



# *ei* AROBOT

로봇, 모두의 곁에 a robot for all

I N V E S T O R   R E L A T I O N S



# INDEX

## I. 수업개요

## II. 비행로봇 하드웨어

1. 하드웨어 구성
2. 하드웨어 조립

## III. 개발환경 설치

1. 운영체제의 선정
2. WSL2 Ubuntu 20.04
3. Visual Studio Code
4. ROS noetic

## IV. ROS 기초 실습

1. ROS 개요
2. ROS 개발환경 구축
3. Package 만들기
4. publisher & subscriber 구현

## V. 모터 제어 실습

1. pigpio
2. PWM
3. 모터 PWM 테스트 코드 작성
4. 비행로봇 환경 구성 및 테스트

## VI. 비행로봇 프로그래밍

1. 원격조종 코드 작성
2. 모터 컨트롤러 코드 작성

## VII. 비행로봇 최종 데모

[https://github.com/arirang2067/drone\\_tutorial](https://github.com/arirang2067/drone_tutorial)  
비행로봇 수업자료와 예제코드를 받을 수 있습니다.

## I. 수업개요

---

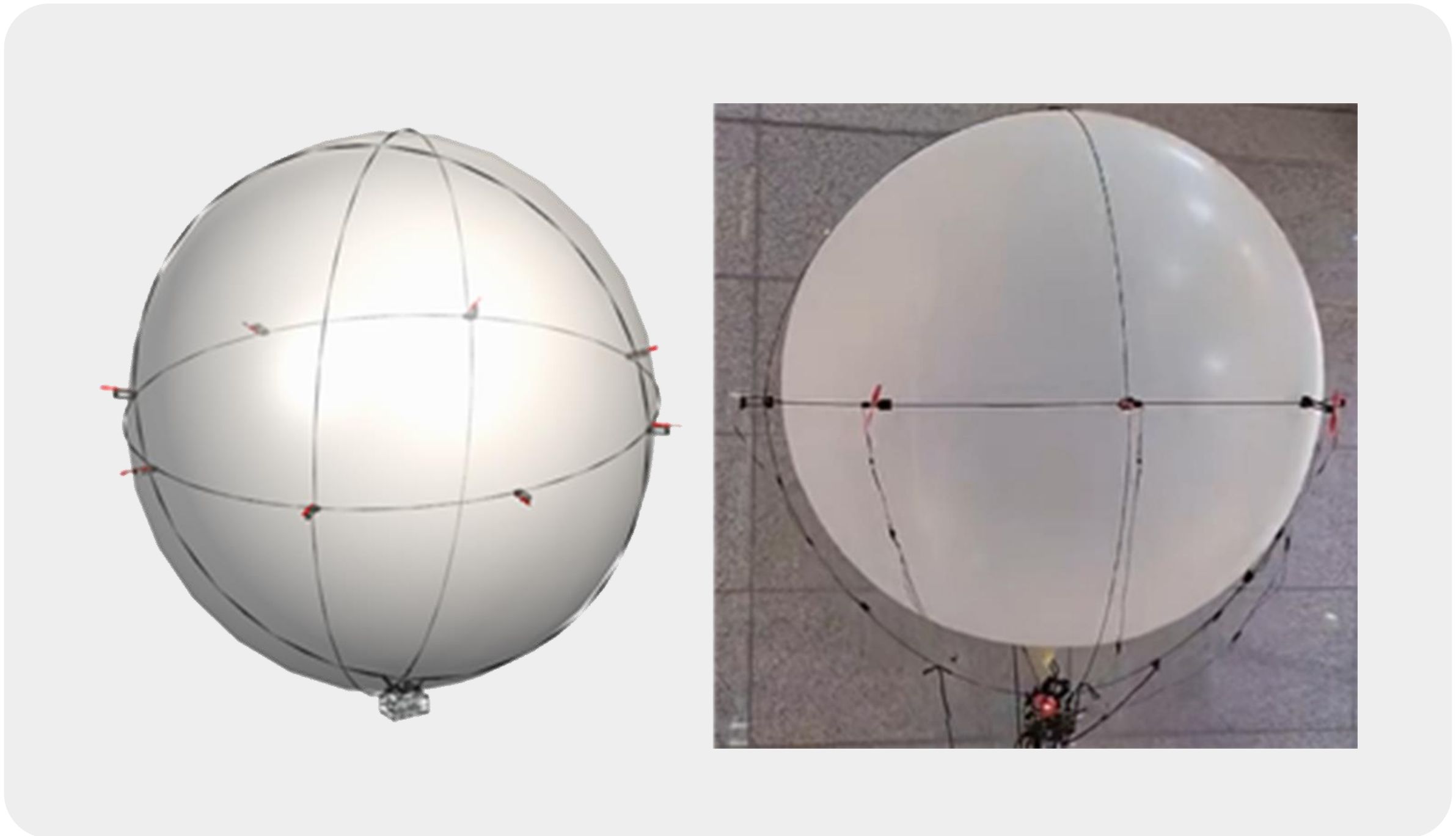
풍선형 비행로봇을 조립하고 프로그래밍하여 조종할 수 있다.

### 〈수업 일정〉

- 1일차 : 비행로봇의 조립
- 2일차 : 리눅스와 C++, ROS 프로그래밍
- 3일차 : 비행로봇 프로그래밍
- 4일차 : 비행로봇 조종하기

## 비행체의 모습

풍선형 비행로봇의 제작된 모습이다.



〈풍선형 비행로봇의 설계(좌), 실제 제작된 비행로봇(우)〉



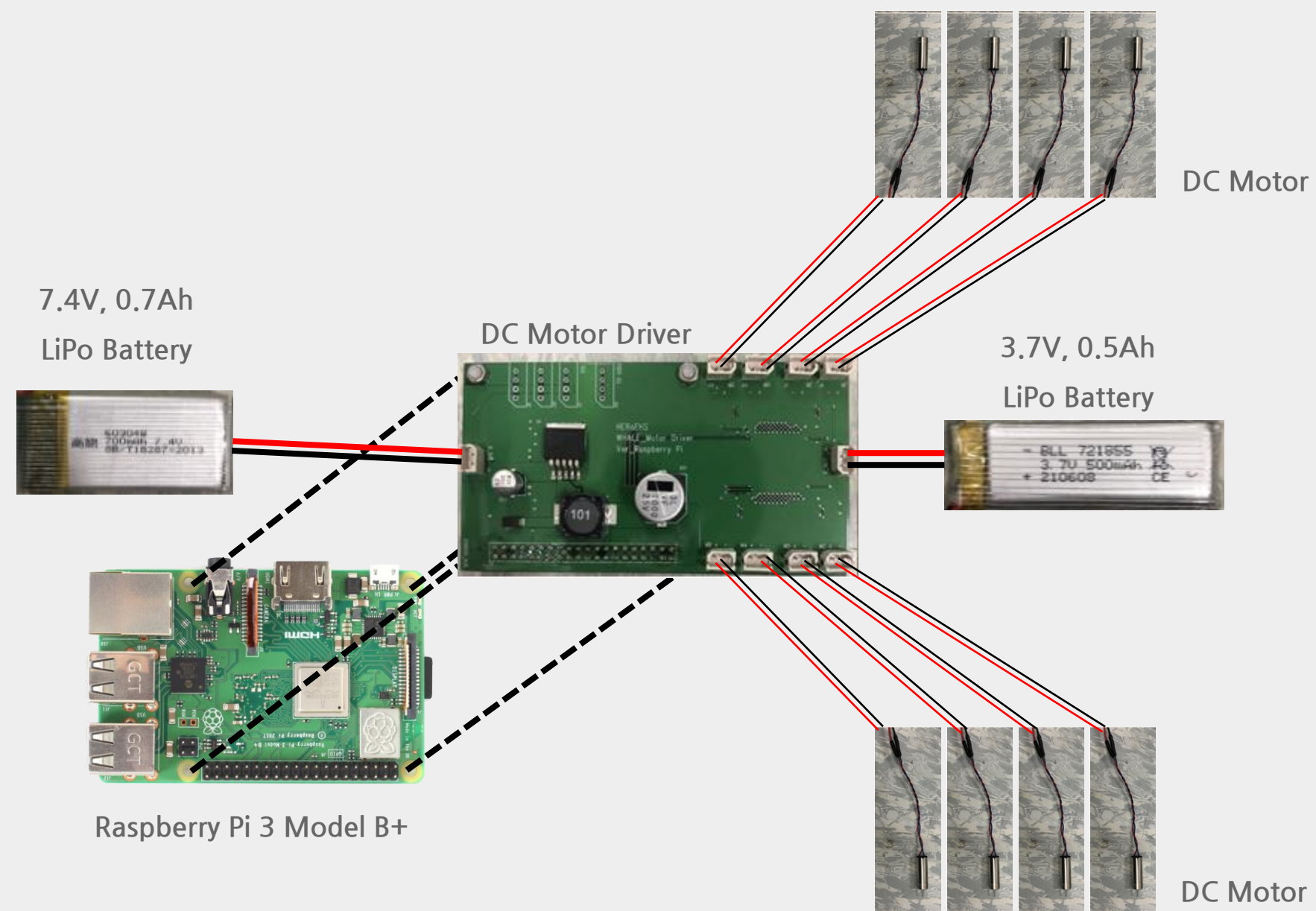
## II. 비행로봇 하드웨어

### 1. 하드웨어 구성



### 전장부의 구성

풍선형 비행로봇의 전장부는 아래와 같이 구성된다.



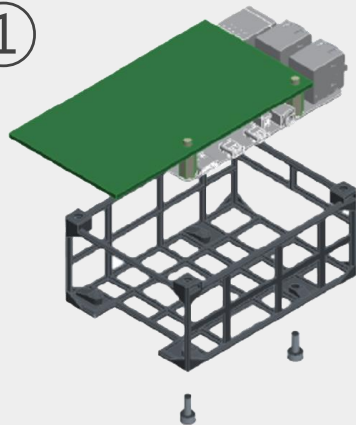
〈비행로봇 전장부의 구성〉

### 기구부의 조립 순서

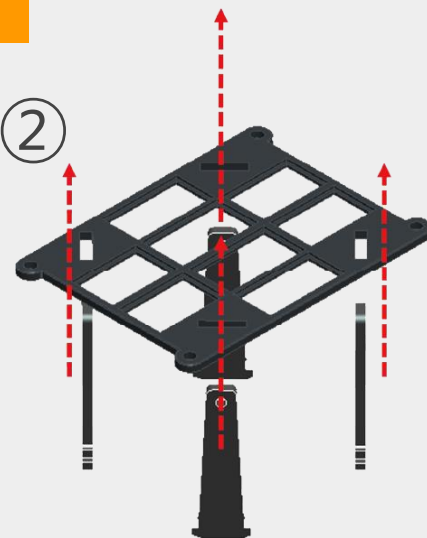
아래의 순서를 따라 보드 프레임을 조립한다.

