

전자종합설계 (2)

최종보고서

Smart Delivery Robot(SDR)

학과 : 전자전기융합공학과
지도교수 : 김상규 교수님
팀명 : SDR
팀원 : 박장원, 박준형, 김나현

Smart Delivery Robot(SDR)

팀원 : 박장원, 박준형, 김나현

목 차

1. 설계 배경	3
2. 설계 목적	3
3. 설계 일정	3
4. 부품 및 사용기기	3
5. 연구비 사용내역	3
6. 설계 구상	4
1) 소프트웨어 구상	
2) 하드웨어 구상	
3) 전체 구상	
7. 주요 기능	4
1) 운영프로그램	
2) 주요 기능	
8. 설계 과정	5
1) SLAM	
2) UBUNTU	
3) ROS	
4) VNC Viewer	
5) 라즈베리파이	
6) Arduino	
7) Teloep Keyboard	
8) Map	
9) Odometry	
10) Navigation	
11) Send goals	
12) 하드웨어 제작	
9. 결과 활용 방안	32
10. 비교 및 고찰	33

1. 설계 배경

이 프로젝트를 진행하게 된 배경은 먼저 현재까지도 택배 기사들의 과로 문제는 사회적 이슈로 떠오르고 있다. 일의 높은 강도와 장시간 근무로 인해 건강이 악화되고 그로 인해 비교적 이른 나이에 퇴직하는 인원들이 증가하고 있다. 그러나 지금 우리 사회는 고령화가 진행되고 있어 택배 기사의 평균 연령이 높아지고 있다. 택배기사의 평균연령이 높아짐에 따라 일을 할 수 있는 인원이 현저히 줄어들게 될 것이다.

다음으로 도로와 보도를 통하여 이동하는 택배 로봇은 어린이와 노약자 등 보행자들의 안전을 위협할 소지를 충분히 갖고 있기 때문에 아직까지는 상용화되기 어렵다고 판단하였다. 또한 노동자의 일자리 감소도 무시할 수 없는 부분이다. 실제 쿠팡에서 물류 자동화 전초 기지를 세우고 나서 순 고용자 수는 감소세로 이어졌다.

이러한 부분을 고려하여 고안한 아이디어가 SDR이다.

SDR은 Smart Delivery Robot의 약자로 아파트 건물 내에서 택배를 배송하는 로봇이다.

2. 설계 목적

건물 동이나 라인 입구에 배치하여 택배기사들이 택배를 배송할 호수를 입력하면 그 장소로 택배를 배송하는 로봇이다. 이러한 로봇을 사용하면 일자리 감소에 대한 우려도 사라질뿐더러 택배기사들의 배달시간 또한 대폭 감소하게 되어 일의 강도가 현저히 줄어들 것이다.

3. 설계 일정

연구내용 \ 월	5	6	7	8	9	10	11
주제선정							
조사 및 분석							
기본 설계							
상세 설계 및 제작							
테스트 및 작품전시 준비							

4. 부품 및 사용기기

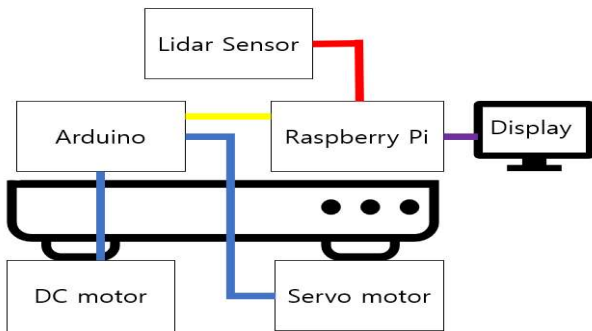
- RP Lidar sensor A2
- Raspberry pi 4
- Display
- Arduino UNO
- DC motor 12V×2
- 휠엔코더
- 브레드보드
- L298N Motor driver
- Servo motor
- IMU sensor
- 보조배터리
- 배터리홀더
- 건전지 AAA×8
- 케이블
- 아크릴 3T
- 비금속 레이저 커터
- 인두기, 납실

5. 연구비 사용내역

사용내역/품목	금액
RPLidar sensor A2	387,420
라즈베리파이	388,000
자동차 인코더 샤프트	123,200
보조배터리	33,900
배터리홀더	5,450
배터리	15,600
아크릴 3T	5,600
양면테이프	12,900
전용 접착제	12,400
지지대 볼트, 너트	6,240
아크릴 경첩	5,050
모터 드라이버	1,890
브레드보드	3,500
IMU sensor	4,900

6. 설계 구상

1) 소프트웨어 구상



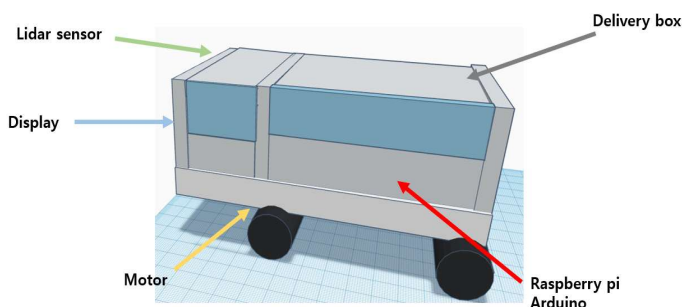
소프트웨어 측면에서 메인보드로 라즈베리파이와 아두이노를 사용하여 제작할 것이다.

RP Lidar sensor를 라즈베리파이와 연동하여 로봇의 사용 장소를 Mapping 하여 지형지물의 데이터를 축적한다.

Mapping 한 데이터는 Raspberry pi와 Display를 연결하여 시각화할 것이다. 이렇게 Lidar를 통해서 얻은 데이터를 기반으로 지도가 갖춰지면 라즈베리파이에서 아두이노로 데이터를 전송하여 아두이노에서 모터 제어를 통해 로봇이 움직이는 방향으로 설계할 것이다.

모터는 앞바퀴에 달린 Servo 모터를 통해서 방향을 제어하고 뒷바퀴에 달린 모터 드라이버와 DC 모터 2개를 통해서 속도를 제어할 것이다.

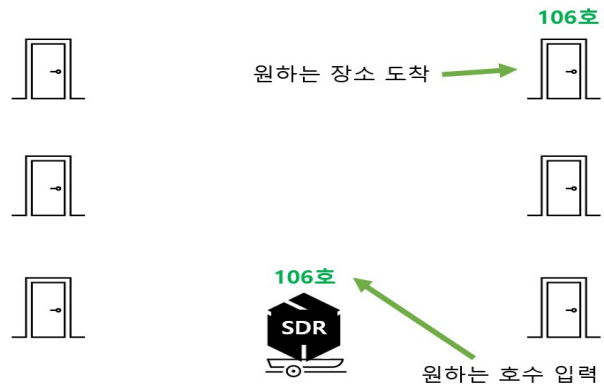
2) 하드웨어 구상



하드웨어 측면에서 비금속 레이저 커터를 활용하여 로봇의 밑단을 제작해 바퀴와 모터들을 결합하고 메인보드인 라즈베리파이와 아두이노를 그 위에 올릴 것이다.

또한 Lidar 센서를 앞쪽 위에 부착하고 Display도 앞쪽에 부착할 것이다. 그리고 뒤쪽에 택배물을 올릴 수 있는 택배함을 제작할 것이다.

3) 전체 구상



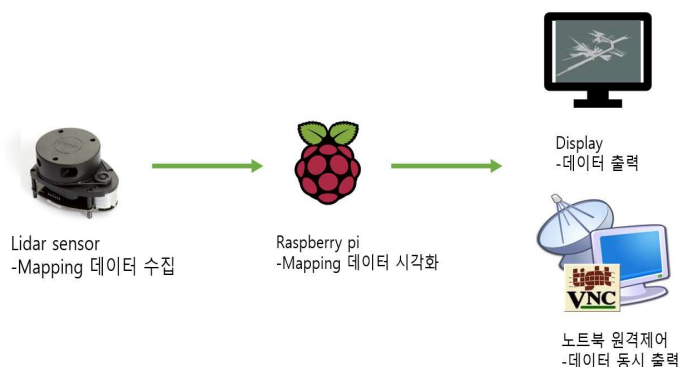
로봇에 원하는 호수인 106호를 입력하면 로봇이 106호 앞으로 택배를 가지고 이동한다.

7. 주요 기능

1) 운영 프로그램

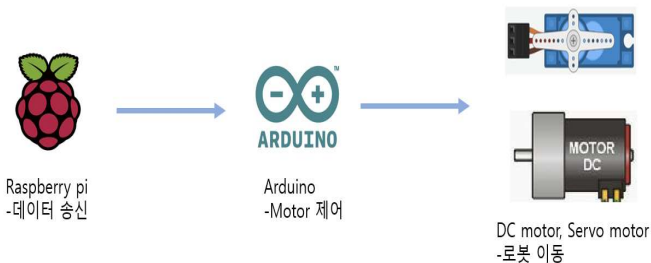
- Ubuntu 20.04
- ROS noetic
- VNC viewer
- Rviz
- Hector slam
- RPlidar node
- Teleop keyboard
- Arduino IDE

2) 주요 기능



라즈베리파이에 Ubuntu 20.04와 ROS noetic을 설치한 후 Lidar sensor와 연동할 것이다. 그 후 Lidar sensor를 통해 Mapping 데이터를 수집하고 수집한 Mapping 데이터는 Raspberry pi의 Rviz에서 시각화한다.

Mapping 한 데이터는 Raspberry pi와 Display를 연결하여 데이터를 출력할 것이다. 또한 Raspberry pi의 데이터 동시 출력을 위해 노트북을 원격제어하기 위한 VNC viewer 프로그램을 사용할 것이다.



Raspberry pi에서 얻은 Mapping 데이터 값을 아두이노와 시리얼 통신을 해 송신할 것이다. 그 데이터 값을 통해서 아두이노는 DC motor 2개와 Servo motor 1개를 제어해 로봇을 원하는 위치로 이동하게 할 것이다.

8. 설계 과정

1) SLAM

SLAM(동시적 위치 추정 및 지도작성)은 자율주행 차량에 사용되어 주변 환경 지도를 작성하는 동시에 차량의 위치를 작성된 지도 안에서 추정하는 방법이다. SLAM 알고리즘을 통해 차량은 미지의 환경에 대한 지도를 작성할 수 있다. 엔지니어는 지도 정보를 사용하여 경로 계획 및 장애물 회피 등의 작업을 수행합니다. 따라서 SLAM은 점점 더 많은 분야의 실제 응용 사례에 사용되고 있다.

2) UBUNTU

2.1 선정 배경

위의 부품에서 볼 수 있듯이 SLAM을 하기 위한 센서 중 하나로 RPlidar A2를 채택하였다. 이러한 로봇 센서를 사용하기 용이한 OS가 ROS이다. ROS는 삼성 노트북에는 Window가, 애플에서는 mac OS가 있듯이 로봇을 제어하는 작동 시스템이라고 생각하면 된다. 이 ROS를 사용하기 위해서는 Linux를 사용해야 하는데 가장 보편적으로 사용하는 배포판 중에 하나인 Ubuntu를 사용하기로 결정했다.

Ubuntu에는 출시된 연도 별로 버전이 다양하게 분포되어 있고 그에 따른 ROS 버전도 다양하다. 그중에서 우리는 Ubuntu 20.04 ROS Noetic을 사용하기로 결정하였다. 그 이유는 20.04가 ROS1의 마지막 버전이기 때문이다. 그 이후로는 ROS2 Fox, Humble 등이 있음에도 ROS1을 선택한 이유는 처음 접해보는 OS였기 때문에 정보의 양이 조금이라도 더 많았어야 했기 때문이다.

2.2 설치 과정

Ubuntu를 바로 라즈베리파이 보드에서 설치를 진행하면 보드 손상 우려가 있어 먼저 노트북에서 Ubuntu 환경을 구축해 보기로 결정하였다.

