Linux ROS 환경에서 SLAM 과 NAVIGATION를 기초로 한 재난 구조 로봇 시스템의 구현

(Implemention of disaster rescue robot system based on SLAM and NAVIGATION in Linux ROS)

신 혜 영, 박 경 하, 유 동 하
한양대학교 로봇공학과
(Hye-Young Shin, Kyoung-Ha Park, Dong-Ha Yoo)
(Hanyang Univ. Robot Engineering)

Abstract : 재난의 피해가 갈수록 증가하고 있는 이유로, 국내와 국외에서 탐색 로봇, 감시용 로봇, 구조 로봇 등 여러 목적으로 재난대응 로봇이 개발되고 있다. 미국, 일본, 유럽 등의 선진국을 중심으로 각종 재난대응 로봇이 개발되고 있으나, 로봇과 통신이 어려운 상황에서 이용은 어려운 상태이다. 이에 본 논문에서는 Linux ROS 환경에서 SLAM과 navigation을 재난구조 로봇에 적용해 사람의 도움 없이 재난환경의 map을 만들어 통신의 여부와 관련 없이 임무를 수행할 수 있는 로봇을 제안한다. 본 논문에서 제시한 로봇은 기본적인 map과 장애물이 있는 map에서 주행 성공 여부와 주행 시간을 측정하여 로봇의 성능을 평가하였다.

Keywords : SLAM, navigation, mapping, ROS, mobile robot

Ⅰ. 서론

1.1 연구배경

자연재해(지진, 황사 등)와 사회재난(화재, 붕괴, 화생방사고, 전염병 등)을 포함하고 있는 재난은 그 피해가 갈수록 증가하고 있다. 이에 대응하기 위해 재난대응 로봇이 개발되고 있는데, 재난대응 로봇이란 자연재해, 사회재난 등 각종 재난 상황에서 재난 확산을 방지하고 피해를 최소화 및 사고 처리를 위한 로봇시스템을 말한다. 가장 최근인 2015년 네팔 지진, 2011년 후쿠시마 원전사고 등의 예로 재난대응 로봇의 필요성이 대두되고 있고, 2016년 경주 지진으로 보아 우리나라도 재난에 안전하지 않다는 점을 알 수 있다. 표 1에서 보는 바와 같이 화재의 경우에서도 2014년 42,135건, 2015년 44,435건, 2016년 43,413건으로 발생 횟수가 증가하는 것을 볼 수 있어 로봇의 필요성을 알 수 있다.

표 1 전국 연도별 화재 발생 현황[1]

시도별	2014	2015	2016
전국	42,135	44,435	43,413
서울특별시	5,815	5,921	6,443
부산광역시	2,026	1,973	2,199
대구광역시	1,767	1,817	1,739
인천광역시	1,818	1,875	1,790
광주광역시	1,010	1,006	956
대전광역시	1,291	1,254	974
울산광역시	890	874	928
세종특별자치시	223	252	300
경기도	9,675	10,333	10,147
강원도	2,182	2,485	2,315
충청북도	1,316	1,373	1,379
충청남도	2,838	3,031	2,825
전라북도	1,652	1,962	1,983
전라남도	2,620	2,647	2,454
경상북도	2,803	3,068	2,651
경상남도	3,622	3,960	3,756
제주특별자치도	587	604	574

재난환경은 기존의 수동 장비를 갖춘 방재 요원의 투입만으로는 방재작업에 한계가 있으며, 화재 상황에서 유해물질 노출로 인한 호흡기 질환, 붕괴

지역에서 탐사 작업에 의한 2차 붕괴 등 방재 요원들의 안전을 심각하게 위협하는 요소들이 산재해 있다. 연평균 순직하는 소방관 수는 7명, 공상자 수는 300명에 달하며, 이뿐만 아니라 광주시 소방본부 소속 소방관들의 건강검진 결과, 24%가 PTSD, 우울증, 수면장애 등을 겪는 것으로 나타났다. 이에 재난대응 로봇은 인간이 접근하기 어려운(접근할 수 없는) 곳에서 훨씬 안전하고 효율적으로 작업을 할 수 있고, 방재 요원들의 2차 피해를 줄일 수 있다는 점에서 필요성이 기대된다.

재난 상황에서 로봇을 활용하는 것은 정형화되어있지 않은 상황에서 반복적이지 않은 작업을 로봇이 정교하게 수행해야 하므로 시설물 등의 부정형 장애물들을 인식하고 장애물 회피 및 통과 기능이 갖춰진 고기능의 로봇이 요구된다.

현재 국내에서는 계단 승/하강, 회전이 가능하고 열 영상 장비 및 카메라와 LRF (Laser Range Finder), 유선/무선 통신 시스템이 탑재된 소형 정찰 로봇[그림. 1-(a)], 한국원자력연구원에서 개발한 인명 탐색용 소형 로봇과 한국생산기술원에서 개발한 재난 감시용 로봇인 Hovering 로봇[그림. 1-(b)] 등이 개발되고 있다.[2]

(a) 소형 정찰 로봇 (b) Hovering 로봇

그림. 1 재난환경 대응 로봇 예시

1.2 연구 목표 및 내용

본 논문에서는 ROS 환경에서 구현된 SLAM (Simultaneous Localization and Mapping)으로 재난현장의 지도를 얻은 뒤 내비게이션 스택을 활용한 재난대응 로봇의 성능향상을 2가지 기술을 중심으로 기술한다. [그림. 2]는 로봇이 인식한 환경에 따라 행동 양식을 결정해주는 핵심 알고리즘이다. 로봇이 인식하는 환경 중 핵심적인 요소는 SLAM 알고리즘을 통해 얻은 MAP data를 이야기할 수 있는데, (1)*과 같이 정제하여 사용하려 한다.

$$(MAP data) => a occupied grid of the map/current\ location --- data_2 \quad (1)$$

인식된 map의 모든 공간을 [그림. 3]과 같이 일정한 간격으로 나누어 node data로 정제한 후 하나의 그래프를 형성한다. 이 데이터로는 로봇의 섬세한 path trajectory를 결정하는 것이 역부족이나 그래프 탐색 알고리즘에서는 탁월한 처리속도를 보장해 줄 것이다. 또한, 그 속도는 map의 resolution에 따라 크게 달라질 것으로 예상하므로 컴퓨터 성능과 환경적 요구사항에 맞게 결정하면 한다.

세부 알고리즘은 Frontier planning([그림 4])과 EXIT planning으로 나뉜다. Frontier planning은 목표지점의 방향조차 인식하지 못한 상태이다. 이 상황에서는 일반적인 그래프 탐색 알고리즘을 사용한다. 현재 위치로부터 방문하지 않은 node(unexplored nodes)를 우선으로 탐색하되 DFS (Depth First Search)방식을 사용한다. DFS 방식을 적용한 이유는 로봇이 직접 방문하기에 현실적이며 Frontier state라는 상황에 더욱 알맞기 때문이다. 우연히 Goal state를 마주친다면 EXIT planning 방식으로 알고리즘을 변경해 가능한 빠르게 재난 현장을 빠져나갈 수 있도록 한다. EXIT planning은 이름 그대로 goal state를 마주쳤을 때 재난환경에서 탈출할 수 있도록 최적의 경로로 움직이는 것이다. 이 상황에서는 입구에서 goal까지의 모든 map을 기억하고 있기 때문에 [그림 2]와 같은 최단경로 탐색 알고리즘을 이용해 경로를 얻어 탈출한다.