#### 1. 보드 프레임 조립

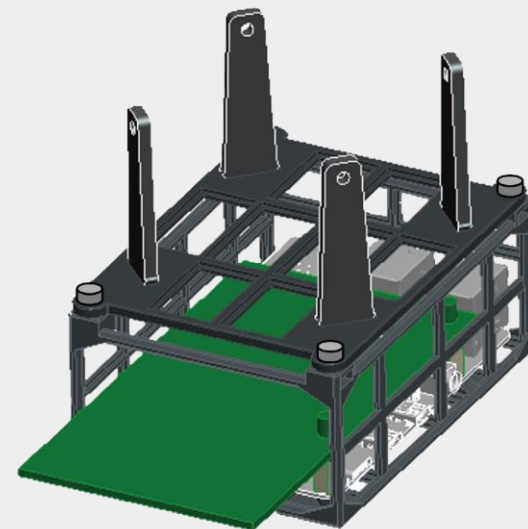
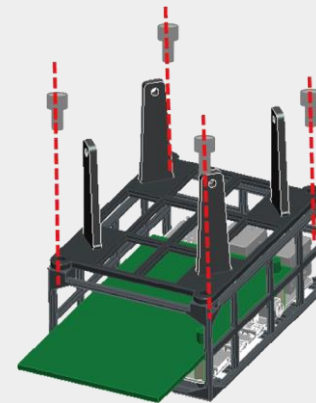
①



②

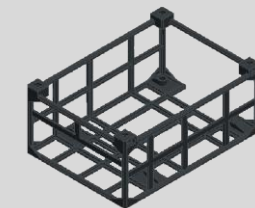


③

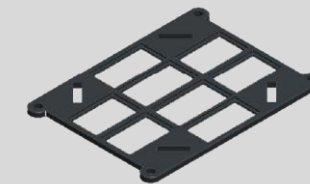


〈보드 프레임 완성 모습〉

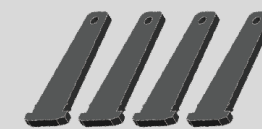
#### <부품 목록>



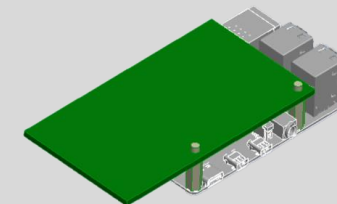
하단 프레임 - 1개



상단 프레임 - 1개



탄소 막대 프레임 - 4개



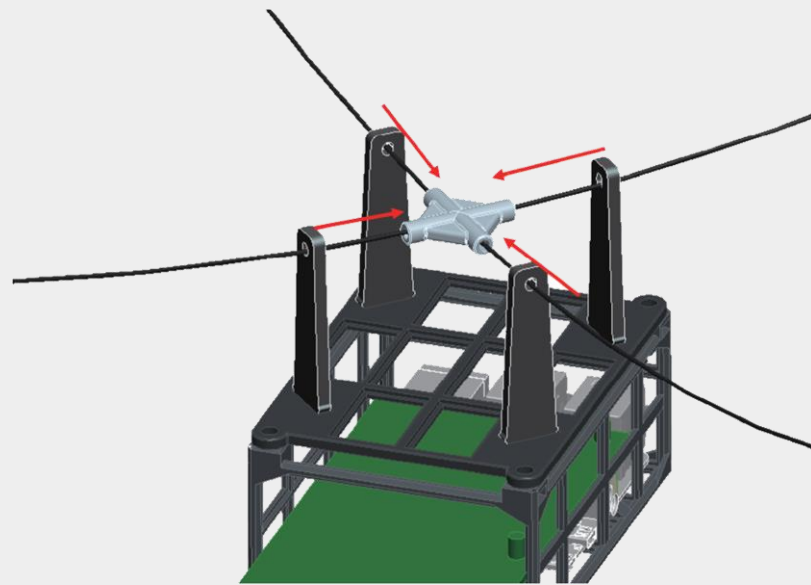
보드 - 1개

### 기구부의 조립 순서

아래의 순서를 따라 세로 방향으로 탄소 막대를 조립한 뒤, 결합지점을 접착제로 고정한다.

#### 2. 세로 방향 프레임 조립

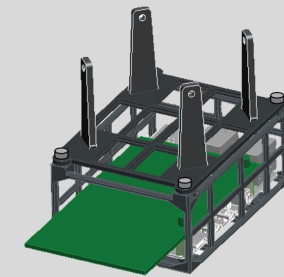
##### ① 하단 프레임



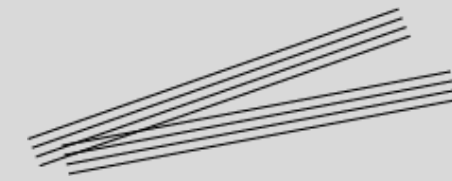
##### ② 상단 프레임



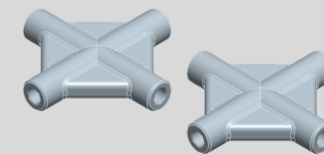
#### <부품 목록>



완성된 보드 프레임 - 1개



탄소 막대(800mm) - 8개

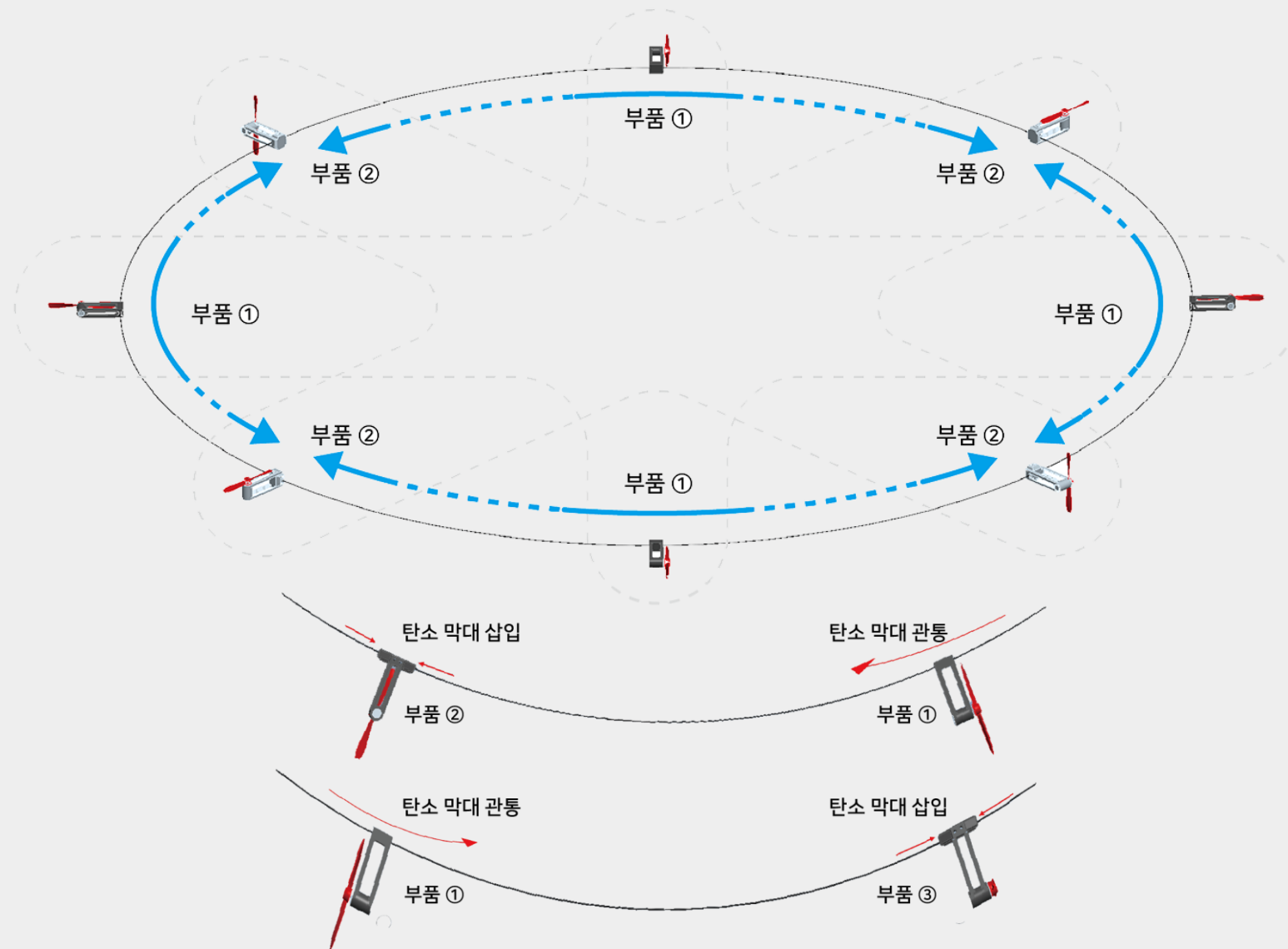


십자모양 결합 부품 - 2개

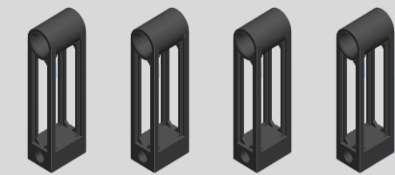
### 기구부의 조립 순서

모터 결합 부품을 잘 구분하여 가로 방향 프레임을 조립한다.