3) ROS 설치

3.1 packages.ros.org의 소프트웨어를 허용하도록 컴퓨터를 설정

```
sudo sh -c 'echo "deb
http://packages.ros.org/ros/ubuntu $(lsb_release
-sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

3.2 키 설정

```
sudo apt install curl # if you haven't already
installed curl
curl -s
https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

3.3 설치

먼저 Debian 패키지 색인이 최신인지 확인

```
sudo apt update
```

데스크탑-전체 설치

```
sudo apt install ros-noetic-desktop-full
```

3.4 환경설정

ROS를 사용하는 모든 bash 터미널 에서 이 스크립트를 소스로 제공

```
source /opt/ros/noetic/setup.bash
```

3.5 패키지 빌드를 위한 종속성

```
sudo apt install python3-rosdep python3-rosinstall
python3-rosinstall-generator python3-wstool
build-essential
```

3.6 설치 확인

```
roscore
```

```
...loggingto/home/vh/.ros/log/l30d8df6-9eee-11e
a-906d-492a6c9dca07/roslaunch-varhowto-com-
26839.log
```

Checking log directory for disk usage. This may take while.

Press Ctrl-C to interrupt

Done checking log file disk usage. Usage is <1GB.

```
started                                roslaunchserver
http://varhowto-con:37317/ ros_ . COMM version
1.15.6
```

SUMMARY

PARAMETERS /rostdistro: noetic

/rosversion: 1.15.6

NODES

auto-starting new master

process[naster]: started with pid[26864]

ROS_MASTER_URI[=http://varhoivto-con:11311/
setting /run_id to

```
b30d8df6-9eee-11ea-906d-492a6c9dca07
process[rosout-1]: started with pid [26874]
started core service [/rosout]
```

위와 같이 코드가 출력되면 ROS Noetic이 잘 설치된 것을 확인해볼 수 있다.

4) VNC Viewer

VNC viewer에는 다양한 회사의 프로그램이 있지만 'Tight VNC Viewer'를 선택하였다. 또한 매번 라즈베리파이에 마우스와 키보드를 연결할 수 없어 데이터를 노트북으로 동시 출력하기 위해 노트북으로 원격 제어를 하기 위해 설치하기로 결정하였다.

4.1 TightVNC서버 설치

```
sudo apt-get update
sudo apt-get install tightvncserver
```

4.2 VNC server 초기 비밀번호 설정

```
vncserver
```

위의 명령어를 입력하여 VNC server를 실행하여 초기 비밀번호를 설정했다.

You will require a password to access your desktops.

Password:

Verify:

Would you like to enter a view-only password (y/n)? n

New 'X' desktop is ubuntu-desktop:1

Creating default startup script /home/ubuntu/ · vnc/xstartup

Starting applications specified in /home/ubuntu/ vnc/xstartup

Log file is /home/ubuntu/vnc/ubuntu-desktop:1.log

노트북에서 원격제어를 하기 위함이므로 view_only는 n으로 거절해준다.

4.3 시작 스크립트를 수정

```
sudo nano ~/.vnc/xstartup
```

위의 명령어를 입력하여 시작 스크립트를 열어준 후 #x-window-manager & 코드 앞에 위치한 #을 지워주어 노트북 window에서 제어할 수 있게 해준다.

```
/bin/sh

xrdb $HOME/.Xresources
xsetroot -solid grey
#x-terminal-emulator -geometry 80x24+10+10 -ls
-title "$VNCDESKTOP Desktop" &
x-window-manager &
# Fix to make GNOME work
export XKL_XMODMAP_DISABLE=1
/etc/X11/Xsession
```

4.4 재부팅 시 Tight VNC가 자동 실행되도록 설정

```
sudo nano /etc/init.d/tightvnc
```

위의 명령어를 통해 init.d에 파일을 만들었다.

```
#!/bin/bash
#### BEGIN INIT INFO
# Provides:          vncserver
# Required-Start:    networking
# Required-Stop:     networking
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
#### END INIT INFO

PATH="/$PATH:/usr/bin/"
export USER="jangwon"

DISPLAY="1"
DEPTH="16"
GEOMETRY="1280x800"

OPTIONS="-depth ${DEPTH} -geometry
${GEOMETRY} :${DISPLAY}"

. /lib/lsb/init-functions

case "$1" in
start)
log_action_begin_msg "Starting vncserver for user
'${USER}' on localhost:${DISPLAY}"
su ${USER} -c "/usr/bin/vncserver ${OPTIONS}"
;;

stop)
log_action_begin_msg "Stopping vncserver for user
'${USER}' on localhost:${DISPLAY}"
su ${USER} -c "/usr/bin/vncserver -kill :${DISPLAY}"
;;

restart)
$0 stop
$0 start
;;
esac
exit 0
```

4.5 Tightvnc파일에 권한을 추가

```
sudo chmod +x /etc/init.d/tightvnc
sudo update-rc.d tightvnc defaults
```

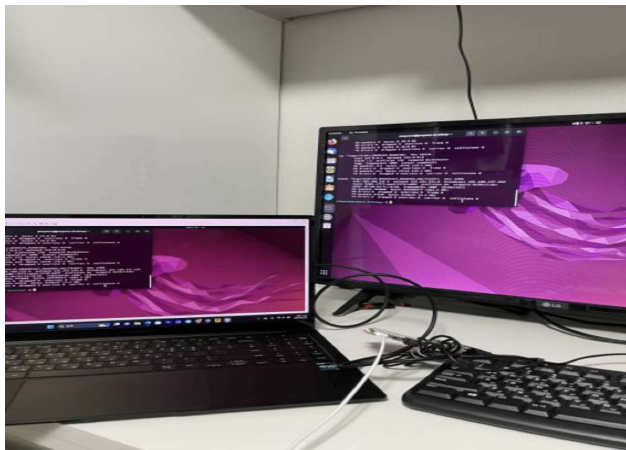
```
init 6
```

init 6로 재부팅을 해준다.

4.6 Tightvnc 실행



초반 Putty를 사용했을 때와 마찬가지로 노트북 핫스팟에서 ip 주소를 얻어 입력해 준다.



그 결과 위의 사진과 같이 Tight VNC가 잘 이루어진 것을 확인해 볼 수 있다. 앞으로 설명할 과정들은 모두 이 원격제어를 통해 이루어질 것이다.

5) 라즈베리파이

5.1 Ubuntu Server

먼저 라즈베리파이를 사용할 때 필요한 SD카드를 32GB를 사용하였다. 이 SD카드에 라즈베리파이 이미

지 파일을 구축해야 하는데 밑의 사진에서 보이는 것처럼 'Raspberry Pi imager'를 사용하면 수월하게 진행할 수 있다. SD카드를 'SDcard Formatter'를 이용하여 포맷을 진행해 준다.

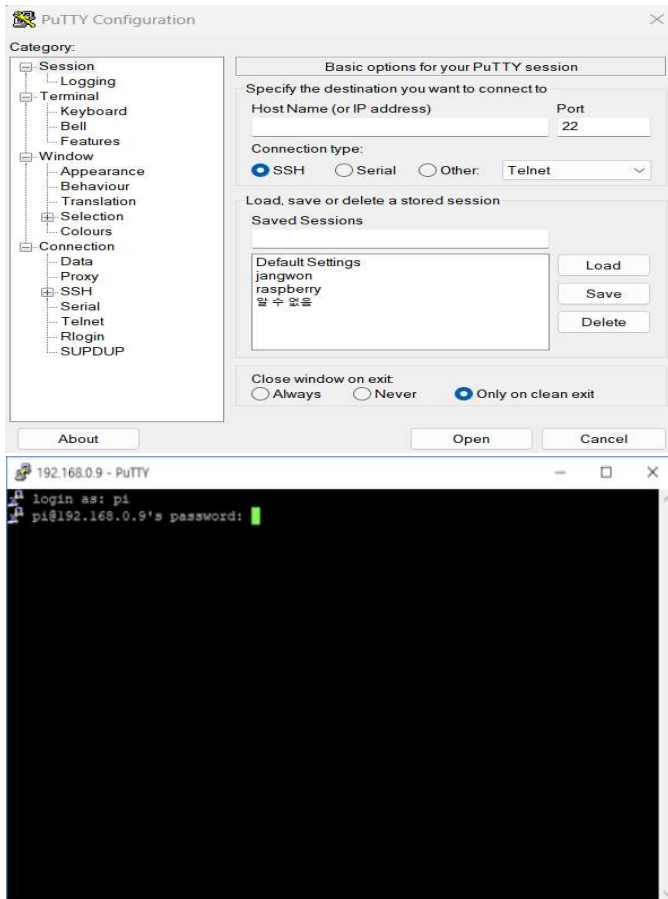
다음으로 운영체제 선택에서 'Ubuntu 20.04'를 선택해 준다. 여기서 문제가 발생하는데 노트북에서 설치하는 것과는 다르게 라즈베리파이에서는 arm 버전으로 설치해야 하는데 Ubuntu 20.04는 Desktop 버전이 아니라 Server 버전밖에 없다는 것이다. 오랜 시간 찾아본 결과 Ubuntu 공식 홈페이지에서 오래된 arm desktop 버전은 더 이상 제공하지 않는다는 것이었다.

따라서 Server 버전을 먼저 설치하고 Desktop 버전으로 덮는 방법으로 진행하기로 결정하였다. Server 버전을 설치 진행할 때 설정에서 노트북의 핫스팟에 네트워크가 연결될 수 있도록 핫스팟 이름과 비밀번호를 설정해 주었다.



이렇게 라즈베리파이에 전원을 인가해 주면 노트북 핫스팟 설정 화면에서 ip 주소가 뜨는 것을 확인할 수 있다.

마지막으로 라즈베리파이 서버와 원격 통신을 하기 위해 아래 그림과 같이 'Putty' 프로그램을 사용해 준다.



연결이 잘 되었으면 위와 같은 Terminal 뜨는 걸 볼 수 있다.

5.2 Ubuntu Desktop

apt-get 도구 업데이트

```
sudo apt-get update
```

apt-get 도구 업그레이드

```
sudo apt-get upgrade
```

Desktop 설치

```
sudo apt-get install ubuntu-desktop
```

Setting up nautilus (1:3.36.3-0ubuntu1)

Setting up libecal-2.0-1:amd64 (3.36.4-0ubuntu1)

Setting up libebook-contacts-1.2-3:amd64 (3.36.4-0ubuntu1)

Setting up gnome-keyring (3.36.0-1ubuntu1)

Setting up libedataserverui-1.2-2:amd64

(3.36.4-0ubuntu1)

Setting up software-properties-gtk (0.98.9.2) Setting up yelp (3.36.0-1)

Setting up libbackend-1.2-10:amd64 (3.36.4-0ubuntu1)

Setting up gstreamer1.0-clutter-3.0:amd64 (3.0.27-1)

Setting up libcheese8:amd64 (3.34.0-1build1) ...

Setting up libedata-cal-2.0-1:amd64 (3.36.4-0ubuntu1)

(3.36.4-0ubuntu1)

Setting up libcheese-gtk25:amd64 (3.34.0-1build1)

Setting up libebook-1.2-20:amd64 (3.36.4-0ubuntu1)

Setting up gnome-control-center (1:3.36.4-0ubuntu1)

Setting up evolution-data-server (3.36.4-0ubuntu1)

Setting up gnome-shell (3.36.4-1ubuntu1"20.04.2)

Setting up gnome-shell-extensions-ion-appindicator (33-1)

Setting up ubuntu-session (3.36.0-2ubuntu1)

Setting up gdm3 (3.34.1-1ubuntu1)

Creating config file /etc/gdm3/greeter.dconf-defaults with new version

gdm.service is not active, cannot reload. invoke-rc.d: initscr ipt gdm3, action "reload" failed.

