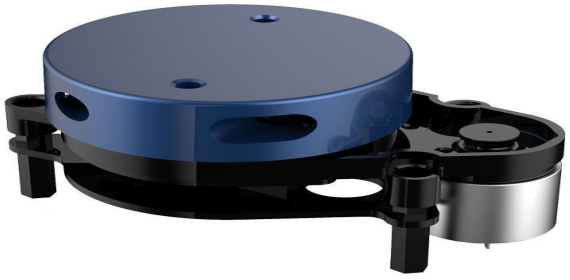
중간 보고서

주제 : 자율주행을 통한 책 인식 및 비교
팀명 : The White
팀장 : 박주언
팀원 : 김장경, 윤상민, 여재영

글쓴이 : 여재영

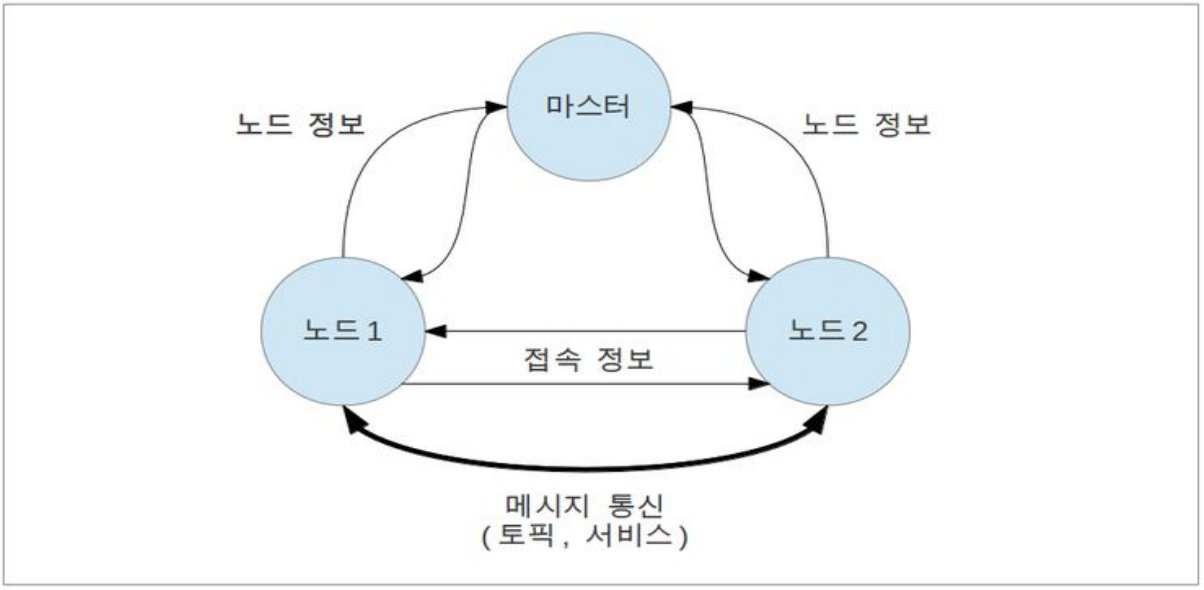
(1) 자율주행

자율 주행을 위해 카메라로 차선을 인식해서 주행 할려 했으나 우리의 목표는 도서관에서 책이 잘 꽂혀있나를 확인하기 때문에 라인을 그리기 힘들 것 같아 lidar를 써서 map을 만들기로 결정 했습니다.
(lidar는 ydlidar-x2를 사용했고 자동차 부분은 jetbot을 사용.(보드는 jetson-nano))



(2) ros란

ros : Robot Operating System의 약자로 로봇 소프트웨어 플랫폼이며 센서나 통신 개발 환경 등을 제공한다.
node : 실행되는 최소 단위의 프로세서, 하나의 프로그램 이기도 하며 topic이나 service를 이용하여 통신이 가능하다.
master : node와 node 사이를 연결하는 역할이며 master를 키지 않는한 통신이 불가능 하다. (roscore로 master를 킨다.)
message : message를 통해 node와 node 사이에 데이터를 주고 받는다.
topic : topic이란 노드들 간에 통신을 할 수 있는 채널의 개념이며 message의 이름이라 생각하면 편하다.
publish -> subscribe란 구조로 한쪽 방향으로 넘겨주며 여러번 통신이 가능하다.
service : service는 topic과는 반대로 서로 통신이 가능하지만 한번 통신 후 연결이 끊긴다.



(node 통신)

(3) ros & google_cartoprapher

ydlidar와 jetbot, 맵을 만들기 위해 google에서 제공하는 cartographer를 돌리기 위해 ros를 사용 했습니다.