#### 3. 가로 방향 프레임 조립



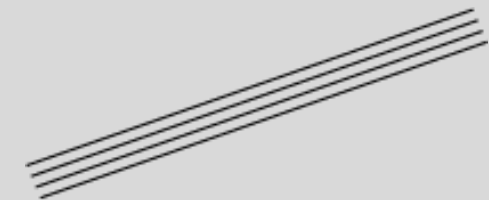
#### <부품 목록>



모터 결합 부품 ① - 4개



모터 결합 부품 ② - 2개



탄소 막대(800mm) - 4개

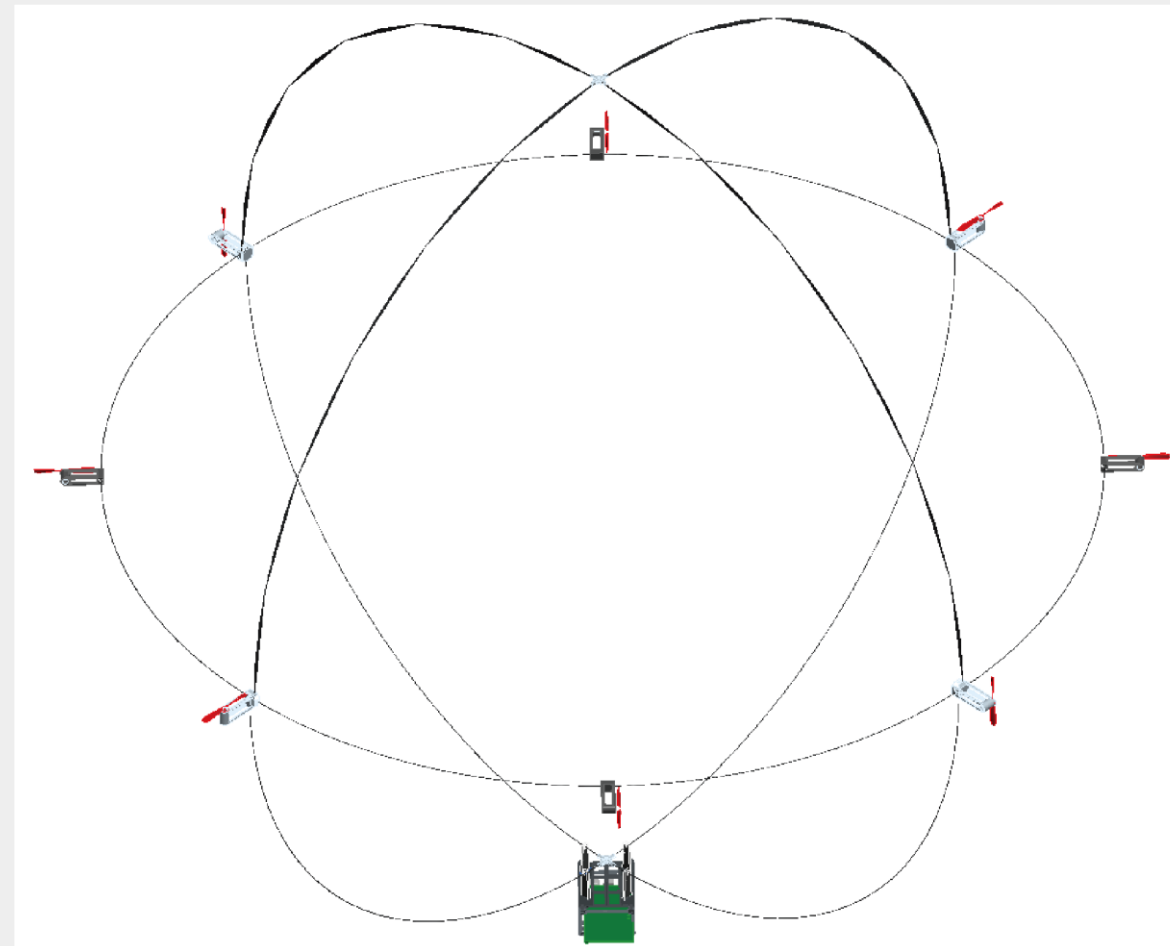
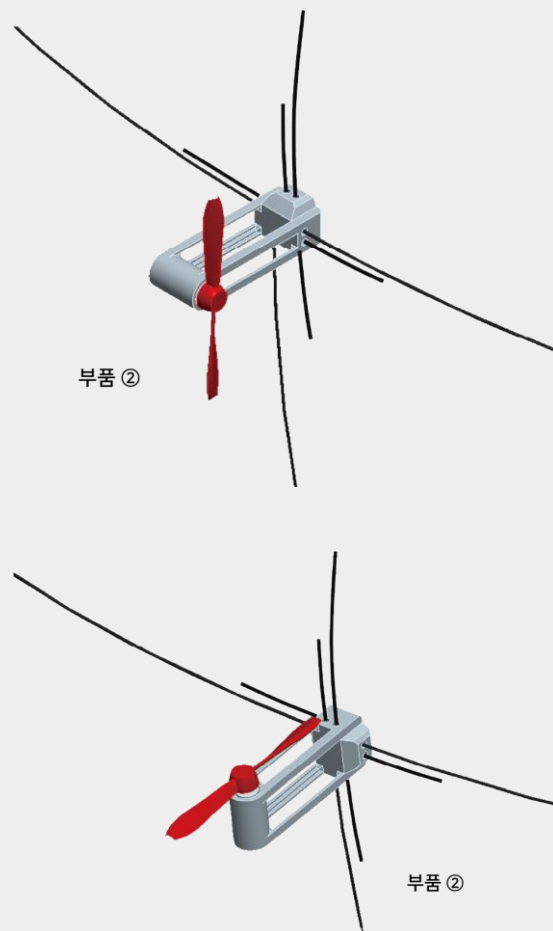
<기구부의 조립 순서>



### 기구부의 조립 순서

모터 결합 부품에 가로 방향 프레임과 세로 방향 프레임을 연결한다.

#### 4. 가로 방향 프레임과 세로 방향 프레임 연결



〈최종 프레임 완성 모습〉

## OS의 종류

OS(Operating System)은 사용자의 하드웨어, 시스템 리소스를 제어하고 프로그램에 대한 일반적 서비스를 지원하는 시스템 소프트웨어이다.





## 왜 Ubuntu를 쓰나요?

- 데비안 리눅스를 포크해 개발된 운영체제
- 남아프리카의 건국이념 ‘우분투’
- 자유 소프트웨어이며 오픈 소스 개발이 가능하다.
- 다양한 하드웨어와 호환된다.
- 생태계가 커져서 다양한 사람들과 문제공유가 가능하다.



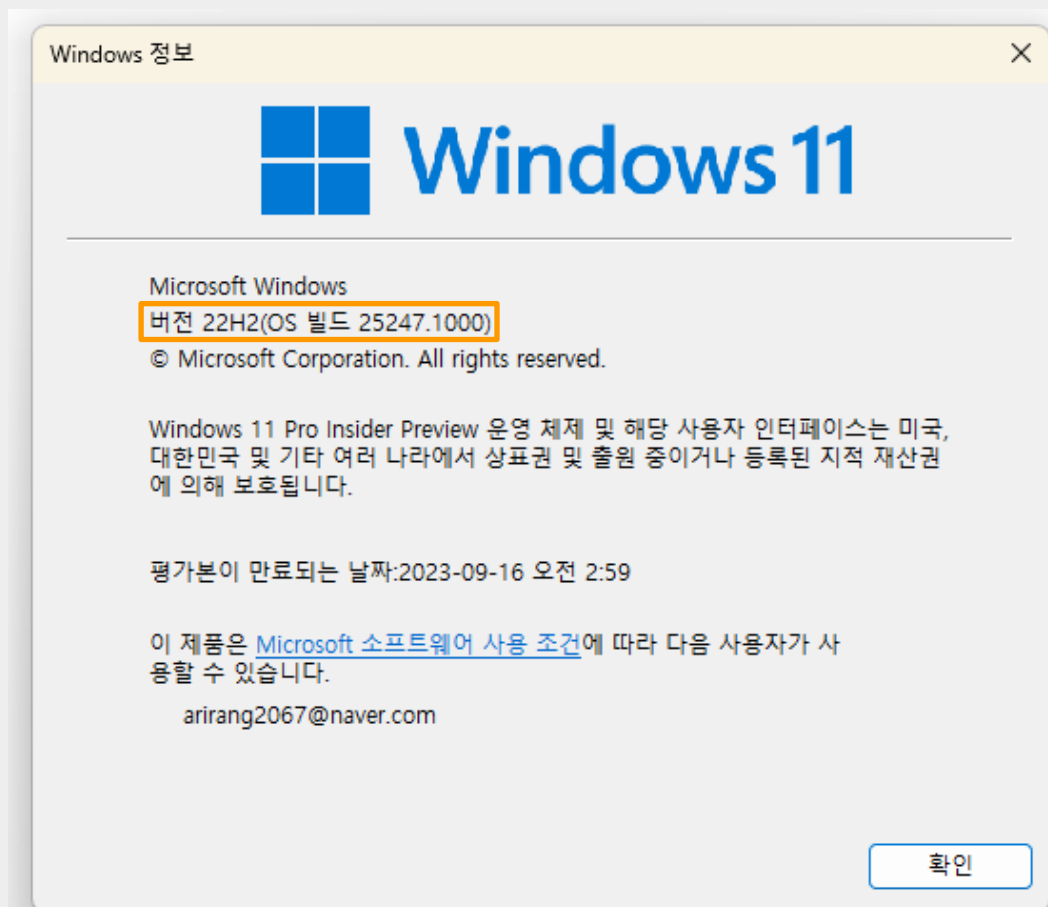


## WSL이란 무엇인가?

- 리눅스용 윈도우 하위 시스템
- (Windows Subsystem for Linux, WSL)
- 윈도우 내에서 리눅스 계열 인터페이스를 제공한다.
- 본 수업에서는 WSL2 Ubuntu 20.04를 쓸 것이다.