Setting up gnome-session (3.36.0-2ubuntu1)

Setting up gnome-shell-extension-desktop-icons (20.04.0-3~ubuntu20.04.1)

Setting up update-manager (1:20.04.10.1)

Setting up gnome-shell-extension-ubuntu-dock (68ubuntu1"20.04.1)

Setting up update-notifier (3.192.30)

Setting up ubuntu-desktop-minimal (1.450.2)

Setting up ubuntu-desktop (1.450.2)

Processing triggers for dictionaries-common (1.28.1)

Processing triggers for libgdk-pixbuf2.0-0:amd64

```
(2.40.0+dfsg-3)
Processing triggers for libc-bin (2.31-0ubuntu9)
Processing triggers for dbus systemd
(1.12.16-2ubuntu2.1)
```

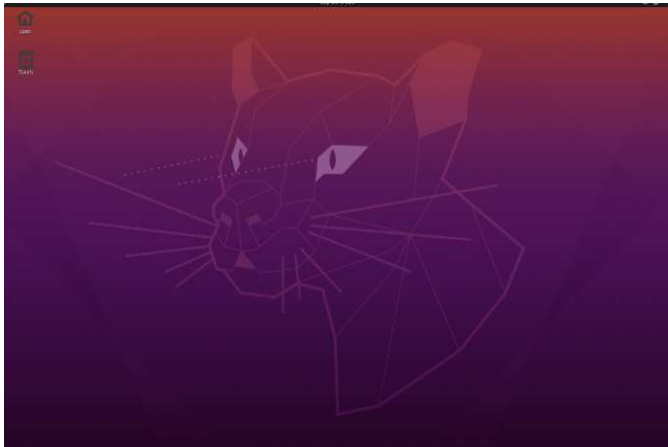
Gui 설치 후 추가 패키지 설치

```
sudo apt-get install indicator-appmenu-tools
sudo apt-get install indicator-session
sudo apt-get install indicator-datetime
sudo apt-get install indicator-applet-complete
```

Gui 실행

```
sudo system reboot
```

시스템을 재부팅하면 라즈베리파이와 HDMI로 연결된 모니터에서 Gui로 전환이 된 것을 확인할 수 있다.



-네트워크 오류 해결

라즈베리파이에서 노트북 핫스팟과 연결은 되어있지만 Gui화면에서 와이파이 설정이 보이지 않아 네트워크를 담당하는 wpa_supplicant를 재부팅 시켜주었다.

```
sudo killall wpa_supplicant
sudo systemctl restart NetworkManager
sudo ifconfig wlan0 up
sudo systemctl restart wpa_supplicant
```

또한 재부팅 시에 초기화되는 현상이 있어서 재부팅 시에 명령어가 자동으로 입력하게 설정해 주었다.

```
sudo nano /etc/systemd/system/wifi-fix.service
Type=oneshot
ExecStart=/bin/bash -c "sudo killall wpa_supplicant
&& sudo systemctl restart NetworkManager &&
sudo ifconfig wlan0 up && sudo systemctl restart
wpa_supplicant"
WantedBy=multi-user.target
sudo systemctl enable wifi-fix.service
```

이제 재부팅을 하여도 와이파이가 잘 연결된 것을 확인할 수 있다.

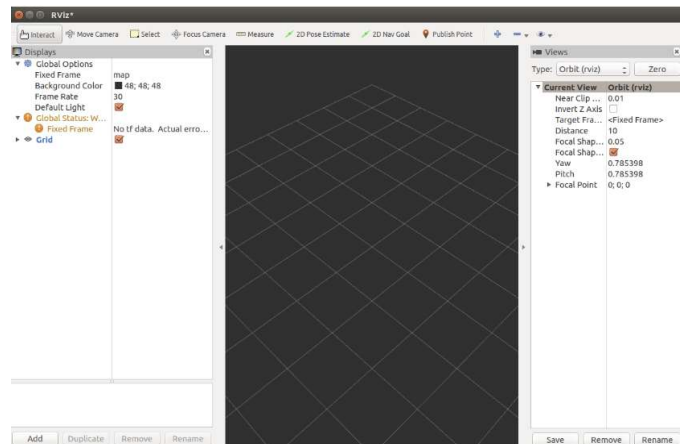
5.3 RPlidar & Hector Slam

Ubuntu에서 RPlidar A2를 활용하여 Slam을 하기 위해선 패키지가 필요하다. 이 패키지를 Launch 파일을 통해 노드를 생성하고 Lidar에서 생성된 data를 slam 노드로 전달해준다. 전달받은 data를 시각화하기 위해 'Rviz'를 사용해준다.

이 때 주의할 점은 lidar와 slam은 한 개의 workspace에 있어야 한다는 것이다.

5.4 Rviz

RViz는 ROS Visualization Tool의 약자로 ROS 도구 중 하나다. 센서 데이터를 시각화하는 기능을 수행한다. 방금 설명한 것처럼 ROS 도구이기 때문에 ROS를 설치하면 자동으로 함께 설치된다.



5.5 패키지 설치

Work Space 생성

```
mkdir catkin_ws && mkdir catkin_ws/src
```

패키지 다운로드

```
git clone github.com/robopeak/rplidar_ros
git clone
github.com/tu-darmstadt-ros-pkg/hector_slam
```

src경로에 rplidar 디렉터리와 hector_slam 디렉터리가 생성된 것을 확인할 수 있다.

이제 패키지의 launch 파일을 수정해야 한다. 먼저 tutorial_tif.launch 파일을 열어준다.

```
1<?xml version="1.0"?>
2
3<launch>
4  <arg name="tf_map_scanmatch_transform_frame_name" default="scanmatcher_frame"/>
5
6  <!--<arg name="base_frame" default="base_footprint"/> MODIFIED BY TIF -->
7  <arg name="base_frame" default="base_link"/>
8
9  <!--<arg name="odom_frame" default="navi"/> MODIFIED BY TIF -->
10 <arg name="odom_frame" default="base_link"/>
11
12 <arg name="pub_map_odom_transform" default="true"/>
13 <arg name="scan_subscriber_queue_size" default="5"/>
14 <arg name="scan_topic" default="scan"/>
15 <arg name="map_size" default="2048"/>
16
17 <node pkg="hector_mapping" type="hector_mapping" name="hector_mapping" output="screen">
18
19   <!-- Frame names -->
20   <param name="map_frame" value="map" />
21   <param name="base_frame" value="$(arg base_frame)" />
22   <param name="odom_frame" value="$(arg odom_frame)" />
23
24   <!-- Tf use -->
25   <param name="use_tf_scan_transformation" value="true"/>
26   <param name="use_tf_pose_start_estimate" value="false"/>
27   <param name="pub_map_odom_transform" value="$(arg pub_map_odom_transform)" />
28 </node>
```

Slam은 평평한 지면에서 진행되고 Base link가 주행 거리계 프레임으로 입력되기 때문에 위의 빨간색으로 표시된 부분을 위의 사진과 같이 수정해 준다.

```
1<?xml version="1.0"?>
2
3<launch>
4
5  <arg name="geotiff_map_file_path" default="$(find hector_geotiff)/maps"/>
6
7  <!--<param name="/use_sim_time" value="true"/> -->
8  <param name="/use_sim_time" value="false"/>
9
10 <node pkg="rviz" type="rviz" name="rviz"
11   args="-d $(find hector_slam_launch)/rviz_cfg/mapping_demo.rviz"/>
12
13 <include file="$(find hector_mapping)/launch/mapping_tif.launch"/>
14
15 <include file="$(find hector_geotiff)/launch/geotiff_mapper.launch">
16   <arg name="trajectory_source_frame_name" value="scanmatcher_frame"/>
17   <arg name="map_file_path" value="$(arg geotiff_map_file_path)" />
18 </include>
19 </launch>
```

Lidar 센서를 활용하여 실시간으로 데이터 값을 받아

들이기 때문에 simulation 시간의 value 값을 false로 변경해 주었다.

Build

```
cd ~/catkin_ws
catkin_make
```

위의 명령어를 통해 catkin_ws 경로에 devel 폴더와 build 폴더가 생성된다.

Hector Slam

위와 같은 과정을 모두 완료하였으니 이제 'Mapping' 을 해볼 것이다.

```
ls /dev/ttyUSB*
```

위의 명령어를 통해 USB로 연결되어 있는 LIDAR센서의 포트를 확인해 볼 수 있다. USB 위치와 순서에 따라 다르지만 지금은 ttyUSB0으로 가정하겠다.

```
sudo chmod a+rw /dev/ttyUSB0
```

위의 명령어로 USB포트와 연동을 시켜준다.

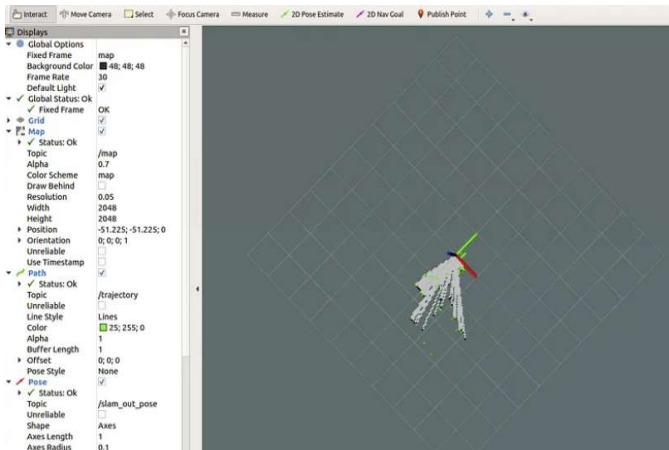
```
cd ~/catkin_ws
source devel/setup.bash
roslaunch rplidar_ros rplidar_a2.launch
```

다음으로 catkin_make으로 생성한 workspace로 진입 후에 source로 변경된 내용을 적용시켜준 후 roslaunch로 launch파일을 실행시켜준다. 그럼 RPlidar A2가 돌아가기 시작하는 것을 확인할 수 있다.

새로운 Terminal을 열어(Ctrl+Alt+T) 위와 동일하게 workspace로 들어간다.

```
source devel/setup.bash
roslaunch hector_slam_launch tutorial_tif.launch
```

위에서 편집했던 tutorial_tif launch 파일을 실행시켜 준다.



이렇게 Lidar sensor로 받은 data 값을 rviz를 통해 시각화한 모습이 잘 보인다.

6) Arduino

6.1 Ubuntu Arduino IDE 설치

먼저 아두이노 홈페이지에 접속하여 Arduino IDE를 설치해 준다.



라즈베리파이에 Ubuntu를 설치할 때와 마찬가지로 Linux ARM 버전으로 다운로드해야 한다.

6.2 Rosserial 설치

```
sudo apt-get install ros-noetic-rosserial-arduino
sudo apt-get install ros-noetic-rosserial
```

ROS 버전이 noetic이기 때문에 noetic으로 입력해 주었다.

6.3 Arduino IDE Libraries에 ros_lib 추가

아두이노가 라즈베리파이의 ROS와 통신이 되려면 ros_lib을 추가해 주어야 한다.

```
cd <sketchbook>/libraries
rosrun rosserial_arduino make_libraries.py
.rosserial/rosserial_arduino/src/rosserial_arduino
```

이제 Arduino IDE의 메뉴에서 메뉴 File > Examples를 보시면 ros examples가 추가된 것을 확인 가능하다.

6.4 Arduino 코드 작성

라즈베리파이에서 필요한 패키지와 프로그램들을 모두 설치한 후에 Teleop Keyboard를 이용하여 Mapping을 진행하기 위한 코드를 Arduino IDE에 작성한다.

