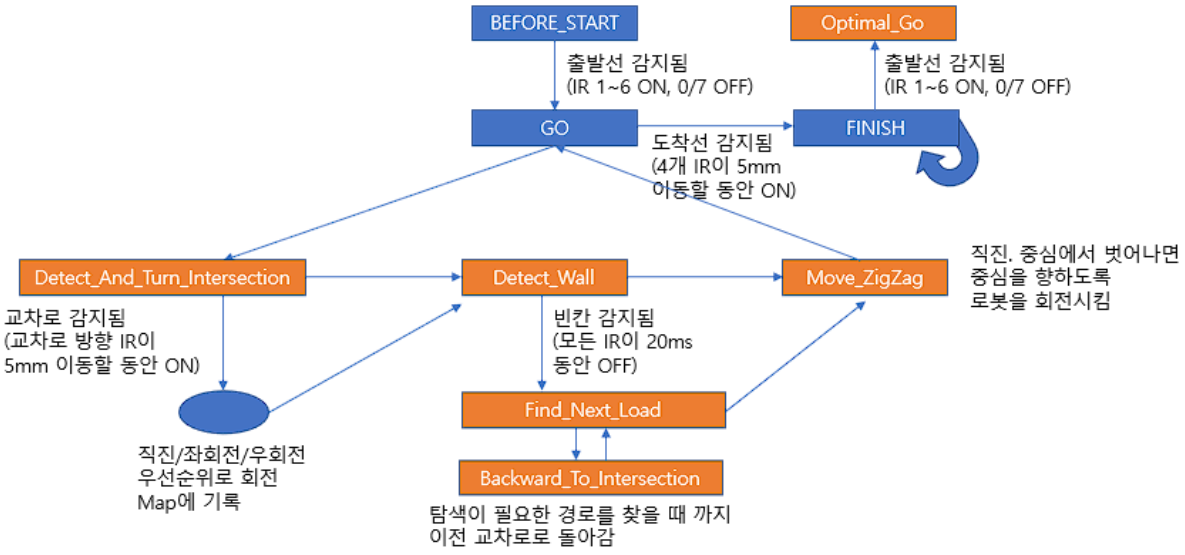
과제 예시로 주어졌던 3x3 맵의 경우 아래 그림과 같이 길 찾기가 이뤄집니다. 탐색 단계에서 두 번째 이미지와 같이 갈 수 있는 길과 갈 수 없는 길을 찾습니다. 이후 여기서 Backtracking을 하여 길을 찾고, 세 번째 이미지와 같이 찾아진 경로로 이동합니다.



2. Explanation of your motor control algorithms

기존 실습 코드에 Tick과 Tachometer를 추가하여 모터를 더 정밀하게 제어했습니다. TimerA2 인터럽트가 1ms마다 발생하도록 설정하고, TA2 Handler에서 tick 변수를 1씩 증가시키게 하였습니다. 이를 통해 로봇의 작동 시간을 1ms 단위로 측정할 수 있었습니다. 또한 모터의 회전수를 알 수 있는 Tachometer를 사용하여, 로봇을 좀 더 정밀하게 이동시키고 정확하게 90도 회전이 가능하게 하였습니다. 이렇게 1ms 단위로 모터를 제어하기 때문에, 모터를 출발시킬 때와 멈출 때 모터의 속도가 빠르면 오차가 생기기 쉽습니다. 따라서 부드럽게 출발하고 멈출 수 있도록 Move_Smooth, Start_Smooth 함수를 구현하였습니다. 두 함수는 목표 이동 거리 dist (0.0001cm단위)를 받습니다. Move_Smooth는 [0, dist/2] 범위에서 [minSpeed, maxSpeed]까지 linear하게 모터 속도를 증가시키고, [dist/2, dist] 범위에서 [maxSpeed, BRAKE_SPEED]까지 linear하게 감소시킵니다. Start_Smooth 함수는 [0, dist] 범위에서 [minSpeed, maxSpeed]까지 linear하게 증가시킵니다. 위 변수 3개를 적절히 조절하여, (미끄러지지 않는 선에서 힘차게 출발 -> 빠르게 이동 -> 1ms 단위 제어 및 관성으로 인한 오차 최소화)가 가능하게 했습니다.

3. Explanation of the entire logic of the program



4. Peer review

맵 탐색 알고리즘, 모터 제어, 교차로 감지 등 모두 동일한 기여도로 작업했습니다.