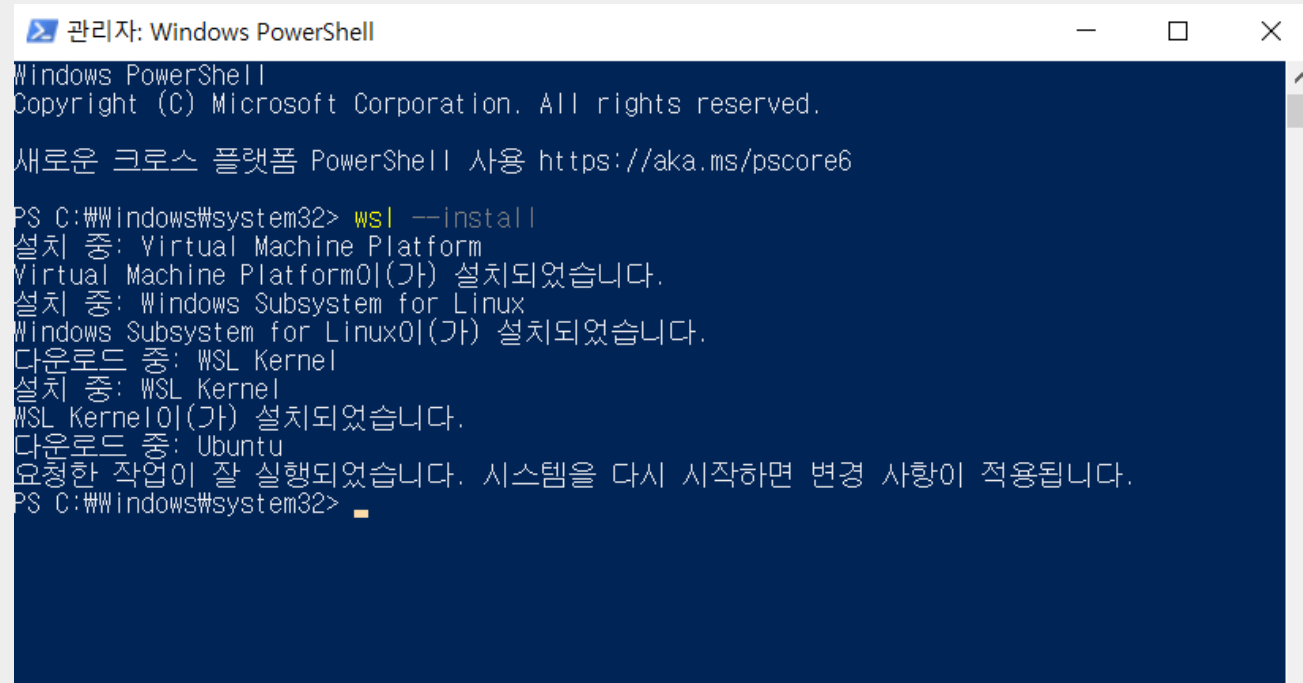
## Windows 버전 확인

- Windows key + R키로 '실행'을 시작.
- winver를 입력 후 확인
- WSL은 Windows10 20H1 버전부터 설치 가능



## WSL 설치 방법

- Windows key + S키로 Windows Terminal이나 Powershell 검색. '관리자로 실행' 을 선택
- 'wsl --install -d Ubuntu-20.04' 입력.
- 설치 후 컴퓨터 다시 시작.



```
관리자: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

새로운 크로스 플랫폼 PowerShell 사용 https://aka.ms/pscore6

PS C:\Windows\system32> wsl --install
설치 중: Virtual Machine Platform
Virtual Machine Platform이(가) 설치되었습니다.
설치 중: Windows Subsystem for Linux
Windows Subsystem for Linux이(가) 설치되었습니다.
다운로드 중: WSL Kernel
설치 중: WSL Kernel
WSL Kernel이(가) 설치되었습니다.
다운로드 중: Ubuntu
요청한 작업이 잘 실행되었습니다. 시스템을 다시 시작하면 변경 사항이 적용됩니다.
PS C:\Windows\system32>
```



## WSL2 Ubuntu20.04 실행

- 다시 시작 후 설치 과정이 이어지고 WSL ubuntu 터미널이 실행됨.
- 자동으로 Ubuntu로 설치되며 버전은 20.04
- 최초 시작 시 username과 password를 설정.
- Ubuntu는 password가 ‘\*\*\*’이 아닌 안보이는 것에 유의
- Ubuntu 패키지 업데이트 및 업그레이드 진행.
- `sudo apt-get update`
- `sudo apt-get upgrade`

```
ubuntu@DESKTOP-TTIPCK5: ~  
Installing, this may take a few minutes...  
Please create a default UNIX user account. The username does not need to match your Windows username.  
For more information visit: https://aka.ms/wslusers  
Enter new UNIX username: ubuntu  
New password:  
Retype new password:  
passwd: password updated successfully  
Installation successful!  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
  
Welcome to Ubuntu 20.04 LTS (GNU/Linux 5.10.16.3-microsoft-standard-WSL2 x86_64)  
  
 * Documentation:  https://help.ubuntu.com  
 * Management:    https://landscape.canonical.com  
 * Support:        https://ubuntu.com/advantage  
  
System information as of Thu Dec  1 11:28:50 KST 2022  
  
System load:  0.06      Processes:            8  
Usage of /:   0.4% of 250.98GB   Users logged in:     0  
Memory usage: 0%          IPv4 address for eth0: 172.18.147.71  
Swap usage:   0%  
  
0 updates can be installed immediately.  
0 of these updates are security updates.  
  
The list of available updates is more than a week old.  
To check for new updates run: sudo apt update  
  
This message is shown once once a day. To disable it please create the  
/home/ubuntu/.hushlogin file.  
ubuntu@DESKTOP-TTIPCK5:~$
```