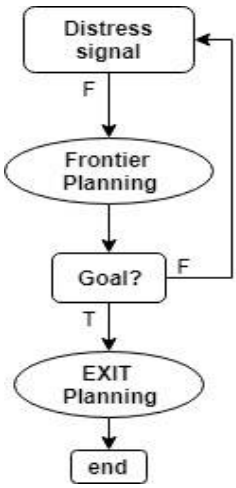


그림. 2 EXIT planning 알고리즘

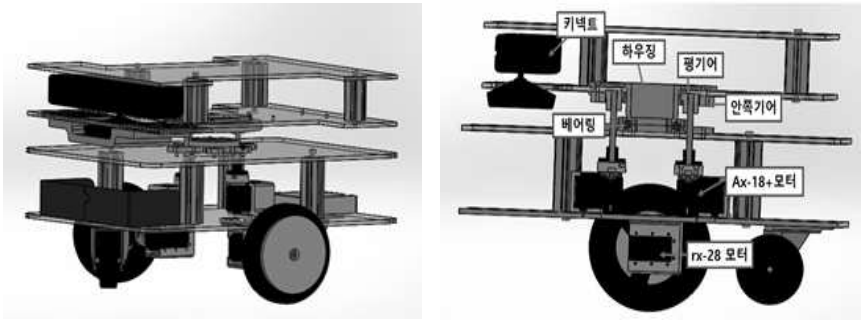


그림. 5 로봇의 전체적인 구성

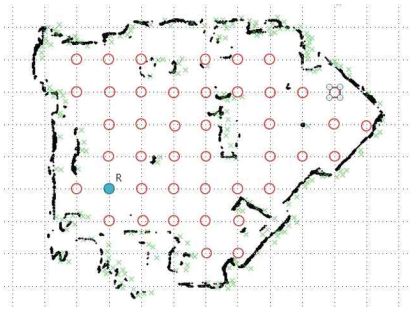


그림. 3 node data로 정제된 map

II. 본 론

- HARDWARE

첫 번째 본론에서는 본 논문에서 제안하는 재난 대응 로봇의 HARDWARE에 관해 이야기 할 것이다. MECHANICAL PART와 CONTROL PART로 나누어 설계하였으며, MECHANICAL PART[그림. 5]에서는 각각 상부와 하부로 나누어 설계하고 KINECT와 PC, 모터의 전력 배선을 설계하였으며, CONTROL PART에서는 dynamixel 모터 제어와 이를 응용하여 평 기어와 바퀴를 제어하였다.

1. MECHANICAL PART

1.1 상부구동

실험 결과 로봇이 회전할 때 센서가 빨리 돌아 map을 잘 그리지 못하는 문제가 있었다. 그래서 이동과 독립적으로 mapping을 하기 위해(환경을 보다 안정적으로 인식하기 위해) 상부를 하부와 독립적으로 회전시킬 필요가 있었다. 이를 구현하기 위해, 안쪽기어([그림. 6])와 평 기어 두개를 유성기어 형태로 배치 및 동력전달을 할 수 있도록 설계하였다. 이는 하나의 모터를 사용하였을 때 출력 토크가 부족할 것을 고려한 것이며, 동시에 백래쉬(back-lash)를 줄일 수 있는 효과도 있다.

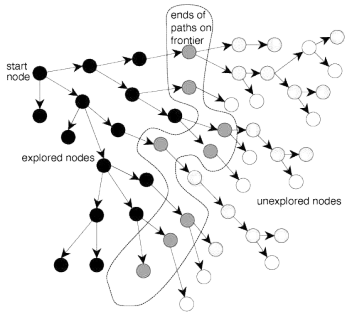


그림. 4 Frontier planning

상부와 하부의 움직임을 독립적으로 동작하기 위하여 베어링을 이용하였다. 또한 베어링을 고정하는 하우징은 3d 프린팅을 이용하여 제작하였다. (안쪽기어와 하우징 사이를 지나는 큰 구멍은 KINECT와 PC 등의 전력배선을 위한 통로로 이용하기 위해 설계되었다.)

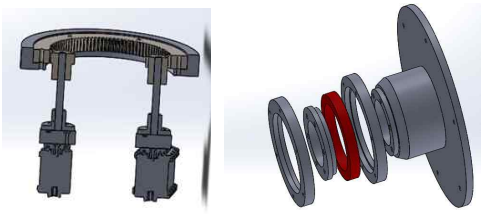


그림. 6 안쪽기어와 하우징 및 베어링

1.2 하부구동

본 구현은 모든 방향으로 이동할 수 있거나 제자리 회전이 쉬운 mobile robot에 적합하게 설계되었다. 경제적인 가격과 $3.7N \cdot m$의 적정한 최대 정지토크의 이유로 RX-28 모터를 선택하였다. 모터와 shaft 바퀴를 허브로 고정했다.[그림. 7]이 모터를 이용하여 이륜구동을 하도록 하고, 2개의 캐스터를 이용하여 로봇을 지지하였다.

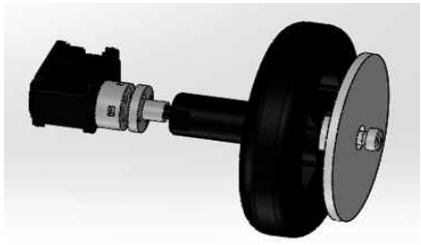


그림. 7 shaft and wheel

1.3 KINECT와 PC, 모터의 전력배선 설계

KINECT와 PC의 경우 각각 구동 전력이 12V 2A/ 5V 2A로 안정적인 전력공급이 필요하였다. 그 때문에 이에 적합한 연축전지 12V 4Ah를 이용하며 각각 레귤레이터를 통해 안정적으로 전력을 공급해줄 수 있도록 설계해 주었다. 동시에 스위치를 이용하여 충전 모드, OFF 모드, ON 모드를 실행할 수 있게 설계하였다.

이 로봇의 전력설계 특징은 모터부와 PC부의 전원이 분리되어 있다는 것인데, 이는 모터 출력이 순간적으로 커졌을 경우 PC나 KINECT에 전류가 충분히 공급되지 못할 것을 염려하여 설계해 준 것이다. [그림. 8]

모터를 구동하기 위해 18650 전지 4개를 직렬로 연결하여 14.8V의 정격전압을 인가해주었으며 SMPS2Dynamixel이라는 모듈을 통해 4개의 모터로 전원을 공급해 줄 수 있었다.[그림. 9] 또한

