

Pixhawk with ROS in Ubuntu environment

RO:BIT 11.5기

지능형배틀로봇팀 이민호

1. 들어가는 말

드론 포럼 등에선 흔히 Pixhawk는 굉장히 쓰기 쉬운 FC이고, 올라가는 펌웨어도 어렵지 않은 수준이라고 이야기한다. 그러나 이는 드론을 조종용으로 세팅할 경우이고, 실제 경험상 우리처럼 자율 주행 등을 연구해야 하는 경우 평소에 사용하던 로봇들의 펌웨어에 비해 미칠듯이 어렵다고 생각한다. 막 자율주행 드론을 시작할 때쯤 약간의 조사를 거친 결과, "DCULab" 이라는 단체에서 Mavros 패키지를 이용해 Ubuntu 상에서 드론을 제어하는 프로젝트를 진행한 자료가 있어 이를 레퍼런스로 삼았다. 이 글에서는 DCULab에서 진행한 프로젝트를 기반으로, 설치법과 필요한 헤더파일, 변수 등을 공유하고자 한다.

1.5. Pixhawk & PX4 세팅

비행을 위한 기본 세팅이 되어 있다는 전제 하에, 드론에 올라가는 PC와의 통신을 위해 따로 세팅해줘야 할 변수가 있다. SYS_COMPANION 변수인데, PC와 똑같이 세팅해주면 된다(PC에서의 세팅은 뒤에 다시 설명하겠다). 이 글에서는 921600으로 설정하고 진행했다.

```
SYS_COMPANION          Companion Link (921600 t TELEM2 as companion computer link
```

↑ SYS_COMPANION 설정

2. 설치

우선 드론에 올라갈 PC에 따라 필요한 OS가 달라지는데,

Odroid XU4, Raspberry Pi : Ubuntu 16.04 **Mate** or Upper

LattePanda(4GB RAM/64GB), NUC 등 : Ubuntu 16.04 or Upper (LattePanda의 경우 리눅스 커스텀 빌드가 따로 있다)

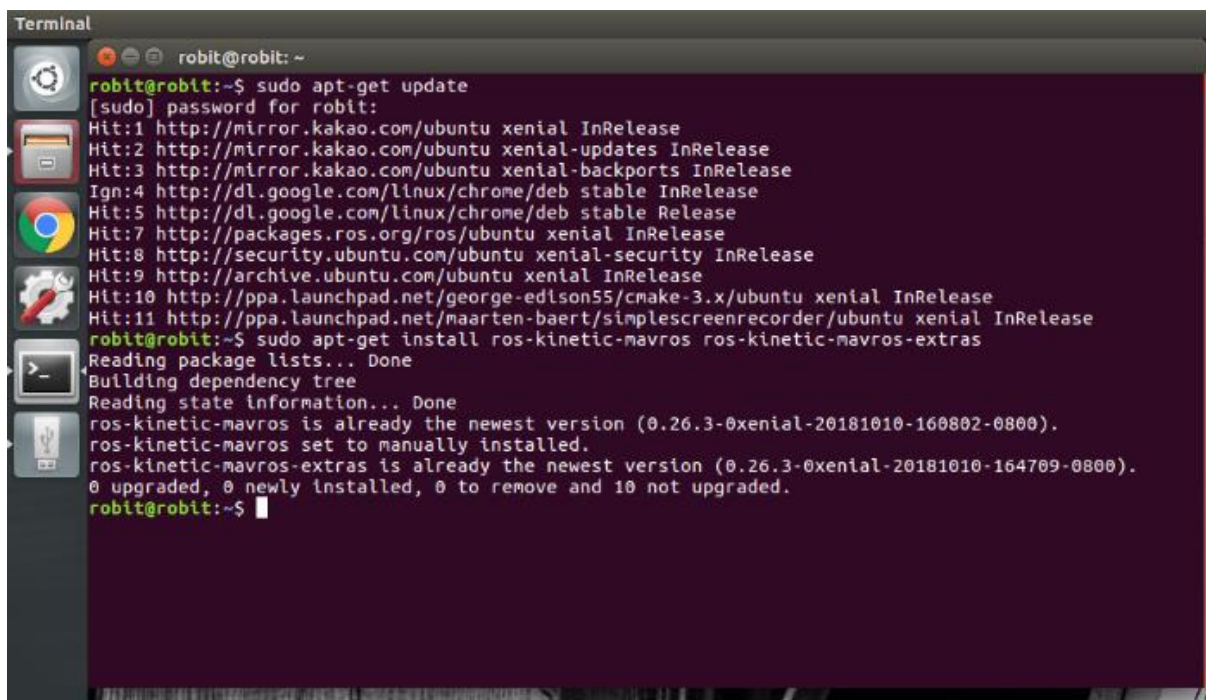
등을 사용하면 된다. Odroid의 경우 OS 설치 후 ROS 설치 과정까지는 같으나, OpenCV 연동을 할 때 ROS 내부에 있는 패키지 및 헤더를 사용하면 OpenGL 에러가 난다.