-아두이노 코드

```
#include <ArduinoHardware.h>
#include <ros.h>
#include <geometry_msgs/Twist.h>
#include <Servo.h>

#define EN_L 3
#define IN1_L 9
#define IN2_L 10

#define EN_R 5
#define IN1_R 7
#define IN2_R 6

double w_r = 0, w_l = 0;

double wheel_rad = 0.0325, wheel_sep = 0.295;

ros::NodeHandle nh;
int lowSpeed = 200;
int highSpeed = 50;
double speed_ang = 0, speed_lin = 0;

void messageCb(const geometry_msgs::Twist&
```

```

msg){
    speed_ang = msg.angular.z;
    speed_lin = msg.linear.x;
    w_r = (speed_lin / wheel_rad) + ((speed_ang *
wheel_sep) / (2.0 * wheel_rad));
    w_l = (speed_lin / wheel_rad) - ((speed_ang *
wheel_sep) / (2.0 * wheel_rad));
}

ros::Subscriber<geometry_msgs::Twist>
sub("cmd_vel", &messageCb );
void Motors_init();
void MotorL(int Pulse_Width1);
void MotorR(int Pulse_Width2);

Servo servoMotor;
int servoPin = 11; // 서보모터의 핀 번호 (원하는
핀 번호로 변경)

void setup(){
    Motors_init();
    servoMotor.attach(servoPin); // 서보모터를 지
정된 핀에 연결
    nh.initNode();
    nh.subscribe(sub);
}

void loop(){
    MotorL(w_l * 4);
    MotorR(w_r * 4);

    int servoAngle = 150; // 서보모터의 초기 각도
(원하는 각도로 변경)
    if (speed_ang == 1.0) {
        servoAngle += 30; // 각도 변화량 (원하는 값
으로 변경)
    } else if (speed_ang == -1.0) {
        servoAngle -= 30; // 각도 변화량 (원하는 값
으로 변경)
    }
}

```

```

}

servoAngle = constrain(servoAngle, 120, 180);
servoMotor.write(servoAngle);

nh.spinOnce();
}

void Motors_init(){
    pinMode(EN_L, OUTPUT);
    pinMode(EN_R, OUTPUT);
    pinMode(IN1_L, OUTPUT);
    pinMode(IN2_L, OUTPUT);
    pinMode(IN1_R, OUTPUT);
    pinMode(IN2_R, OUTPUT);
    digitalWrite(EN_L, LOW);
    digitalWrite(EN_R, LOW);
    digitalWrite(IN1_L, LOW);
    digitalWrite(IN2_L, LOW);
    digitalWrite(IN1_R, LOW);
    digitalWrite(IN2_R, LOW);
}

void MotorL(int Pulse_Width1){
    if (Pulse_Width1 > 0){
        analogWrite(EN_L, Pulse_Width1);
        digitalWrite(IN1_L, HIGH);
        digitalWrite(IN2_L, LOW);
    }
    if (Pulse_Width1 < 0){
        Pulse_Width1=abs(Pulse_Width1);
        analogWrite(EN_L, Pulse_Width1);
        digitalWrite(IN1_L, LOW);
        digitalWrite(IN2_L, HIGH);
    }
    if (Pulse_Width1 == 0){
        analogWrite(EN_L, Pulse_Width1);
        digitalWrite(IN1_L, LOW);
        digitalWrite(IN2_L, LOW);
    }
}

```

```

    }
}

void MotorR(int Pulse_Width2){
    if (Pulse_Width2 > 0){
        analogWrite(EN_R, Pulse_Width2);
        digitalWrite(IN1_R, LOW);
        digitalWrite(IN2_R, HIGH);
    }
    if (Pulse_Width2 < 0){
        Pulse_Width2=abs(Pulse_Width2);
        analogWrite(EN_R, Pulse_Width2);
        digitalWrite(IN1_R, HIGH);
        digitalWrite(IN2_R, LOW);
    }
    if (Pulse_Width2 == 0){
        analogWrite(EN_R, Pulse_Width2);
        digitalWrite(IN1_R, LOW);
        digitalWrite(IN2_R, LOW);
    }
}
}

```

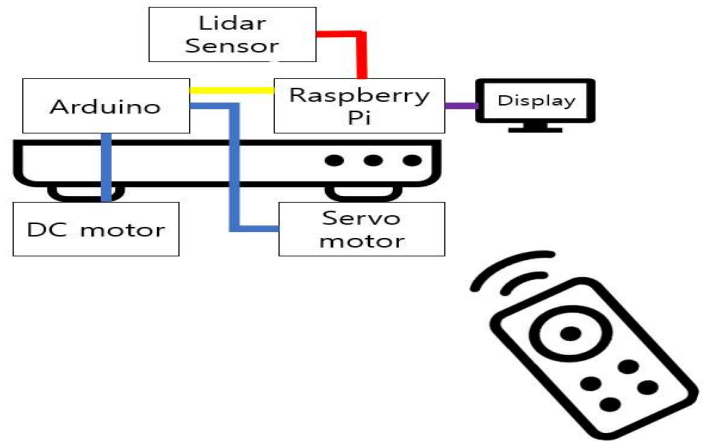
위 코드는 Arduino IDE에 작성한 코드이다. 로봇에 사용될 12V DC 모터, 아두이노의 최대 전압을 고려한 L298N 모터 드라이버, 방향 전환을 하기 위한 Servo 모터에 관한 코드이다.

여기서 모터의 초기 각도와 각도의 변화치, mapping 할 때 지도가 깨지지 않을 DC 모터 속도 등을 고려하여 수치를 조정하면서 코드를 완성한다.

7) Teleop Keyboard

우리 조는 로봇으로 mapping 하기 위한 프로그램으로 Teleop Keyboard를 선정하였다.

7.1 Teleop Keyboard 정의



Teleop Keyboard는 ROS 패키지 중 하나로 키보드 입력을 사용하여 로봇을 움직이는데 사용되는 간단한 teleoperation 도구이다. 이 패키지는 주로 시뮬레이션 환경에서 로봇을 조종하거나 로봇 개발 및 테스트 중에 편리하게 사용된다.

로봇을 수동으로 작동시킬 수 있는 teleop 키를 활용하여 로봇을 제어해서 mapping을 하려고 계획했다. Rviz에서 mapping을 하기 위해서는 하드웨어와 하드웨어를 움직이는 소프트웨어가 필요하다. 우리 조는 로봇을 수동으로 작동시킬 수 있는 teleop 키를 활용하여 로봇을 제어해서 mapping을 하려고 계획했다.

7.2 Teleop Keyboard 사용방법

Ubuntu에서 새 터미널 창을 연다.

```
source /opt/ros/neotic/setup.bash
```

roserial로 ros와 아두이노의 통신을 열어준다.

```
roslaunch roserial_python serial_node.py
_port:=/dev/ttyUSB0
```

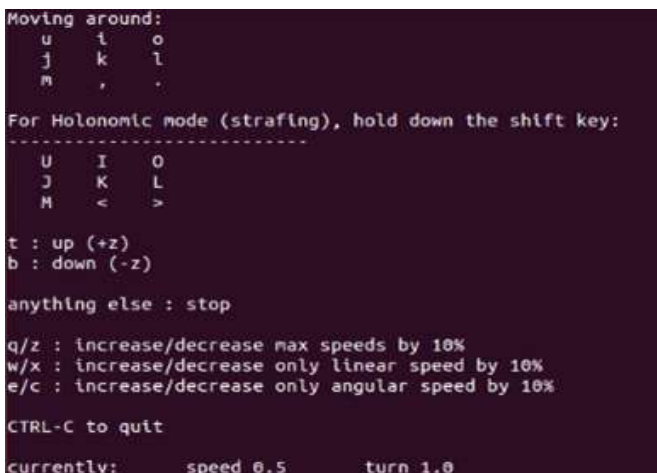
()안에 숫자는 포트가 꽂혀있는 번호를 작성해준다.

이후에 새 터미널 창을 열어서 teleop 패키지 설치

```
source /opt/ros/noetic/setup.bash
sudo apt-get install
ros-noetic-teleop_twist_keyboard
```

설치가 끝나고 teleop 실행

```
roslaunch teleop_twist_keyboard
teleop_twist_keyboard.py
```



Moving around:
u i o
j k l
m , .
For Holonomic mode (strafing), hold down the shift key:

U I O
J K L
M < >
t : up (+z)
b : down (-z)
anything else : stop
q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
CTRL-C to quit
currently: speed 0.5 turn 1.0

코드를 입력하면 다음과 같은 화면이 뜨고 키보드를 이용하여 전진, 후진, 좌회전, 우회전 등 작동이 가능해진다.

8) MAP

8.1 Map 제작

아두이노 코드 입력과 teleop key 설정까지 마친 후에 Ubuntu에서 여러 터미널에 각각의 명령어를 입력하여 Mapping을 진행한다.

터미널에서 rplidar a2를 실행시킨다.

```
sudo chmod a+rw /dev/ttyUSB0
cd ~/catkin_ws
source devel/setup.bash
roslaunch rplidar_ros rplidar_a2.launch
```

터미널에서 hector slam이 구현되는 rviz가 실행된다.

```
cd ~/catkin_ws
source devel/setup.bash
roslaunch hector_slam_launch tutorial_tif.launch
```

```
source /opt/ros/noetic/setup.bash
roscore
```

ros를 실행시킨다.

```
arduino
```

teleop key로 로봇이 움직이게 하기 위한 코드 작성을 위해 arduino를 실행한다.

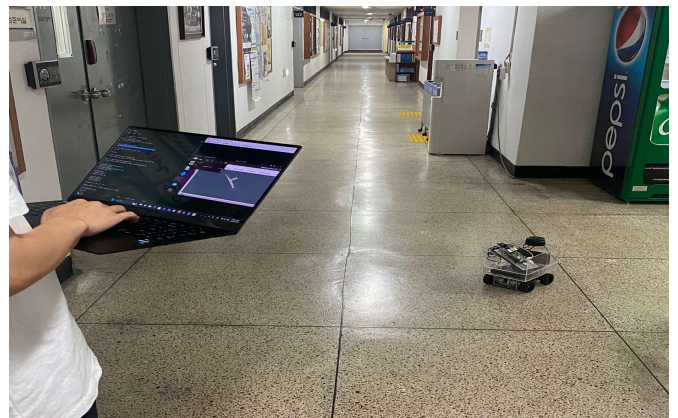
```
source /opt/ros/noetic/setup.bash
roslaunch rosserial_python serial_node.py
_port:=/dev/ttyUSB0
```

ros와 arduino가 통신되도록 설정해주는 터미널이다.

```
source /opt/ros/noetic/setup.bash
roslaunch teleop_twist_keyboard
teleop_twist_keyboard.py
```

teleop key를 실행시킬 수 있는 터미널이다.

위의 코드들이 각각의 터미널에서 입력되어 실행시키면 rviz에서 화면에서 lidar를 통해서 얻는 데이터를 확인할 수 있다. 그리고 teleop에서 나타내는 key를 사용하여 로봇을 움직여 지도를 만들어 완성시킨다.



이렇게 mapping 한 지도를 저장하기 위해서는 map_server를 설치해주어야 한다.

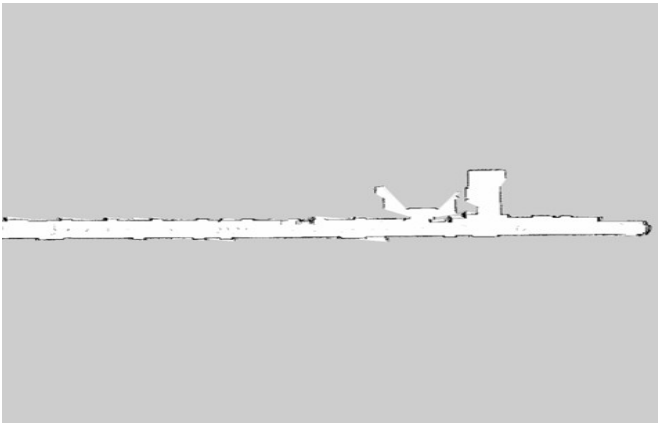
```
sudo apt install ros-noetic-map-server
```

그 후 새로운 터미널에서 완성된 지도를 저장하는 코드를 작성한다.

```
roslaunch map_server map_saver -f my_map
```

저장한 후 지도파일을 다시 열 수 있는 터미널이다.

```
roslaunch map_server map_server my_map.yaml
```



로봇을 움직여가며 완성시킨 맵을 저장하고 파일로 다시 열었을 때의 사진이다.