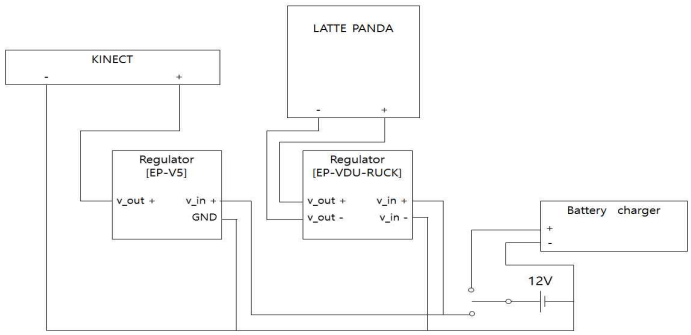


그림. 8 KINECT와 PC의 전력배선 및 설계

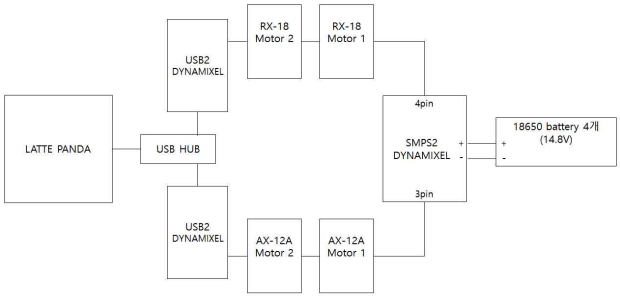


그림 9. 모터구동을 위한 전력 및 통신 설계

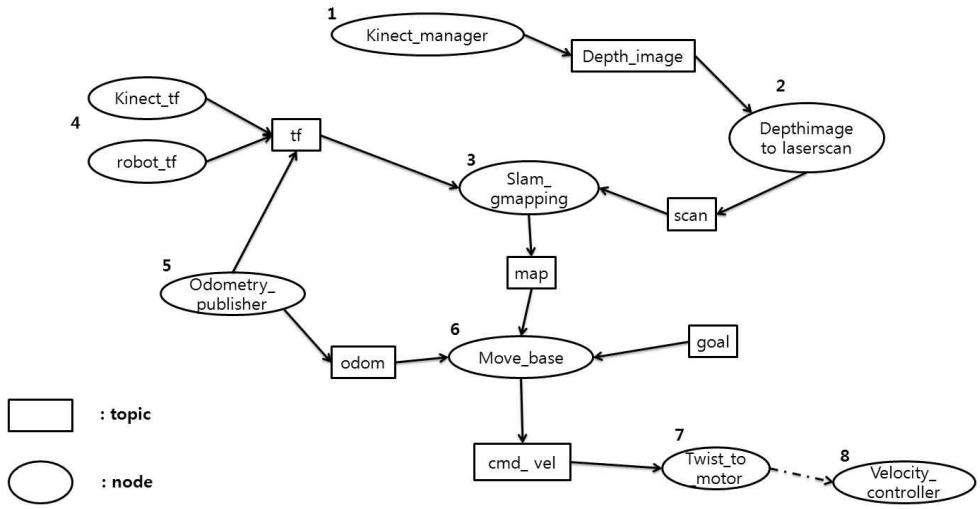


그림 10. 로봇 시스템 프로세스의 구조와 IPC 구조 요약 및 도식화

모터의 제어는 하드웨어 제어기 없이 PC를 이용해 직접 해주는 ROS의 dynamixel_sdk 패키지를 이용하였다.

2. CONTROL PART

dynamixel 모터 제어는 GitHub에 upload 되어 있는 dynamixel workbench를 이용하여 제어했다. dynamixel workbench 와 dynamixel sdk파일을 work space 내부에 src 파일에 넣고 dynamixel workbench 폴더에서 velocity control을 이용해 제어했다. 기본 제공 velocity control은 w, a, s, d, x 버튼을 이용하여 모터 2개를 컨트롤하는 코드이다. w는 앞, a는 왼쪽, s는 멈춤, d는 오른쪽, x는 뒤로 모터를 제어한다. (dynamixel_workbench_wheel.cpp 파일에 모터를 제어하는 example 코드가 있다.) 이를 응용하여 평 기어와 바퀴를 제어하는 코드를 작성하였다.

이때 평 기어와 바퀴 2개를 구동시키는 모터가 다르고 제어하는 코드도 다르기 때문에 launch 파일의 parameter 값과 구동 코드도 바뀌어야 한다. 이를 위해 launch 파일에서는 group을 이용해 2개의 node를 실행시키기로 했으며 각 group node 마다 model 파라미터 값을 넣어주어 velocity control에서 움직이는 motion이 다르게 하게끔 했다.

평 기어를 움직이는 모터는 평 기어가 약 60도 정도 돌아가게끔 만들었는데, 이 만큼 돌리기 위해서는 모터가 360도 이상을 돌아야하기 때문에 position 제어만으로는 불가능하다. 따라서 우리는 wheel 모드로 모터를 돌려 컨트롤을 했으며, 기어비와 모터의 스피드를 계산하여 제어했다. (2)*(3)*

$$\text{모터가회전한거리} = \text{모터의스피드} * \text{시간} \quad (2)$$

$$\text{평기어가돌아간거리} = \text{모터가회전한거리} * \text{기어비}(\frac{100}{19}) \quad (3)$$

이때 시간은 처음에 clock()함수를 이용하여 시간을 측정해 계산하려 했지만, 코드의 길이나 컴퓨터의 상황에 따라 시간이 달라진다는 것을 알게 되어 clock() 함수를 사용하지 않고 sleep() 함수를 이용해 시간을 주었다.

Ⅲ. 본 론(2)
- SOFTWARE

[그림. 10]은 ROS를 이용해 구현한 로봇 시스템을 요약하여 도식화한 것이다. 이 시스템은 여러 개의 프로세스 간 통신을 통해 더욱 빠른 반응속도와 모듈화를 통해 응용이 쉽게 만들어 주었다. 대부분의 node는 GitHub를 통해 공유되어있는 ROS 패키지를 이용하거나 개선하여 시스템을 구성한 것

이다. ROS에서는 각 프로세스를 node, 프로세스간 주고받는 데이터를 topic 및 service라 부르며 master node의 제어 아래 topic을 주고받는 구조로 더욱 쉽게 시스템을 구성할 수 있다. 본 논문에서는 [표 2]와 같은 node와 [표 3]과 같은 topic을 이용한다.

3. RViz

RViz는 ROS의 3차원 시각화 도구이다. ROS 메시지를 3차원으로 표시하는 것이 주요 목적으로 데이터를 시각화하여 확인할 수 있다. 예를 들어 레이저 레인지 파인더(LRF)의 센서로부터 장애물과의 거리, RealSense 및 KINECT나 Xtion와 같은 3차원 거리 센서의 포인트 클라우드 데이터(PCD, Point Cloud Data), 카메라로부터 취득한 이미지 값 등을 별도의 프로그래밍 없이 표시할 수 있다. 또한, 사용자가 지정한 폴리곤(polygon)을 이용하여 각종 표시를 지원하고 있고, 인터렉티브 마커(Interactive Markers)를 이용하면 사용자 node로부터 명령이나 데이터를 수신하여 상호작용하는 움직임을 나타낼 수 있다. 더불어, ROS에서는 로봇을 URDF (Unified Robot Description Format) 포맷으로 기술하는데 이를 2차원 모델로 표현하고, 각각의 모델은 자유도에 따라 이동하거나 구동할 수

표 3 topic 설명

1. tf	tf란 transform의 약자로 로봇의 센서 및 액추에이터의 좌표 혹은 로봇과 map 사이의 좌표를 기록해 놓은 자료구조로 각 좌표에서의 상대좌표만으로 절대 좌표를 직관적으로 계산할 수 있도록 도와준다.
2.scan	scan은 주로 distant sensor나 depth sensor를 이용해 추정된 거리를 담은 2d기반 자료구조이다.
3. odom	로봇의 이동경로인 Odometry를 담은 자료구조로 주로 로봇의 움직임을 통해 위치파악을 하는 것에 쓰인다.
4. goal	STATIC MAP의 경우 위치를 의미하며 이번 프로젝트에서는 KINECT의 소리를 통해 목표를 설정해준다. 하지만 현재는 직접 GUI툴을 이용해 설정해주고 있는 상황이다.
5. cmd_vel	move_basenode에서 나오는 Twist값을 담은 자료구조이다.
6. depth_image	KINECT의 depth sensor로부터 받은 센서 값 그 자체이다.