본 내용은 GIGABYTE Mini PC, Ubuntu 16.04, ROS Kinetic 환경에서 진행했다.

OS 및 ROS(indigo 이상을 권한다) 설치, 환경 설정 등은 완료되었다고 치고, Mavros 패키지를 설치해줘야 한다.

```
sudo apt-get update
```

```
sudo apt-get install ros-kinetic-mavros ros-kinetic-mavros-extras
```



```
Terminal
robbit@robbit: ~
robbit@robbit:~$ sudo apt-get update
[sudo] password for robbit:
Hit:1 http://mirror.kakao.com/ubuntu xenial InRelease
Hit:2 http://mirror.kakao.com/ubuntu xenial-updates InRelease
Hit:3 http://mirror.kakao.com/ubuntu xenial-backports InRelease
Ign:4 http://dl.google.com/linux/chrome/deb stable InRelease
Hit:5 http://dl.google.com/linux/chrome/deb stable Release
Hit:7 http://packages.ros.org/ros/ubuntu xenial InRelease
Hit:8 http://security.ubuntu.com/ubuntu xenial-security InRelease
Hit:9 http://archive.ubuntu.com/ubuntu xenial InRelease
Hit:10 http://ppa.launchpad.net/george-edison55/cmake-3.x/ubuntu xenial InRelease
Hit:11 http://ppa.launchpad.net/maarten-baert/simplescreenrecorder/ubuntu xenial InRelease
robbit@robbit:~$ sudo apt-get install ros-kinetic-mavros ros-kinetic-mavros-extras
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-kinetic-mavros is already the newest version (0.26.3-0xenial-20181010-160802-0800).
ros-kinetic-mavros set to manually installed.
ros-kinetic-mavros-extras is already the newest version (0.26.3-0xenial-20181010-164709-0800).
0 upgraded, 0 newly installed, 0 to remove and 10 not upgraded.
robbit@robbit:~$
```

↑ Mavros 설치

3. CMakeList.txt 수정, 헤더 추가

설치가 되었다면 각 패키지 CMakeList.txt 파일, find_package 부분에 catkin REQUIRED COMPONENTS roscpp rospy sensor_msgs std_msgs 등등을 넣어 준다.

```
Changes to cmake files are shown in the project tree after building.
1  cmake_minimum_required(VERSION 2.8.3)
2  project(modudculab_ros)
3
4  ## Find catkin macros and libraries
5  ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
6  ## is used, also find other catkin packages
7  find_package(catkin REQUIRED COMPONENTS
8      roscpp
9      rospy
10     std_msgs
11 )
12
13 ## System dependencies are found with CMake's conventions
```