8.2 Map editor

Teleop Keyboard를 이용하여 mapping을 진행한 후에 mapping 한 Map에 D동 3층 각각의 연구실마다 호수를 지정해 주기 위해 Map을 편집해 주었다.

ROS에서 Map을 편집하기 위해서 사용한 프로그램은 gimp라는 프로그램이다.

gimp는 GNU Image Manipulation Program의 약자로 오픈 소스 이미지 편집 소프트웨어로 ROS와 함께 사용할 수 있다.

-gimp 사용 방법

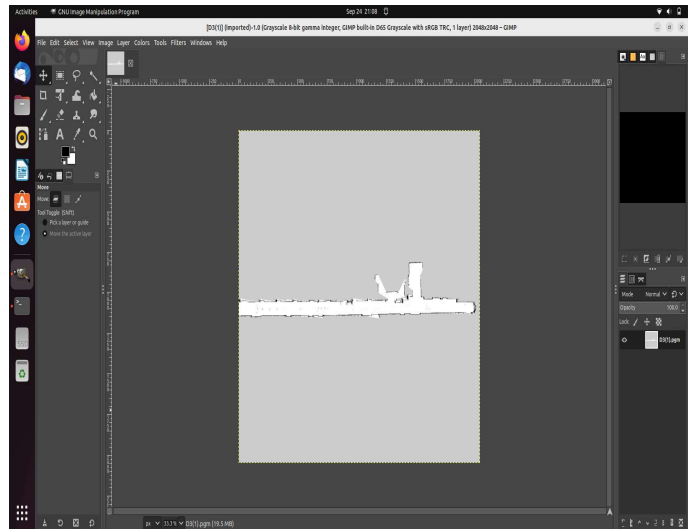
gimp에서 map을 열기 위해서는 파일 형식을 pgm에서 png로 변경해 주어야 한다. 그 과정을 진행하기 위해 imagemagick이라는 프로그램을 설치 해주고 map의 형식을 바꿔주었다.

```
sudo apt-get install imagemagick
convert my_map.pgm my_map.png
```

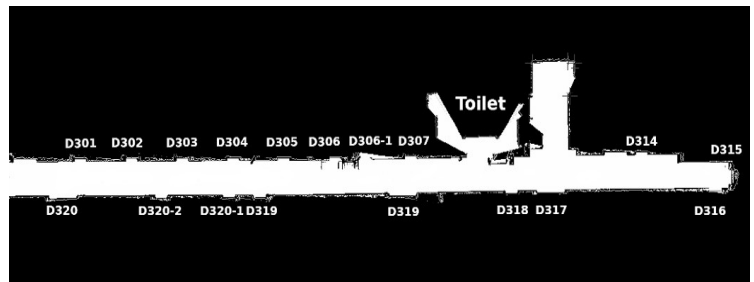
gimp 프로그램을 설치해주고 실행시킨다.

```
sudo apt-get update
sudo apt-get install gimp
gimp
```

gimp를 실행시킨 후 map을 열면 아래와 같이 나타나게 된다.



그 후 gimp에서 map을 편집해주었다. 아래는 map을 편집해주고 난 후의 map이다.



map을 편집하고 난 후 export 한 후 맵을 다시 png형식에서 pgm형식으로 바꿔주었다.

```
convert my_map.png my_map.pgm
```

그러면 이제 다시 Rviz에서 아래 코드를 통해 내가 새로 편집한 map을 열 수 있게 된다.

```
cd ~/catkin_ws/maps
roscore
```

내가 mapping한 map을 Rviz에서 열어준다.

```
roslaunch map_server map_server my_map.yaml
rviz
```

9) Odometry

9.1 tick 수 측정

지도에서 로봇의 위치를 파악하기 위해서 Wheel Encoder를 사용합니다. 먼저 우리가 사용하는 DC 모터의 Wheel ticks 수를 먼저 측정한다.

```
// Encoder output to Arduino Interrupt pin
#define ENC_IN_RIGHT_A 2

// Keep track of the number of right wheel pulses
volatile long right_wheel_pulse_count = 0;

void setup() {
    // Open the serial port at 9600 bps
    Serial.begin(9600);

    // Set pin states of the encoder
    pinMode(ENC_IN_RIGHT_A, INPUT_PULLUP);

    // Every time the pin goes high, this interrupt will be triggered
    attachInterrupt(digitalPinToInterrupt(ENC_IN_RIGHT_A), right_wheel_pulse_count++, RISING);
}

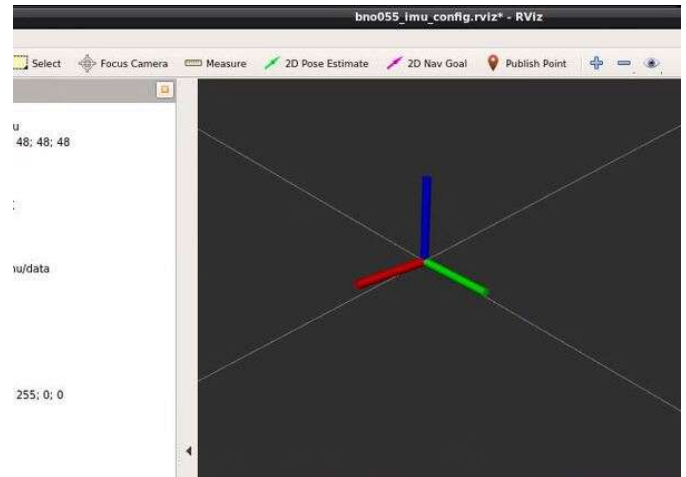
void loop() {
    Serial.print(" Pulses: ");
    Serial.println(right_wheel_pulse_count);
}
```

아두이노 코드 작성 이후 모터를 한 바퀴 돌린 결과를 위 사진으로 모터 당 391의 ticks가 나오는 것을 확인할 수 있다. 이를 바탕으로 Tick Publisher Node를 만들어주고 Launch file도 생성해 준다.

9.2 IMU Sensor

위의 Encoder를 사용한 이유와 동일하게 위치의 정확성을 더욱 향상시키기 위해서 IMU Sensor를 사용한다.

IMU Sensor란 물체가 기울어진 각도를 정확하게 측정해 주는 장치이다. 이를 통해 Rviz에서 각속도를 측정하고 시간 당 몇 도를 회전하는지 알 수 있다. 이를 토대로 로봇의 전진뿐 아니라 좌우 회전에 대한 정확한 위치표기가 가능해진다.



9.3 EKF package를 사용한 Sensor Fusion

Wheel Encoder와 IMU Sensor를 활용한 데이터에 EKF package를 추가하여 보다 정확한 데이터를 얻는다. ROS 터미널에 Robot_pose_ekf Package를 설치하고 이 Package node를 실행하기 위한 launch file에 추가한다.

10) Navigation

10.1 아두이노 모터 제어

먼저 Navigation 이란 위의 과정을 통해 작성한 map을 rviz에 나타내고 2D Pose Estimate로 로봇의 초기 위치를 설정해 준 뒤 2D Nav Goal을 찍어주면 최단 경로를 생성하며 원하는 지점으로 이동하게 해주는 알고리즘이다. 그러기 위해서는 먼저 아두이노의 휠 인코더 모터를 제어해야 한다.

위에서 휠 인코더의 적수를 게시한 데이터를 바탕으로 아두이노를 ROS 노드와 연결된 코드를 작성해 준다.

```
roscore
```

```
arduino
```

```
#include <ros.h>
#include <std_msgs/Int16.h>
#include <geometry_msgs/Twist.h>
// Handles startup and shutdown of ROS
ros::NodeHandle nh;

//////////////////// Tick Data Publishing Variables
and Constants //////////////////

// Encoder output to Arduino Interrupt pin.
Tracks the tick count.
#define ENC_IN_LEFT_A 2
#define ENC_IN_RIGHT_A 3

// Other encoder output to Arduino to keep
track of wheel direction
// Tracks the direction of rotation.
#define ENC_IN_LEFT_B 4
#define ENC_IN_RIGHT_B 11

// True = Forward; False = Reverse
boolean Direction_left = true;
boolean Direction_right = true;

// Minumum and maximum values for 16-bit
integers
// Range of 65,535
const int encoder_minimum = -32768;
const int encoder_maximum = 32767;

// Keep track of the number of wheel ticks
std_msgs::Int16 right_wheel_tick_count;
ros::Publisher rightPub("right_ticks",
&right_wheel_tick_count);

std_msgs::Int16 left_wheel_tick_count;
ros::Publisher leftPub("left_ticks",
&left_wheel_tick_count);
```

```
// Time interval for measurements in milliseconds
const int interval = 30;
long previousMillis = 0;
long currentMillis = 0;

//////////////////// Motor Controller Variables and
Constants //////////////////

// Motor A connections
const int enA = 9;
const int in1 = 5;
const int in2 = 6;

// Motor B connections
const int enB = 10;
const int in3 = 7;
const int in4 = 8;

// How much the PWM value can change each
cycle
const int PWM_INCREMENT = 1;

// Number of ticks per wheel revolution. We
won't use this in this code.
const int TICKS_PER_REVOLUTION = 390;

// Wheel radius in meters
const double WHEEL_RADIUS = 0.033;

// Distance from center of the left tire to the
center of the right tire in m
const double WHEEL_BASE = 0.16;

// Number of ticks a wheel makes moving a
linear distance of 1 meter
// This value was measured manually.
const double TICKS_PER_METER = 1950; //
Originally 2880
```

```
// Proportional constant, which was measured by
measuring the
// PWM-Linear Velocity relationship for the
robot.
const int K_P = 278;

// Y-intercept for the PWM-Linear Velocity
relationship for the robot
const int b = 52;

// Correction multiplier for drift. Chosen through
experimentation.
const int DRIFT_MULTIPLIER = 120;

// Turning PWM output (0 = min, 255 = max for
PWM values)
const int PWM_TURN = 80;

// Set maximum and minimum limits for the
PWM values
const int PWM_MIN = 80; // about 0.1 m/s
const int PWM_MAX = 100; // about 0.172 m/s

// Set linear velocity and PWM variable values for
each wheel
double velLeftWheel = 0;
double velRightWheel = 0;
double pwmLeftReq = 0;
double pwmRightReq = 0;

// Record the time that the last velocity
command was received
double lastCmdVelReceived = 0;

////////// Tick Data Publishing
Functions //////////

// Increment the number of ticks
void right_wheel_tick() {
```

```
// Read the value for the encoder for the right
wheel
int val = digitalRead(ENC_IN_RIGHT_B);

if (val == LOW) {
    Direction_right = false; // Reverse
}
else {
    Direction_right = true; // Forward
}

if (Direction_right) {

    if (right_wheel_tick_count.data ==
encoder_maximum) {
        right_wheel_tick_count.data =
encoder_minimum;
    }
    else {
        right_wheel_tick_count.data++;
    }
}
else {
    if (right_wheel_tick_count.data ==
encoder_minimum) {
        right_wheel_tick_count.data =
encoder_maximum;
    }
    else {
        right_wheel_tick_count.data--;
    }
}
}

// Increment the number of ticks
void left_wheel_tick() {

    // Read the value for the encoder for the left
```

```
wheel
  int val = digitalRead(ENC_IN_LEFT_B);

  if (val == LOW) {
    Direction_left = true; // Reverse
  }
  else {
    Direction_left = false; // Forward
  }

  if (Direction_left) {
    if (left_wheel_tick_count.data ==
encoder_maximum) {
      left_wheel_tick_count.data =
encoder_minimum;
    }
    else {
      left_wheel_tick_count.data++;
    }
  }
  else {
    if (left_wheel_tick_count.data ==
encoder_minimum) {
      left_wheel_tick_count.data =
encoder_maximum;
    }
    else {
      left_wheel_tick_count.data--;
    }
  }
}

////////// Motor Controller Functions
//////////

// Calculate the left wheel linear velocity in m/s
every time a
// tick count message is rpublished on the
/left_ticks topic.
```

```
void calc_vel_left_wheel(){

  // Previous timestamp
  static double prevTime = 0;

  // Variable gets created and initialized the first
time a function is called.
  static int prevLeftCount = 0;

  // Manage rollover and rollunder when we get
outside the 16-bit integer range
  int numOfTicks = (65535 +
left_wheel_tick_count.data - prevLeftCount) %
65535;

  // If we have had a big jump, it means the tick
count has rolled over.
  if (numOfTicks > 10000) {
    numOfTicks = 0 - (65535 - numOfTicks);
  }

  // Calculate wheel velocity in meters per
second
  velLeftWheel =
numOfTicks/TICKS_PER_METER/((millis()/1000)-pre
vTime);

  // Keep track of the previous tick count
prevLeftCount = left_wheel_tick_count.data;

  // Update the timestamp
prevTime = (millis()/1000);
}

// Calculate the right wheel linear velocity in m/s
every time a
// tick count message is published on the
/right_ticks topic.
```

```
void calc_vel_right_wheel(){

    // Previous timestamp
    static double prevTime = 0;

    // Variable gets created and initialized the first
    time a function is called.
    static int prevRightCount = 0;

    // Manage rollover and rollunder when we get
    outside the 16-bit integer range
    int numOfTicks = (65535 +
    right_wheel_tick_count.data - prevRightCount) %
    65535;

    if (numOfTicks > 10000) {
        numOfTicks = 0 - (65535 - numOfTicks);
    }

    // Calculate wheel velocity in meters per
    second
    velRightWheel =
    numOfTicks/TICKS_PER_METER/((millis()/1000)-pre
    vTime);

    prevRightCount = right_wheel_tick_count.data;

    prevTime = (millis()/1000);

}

// Take the velocity command as input and
calculate the PWM values.
void calc_pwm_values(const
geometry_msgs::Twist& cmdVel) {

    // Record timestamp of last velocity command
    received
    lastCmdVelReceived = (millis()/1000);
```

```
    // Calculate the PWM value given the desired
    velocity
    pwmLeftReq = K_P * cmdVel.linear.x + b;
    pwmRightReq = K_P * cmdVel.linear.x + b;

    // Check if we need to turn
    if (cmdVel.angular.z != 0.0) {

        // Turn left
        if (cmdVel.angular.z > 0.0) {
            pwmLeftReq = -PWM_TURN;
            pwmRightReq = PWM_TURN;
        }
        // Turn right
        else {
            pwmLeftReq = PWM_TURN;
            pwmRightReq = -PWM_TURN;
        }
    }
    // Go straight
    else {

        // Remove any differences in wheel velocities
        // to make sure the robot goes straight
        static double prevDiff = 0;
        static double prevPrevDiff = 0;
        double currDifference = velLeftWheel -
        velRightWheel;
        double avgDifference =
        (prevDiff+prevPrevDiff+currDifference)/3;
        prevPrevDiff = prevDiff;
        prevDiff = currDifference;

        // Correct PWM values of both wheels to
        make the vehicle go straight
        pwmLeftReq -= (int)(avgDifference *
        DRIFT_MULTIPLIER);
        pwmRightReq += (int)(avgDifference *
```

```

DRIFT_MULTIPLIER);
}

// Handle low PWM values
if (abs(pwmLeftReq) < PWM_MIN) {
    pwmLeftReq = 0;
}
if (abs(pwmRightReq) < PWM_MIN) {
    pwmRightReq = 0;
}
}

void set_pwm_values() {

    // These variables will hold our desired PWM
    values
    static int pwmLeftOut = 0;
    static int pwmRightOut = 0;

    // If the required PWM is of opposite sign as
    the output PWM, we want to
    // stop the car before switching direction
    static bool stopped = false;
    if ((pwmLeftReq * velLeftWheel < 0 &&
    pwmLeftOut != 0) ||
        (pwmRightReq * velRightWheel < 0 &&
    pwmRightOut != 0)) {
        pwmLeftReq = 0;
        pwmRightReq = 0;
    }

    // Set the direction of the motors
    if (pwmLeftReq > 0) { // Left wheel forward
        digitalWrite(in1, HIGH);
        digitalWrite(in2, LOW);
    }
    else if (pwmLeftReq < 0) { // Left wheel reverse
        digitalWrite(in1, LOW);
        digitalWrite(in2, HIGH);
    }
}