표 2 node 설명

1. KINECT_manager	KINECT를 오픈 소스로서 이용할 수 있게 해준 openni의 rosnode이다. 해당 node를 이용해 KINECT의 센서값을 받아올 수 있다.
2. depthimage_to_laserscan	KINECT로부터 받은 depthimage를 scan data형식으로 가공해 퍼블리싱 해주는 node이다.
3. SLAM_gmapping	로봇의 tf모델과 scan 데이터를 이용해 occupied Grid를 형성 한다. 또한 자체적으로도 map 데이터를 이용해 로봇의 현재 위치를 추정해준다.
4. KINECT_tf, robot_tf	KINECT와 mobile robot의 tf 모델을 퍼블리싱 해주는 node이다.
5. odometry_publisher	로봇의 이동경로인 odometry를 모터의 속도 등을 반영해 추정한 후 odom이라는 데이터로 publishing 해주는 node이다.
6. Move_base	map과 odom을 받아 node 내부에서 global path, local path를 형성하며 자체 planner를 이용해 실시간으로 Twist 데이터를 퍼블리싱 해준다.
7. Twist_to_모터	Twist 데이터를 로봇의 모터로 구현할 수 있도록 공식화 해주며 각 모터에 어떤 angular velocity를 할당할지 결정한다.
8. velocity_controller	ROBOTIS의 Dynamixel을 돌리기 위해 필요한 contoller node로 받은 angular velocity를 이용해 모터를 제어한다.

있어서 시뮬레이션이나 제어에 사용할 수 있다.[그림. 11] 또한, KINECT, LRF, RealSense 등의 다양한 센서로부터 데이터를 얻어 2차원으로 시각화할 수 있다.[3]

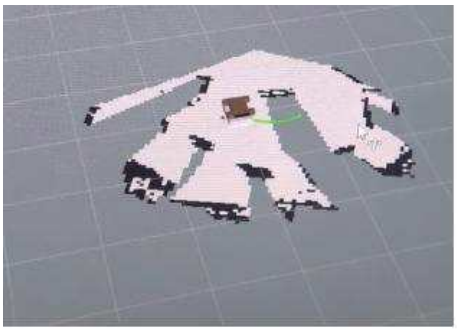


그림. 11 Simulation using RViz

4. ROS serial

Rosserial은 ROS의 메시지, topic 그리고 service들을 시리얼 통신 기반으로 사용할 수 있도록 변환해주는 패키지이다.[그림. 12] 일반적으로 마이크로컨트롤러는 ROS에서 기본 통신으로 사용하는 TCP/IP보다는 UART와 같은 시리얼 통신을 많이 사용한다. 따라서 마이크로컨트롤러와 ROS를 사용하는 컴퓨터 간의 메시지 통신을 위해서는 rosserial과 같은 중계자 역할이 필요하다.

ROS가 구동되는 PC는 rosserial server이고 PC와 연결되는 마이크로컨트롤러가 rosserial client가 된다. Server와 client는 rosserial protocol을 이용해 데이터를 송/수신하기 때문에 데이터를 송/수신할 수 있는 하드웨어는 모두 사용할 수 있다. 이를 통해서 마이크로컨트롤러에서 많이 사용하는 UART를 이용해서도 ROS 메시지나 topic들을 사용할 수 있게 된다.

예를 들어 마이크로컨트롤러에 연결된 센서 값을 ADC로 디지털화 시킨 후 시리얼로 전송하게 되면, 컴퓨터의 rosserial_server node는 시리얼값을 받아서 ROS에서 사용하는 topic으로 변환하여 전송하게 된다. 반대로 ROS의 다른 node에서 모터 제어 속도 값을 topic으로 전송하면 rosserial_server node는 이를 패킷으로 마이크로컨트롤러에 전송하고 연결된 모터를 직접 제어하게 되는 것이다.

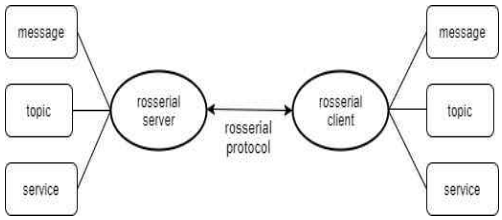


그림. 12 rosserial 기본 구조

Rosserial은 일반 컴퓨터의 경우, 제어에 있어서 실시간성 보장이 안되기 때문에 이렇게 보조 하드웨어 제어기로 마이크로컨트롤러를 사용하면 실시간성을 확보할 수 있게 된다.[3]

4.1 목표좌표 ('/move_base_simple/goal',geometry_msgs/PoseStamped)

목표좌표는 사용자가 직접 지정할 수 있다. 이는 태블릿과 같은 장비로 별도의 목표 좌표 명령 패키지를 작성하여 사용할 수도 있으나, 이 논문에서는 ROS의 rosserial을 통해 목표 좌표를 지정할 예정이다. 목표 좌표는 2차원상에 x,y,w로 구성되어 있다.[3]

4.2 ROS에서 rosserial 패키지의 이해

rosserial 패키지는 ROS에서 시리얼 포트 및 소켓과 같은 문자 장치로, 혹은 그 반대로 통신하기 위해 시행하는 표준화된 통신 프로토콜이다. rosserial 프로토콜은 표준 ROS 메시지 및 service 데이터 유형을 임베디드 장치와 동등한 데이터 유형으로 변환할 수 있다. 또한 ros 프로토콜은 디바이스 시리얼 데이터를 묶은 multitopic을 실행시킨다. 시리얼 데이터는 패킷에 header 및 tail bytes을 추가하여 데이터 패킷으로 전송된다.
패킷 표현은 [그림. 13]와 같이 표시된다. 패킷의 세부 기능은 [표 4]와 같다.

roscpp, rospy와 같은 ROS client 라이브러리는 다양한 기기에서 ROS node를 실행시킬 수 있다. Arduino에서 ROS node를 실행시킬 수 있는 ROS client의 port 중 하나를 rosserial_client 라이브러리라고 한다. rosserial_client 라이브러리를 이용하면 Arduino에서 ROS node를 실행시킬 수 있다. 다음은 rosserial_client 라이브러리 중 rosserial_arduino에 관한 설명이다.