↑ 코딩 및 컴파일에 필요한 패키지 요소

다음, Mavros에 필요한 토픽을 publishing 하기 위해, 사용하는 소스 파일 또는 헤더 파일에 ROS 및 패키지 헤더를 추가한다. Mavros 예제에서는 Setpoint를 이용한 위치 제어를 하는데, 이때 Setpoint는 PoseStamped 클래스 형태로 생성 및 전달되므로 PoseStamped.h를 추가한다.

```
_ros - Qt Creator
+ [Icons] pub setpoints pos.cpp [C++] main(int, char
1  #include <ros/ros.h>
2  #include <std_msgs/String.h>
3  #include <stdio.h>
4  #include "geometry_msgs/PoseStamped.h"
5  #include "geometry_msgs/Vector3Stamped.h"
6
7  int main(int argc, char **argv)
8  {
9      ros::init(argc, argv, "pub_setpoints");
10     ros::NodeHandle n;
```

↑ 필요한 헤더 파일들

4. 소스 코딩

다음 내용은 ROS 시스템의 publish 및 subscribe를 조금 공부하고 보면 좋다. 계속해서 Setpoint 토픽을 publishing 한다는 이야기를 하고 있는데, 무슨 뜻인지 (드디어) 설명을 하고 가려고 한다.

Pixhawk의 펌웨어인 PX4는 여러가지 비행 모드를 지원하는데, 흔히 카메라와의 연동 등을 통해 자율 비행을 하고자 할 때 사용되는 모드는 "Offboard" 모드이다. 이 모드에서는 Kill Switch와 모드 변경을 제외한 모든 리모컨 조작이 무시되는데, 그렇게 때문에 제어를 하려면 드론에 장치된 PC를 통해 Setpoint 라는 토픽을 PX4로 보내야 한다.

```
int main(int argc, char **argv)
{
    ros::init(argc, argv, "pub_setpoints");
    ros::NodeHandle n;

    ros::Publisher chatter_pub = n.advertise<geometry_msgs::PoseStamped>("/mavros/setpoint_position/local",100);
    ros::Rate loop_rate(100);
    ros::spinOnce();

    geometry_msgs::PoseStamped msg;
    int count = 1;

    //PositionReceiver qp;
    //Body some_object;
    //qp.connect_to_server();

    while(ros::ok())
    {
        //some_object = qp.getStatus();
        // some_object.print();
        //printf("%f\n", some_object.position_x);

        msg.header.stamp = ros::Time::now();
        msg.header.seq=count;
        msg.header.frame_id = 1;
        msg.pose.position.x = 0.0;//0.001*some_object.position_x;
        msg.pose.position.y = 0.0;//0.001*some_object.position_y;
        msg.pose.position.z = 1.0;//0.001*some_object.position_z;
        msg.pose.orientation.x = 0;
        msg.pose.orientation.y = 0;
        msg.pose.orientation.z = 0;
        msg.pose.orientation.w = 1;

        chatter_pub.publish(msg);
        ros::spinOnce();
        count++;
        loop_rate.sleep();
    }
}
```

↑ DCULab의 예제 소스

파란색 네모 안이 중요한 부분인데, 차례대로 설명하도록 하겠다.

```
ros::Publisher chatter_pub = n.advertise<geometry_msgs::PoseStamped>("/mavros/setpoint_position/local",100);
ros::Rate loop_rate(100);
```

맨 위 네모 안의 코드이다. "/mavros/setpoint_position/local" 이라는 토픽에 우리가 만든 메시지인 chatter_pub를 publish 하겠다는 선언이라고 보면 되겠다.

```
geometry_msgs::PoseStamped msg;
```

두번째 네모 안의 코드인데, geometry_msgs 클래스의 PoseStamped 구조체 msg를 선언한다는 의미이다. 이 msg 내부 내용을 세번째 네모 안의 코드로 정해 주고, 마지막에 publish 해준다.

```
msg.header.stamp = ros::Time::now();  
msg.header.seq=count;  
msg.header.frame_id = 1;  
msg.pose.position.x = 0.0;//0.001*some_object.position_x;  
msg.pose.position.y = 0.0;//0.001*some_object.position_y;  
msg.pose.position.z = 1.0;//0.001*some_object.position_z;  
msg.pose.orientation.x = 0;  
msg.pose.orientation.y = 0;  
msg.pose.orientation.z = 0;  
msg.pose.orientation.w = 1;  
chatter_pub.publish(msg);
```

이 부분까지 정상작동 한다면, rqt나 rostopic echo 등을 이용해 토픽을 봤을 때 /mavros/setpoint_position/local 이라는 토픽 내부 내용이 이렇게 세팅된다.