```

```

}
else if (pwmLeftReq == 0 && pwmLeftOut ==
0 ) { // Left wheel stop
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
}
else { // Left wheel stop
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
}

if (pwmRightReq > 0) { // Right wheel forward
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
}
else if(pwmRightReq < 0) { // Right wheel
reverse
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);
}
else if (pwmRightReq == 0 && pwmRightOut
== 0) { // Right wheel stop
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);
}
else { // Right wheel stop
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);
}

// Increase the required PWM if the robot is
not moving
if (pwmLeftReq != 0 && velLeftWheel == 0) {
    pwmLeftReq *= 1.5;
}
if (pwmRightReq != 0 && velRightWheel == 0)
{
    pwmRightReq *= 1.5;
}
}

```

```
// Calculate the output PWM value by making
slow changes to the current value
if (abs(pwmLeftReq) > pwmLeftOut) {
    pwmLeftOut += PWM_INCREMENT;
}
else if (abs(pwmLeftReq) < pwmLeftOut) {
    pwmLeftOut -= PWM_INCREMENT;
}
else{}

if (abs(pwmRightReq) > pwmRightOut) {
    pwmRightOut += PWM_INCREMENT;
}
else if (abs(pwmRightReq) < pwmRightOut) {
    pwmRightOut -= PWM_INCREMENT;
}
else{}

// Conditional operator to limit PWM output at
the maximum
pwmLeftOut = (pwmLeftOut > PWM_MAX) ?
PWM_MAX : pwmLeftOut;
pwmRightOut = (pwmRightOut > PWM_MAX)
? PWM_MAX : pwmRightOut;

// PWM output cannot be less than 0
pwmLeftOut = (pwmLeftOut < 0) ? 0 :
pwmLeftOut;
pwmRightOut = (pwmRightOut < 0) ? 0 :
pwmRightOut;

// Set the PWM value on the pins
analogWrite(enA, pwmLeftOut);
analogWrite(enB, pwmRightOut);
}

// Set up ROS subscriber to the velocity
command
```

```
ros::Subscriber<geometry_msgs::Twist>
subCmdVel("cmd_vel", &calc_pwm_values );

void setup() {

    // Set pin states of the encoder
    pinMode(ENC_IN_LEFT_A , INPUT_PULLUP);
    pinMode(ENC_IN_LEFT_B , INPUT);
    pinMode(ENC_IN_RIGHT_A , INPUT_PULLUP);
    pinMode(ENC_IN_RIGHT_B , INPUT);

    // Every time the pin goes high, this is a tick

    attachInterrupt(digitalPinToInterrupt(ENC_IN_LEFT
_A), left_wheel_tick, RISING);

    attachInterrupt(digitalPinToInterrupt(ENC_IN_RIGH
T_A), right_wheel_tick, RISING);

    // Motor control pins are outputs
    pinMode(enA, OUTPUT);
    pinMode(enB, OUTPUT);
    pinMode(in1, OUTPUT);
    pinMode(in2, OUTPUT);
    pinMode(in3, OUTPUT);
    pinMode(in4, OUTPUT);

    // Turn off motors - Initial state
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);

    // Set the motor speed
    analogWrite(enA, 0);
    analogWrite(enB, 0);

    // ROS Setup
    nh.getHardware()->setBaud(115200);
```

```

nh.initNode();
nh.advertise(rightPub);
nh.advertise(leftPub);
nh.subscribe(subCmdVel);
}

void loop() {

    nh.spinOnce();

    // Record the time
    currentMillis = millis();

    // If the time interval has passed, publish the
    number of ticks,
    // and calculate the velocities.
    if (currentMillis - previousMillis > interval) {

        previousMillis = currentMillis;

        // Publish tick counts to topics
        leftPub.publish( &left_wheel_tick_count );
        rightPub.publish( &right_wheel_tick_count );

        // Calculate the velocity of the right and left
        wheels
        calc_vel_right_wheel();
        calc_vel_left_wheel();

    }

    // Stop the car if there are no cmd_vel
    messages
    if((millis()/1000) - lastCmdVelReceived > 1) {
        pwmLeftReq = 0;
        pwmRightReq = 0;
    }

    set_pwm_values(); }

```

위의 코드를 Arduino ide에 업로드를 해준다. 이때 메모리 부족 경고가 발생하지만 100%를 모두 사용하는 것은 아니기 때문에 무시해도 된다.

10.2 Initial Pose and Goal Publisher

10.1에서 Navigation의 간단한 알고리즘 중에서 로봇의 초기 위치와 원하는 목적지를 설정하는 방법에 대한 설명이다.

-2D Pose Estimate

게시된 topic은 initialpose이며 유형은 geometry_msgs/PoseWithCovarianceStamped 이다. 이 버튼의 기능은 로봇의 자세를 설정하여 위치를 파악하는 시스템을 초기화해준다. 즉, 초기 위치를 지정해주는 역할을 한다.

-2D Nav Goal

게시된 topic은 move_base_simple/goal이며 유형은 geometry_msgs/PoseStamped 이다. 기능적으로는 로봇이 최종적으로 이동할 위치 값을 설정하여 그 목표로 보낼 수 있다.

이제 Rviz에서 위의 기능들을 실행시켜주기 위한 코드를 C++ 코드를 localization_data_pub 파일에 작성해준다.

```

CD ~/catkin_ws/src/jetson_nano_bot/
localization_data_pub /src

```

```

gedit rviz_click_to_2d.cpp

```

```

#include "ros/ros.h"
#include "geometry_msgs/PoseStamped.h"
#include
"geometry_msgs/PoseWithCovarianceStamped.h"
#include <tf/transform_broadcaster.h>
#include <iostream>

using namespace std;

```



```
// Initialize ROS publishers
ros::Publisher pub;
ros::Publisher pub2;

// Take move_base_simple/goal as input and
publish goal_2d
void handle_goal(const
geometry_msgs::PoseStamped &goal) {
    geometry_msgs::PoseStamped rpyGoal;
    rpyGoal.header.frame_id = "map";
    rpyGoal.header.stamp = goal.header.stamp;
    rpyGoal.pose.position.x = goal.pose.position.x;
    rpyGoal.pose.position.y = goal.pose.position.y;
    rpyGoal.pose.position.z = 0;
    tf::Quaternion q(0, 0, goal.pose.orientation.z,
goal.pose.orientation.w);
    tf::Matrix3x3 m(q);
    double roll, pitch, yaw;
    m.getRPY(roll, pitch, yaw);
    rpyGoal.pose.orientation.x = 0;
    rpyGoal.pose.orientation.y = 0;
    rpyGoal.pose.orientation.z = yaw;
    rpyGoal.pose.orientation.w = 0;
    pub.publish(rpyGoal);
}

// Take initialpose as input and publish initial_2d
void handle_initial_pose(const
geometry_msgs::PoseWithCovarianceStamped
&pose) {
    geometry_msgs::PoseStamped rpyPose;
    rpyPose.header.frame_id = "map";
    rpyPose.header.stamp = pose.header.stamp;
    rpyPose.pose.position.x =
pose.pose.pose.position.x;
    rpyPose.pose.position.y =
pose.pose.pose.position.y;
    rpyPose.pose.position.z = 0;
```

```
tf::Quaternion q(0, 0,
pose.pose.pose.orientation.z,
pose.pose.pose.orientation.w);
    tf::Matrix3x3 m(q);
    double roll, pitch, yaw;
    m.getRPY(roll, pitch, yaw);
    rpyPose.pose.orientation.x = 0;
    rpyPose.pose.orientation.y = 0;
    rpyPose.pose.orientation.z = yaw;
    rpyPose.pose.orientation.w = 0;
    pub2.publish(rpyPose);
}

int main(int argc, char **argv) {
    ros::init(argc, argv, "rviz_click_to_2d");
    ros::NodeHandle node;
    pub =
node.advertise<geometry_msgs::PoseStamped>("
goal_2d", 0);
    pub2 =
node.advertise<geometry_msgs::PoseStamped>("i
nitial_2d", 0);
    ros::Subscriber sub =
node.subscribe("move_base_simple/goal", 0,
handle_goal);
    ros::Subscriber sub2 =
node.subscribe("initialpose", 0,
handle_initial_pose);
    ros::Rate loop_rate(10);
    while (ros::ok()) {
        ros::spinOnce();
        loop_rate.sleep();
    }
    return 0;
}
```

10.3 Costmap 구성

ROS Navigation Stack은 세 개의 costmap을 사용하여 맵의 장애물에 대한 정보를 저장한다.