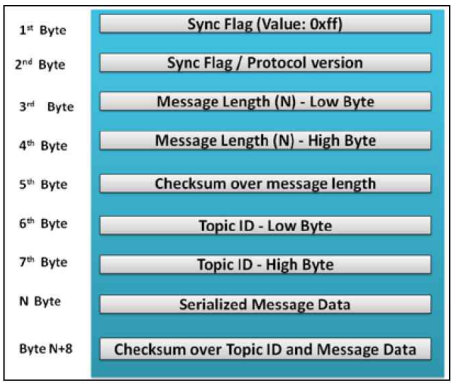


그림. 13 rosserial packet representation

rosserial_arduino : 이 rosserial_client는 Arduino UNO, Leonardo 그리고 Mega series에서 동작한다. 이러한 이유로 robotics project에서 쓰인다. 이 논문에서는 Arduino mega 2560을 사용한다.

latte panda에서 serial message와 rosserial_client 라이브러리의 topic들을 해석하기 위해 다른 패키지들이 필요하다. 다음의 패키지들이 serial data를 해석하는 데 도움이 된다.

rosserial_python : 기기로부터 온 시리얼 데이터를 다루기 위해 권장된 Python으로 작성된 PC 쪽의 node이다.

rosserial_server : PC에서 C++로 구현된 rosserial이다. 내재된 기능들은 rosserial_python보다 떨어지지만, 높은 성능의 application에 쓰일 수 있다.

rosserial_java : JAVA를 기본으로 구현된 rosserial이다. 그러나 아직 활발히 지원되지 않는다. 이것은 주로 안드로이드 장치와 통신하기 위해 사용된다.[8]

이 논문에서는 rosserial_python을 사용했다.

4.3 목표 및 경로 계획 보내기

SLAM을 이용하여 로봇의 현재 위치를 얻게 되면 move_base node로 목표 위치를 보낼 수 있다. move_base node는 이 목표 위치를 현재 로봇의 위치에서 목표 지점까지의 경로를 계획하는 global planner로 보낸다. 이 plan은 map server에서 주는 costmap을 기준으로 한다. global planner는 global plan의 각 부분을 실행하는 local planner로 경로를 보낼 것이다. local planner는 move_base에서의 주행거리와 센서값을 받고 로봇을 위한 충돌 없는 local plan을 찾는다. local planner는 로봇 주위의 장애물을 볼 수 있는 local costmap을 기준으로 한다.[8]

4.4 ROS node에서 Navigation stack으로 목표 지점 보내기

move_base node는 SimpleActionServer 이다. 만약 목표지점까지 도달하는 시간이 너무 오래 걸린다면 로봇에게 목표를 취소하도록 할 수 있다.

Simple_navigation_goals 패키지는 변수 x, y 및 θ 값을 보낼 수 있는 move_base node의 SimpleActionClient이다. simple_navigation_goals.cpp는 simple_navigation_goals/src 폴더 안에 있다.[8]

표 4 패킷 byte의 기능

(1) Sync Flag	패킷의 첫 번째 byte이며, 항상 0xff이다.
(2) Sync Flag/Protocol version	ROS Groovy에서 0xff였고, 그 후 0xfe로 설정된다.
(3) Message Length	패킷의 길이를 나타낸다.
(4) Checksum over message length	패킷 손상을 찾기 위한 길이의 checksum이다.
(5) Topic ID	각각의 topic에 할당되어진 ID이다. 0-100의 범위가 시스템 관련 기능에 할당되어진다.
(6) Serialized Message data	각 topic과 관련된 데이터이다.
(7) Checksum of Topic ID and data	topic에 대한 checksum과 패킷 손상을 찾기 위한 데이터이다.

4.5 DWA (Dynamic Window Approach) Planner

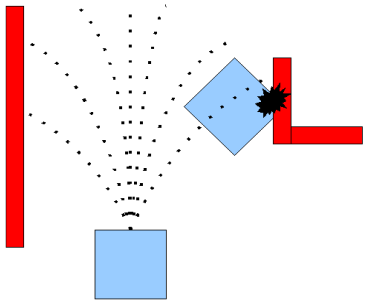


그림. 14 Local path로 사용된 DWA Planner의 details

(1) 로봇의 control space에서 각각 sample을 취한다.(dx, dy, dΘ)

(2) 각각 뽑힌 속도들에 대해, 로봇의 현재 상태로부터 해당 속도로 진행하게 된다면(짧은 시간 동안), 앞으로 어떤 일이 일어날 것인지에 대해 시뮬레이션해본다.

(3) 앞선 시뮬레이션 결과를 통해 trajectory 결과를 평가한다. 다음과 같은 평가요소를 이용해 필요 없는 trajectory를 제외한다.

평가요소 : proximity to obstacles, proximity to the goal, proximity to the global path, and speed.

(4) 가장 평가점수가 높은 trajectory를 골라 mobile base로 속도를 보낸다.

(5) 위의 내용을 반복한다.

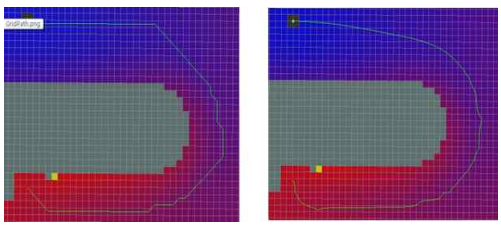


그림. 15 Global Planner의 grid_based mode와 polynomial_filter mode

DWA는 이동 경로 계획 및 장애물을 회피할 때 많이 사용되는 방법으로 로봇의 속도 탐색 영역(velocity search space)에서 로봇과 충돌 가능한 장애물을 회피하면서 목표 지점까지 빠르게 다다를 수 있는 속도를 선택하는 방법을 말한다. ROS에서는 지역 이동 계획에 Trajectory planner를 많이 사용했으나 최근 들어서는 DWA가 그 성능 면에서 뛰어나기에 대체되어가고 있다.

우선, 로봇은 X, Y축의 위치 좌표가 아니라 병진속도 v와 회전속도 w를 축으로 하는 속도 탐색 영역(velocity search space)에서 생각하면 [그림. 16]와 같다. 이 공간에서 로봇은 하드웨어의 한계로 인하여 최대 허용 속도가 존재하고 이를 다이나믹 윈도우(Dynamic Window)라 한다.

이 다이나믹 윈도우상에서 목적함수G (v, w)를 이용하여 로봇의 방향, 속도, 충돌을 고려한 목적함수가 최대가 되는 병진속도 v와 회전속도 w를 구하게 된다.[9]

Ⅳ. 본 론(3)
- SOUND LOCALIZATION

이 논문에서는 사람의 목소리를 버저 소리라고 가정하고 로봇이 버저가 울리는 위치를 찾아가는 것을 목표로 했다. 로봇은 전체 지도와 원래 버저 소리의 크기를 모르기 때문에, 로봇으로부터 버저까지의 정확한 거리를 구할 수는 없다. 즉, 로봇은 버저가 울리는 거리는 알지 못한 채 버저가 울리는 각도만 알고 있는 상태로 움직이게 된다.

1. 필터

단순히 버저 소리의 크기를 이용하여 인식하기에는 주변의 잡음(특히 모터의 소리와 같은 로봇 자체의 잡음)이 너무 많기 때문에 버저의 각도를 계산하기에는 부적절하다. 따라서 필터를 이용하여 버저의 소리만을 걸러낼 필요가 있다.

우선 버저의 소리를 필터로 걸러내기 위해 센서로 들어오는 소리의 크기에 대한 정보를 주파수로 변환할 필요가 있다. 센서로 들어오는 소리의 크기 정보를 256개 누적시킨 후 이를 FFT 변환을 통해 주파수 영역으로 데이터를 변환을 시킨다.