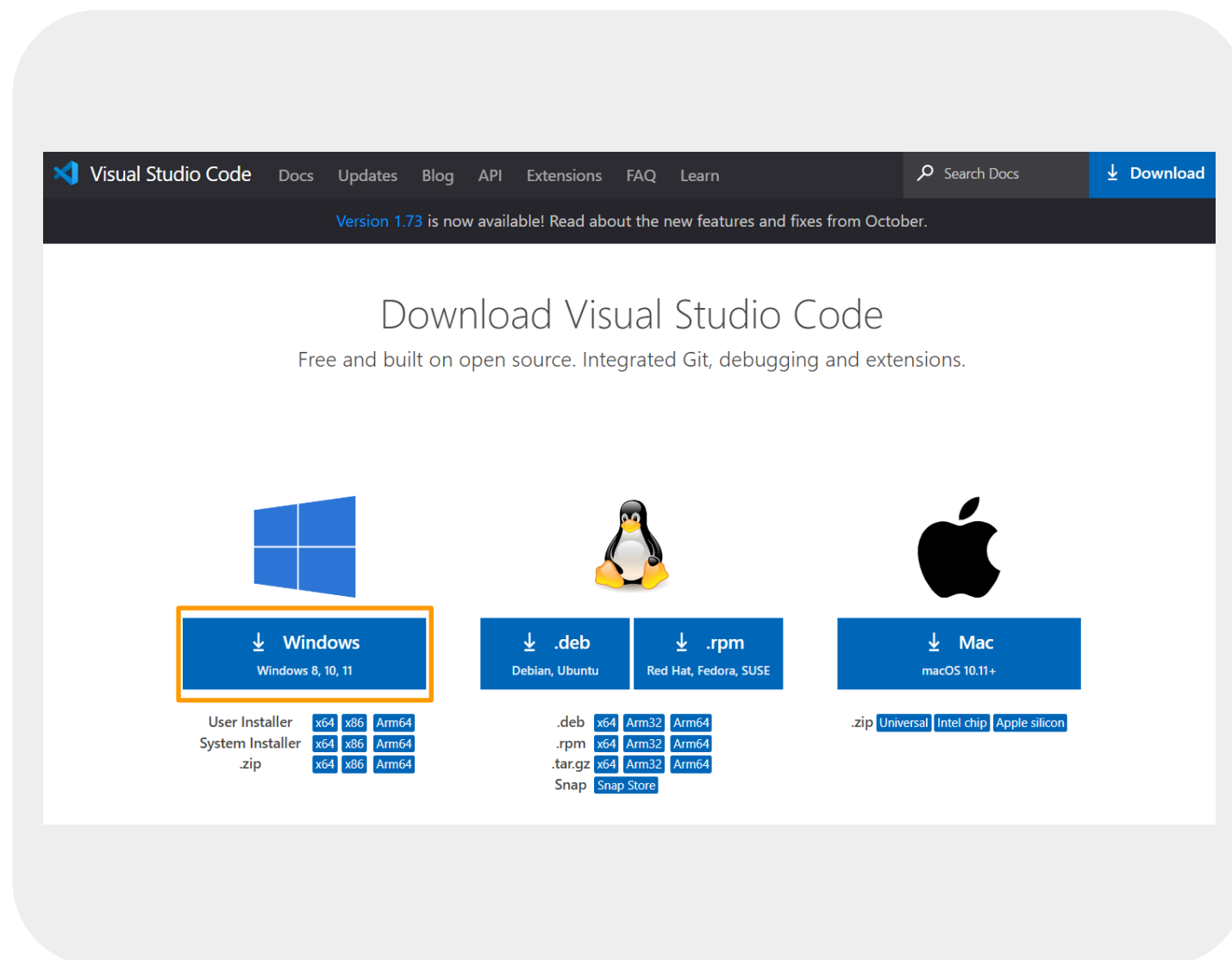


## Visual Studio Code

- 마이크로소프트에서 개발한 텍스트 에디터
- 강력하고 다양한 확장기능으로 **애용**되고 있음

## Visual Studio Code 설치 (1)

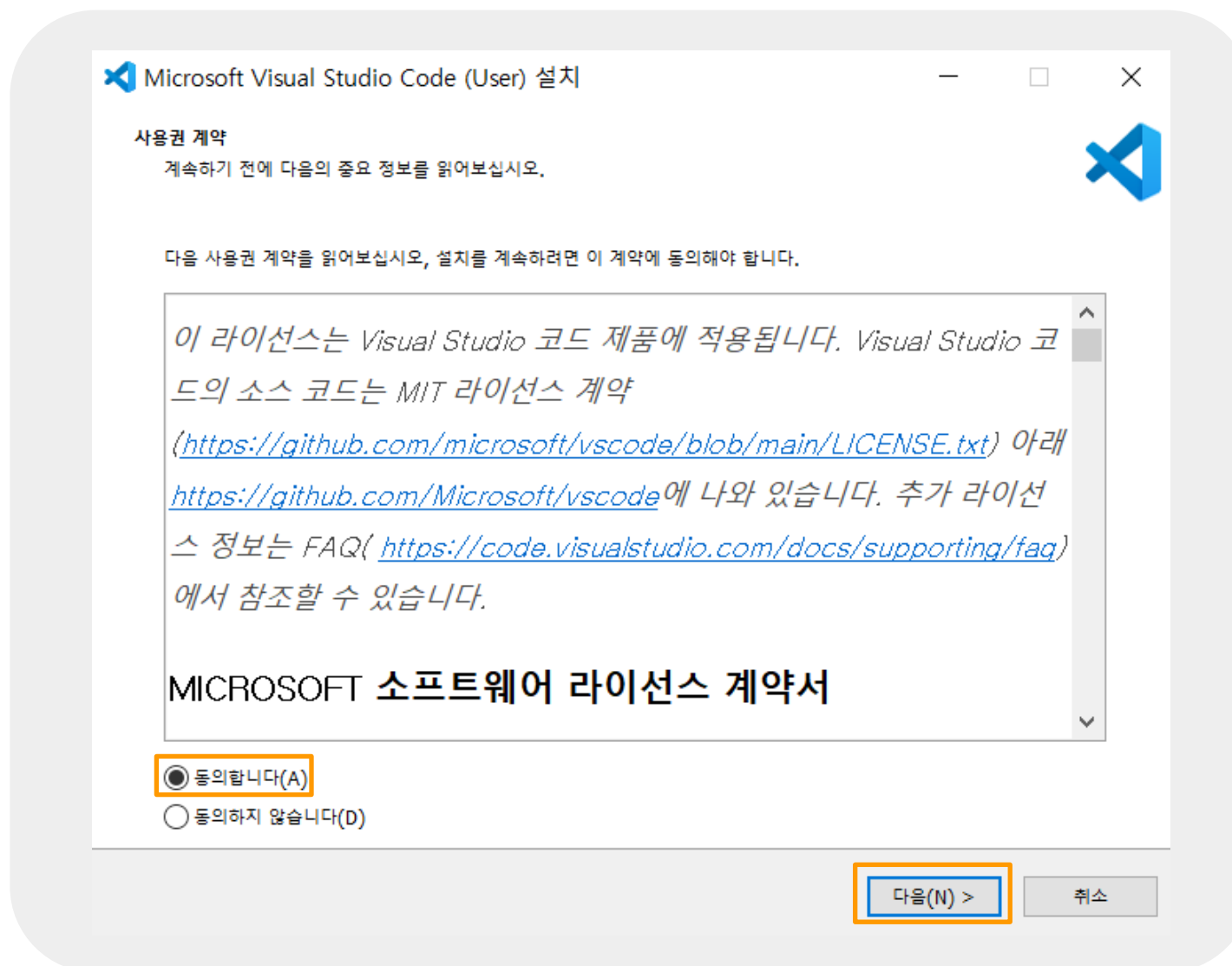
- <https://code.visualstudio.com/Download>
- Windows에서 다운로드 클릭.
- 다운로드 파일 실행



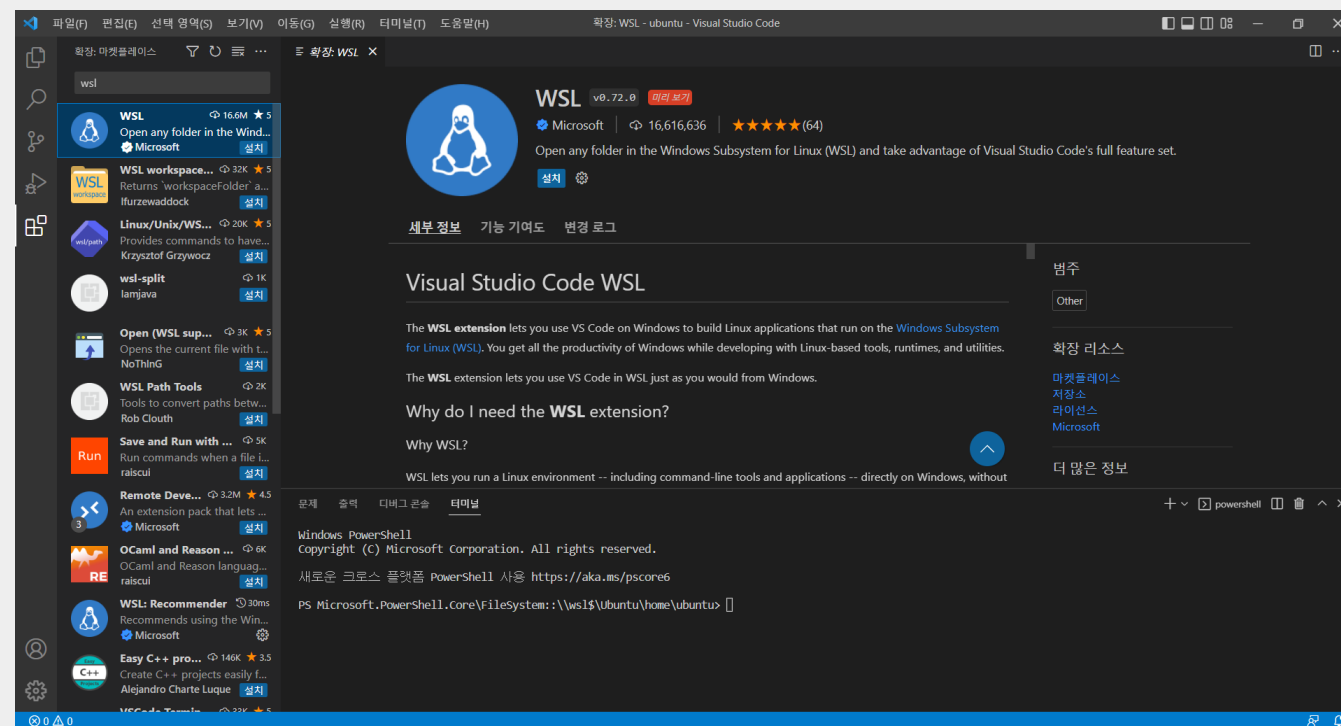


## Visual Studio Code 설치 (2)

- ‘동의합니다’ 체크하고 ‘다음’ 클릭.
- 계속 다음 누르면서 설치 진행.



# WSL 확장기능 설치



- 필요에 따라 다양한 확장기능 설치 가능.
- 왼쪽 '확장' 탭에서 WSL 검색 후 설치.
- Korean Language Pack 설치.
- C/C++ Extension Pack 설치.
- WSL Ubuntu에서 'code .' 입력하여 재실행



## VS Code에서 Ubuntu 다루기

### Ubuntu 작업환경에서 자주 쓰는 명령어 10가지

<pre>\$ ls</pre> <p>// 현재 디렉토리 안의 파일과 디렉토리를 나열한다.</p>	<pre>\$ sudo</pre> <p>// root 권한을 부여하여 명령어를 실행한다.</p>
<pre>\$ cd [directory]</pre> <p>// 디렉토리를 이동한다.</p>	<pre>\$ mv [file or directory]</pre> <p>// 파일 또는 디렉토리를 이동한다.</p>
<pre>\$ mkdir [directory]</pre> <p>// 디렉토리를 생성한다.</p>	<pre>\$ cp [file or directory]</pre> <p>// 파일 또는 디렉토리를 복사한다.</p>
<pre>\$ vi [file]</pre> <p>// 파일을 vi 텍스트 편집기로 연다.</p>	<pre>\$ rm [file or directory]</pre> <p>// 파일 또는 디렉토리를 삭제한다.</p>
<pre>\$ ifconfig</pre> <p>// 네트워크 인터페이스 및 ip 주소를 표시한다.</p>	<pre>\$ clear</pre> <p>// 터미널의 내용을 지운다.</p>