- Global costmap : 지도 환경에 대한 경로 계획을 생성하는 데 사용된다. 지도의 초기 좌표와 최종 좌표까지의 경로를 최단 거리로 계산한다.
- Local costmap : 장애물을 피하기 위해 맵의 환경에 대한 단기 계획을 생성한다.
- Common Configuration : Global costmap과 Local costmap의 공통적인 매개변수를 포함하는 파일을 작성한다.
- Base Local Planner : 로봇 기본 컨트롤러로 전송되는 속도 명령을 계산한다.

10.4 Navigation Stack

먼저 터미널을 열어 ROS Navigation Stack을 설치한다.

```
sudo apt-get install ros-noetic-navigation
```

navstack_pub 패키지를 만들어 준다.

```
catkin_create_pkg navstack_pub roscpp
std_msgs tf tf2_ros geometry_msgs sensor_msgs
nav_msgs move_base
```

param 파일을 만들어 준다.

```
mkdir param
```

CMakeLists.txt 파일을 열어 C++ 지원을 활성화하기 위해 앞에 있는 해시태그를 제거해 준다.

이제 지금까지 진행한 Navigation 노드들에 대하여 하나의 Launch 파일을 작성해 준다.

-변환 구성

-센서 정보(장애물을 센서 정보로 환경의 장애물을 피할 수 있게 해준다.)

-라이다 정보

-주행 정보(휠 엔코더, IMU, ekf)

-기본 컨트롤러(속도 명령)

-매핑 정보

-costmap 구성

-공통 구성

-Base Local Planner

-move base node

-AMCL 구성(확률적 위치 파악 시스템)

이에 대한 Launch 파일은 아래와 같다.

```
<launch>
  <!-- Transformation Configuration ... Setting
Up the Relationships Between Coordinate Frames
-->
  <node pkg="tf"
type="static_transform_publisher"
name="base_link_to_laser" args="0.06 0 0.08 0 0
0 base_link laser 30" />
  <node pkg="tf"
type="static_transform_publisher"
name="imu_broadcaster" args="0 0.06 0.02 0 0 0
base_link imu 30" />
  <node pkg="tf"
type="static_transform_publisher"
name="base_link_broadcaster" args="0 0 0.09 0 0
0 base_footprint base_link 30" />
  <!-- odom to base_footprint transform will be
provided by the robot_pose_ekf node -->
  <!-- map to odom will be provided by the
AMCL -->
  <node pkg="tf"
type="static_transform_publisher"
name="map_to_odom" args="0 0 0 0 0 0 map
odom 30" />
  <!-- Wheel Encoder Tick Publisher and Base
Controller Using Arduino -->
  <!-- motor_controller_diff_drive_2.ino is the
Arduino sketch -->
```

```

<!-- Subscribe: /cmd_vel -->
<!-- Publish: /right_ticks, /left_ticks -->
<node pkg="roscpp" name="serial_node"
type="serial_node.py" name="serial_node">
  <param name="port"
value="/dev/ttyUSB0"/>
  <param name="baud" value="115200"/>
</node>
<!-- Wheel Odometry Publisher -->
<!-- Subscribe: /right_ticks, /left_ticks,
/initial_2d -->
<!-- Publish: /odom_data_euler,
/odom_data_quat -->
<node pkg="localization_data_pub"
type="ekf_odom_pub" name="ekf_odom_pub">
</node>

<!-- IMU Data Publisher Using the
BNO055 IMU Sensor -->
<!-- Publish: /imu/data -->
<node ns="imu" name="imu_node"
pkg="imu_MPU6050" type="MPU6050_i2c_node"
respawn="true" respawn_delay="2">
  <param name="device" type="string"
value="/dev/i2c-1"/>
  <param name="address" type="int"
value="40"/> <!-- 0x28 == 40 is the default -->
  <param name="frame_id" type="string"
value="imu"/>
</node>

<!-- Extended Kalman Filter from
robot_pose_ekf Node-->
<!-- Subscribe: /odom, /imu_data, /vo -->
<!-- Publish: /robot_pose_ekf/odom_combined
-->
<remap from="odom" to="odom_data_quat"
/>
<remap from="imu_data" to="imu/data" />
<node pkg="robot_pose_ekf"
type="robot_pose_ekf" name="robot_pose_ekf">

```

```

  <param name="output_frame"
value="odom"/>
  <param name="base_footprint_frame"
value="base_footprint"/>
  <param name="freq" value="30.0"/>
  <param name="sensor_timeout"
value="1.0"/>
  <param name="odom_used" value="true"/>
  <param name="imu_used" value="true"/>
  <param name="vo_used" value="false"/>
  <param name="gps_used" value="false"/>
  <param name="debug" value="false"/>
  <param name="self_diagnose"
value="false"/>
</node>

<!-- Initial Pose and Goal Publisher
-->
<!-- Publish: /initialpose,
/move_base_simple/goal -->
<node pkg="rviz" type="rviz" name="rviz"
args="-d
/home/automaticaddison/catkin_ws/src/SDR/navigation_data_pub/maps/D3.rviz">
</node>
<!-- Subscribe: /initialpose,
/move_base_simple/goal -->
<!-- Publish: /initial_2d, /goal_2d -->
<node pkg="localization_data_pub"
type="rviz_click_to_2d" name="rviz_click_to_2d">
</node>

<!-- Lidar Data Publisher Using RPLIDAR from
Slamtec -->
<!-- Used for obstacle avoidance and can be
used for mapping -->
<!-- Publish: /scan -->
<node name="rplidarNode"
pkg="rplidar_ros" type="rplidarNode"
output="screen">
  <param name="serial_port"

```

```

type="string" value="/dev/ttyUSB1"/>
  <!--param name="serial_baudrate"
type="int" value="115200"/><!--A1/A2 -->
  <param name="serial_baudrate" type="int"
value="256000"--><!--A3 -->
  <param name="frame_id"
type="string" value="laser"/>
  <param name="inverted"
type="bool" value="false"/>
  <param name="angle_compensate"
type="bool" value="true"/>
</node>
<!-- Map File -->
<arg name="map_file" default="$(find
navigation_data_pub)/maps/D3.yaml"/>
  <!-- Map Server -->
  <!-- Publish: /map, /map_metadata -->
  <node pkg="map_server" name="map_server"
type="map_server" args="$(arg map_file)" />
  <!-- Add AMCL example for
differential drive robots for Localization -->
  <!-- Subscribe: /scan, /tf, /initialpose, /map -->
  <!-- Publish: /amcl_pose, /particlecloud, /tf -->
  <include file="$(find
amcl)/examples/amcl_diff.launch"/>
  <!-- Move Base Node -->
  <!-- Subscribe: /move_base_simple/goal -->
  <!-- Publish: /cmd_vel -->
  <node pkg="move_base" type="move_base"
respawn="false" name="move_base"
output="screen">
    <rosparam file="$(find
navstack_pub)/param/costmap_common_params.
yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find
navstack_pub)/param/costmap_common_params.
yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find
navstack_pub)/param/local_costmap_params.yaml

```

```

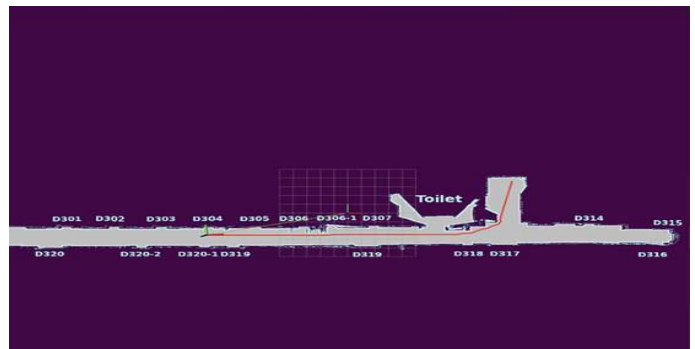
" command="load" ns="local_costmap" />
  <rosparam file="$(find
navstack_pub)/param/global_costmap_params.ya
ml" command="load" ns="global_costmap" />
  <rosparam file="$(find
navstack_pub)/param/base_local_planner_params.
yaml" command="load" />
</node>
</launch>

```

이제 Navigation을 실행 시켜준다.

```
roslaunch navstack_pub SDR_bot.launch
```

RViz 상단의 2D Pos Estimate 버튼을 클릭한 후 지도를 클릭하여 로봇의 초기 자세를 설정한 후, 2D Nav Goal 버튼을 클릭한 다음 지도를 클릭하여 로봇에게 목표를 부여한다.



지도에 자동으로 그려진 계획 경로가 표시된다. 그러면 로봇이 이 경로를 따르기 시작한다.

11) Send goals

로봇을 원하는 위치로 이동시키기 위해 앞서 말했듯이 지도에서 위치를 지정해 주어 경로를 생성해 원하는 위치로 이동하는 것을 구현하였다. 여기서 더 나아가 터미널에 D317과 같은 목표 위치 즉 호수를 입력해 주었을 때 로봇이 목표 위치로 이동하게 하는 방법도 구현하였다.

11.1 구현 방법

터미널 창에 SDR_bot launch 파일을 실행시켜준다.

```
roslaunch navstack_pub SDR_bot.launch
```

Rviz에서 Point Publish 버튼을 사용하여 원하는 각 목표 위치의 X 및 Y 좌표 값을 측정해 준다. 터미널에 다음 명령을 입력하여 좌표 값을 측정해 준다.

```
ros topic echo /clicked_point
```

1 = D304

2 = D306

3 = D316

4 = D319

위와 같이 각각의 강의실과 연구실의 호수를 숫자로 지정해 주었다. 그러면 터미널 창에 지정한 숫자를 입력해 주면 로봇이 해당 위치로 이동하게 된다.