[그림. 17], [그림.18]는 센서로 들어온 크기 정보를 FFT 변환 시켜 주파수 영역으로 데이터를 변환시킨 그림이다. FFT 결과를 보면 버저의 소리에서 2개의 peak가 뜨는 것을 확인할 수 있다. 따라서 버저의 주파수는 저 2개의 peak라고 할 수 있다.

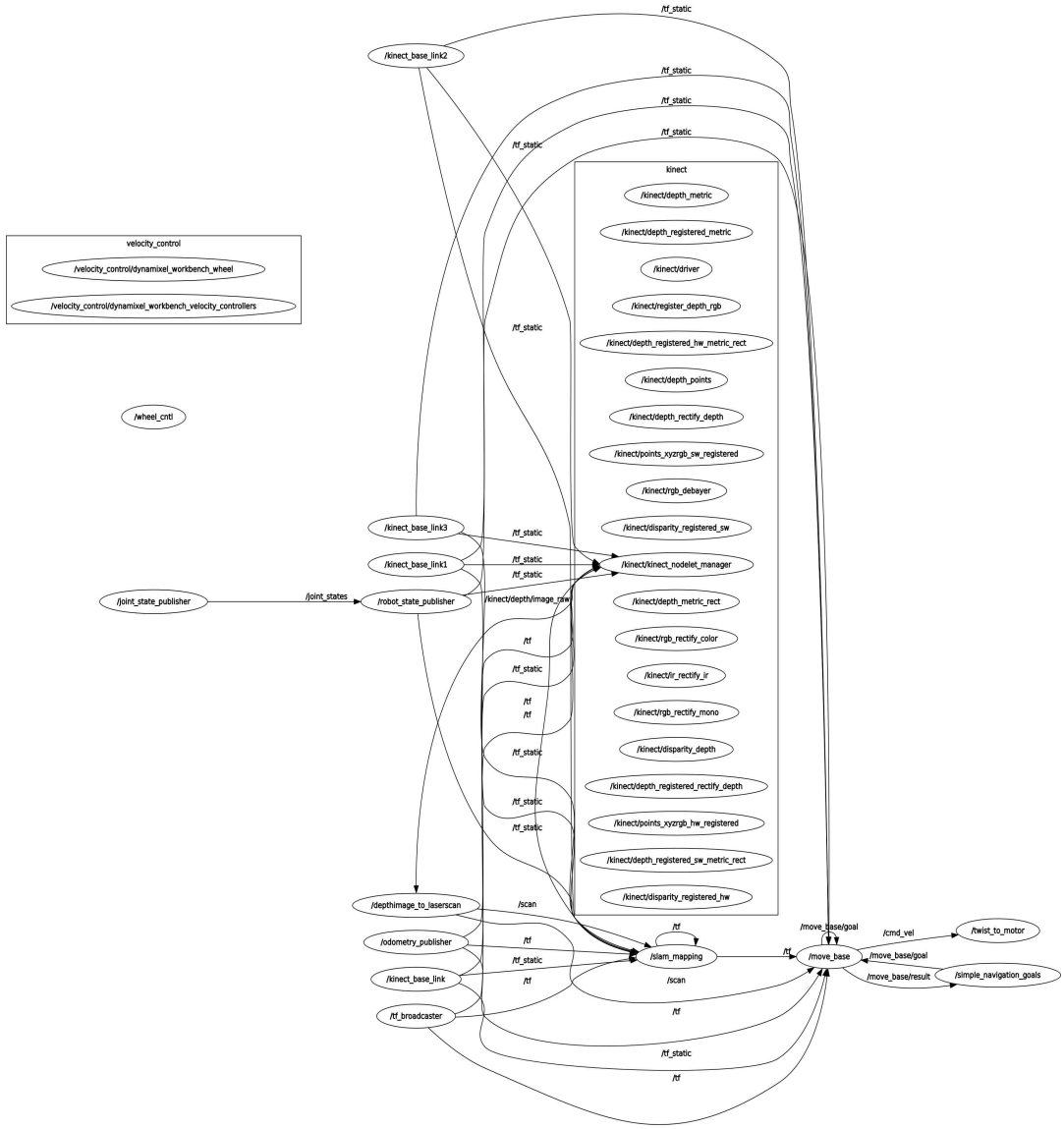


그림. 16 ROS topic list

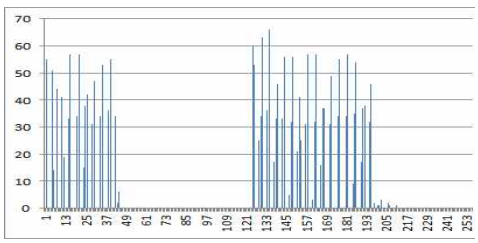


그림. 17 raw 데이터

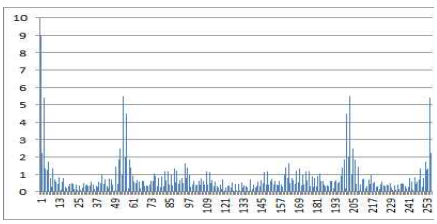


그림. 18 FFT 데이터

여러 소리가 들어올 때 FFT 변환을 시키면 버저의 주파수 이외에 다른 주파수도 peak가 뜰 것이기 때문에 버저 소리를 필터링시켜줄 필요가 있다. 필터는 ideal filter와 Gaussian filter 두 개를 사용했으며 그 결과는 [그림. 19], [그림. 20]과 같았다.

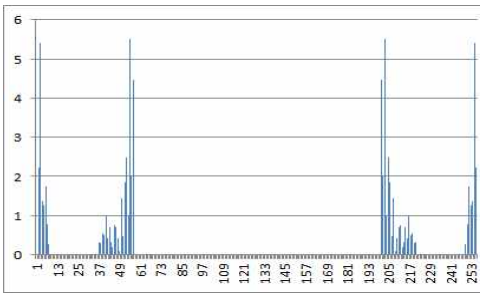


그림. 19 ideal filter

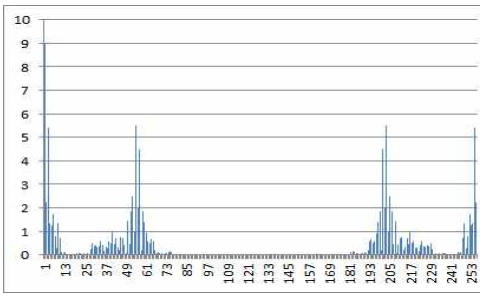


그림. 20 Gaussian filter

위 그래프 결과 처음 DC 소리에 나오는 주파수인 0의 값을 거르기 위해서는 Gaussian filter보다는 ideal filter를 사용하는 것이 좋다는 판단이 되어 필터는 ideal filter를 이용하여 버저의 소리를 거른다.

2. 소리 각도 계산

버저 소리의 각도 계산은 센서 간의 위치 관계로부터 계산 할 수 있다. 각 센서는 버저로부터의 거리가 다르기 때문에 소리 신호가 들어오는 시간 차이가 생기게 된다. 이러한 시간 차이를 가지고 [그림. 21]과 같이 계산할 수 있다.