//cartographer install//
mkdir -p ~/carto/src // 작업을 할 파일을 생성합니다
cd ~/carto/src // 그 파일로 이동 후

catkin_init_workspace // (작업공간 초기화인데 안해도 상관 없음)
cd ~/carto&&catkin_make // carto파일로 이동후 compile을 해줍니다.

echo "source ~/carto/devel/setup.bash" >> ~/.bashrc // bashrc에 경로 추가
source ~/.bashrc // 경로 추가 후 껐다 켜야하지만 이 코드를 입력 하면 그럴 필요가 없어짐.

cd src / src(script)파일로 이동후
git clone https://github.com/msadowski/x2_cartographer.git // git에서 코드를 다운
cd x2_cartographer // 받은 파일로 이동을 합니다.

rm -y ydlidar // 그리고 ydlidar 파일을 삭제(ydlidar 주소가 잘못 되어있음)
git clone git://github.com/YDLIDAR/ydlidar_ros.git ydlidar // 그리고 ydlidar를 직접 받습니다.

cd ../.. // carto 파일로 다시 이동
rosdep install --from-paths src --ignore-src --rosdistro melodic -r -y
source devel/setup.bash //
catkin_make // compile(carto 파일에서만 해야함)

roslaunch carto_mapper mapper.launch // cartographer를 실행 해 줍니다.

(4) topic으로 message 주고 받기

ros의 장점은 node간에 언어가 달라도 topic명과 형식을 안다면 호환이 됩니다.

ros pkg를 만들고 안에 CMakeLists와 package를 수정해 사용이 가능합니다.
만드는 법은 catkin_create_pkg (원하는 이름) rospy(파이썬일때) roscpp(c++일때)
std_msgs(std_msgs를 사용할때)로 사용합니다.
ex) catkin_create_pkg new roscpp std_msgs

위에 방법으로 new 폴더를 생성하고 안에 들어가보면 src파일이 있는데 거기에서 c++ 코드로 프로그램을 작성해주면 됩니다.

처음에는 c++ 를 사용해서 pulisher -> subscriber에게 메세지를 전달해주는 것을 해보겠습니다.

topic_publisher.cpp

```
#include "ros/ros.h"  // ROS 기본 헤더파일
#include "ros_tutorials_topic/MsgTutorial.h"// MsgTutorial 메시지 파일 헤더(빌드 후 자동 생성됨)
int main(int argc, char **argv)
{
ros::init(argc, argv, "topic_publisher"); //"topic_publisher"라는 이름을 가진 node
ros::NodeHandle nh;

// 퍼블리셔 선언, ros_tutorials_topic 패키지의 MsgTutorial 메시지 파일을 이용한
// 퍼블리셔 ros_tutorial_pub 를 작성한다. 토픽명은 "ros_tutorial_msg" 이며,
// 퍼블리셔 큐(queue) 사이즈를 100개로 설정한다는 것이다
ros::Publisher ros_tutorial_pub =
nh.advertise<ros_tutorials_topic::MsgTutorial>("ros_tutorial_msg", 100);
// 루프 주기를 설정한다. "10" 이라는 것은 10Hz를 말하는 것으로 0.1초 간격으로 반복된다
ros::Rate loop_rate(10);
// MsgTutorial 메시지 파일 형식으로 msg 라는 메시지를 선언
ros_tutorials_topic::MsgTutorial msg;
// 메시지에 사용될 변수 선언
int count = 0;

while (ros::ok())
{
msg.stamp = ros::Time::now();
msg.data = count;
// 현재 시간을 msg의 하위 stamp 메시지에 담는다
// count라는 변수 값을 msg의 하위 data 메시지에 담는다
ROS_INFO("send msg = %d", msg.stamp.sec);
ROS_INFO("send msg = %d", msg.stamp.nsec);
ROS_INFO("send msg = %d", msg.data); // stamp.sec 메시지를 표시한다
// stamp.nsec 메시지를 표시한다
// data 메시지를 표시한다
ros_tutorial_pub.publish(msg); // 메시지를 발행한다
loop_rate.sleep(); // 위에서 정한 루프 주기에 따라 슬립에 들어간다
++count; // count 변수 1씩 증가
}
return 0;
}
```