5. Mavros 사용

PX4와 Mavros의 연동에서 중요한 2번째 요소는 launch 파일의 제작이다. 사실 사용하는 입장에서조차 완전히 알고 사용하지는 못하지만, 일단 동작에 필요한 많은 파라미터를 한번에 설정해줄 수 있는 만큼 사용은 하고 있다.



```
<launch>

<arg name="fcu_url" default="udp://:14540@192.168.0.5:14557"/>
<!-- arg name="fcu_url" default="/dev/ttyUSB0:921600" / -->
<arg name="gcs_url" default="" />
<arg name="tgt_system" default="1" />
<arg name="tgt_component" default="1" />

<node name="mavros" pkg="mavros" type="mavros_node" output="screen">
  <param name="fcu_url" value="$(arg fcu_url)" />
  <param name="gcs_url" value="$(arg gcs_url)" />
  <param name="target_system_id" value="$(arg tgt_system)" />
  <param name="target_component_id" value="$(arg tgt_component)" />
  <rosparam command="load" file="$(find mavros)/launch/px4_config.yaml" />

  <!-- rosparam command="load" file="$(find mavros)/launch/px4_blacklist.yaml" -->

  <!-- enable heartbeat send and reduce timeout -->
  <param name="conn_heartbeat" value="5.0" />
  <param name="conn_timeout" value="5.0" />
  <!-- automatically start mavlink on USB -->
  <param name="startup_px4_usb_quirk" value="true" />
  <param name="mocap/use_tf" value="true"/>
  <param name="mocap/use_pose" value="false"/>
</node>

<node name="irc_drone_vision" pkg="irc_drone_vision" type="irc_drone_vision" output="screen">
</node>

</launch>
```

↑ 사용 중인 launch 파일 예시

위 내용 중 가장 첫번째로 세팅해줘야 할 부분이 arg name="fcu_url" 부분이다. 맨 위에 Pixhawk 세팅에서 SYS_COMPANION 변수를 921600으로 세팅했는데, 이 변수와 위 fcu_url의 baudrate가 같아야 한다(위 사진에선 그 부분이 <!-- -->로 주석 처리된 상태이다).

Pixhawk와 PC를 USBtoUART로 연결했다면 우분투 터미널에 다음 명령어를 입력해보자.

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

터미널에 '/dev/ttyUSB0' 라고 뜬다면, 일단 소자나 USB 포트가 고장나진 않았

다는 증거이다. '/dev/ttyUSB0(간혹 /dev/ttyACM0로 연결되는 경우도 있다)' 다음 아래 명령어를 입력해보자.

```
sudo chmod 777 /dev/ttyUSB0
```

이 명령어는 /dev/ttyUSB0에 연결된 것에 읽고 쓰는 것이 가능하게 하는 명령어이다(chmod가 명령어, 777이 권한 설정 부분). 간혹 연결은 됐는데 Permission Denied 오류가 날 경우 실행해주면 된다. 그냥 PC 부팅 후에 한번씩 써주자.

이상 Pixhawk와 Ubuntu PC와 연결은 끝이다. 터미널에서,

```
roslaunch (패키지 명) (파일이름).launch
```

라고 입력하면, Mavros 패키지와 함께 직접 빌드한 패키지가 실행된다.

6. 참고문헌 및 출처

DCULab - Pixhawk와 ROS를 이용한 자율주행 - 가끔 자료의 명령어 중에 오류가 나는 경우가 있는데, 검색을 통해 해결하도록 하자.

(http://www.modulabs.co.kr/board_GDCH80/1543)

(WW192.168.0.49Wrobit_ftp2W053 지능형배틀로봇팀W2017_DroneW00_연구자료_바이블WModuDCULab_Autonomous_Drone.pdf : PDF 파일 버전도 있다)