버저는 센서 사이의 거리보다 충분히 멀다고 가정하면 두 센서로부터 들어오는 버저의 소리는 거의 평행하다고 가정할 수 있다.

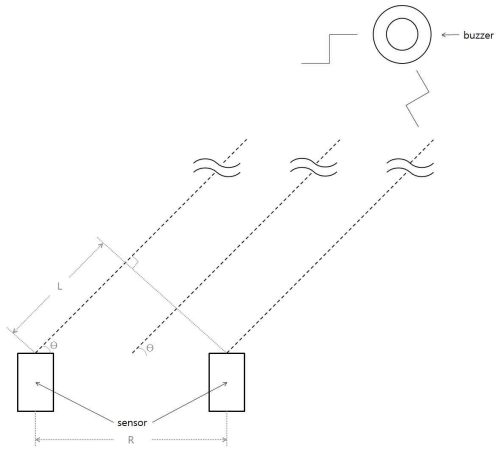


그림. 21 소리 각도 계산

이때 filtering 결과에 의해 나온 버저의 peak 사이 시간 차이(t)를 알고 있다고 하면, 로봇으로부터 버저까지의 각도(Θ)를 아래와 같이 구할 수 있다.

$$L = t \times \text{소리속도}(340m/s)$$

$$\Theta = \cos^{-1}\frac{L}{R}$$

위 계산 결과 나온 Θ값을 rosserial을 이용해 simple_navigation_goals node에 Θ값에 넘겨준다.

Ⅴ. 성능 분석 및 평가

재난 로봇의 구동 성능을 평가하기 위해 재난현장과 유사한 장애물을 모사하여 주행시험을 시행하였다. 주행 시험은 로봇의 기본적인 구동성능을 평가하기 위한 것으로 장애물이 있는 환경과 그렇지 않은 환경에서의 로봇이 Frontier planning을 통해 map을 그릴 수 있는지를 파악하였다. 한편 로봇이 목표 지점에 도달하는 시간의 측정을 통해 로봇의 성능평가를 진행하였다.

구현한 로봇의 성능평가를 위해 장애물이 없는 3개의 map과 장애물이 있는 2개의 map에서 로봇의 시작 위치에서 3m 앞을 goal 지점으로 하는 실험을 시행했다. map1은 직선, map2는 s자, map3은 ㄷ자, map4는 장애물1, map5는 장애물2 코스로 각각의 코스에서 9번 구동한 실험결과를 나타낸다.[그림. 22], [표 5], [그림, 23]

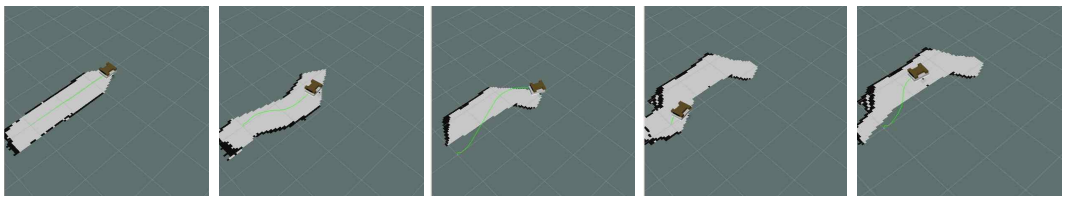
표 5 실험 결과

횟수 \ map	직선	s자	ㄷ 자	장애물1	장애물2
1회	24.36	25.34	1.56.04(oscil)	3.51.00(x)	30.51
2회	54.42(x)	1.41.07	28.24	31.11	27.82
3회	25.66	25.65	31.41	36.79	29.8
4회	24.44	25.28	1.47.31(x)	45.28	33.13
5회	25.31(oscil)	2.28.79(x)	41.26	1.46.43(x)	30.36
6회	23.72	1.30.87(x)	34.67	23.31	39.86
7회	24.5	1.10.88(x)	28.64	1.18.71	45.39
8회	x	1.58.12	54.31	1.32.94(x)	25.12
9회	24.76	36	33.44	47.61	35.24
평균	24.57	55.24	36.00	43.80	33.03
성공률(%)	66.67	66.67	77.78	66.67	100
oscillation(%)	11.11	0	11.11	0	0

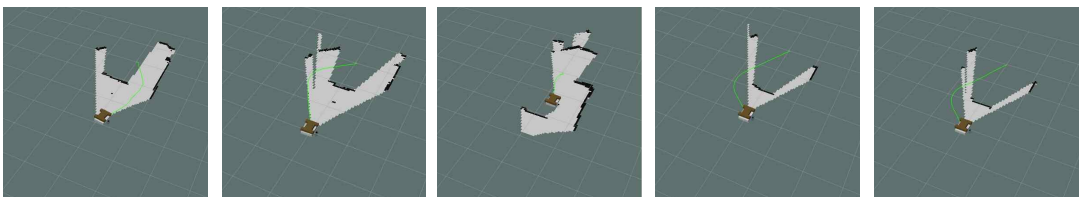
*단위 : sec, x : 실패, oscil : oscillation

(a) map1 (b) map2 (c) map3 (d) map4 (e) map5

그림. 22 성능평가를 위한 맵



(a) map1 (b) map2 (C) map3-1 (d) map3-2 (e) map3-3



(f) map4-1 (g) map4-2 (h) map5-1 (i) map5-2 (j) map5-3

그림. 23 Rviz를 통한 SLAM 시뮬레이션

1. 성능평가

로봇을 구동하게 되면 theta값과 x값을 이용하여 목표지점을 지정하게 되고 global plan과 local plan을 갖는다. SLAM으로 만들어진 지도를 이용하여 목표지점으로 갈 수 있는 global plan과 local plan을 업데이트한다. 목표지점에 도달하게 되면 명령 창에 Goal reached!가 뜨면서 로봇이 정지한다. 시간은 로봇이 움직이기 시작한 시간부터 Goal reached!가 뜨기까지의 시간을 측정하여 기록한다.

본래는 센서 값을 받아 로봇에서 소리가 울리는 지점과의 각도를 rosserial로 통신해야 하지만, sound localization이 센서의 문제로 구현되지 않았기 때문에 rosserial에서 theta값을 1.0으로 고정한 상태로 통신을 하는 방식으로 실험을 진행한다. 이와 같은 이유로 목표지점은 로봇의 시작 위치에서 3m 앞을 목표지점으로 한다.

map1, map2, map3은 주어진 길에서 목표지점을 찾아가는 것을 목표로 하여 가장 간단한 map에서 SLAM, navigation과 연동되어 로봇이 구동됨을 확인했다. map4는 목표지점까지 가는 길에서 장애물이 놓여있는 상황에서 회피함을 확인하였고, map5는 장애물이 놓여있는 상황에서 최단 거리를 찾는 A^* 알고리즘이 제대로 동작하는 것을 확인하였다.

2. 성능평가 분석

실험 결과 장애물이 있는 환경과 없는 환경에서의 차이가 크지 않은 거로 보아 장애물의 여부보다는 코스의 폭이나 모양이 로봇의 주행에 많은 영향을 끼친 것으로 파악된다. 직선 코스에서 평균이 24.57초로 가장 빨랐고, s자 코스에서는 평균 55.24초로 가장 느렸다.

goal이 형성되는 시점에서 SLAM의 추정에 의한 오차에 의해 [그림. 24]처럼 경로가 휘게 형성되는 문제가 발생한다.