topic_subscriber.cpp

```
#include "ros/ros.h"
// ROS 기본 헤더파일
#include "ros_tutorials_topic/MsgTutorial.h"
// MsgTutorial 메시지 파일 헤더 (빌드 후 자동 생성됨)
// 메시지 콜백 함수로써, 밑에서 설정한 ros_tutorial_msg라는 이름의 토픽
// 메시지를 수신하였을 때 동작하는 함수이다
// 입력 메시지로는 ros_tutorials_topic 패키지의 MsgTutorial 메시지를 받도록 되어있다
void msgCallback(const ros_tutorials_topic::MsgTutorial::ConstPtr& msg)
{
ROS_INFO("recieve msg = %d", msg->stamp.sec);
// stamp.sec 메시지를 표시한다
ROS_INFO("recieve msg = %d", msg->stamp.nsec);
// stamp.nsec 메시지를 표시한다
ROS_INFO("recieve msg = %d", msg->data);
// data 메시지를 표시한다
}

int main(int argc, char **argv)
{
ros::init(argc, argv, "topic_subscriber");
ros::NodeHandle nh;
// 노드 메인 함수
// 노드명 초기화
// ROS 시스템과 통신을 위한 노드 핸들 선언
// 서브스크라이버 선언, ros_tutorials_topic 패키지의 MsgTutorial 메시지 파일을 이용한
// 서브스크라이버 ros_tutorial_sub 를 작성한다. 토픽명은 "ros_tutorial_msg" 이며,
// 서브스크라이버 큐(queue) 사이즈를 100개로 설정한다는 것이다
ros::Subscriber ros_tutorial_sub = nh.subscribe("ros_tutorial_msg", 100, msgCallback);
// 콜백함수 호출을 위한 함수로써, 메시지가 수신되기를 대기,
// 수신되었을 경우 콜백함수를 실행한다
ros::spin();
return 0;
}
```

그 후 다시 new 폴더에서 msg폴더를 만들고 폴더 안에 MsgTutorial.msg를 만들어줍니다.
안에 내용은

```
time stamp
int32 data
```

으로 작성 후 다시 new 폴더에 package파일을 열어
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend> 이걸 비슷한 부분에 추가해 줍니다.

그리고 CMakeLists를 열어

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_tutorials_topic)
## 캐킨 빌드를 할 때 요구되는 구성요소 패키지이다.
## 의존성 패키지로 message_generation, std_msgs, roscpp이며 이 패키지들이 존재하지 않으면 빌드 도중에 에러가 난다.
find_package(catkin REQUIRED COMPONENTS message_generation std_msgs roscpp)
## 메시지 선언: MsgTutorial.msg
add_message_files(FILES MsgTutorial.msg)
## 의존하는 메시지를 설정하는 옵션이다.
## std_msgs가 설치되어 있지 않다면 빌드 도중에 에러가 난다.
generate_messages(DEPENDENCIES std_msgs)
## 캐킨 패키지 옵션으로 라이브러리, 캐킨 빌드 의존성, 시스템 의존 패키지를 기술한다.
catkin_package(
LIBRARIES ros_tutorials_topic
CATKIN_DEPENDS std_msgs roscpp
)

## 인클루드 디렉터리를 설정한다.
include_directories(${catkin_INCLUDE_DIRS})
## topic_publisher 노드에 대한 빌드 옵션이다.
## 실행 파일, 타깃 링크 라이브러리, 추가 의존성 등을 설정한다.
add_executable(topic_publisher src/topic_publisher.cpp)
add_dependencies(topic_publisher ${${PROJECT_NAME}_EXPORTED_TARGETS}
${catkin_EXPORTED_TARGETS})
target_link_libraries(topic_publisher ${catkin_LIBRARIES})
## topic_subscriber 노드에 대한 빌드 옵션이다.
add_executable(topic_subscriber src/topic_subscriber.cpp)
add_dependencies(topic_subscriber ${${PROJECT_NAME}_EXPORTED_TARGETS}
${catkin_EXPORTED_TARGETS})
target_link_libraries(topic_subscriber ${catkin_LIBRARIES})
```

으로 바꾸어 줍니다.

그리고 rosrun new(pkg_name) topic_publisher로 실행합니다.(subscriber도 마찬가지)
그러면 통신을 하는 것을 볼 수 있습니다.

-

(5) 다른 컴퓨터와 통신

같은 컴퓨터와 node 통신을 했으니 다른 컴퓨터와 연결을 해보겠습니다.
vi ~/.bashrc를 입력해 아래에다가 밑에 두문장을 추가해 줍니다
export ROS_MASTER_URI=http://000.000.000.000:11311
export ROS_HOSTNAME=000.000.000.000

주소는 ifconfig를 쳐서 나오는 주소값을 넣는데 roscore를 킬 컴퓨터는 master와 host를 넣고 다른 컴퓨터에서는 master uri에 roscore가 켜있는 주소를 넣고(11311은 동일) hostname에는 자기 주소를 넣으면 됩니다.

※ 주의 : 연결하려는 컴퓨터는 같은 공유기에 연결이 되있어야 합니다.

(6)