## VS Code에서 Ubuntu 다루기

디렉토리를 나타내는 특수문자

/  
// 루트 디렉토리

~  
// 홈 디렉토리

.  
// 현재 디렉토리

..  
// 이전 디렉토리

자주 사용하는 특수문자와 명령어

\*  
// 모두

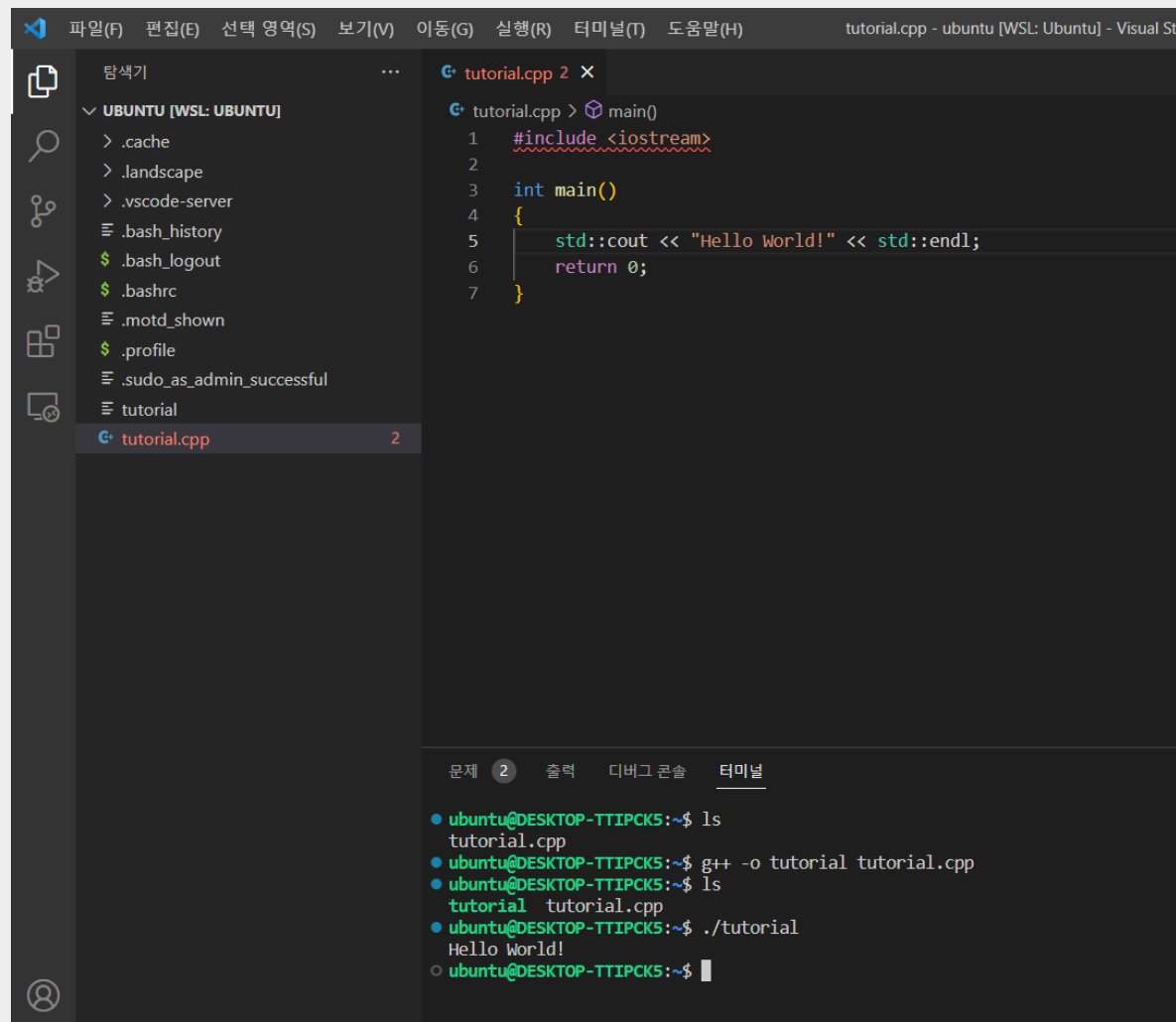
-  
// 옵션

<tab>  
// 디렉토리, 파일, 패키지 이름의 자동 완성

<↑>, <↓>  
// 이전에 사용한 명령어, 최근에 사용한 명령어로 이동

## VS Code에서 코드작성

- 상단의 '터미널' - '새 터미널'로 Ubuntu 터미널 열기
- 하단에 터미널이 생성됨.
- 터미널에 '**sudo apt-get install g++**' 입력하여 설치.
- g++ : c++언어를 컴파일하는 컴파일러

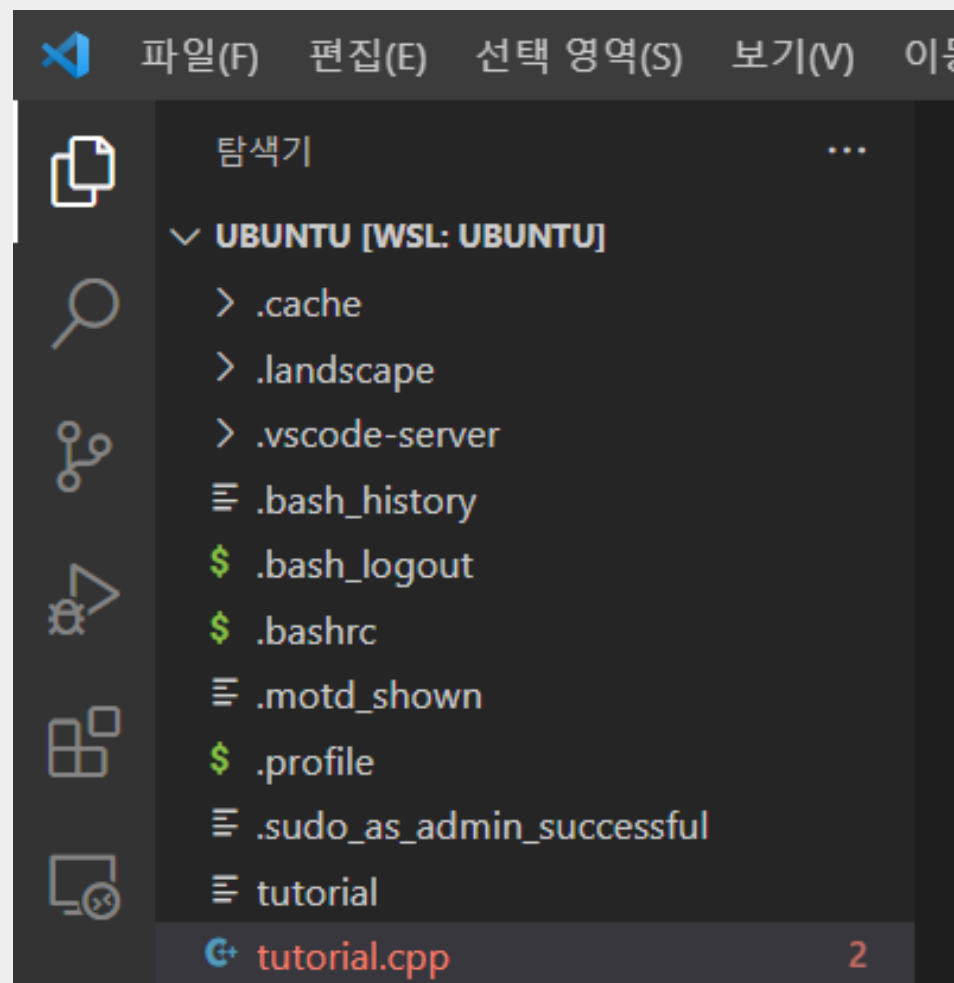


The screenshot shows the Visual Studio Code interface. On the left, the Explorer panel shows the file structure of the 'UBUNTU [WSL: UBUNTU]' workspace, with 'tutorial.cpp' selected. The main editor displays the code for 'tutorial.cpp', which includes a C++ program that prints 'Hello World!'. The bottom panel shows the 'Terminal' tab, which contains the following commands and output:

```
ubuntu@DESKTOP-TTIPCK5:~$ ls
tutorial.cpp
ubuntu@DESKTOP-TTIPCK5:~$ g++ -o tutorial tutorial.cpp
ubuntu@DESKTOP-TTIPCK5:~$ ls
tutorial  tutorial.cpp
ubuntu@DESKTOP-TTIPCK5:~$ ./tutorial
Hello World!
ubuntu@DESKTOP-TTIPCK5:~$
```

## VS Code에서 코드작성

- 좌측 프로젝트 폴더 창에서 'tutorial.cpp' 생성
- 코드 내용 작성
- 'g++ -o tutorial tutorial.cpp'로 컴파일



```
#include <iostream>

int main()
{
    std::cout << "Hello World" << std::endl;
    return 0;
}
```

## VS Code에서 코드작성

- 좌측 프로젝트 폴더 창에서 'tutorial.cpp' 생성
- 코드 내용 작성
- 'g++ -o tutorial tutorial.cpp'로 컴파일

## VS Code에서 코드작성

- 좌측 프로젝트 폴더 창에서 'tutorial.cpp' 생성
- 코드 내용 작성
- 'g++ -o tutorial tutorial.cpp'로 컴파일

```
문제 2 출력 디버그 콘솔 터미널
• ubuntu@DESKTOP-TTIPCK5:~$ ls
  tutorial.cpp
• ubuntu@DESKTOP-TTIPCK5:~$ g++ -o tutorial tutorial.cpp
• ubuntu@DESKTOP-TTIPCK5:~$ ls
  tutorial tutorial.cpp
• ubuntu@DESKTOP-TTIPCK5:~$ ./tutorial
  Hello World!
○ ubuntu@DESKTOP-TTIPCK5:~$
```



```
#include <iostream>
int main()
{
    int a, b;
    std::cin >> a >> b;
    std::cout << a + b << std::endl;
    return 0;
}
```

## C++ 프로그래밍 실습

- 두 정수 a, b를 입력 받고, a+b를 출력하는 프로그램
- 입력 : a, b
- 출력 : a+b
- 심화 : 사칙연산을 출력하는 프로그램을 작성해보자

## ROS - Robot Operating System

The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source.

## ROS란 무엇인가?

- 노드와 노드 사이의 데이터 통신
- 로봇 관련 다양한 기능
- 다양한 개발 도구 제공

## ROS 설치(1)

- <http://wiki.ros.org/noetic/Installation/Ubuntu>
- Ubuntu20.04에 호환되는 ROS는 Noetic 버전이다.





## ROS 설치(2)

WSL의 터미널에서 다음과 같이 ROS 설치 과정을 진행하자.

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

// ros.org의 패키지를 설치할 수 있도록 설정한다.

```
$ sudo apt install curl
```

// curl : 원격 서버에서 데이터를 다운로드 할 수 있는 패키지이다.

```
$ curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo
apt-key add -
```

// apt가 패키지를 인증하는데 사용하는 키 목록에 ros를 등록한다.



# ROS 설치(3)

WSL의 터미널에서 다음과 같이 ROS 설치 과정을 진행하자.

```
$ sudo apt update
```

// apt에서 다운로드 할 수 있는 패키지 목록을 최신으로 업데이트한다.

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

// ROS noetic을 풀버전으로 설치한다.

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

// ROS의 설정파일을 현재 터미널 shell에 반영한다.





# ROS 설치(4)

WSL의 터미널에서 다음과 같이 ROS 설치 과정을 진행하자.

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

// ROS에서 의존하는 중요한 패키지들을 설치한다.

```
$ sudo rosdep init
```

```
$ rosdep update
```

// ROS 도구들을 사용하기 위해 rosdep을 초기화한다.

## ROS 실행

\$ roscore

// ROSMASTER를 실행한다.