위와 같은 문제가 발생하기 때문에, goal 지점이 [그림. 25-(a)]의 형태처럼 goal에 도착하는 경로와 map이 충돌하지 않는 경우, oscillation이 일어나지 않았지만, [그림. 25-(b)]의 형태처럼 goal에 도착하는 경로와 map이 충돌하게 되면 로봇이 막힌 길과 경로 사이에서 길을 정하지 못하게 되어 oscillation이 일어난다.

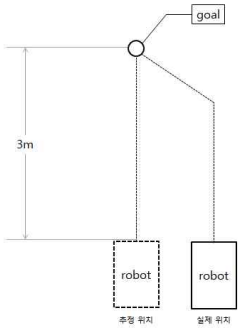
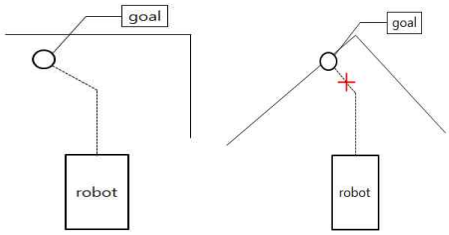


그림. 24 SLAM의 추정에 의한 goal 위치 오차



(a) oscillation이 일어나지 않는 경우 (b) oscillation이 일어나는 경우

그림. 25 map의 최종 모양에 따른 oscillation 발생 유무

map5에서 성공률이 다른 map에서 더욱 큰 이유는 goal 지점과 map의 마지막 지점이 멀기 때문에 이와 같은 충돌이 일어나지 않기 때문이다.

map2에서는 s자 코스의 너비가 작을 경우, 로봇이 global plan을 제대로 잡지 못하는 경우가 발생하여 실패하는 경우가 생겼고, s자 코스의 너비를 크게 줄 경우에는 이런 문제가 발생하지 않았다. global plan이 코스 바깥으로 가는 길과 코스를 통해 목표지점으로 가는 plan 중 선택하지 못하는 문제가 발생했다.

너비가 작을 경우 이런 문제가 발생하는 것은 KINECT에서 받은 depth image를 업데이트하는 시간보다 로봇이 장애물에 도달하는 시간이 짧아서 장애물을 제대로 인식하지 못하고 경로를 이탈하는 문제가 발생했다. 시뮬레이션 상에서는 아직 SLAM이 업데이트되지 않았기 때문에 갈 수 있는 길로 추정하는 것이 문제이다.

그리고 성능평가 중 1분이 넘어가게 되는 경우, 목표지점에 도달하지 못하는 경우가 대부분이었으며, 이는 탐색이 실패한 경우로 간주하였다.

Ⅵ. 결론

본 논문의 주된 목적은 재난 상황에서의 생존자 확인 및 정확한 위치 정보를 습득하는 방법을 고안하는 것이다. 특히 전력보급이 모두 끊겨 주변의 위치를 제대로 확인하기 힘들고 추가적인 인명피해 등의 위협으로 인력이 투입되기 힘든 상황에서 로봇의 투입은 불가피하다. 더군다나 방사능 등의 통신 교란 상태가 온다면 로봇 스스로 주변 환경을 인식하고 판단할 줄 아는 AI 시스템이 필수적일 것이다. 이 프로젝트는 이러한 로봇을 위한 Solution을 연구하고 개발하는 것을 목적으로 한다.

SLAM을 이용하면 주변 환경을 인지할 수 있으며, 로봇이 이동하면서 저장된 map 데이터와의 매칭을 통해 로봇의 현재 위치를 파악할 수 있게 된다. 하지만 이동하는 방향이나 목적 등을 결정할 의사결정 알고리즘이 필요한데 이를 설계하는 것이 향후 과제로 남을 것이다. 재난 현장에서 이동 방향을 결정하는 것은 재난현장에서의 사람의 목소리다. 소리 데이터를 이용해 이동 방향을 결정하며 SLAM 데이터를 토대로 장애물을 피하는 등의 의사결정 알고리즘을 새롭게 설계할 수 있어야 할 것이다.

이번 프로젝트로 만들어질 로봇은 바로 재난현장에 투입하기엔 무리일 것이다. 하드웨어적으로 보았을 땐 재난현장과 같이 험난한 지형이나 복잡한 구조에서는 제대로 작동하기 힘들 것이며, 소프트웨어적으로는 3D 환경을 2D 환경으로 인식하고 사운드 센서로부터 얻은 위치추정치 역시 2차원적 추정뿐이기 때문이다.

이 프로젝트는 더 높은 차원으로의 연구의 초석으로써 사용될 수 있다. 본 프로젝트의 로봇은 기본적으로 KINECT의 센서를 기반으로 설계되었으며 전체적인 알고리즘 역시 KINECT의 역량을 그대로 반영해 설계한 것이다. 하지만 문제점이 있다면 KINECT의 소리 인식 센서가 1차원의 어레이-마이크라는 점이며 이는 최대 2차원적으로 소리의 위치를 추정할 수 있게 된다. 또한 KINECT의 depth 센서는 고가의 3D Scanner와 다르게 정확한 Scan 능력이 떨어지며 실시간으로 3차원의 데이터를 인식하기엔 하드웨어적으로 불가능하다. 또한, 추후 연구 과제는 주변 환경을 인식한 SLAM 데이터와 소리 데이터의 각도와 세기를 이용해 더욱 정확한 위치를 추정하는 것이다. 이 부분에서 딥러닝 알고리즘을 적용하게 된다면, 학습된 데이터를 이용해 소리의 회절 및 노이즈 등을 고려한 최적의 위치추정 및 의사결정을 할 수 있는 능력이 있는 로봇을 구현할 수 있을 것이다.

참 고 문 헌

[1] http://kosis.kr/statHtml/statHtml.do?orgId=101&tblId=DT_1YL8601&vw_cd=&list_id=&scrId=&seqNo=&lang_mode=ko&obj_var_id=&itm_id=&conn_path=E1

[2] https://www.kitech.re.kr/main/

[3] 표윤석 외 3명, “ROS 로봇 프로그래밍 [기초 개념부터 프로그래밍 학습, 실제로봇에 적용까지]” ,2017

[4] http://wiki.ros.org/ko

[5] Leonard T. Park, Songhwai Oh, Implementation of Mapping and Automatic Navigation on a Mobile Robot Using Linux ROS“ , 2013

[6] Quigley, Morgan., Gerkey, Brian., Smart, William D.“Programming robots with ROS” ,2015

[7] Joseph, Lentin,“Mastering ROS for robotics programming : design, build, and simulate complex robots using Robot Operating System and master its out-of-the-box functionalities”,2015

[8] Joseph, Lentin,“ROS 로보틱스 프로그래밍 : 4차 산업혁명 시대의 핵심 기술, 로봇 프로그래밍”, 2017]

[9] Mastering ROS for Robotics Programming, Joseph, Lentin,2015

[10] Dieter Fox, Wolfram Burgard, Sebastian Thrun, “The dynamic window approach to collision avoidance”, IEEE Robotics & Automation Magazine, 제 4권, 제 1호, 23-33쪽, 1997