터미널 창을 열어 C++ 코드를 작성해 준다.

```
roscd navstack_pub
cd src
gedit send_goals.cpp
```

```
#include <ros/ros.h>
#include <move_base_msgs/MoveBaseAction.h>
#include <actionlib/client/simple_action_client.h>
#include <iostream>
```

```
using namespace std;
```

```
// Action specification for move_base
typedef
actionlib::SimpleActionClient<move_base_msgs::
MoveBaseAction> MoveBaseClient;
```

```
int main(int argc, char** argv){
```

```
// Connect to ROS
```

```
ros::init(argc, argv, "simple_navigation_goals");
```

```
//tell the action client that we want to spin a
thread by default
```

```
MoveBaseClient ac("move_base", true);
```

```
// Wait for the action server to come up so
that we can begin processing goals.
```

```
while(!ac.waitForServer(ros::Duration(5.0))){
```

```
    ROS_INFO("Waiting for the move_base
action server to come up"); }
```

```
int user_choice = 4;
```

```
char choice_to_continue = 'Y';
```

```
bool run = true;
```

```
while(run) {
```

```
// Ask the user where he wants the robot to go?
```

```
    cout << "\nWhere do you want the robot
to go?" << endl;
```

```
    cout << "\n1 = D304" << endl;
```

```
    cout << "2 = D306" << endl;
```

```
    cout << "3 = D316" << endl;
```

```
    cout << "4 = D319" << endl;
```

```
    cout << "\nEnter a number: ";
```

```
    cin >> user_choice;
```

```
// Create a new goal to send to move_base
move_base_msgs::MoveBaseGoal goal;
```

```
// Send a goal to the robot
```

```
goal.target_pose.header.frame_id = "map";
```

```
goal.target_pose.header.stamp =
```

```
ros::Time::now();
```

```
bool valid_selection = true;
```

```
// Use map_server to load the map of the
environment on the /map topic.
```

```
// Launch RViz and click the Publish Point
```

```

button in RViz to
    // display the coordinates to the
    /clicked_point topic.
    switch (user_choice) {
        case 1:
            cout << "\nGoal Location: D304\n" <<
endl;
            goal.target_pose.pose.position.x = 10.0;
            goal.target_pose.pose.position.y = 3.7;
            goal.target_pose.pose.orientation.w =
1.0;
            break;
        case 2:
            cout << "\nGoal Location: D306\n" <<
endl;
            goal.target_pose.pose.position.x = 8.1;
            goal.target_pose.pose.position.y = 4.3;
            goal.target_pose.pose.orientation.w =
1.0;
            break;
        case 3:
            cout << "\nGoal Location: D316\n" <<
endl;
            goal.target_pose.pose.position.x = 10.5;
            goal.target_pose.pose.position.y = 2.0;
            goal.target_pose.pose.orientation.w =
1.0;
            break;
        case 4:
            cout << "\nGoal Location: D319\n" <<
endl;
            goal.target_pose.pose.position.x = 5.3;
            goal.target_pose.pose.position.y = 2.7;
            goal.target_pose.pose.orientation.w =
1.0;
            break;

        default:
            cout << "\nInvalid selection. Please try

```

```

again.\n" << endl;
        valid_selection = false;
    }

    // Go back to beginning if the selection is
    invalid.
    if(!valid_selection) {
        continue;
    }

    ROS_INFO("Sending goal");
    ac.sendGoal(goal);
    // Wait until the robot reaches the goal
    ac.waitForResult();
    if(ac.getState() ==
actionlib::SimpleClientGoalState::SUCCEEDED)
        ROS_INFO("The robot has arrived at the
goal location");
    else
        ROS_INFO("The robot failed to reach the
goal location for some reason");

    // Ask the user if he wants to continue
    giving goals
    do {
        cout << "\nWould you like to go to
another destination? (Y/N)" << endl;
        cin >> choice_to_continue;
        choice_to_continue =
tolower(choice_to_continue); // Put your letter to
its lower case
    } while (choice_to_continue != 'n' &&
choice_to_continue != 'y');

    if(choice_to_continue == 'n') {
        run = false;
    }
}
return 0; }

```

파일을 저장해 주고 난 후 새 터미널 창에 아래 명령어를 입력해 주어 파일을 열어준다.

```
roscd navstack_pub
gedit CMakeLists.txt
```

그 후 파일에 아래의 명령어를 추가해 준다.

```
INCLUDE_DIRECTORIES(/usr/local/lib)
LINK_DIRECTORIES(/usr/local/lib)

add_executable(send_goals src/send_goals.cpp)
target_link_libraries(send_goals
${catkin_LIBRARIES})
```

그 후 패키지를 컴파일 하고 노드를 실행시켜준다.

```
CD ~/catkin_ws/
catkin_make --only-pkg-with-deps navstack_pub
roslaunch navstack_pub SDR_bot.launch
roslaunch navstack_pub send_goals
```

```
jangwon@jangwon-950XED: ~
jangwon@jangwon-950XED:~$ roslaunch navstack_pub send_goals

Where do you want the robot to go?

1 = D304
2 = D306
3 = D316
4 = D319

Enter a number: 3

Goal Location: D316

[ INFO ] [1698249422.389507886]: Sending goal
```

노드를 실행 시켜주면 위치와 번호가 함께 뜨게 되고 Enter a number에 원하는 위치의 숫자를 써주면 그 위치로 로봇이 이동하게 된다.

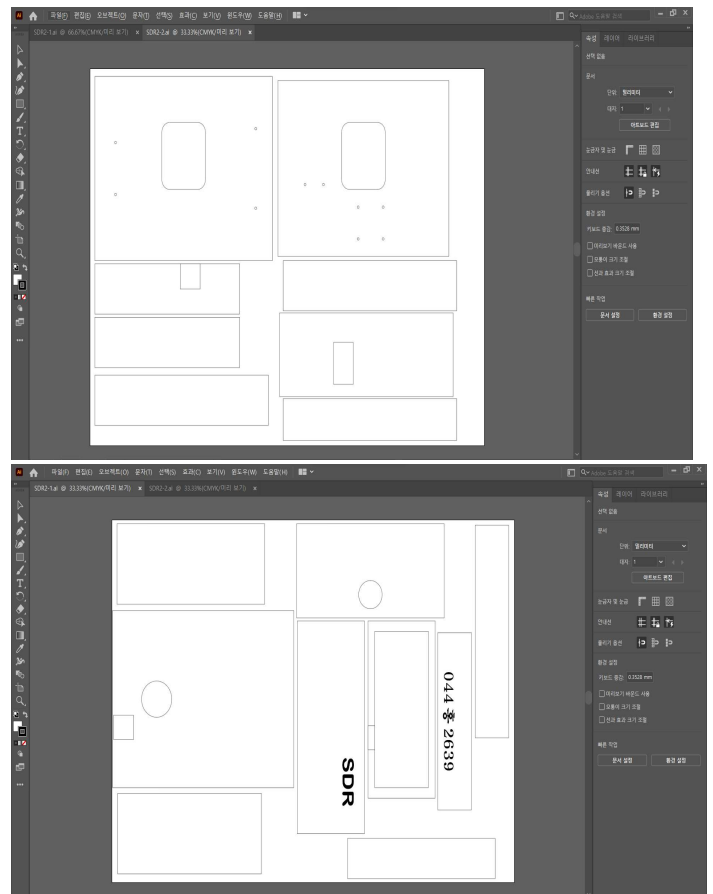
12) 하드웨어 제작

우리 조는 로봇의 하드웨어를 제작하기 위해 비금속 레이저 커터를 사용하기로 결정했다. 또한 내구성과 로봇의 motor들을 고려하여 아크릴 3T 사용하기로 결정하였다.

12.1 도안 제작

도안은 AI illustration program을 사용하여 제작하였다.

아래 사진은 도안을 제작한 모습이다.



도안에 구멍을 뚫어주어 Arduino와 motor driver가 아크릴에 고정될 수 있게 해주었고 DC motor, Servo motor, motor drvier, 아두이노, Lidar sensor에 연결된 선들을 구멍을 통해 서로 연결 될 수 있게 해주었다. 그리고 SDR이 각인될 수 있게 해주었다.

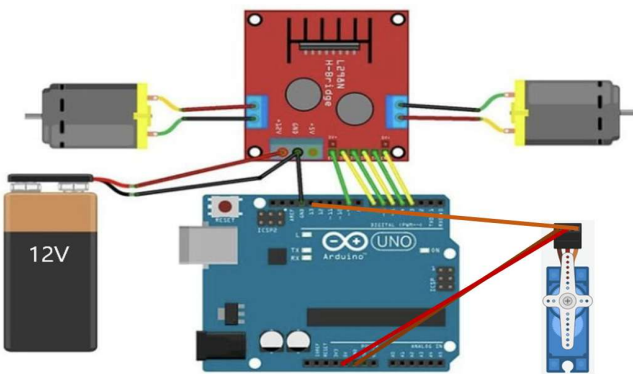
12.2 비금속 레이저 커터



비금속 레이저 커터를 진행하기 위해 먼저 F동에서 교육을 들은 후 검수를 받아 테스트를 통과하였다. 그 후 장비 사용신청을 해 제작한 도안을 가지고 비금속 레이저 커터를 진행하였다.

12.3 로봇 제작

먼저 로봇의 motor 부분의 제작을 해주었다. 그 후 나사로 아크릴과 모터 부분을 연결해 주어 로봇을 제작해 주었다.



위에 사진과 같이 DC motor 2개를 motor driver에 연결해 주었고 12V 전원을 인가해 주었다. 그리고 서보모터 또한 아두이노에 연결해 주었다.

그 후에 로봇의 전체적인 부분을 제작해 주었다. 먼저 밑에는 DC motor와 Servo motor를 바퀴와 연결해 주었고 그 위에 Arduino와 motor driver, 배터리 홀더를 배치하였다.

그리고 그 위에 라즈베리파이와 디스플레이, 보조배터리를 배치하였고 맨 위에 Lidar sensor를 배치하였다.



그 결과 위 사진과 같이 로봇을 제작하였다. 비금속 레이저 커터로 제작한 아크릴들을 아크릴 본드로 붙여주었고 나사로 Arduino와 motor driver를 고정해주었다. 그리고 디스플레이, 보조배터리, Lidar sensor들을 양면테이프로 고정해 주었다.

9. 결과 활용 방안

로봇이 택배를 배달함에 따라 기대되는 효과는 먼저 택배 기사님들의 과로 문제를 해결할 수 있다. 택배를 아파트 내에 있는 로봇에 올려놓고 몇 호인지를 입력하면 로봇이 택배를 직접 배송하게 되어 택배 기사님들의 시간을 절약할 수 있다. 또한 택배기사님들이 모든 층에 택배를 직접 들고 이동할 필요가 없어져 움직이는 동선이 줄어들게 된다. 그로 인해 과로 문제를 해결할 수 있다.

또한 배달음식 기사님들도 마찬가지로 기사님들이 직접 들고 이동할 필요 없이 로봇이 운반하게 되면 노동력을 줄일 수 있게 된다. 요즘 같은 코로나 시대에 사람과 대면하는 것이 아닌 로봇이 배송을 하게 되면 더욱 안전하다는 장점도 있다.

그리고 택배 로봇을 택배를 배송하는 용도로만 쓰이는 것이 아니라 다른 용도로도 활용할 수 있다. 무거운 짐을 들고 이동하기 어려운 임산부, 장애인 등과 같은 사람들에게도 사용하기에 아주 적합하다. 택배 로봇에 자신의 짐을 싣고 집 호수를 입력하게 되면 택배 로봇이 알아서 짐들을 집 앞까지 배송해 줍니다. 그러면 무거운 짐들을 직접 들지 않고 편안하게 집에 도착할 수 있고 물건도 안전하게 집에 도착할 수 있게 된다.

이렇듯 저희의 SDR은 택배기사, 임산부, 장애인 등 많은 사람들의 일상생활에 좋은 영향을 끼칠 것으로 예상된다.

10. 비고 및 고찰

우리 조는 라즈베리파이에 Ubuntu를 설치하고 거기에 ROS, Rviz, hector slam, Lidar sensor, Teleop Keyboard 패키지를 다운로드하여 홍익대학교 D동 3층을 mapping 해주었다. mapping 된 지도를 토대로 로봇의 navigation stack 패키지를 실행시켜주었다.

그 패키지를 통해 우리 조의 최종 목표인 로봇을 원하는 위치로 이동시키는 것을 구현하였다. 이 프로젝트를 진행함에 따라 우리 조는 경제적으로 한계점이 있어 라즈베리파이를 사용하였다. 만약 더 여유가 있었다면 메인보드로 jeston nano, NVIDIA 등을 사용하여 보드 자체의 성능을 향상시켜 프로젝트에서 지연되는 현상들을 개선했을 것이다. 더 나아가 로봇에 GPS, 블루투스 등의 센서들을 추가적으로 사용하여 프로젝트의 정확성을 높일 수 있을 것이다. 마지막으로 SDR에 더 추가할 수 있는 기능들로는 엘리베이터 이용, 택배 내려놓기 등의 기능들을 추가할 수 있다고 생각한다.