왼쪽과 같은 화면이 출력되면 성공!

```
ubuntu@DESKTOP-TTIPCK5:~$ roscore
... logging to /home/ubuntu/.ros/log/f4d785dc-7157-11ed-91a4-00155dd752be/roslaunch-DESKTOP-TTIPCK5-7000.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://DESKTOP-TTIPCK5:39457/
ros_comm version 1.15.15

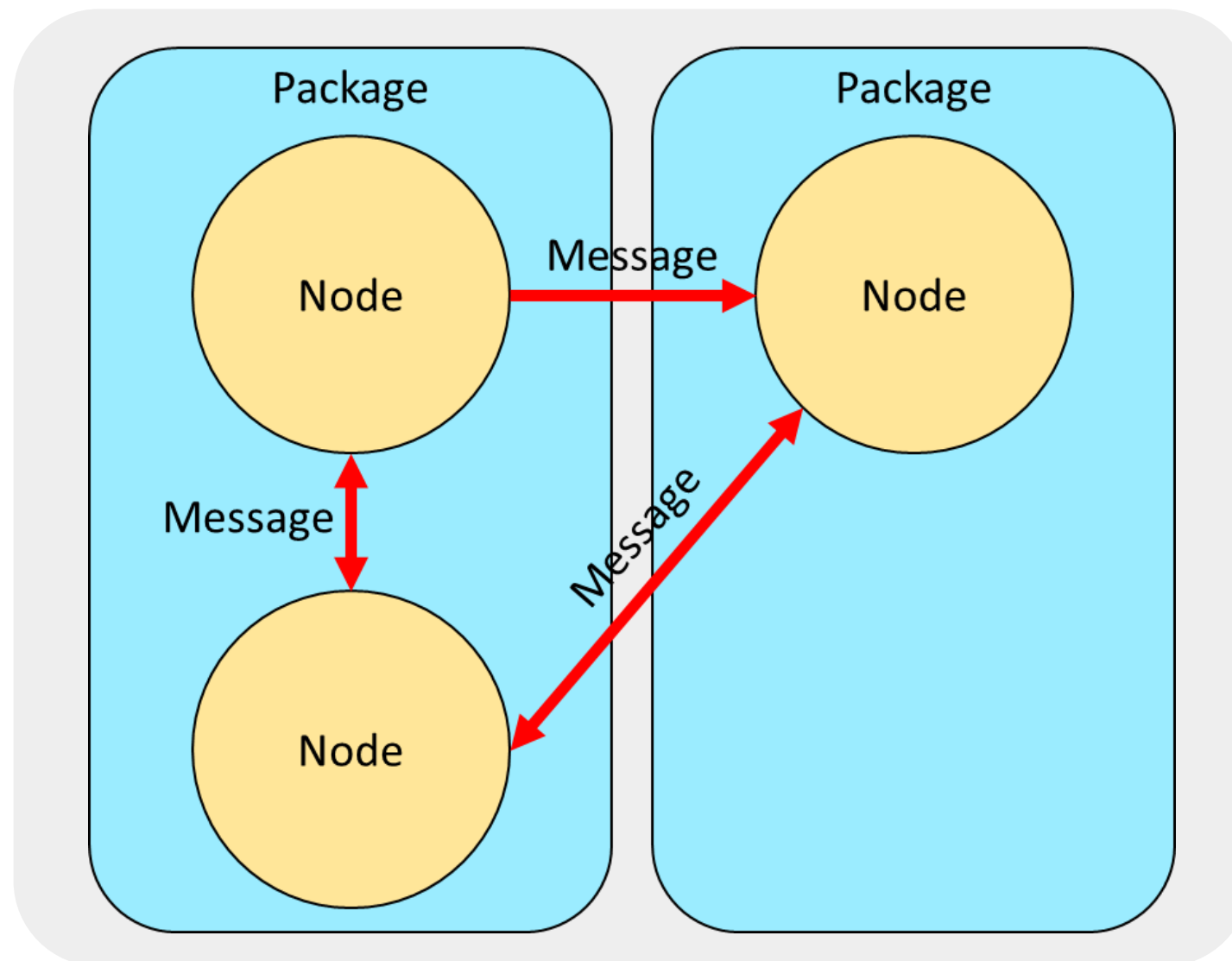
SUMMARY
=====

PARAMETERS
* /rostdistro: noetic
* /rosversion: 1.15.15

NODES

auto-starting new master
process[master]: started with pid [7008]
ROS_MASTER_URI=http://DESKTOP-TTIPCK5:11311/

setting /run_id to f4d785dc-7157-11ed-91a4-00155dd752be
process[rosout-1]: started with pid [7018]
started core service [/rosout]
```



## ROS 용어정리

- Node
  - 최소 단위의 실행가능한 프로세스
- Package
  - 하나 이상의 노드, 노드 실행을 위한 정보 등을 묶어 놓은 것.
- Message
  - 노드 간에 주고받는 데이터



# Workspace 만들기

로컬PC에 앞서 배운 ROS Package들을 저장하고 빌드할 수 있는 Workspace(작업공간)을 만들어 보자.

```
$ mkdir -p ~/catkin_ws/src
```

// mkdir : 디렉토리를 생성한다, -p : 존재하지 않는 중간 디렉토리를 자동 생성하는 옵션

```
$ cd ~/catkin_ws/
```

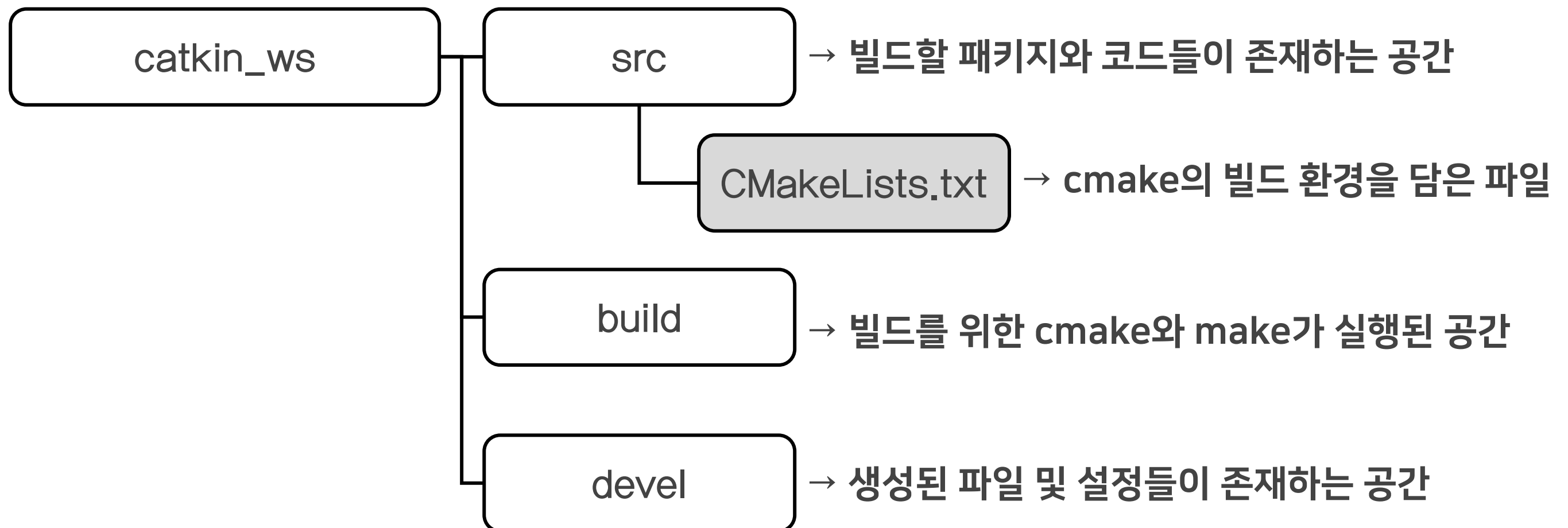
// cd : 디렉토리를 이동한다.

```
$ catkin_make
```

// catkin\_make : 현재 디렉토리의 src에 있는 모든 패키지를 빌드한다.

## Workspace 만들기

‘catkin\_make’를 수행하면 아래와 같이 디렉토리가 구성된다.







### source 명령어

‘source’ 명령어는 설정 파일에 있는 설정을 현재 터미널 shell에 반영한다.

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

// ROS의 설정파일을 현재 터미널 shell에 반영한다.

```
$ source ~/catkin_ws/devel/setup.bash
```

// workspace의 설정파일을 현재 터미널 shell에 반영한다.

→ 터미널을 열 때마다 source 명령어를 실행하는 것은 너무 번거롭다!



## ~/.bashrc 설정 파일

‘~/.bashrc’ 설정 파일은 터미널 shell을 새로 열 때마다 반영된다.  
따라서 ‘~/.bashrc’에 필요한 환경설정과 단축키를 넣으면 편리하다.

```
$ vi ~/.bashrc
```

// bashrc를 vi 에디터로 열람한다.

```
$ source ~/.bashrc
```

// 수정된 bashrc를 현재 터미널 shell에 반영한다.



## 간단한 **bashrc** 설정

```
export ROS_WS=$HOME/catkin_ws #define workspace
export ROS_MASTER_URI=http://localhost:11311 #define ROS_MASTER
export ROS_HOSTNAME=localhost #define ROS_HOSTNAME
export ROSCONSOLE_FORMAT='[${node}], ${walltime:%H:%M:%S.%f}]  ${message}'
#change console format

source /opt/ros/noetic/setup.bash #set ROS setting
source $ROS_WS/devel/setup.bash #set workspace setting

alias eb='vi ~/.bashrc' #edit bashrc
alias sb='source ~/.bashrc' #source bashrc
alias cw='cd $ROS_WS' #change directory as workspace
alias cs='cd $ROS_WS/src' #change directory as workspace/src
alias cm='catkin_make -C $ROS_WS' #build workspace
alias rs='rm -rf $ROS_WS/devel/* && rm -rf $ROS_WS/build/*' #remove devel & build
alias rl='rosclean purge -y' #clean ROS log
```

// 작성이 끝나면 `esc → shift+; → wq`로 저장한 뒤 vi 에디터를 닫는다.

// 터미널에서 `'source ~/.bashrc'`를 입력하여 설정을 적용한다.



## package 만들기

ROS에서 package는 하나 이상의 노드, 노드 실행을 위한 정보 등을 묶어 놓은 것이다.

```
$ cd ~/catkin_ws/src
```

```
// catkin_ws/src 디렉토리로 이동한다. package는 이곳에 저장되어야 한다.
```

```
$ catkin_create_pkg tutorial_package roscpp std_msgs
```

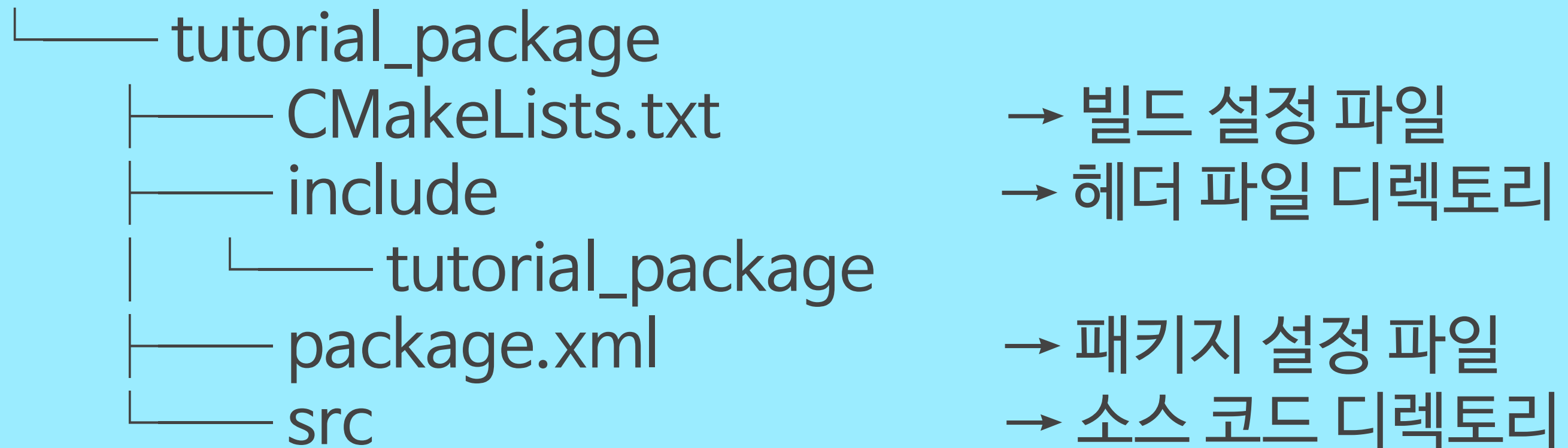
```
// catkin_create_pkg : 패키지를 생성하는 명령어이다. 아래 규칙대로 작성한다.
```

```
// catkin_create_pkg [package_name] [depend1] [depend2] ...
```

```
// 위 명령은 roscpp과 std_msgs 패키지를 의존하는 tutorial_package를 생성한다.
```

## package 만들기

생성된 tutorial\_package는 아래와 같이 자동으로 구성된다.



<tutorial\_package의 구성>



## package.xml 둘러보기

package.xml은 패키지의 정보를 담은 xml파일이며, 패키지 이름, 저작자, 라이선스, 의존성 패키지 등을 기술하고 있다.

```
$ cd ~/catkin_ws/src/tutorial_package  
// tutorial_package 디렉토리로 이동한다.
```

```
$ vi package.xml  
// package.xml을 vi에디터로 열람한다.
```





## package.xml 둘러보기

```

<?xml version="1.0"?> → 문서 문법 정의
<package format="2"> → 패키지, 포맷 버전
  <name>tutorial_package</name> → 이름
  <version>0.0.0</version> → 버전
  <description>The tutorial_package package</description> → 설명
  <maintainer email="chris20@todo.todo">chris20</maintainer> → 관리자 정보
  <license>TODO</license> → 라이선스 정보(BSC, MIT, Apache, GPLv3, LGPLv3 ...)
  <buildtool_depend>catkin</buildtool_depend> → 의존하는 빌드 시스템
  <build_depend>roscpp</build_depend> → 빌드할 때 의존하는 패키지
  <build_export_depend>std_msgs</build_export_depend>
  <build_export_depend>roscpp</build_export_depend> → build_depend가 포함된 소스를 export할 때 의존하는 패키지
  <build_export_depend>std_msgs</build_export_depend>
  <exec_depend>roscpp</exec_depend> → 패키지 실행할 때의 의존하는 패키지
  <exec_depend>std_msgs</exec_depend>
  <export></export>
</package>

```

[Package format 2 \(recommended\) — catkin 0.6.19 documentation \(ros.org\)](#) // 포맷에 대한 자세한 설명은 링크를 참조.



# CMakeLists.txt 둘러보기

Catkin 툴로 빌드를 수행할 때 설정할 정보가 담긴 파일이다.

```
$ cd ~/catkin_ws/src/tutorial_package  
// tutorial_package 디렉토리로 이동한다.
```

```
$ vi CMakeLists.txt  
// CMakeLists.txt를 vi에디터로 열람한다.
```



## CMakeLists.txt 둘러보기

```

cmake_minimum_required(VERSION 3.0.2) → 소스 코드가 빌드될 때 필요한 최소한의 cmake 버전
project(tutorial_package) → Package 이름. package.xml의 <name>과 동일해야 함.

find_package(catkin REQUIRED COMPONENTS → 특정 라이브러리의 경로를 찾는다. (이후 library를 링크하고 include 해야 한다.)
→ REQUIRED : 찾지 못하면 cmake 종료. COMPONENT : 특정 컴포넌트 호출)
  roscpp
  std_msgs
) → 소스 코드가 빌드될 때 요구되는 패키지 구성요소. package.xml의 <build_depend>을 참조

catkin_package(
#  INCLUDE_DIRS include → cmake 문법을 준수하지 않는 패키지
#  LIBRARIES tutorial_package → cmake 문법을 준수하지 않는 패키지
#  CATKIN_DEPENDS roscpp std_msgs → 다른 패키지가 export할 때 의존하는 패키지
#  DEPENDS system_lib → cmake 문법을 준수하는 패키지
) → catkin package의 옵션 기술

include_directories(→ include할 directory 추가
# include
  ${catkin_INCLUDE_DIRS} → catkin_이 붙은 include directory를 include하겠다.
)

```

[catkin/CMakeLists.txt - ROS Wiki](#) // 포맷에 대한 자세한 설명은 링크를 참조



# publisher & subscriber 구현하기

0부터 시작하여 1초마다 1씩 늘어나는 메시지를 보내는 publisher를 구현해보자!

0초 : 0

1초 : 1

2초 : 2

3초 : 3

4초 : 4

5초 : 5

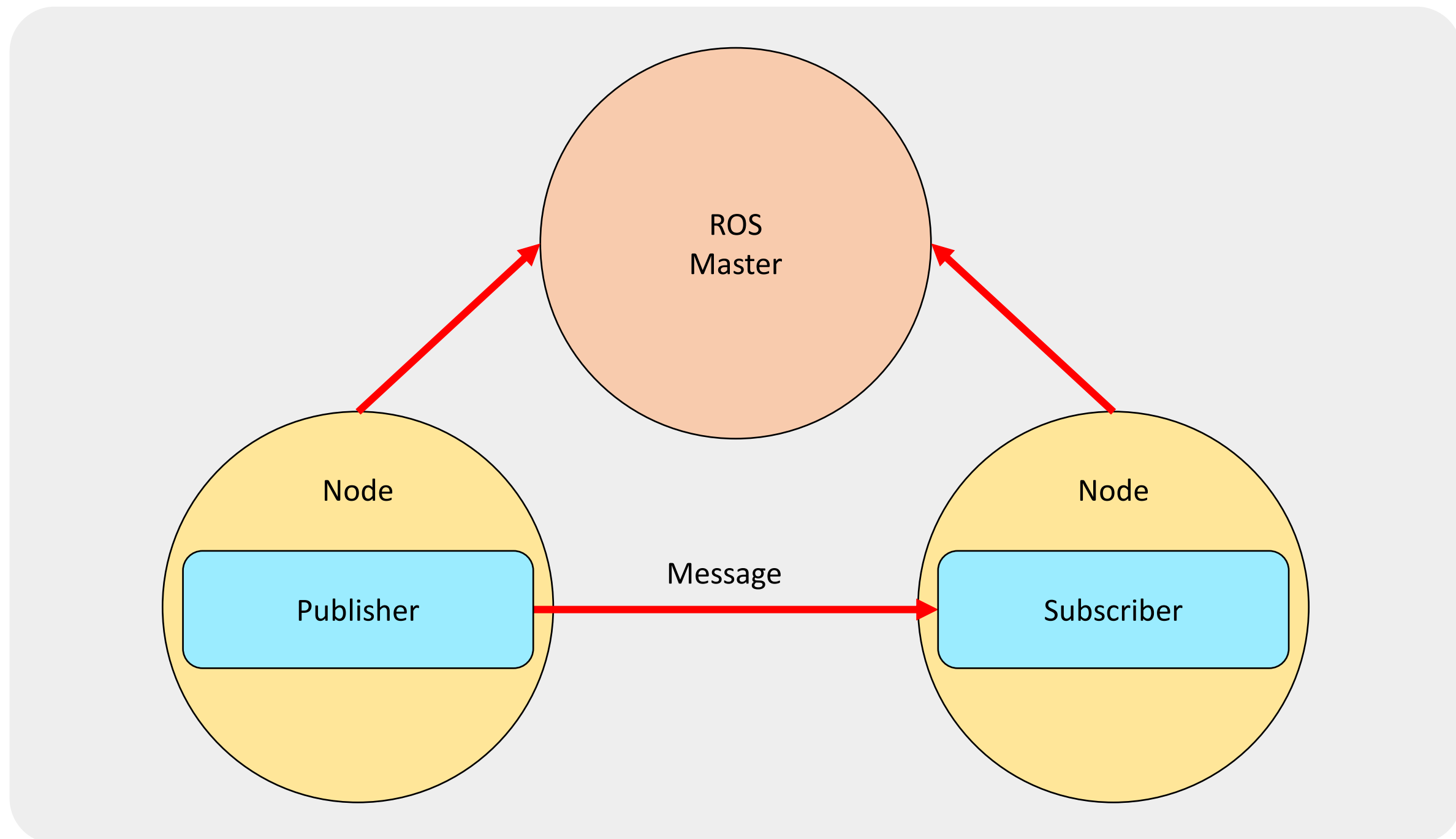
...

publisher에서 보내는 메시지를 받아서 출력하는 subscriber를 구현해보자!

## Topic의 이해

모든 노드는 통신 전에 **ROS MASTER**에 정보를 등록해야 한다.

Topic은 단방향이며 연속적으로 통신한다.



〈Topic을 통해 Node들이 통신하는 모습〉



## Publisher Node 생성하기

```
$ cd ~/catkin_ws/src/tutorial_package/src  
// tutorial_package의 src 디렉토리로 이동한다.
```

```
$ vi pub_node.cpp  
// vi 에디터로 pub_node 작성을 시작한다.
```

## Publisher Node 생성하기

```
#include <ros/ros.h> → ROS 기본 헤더파일
#include <std_msgs/Int16.h> → std_msgs의 Int16 메시지 헤더파일

int main(int argc, char** argv)
{
    ros::init(argc, argv, "pub_node"); → 노드명 초기화
    ros::NodeHandle nh; → ROS 시스템과 통신을 위한 노드 핸들 선언

    ros::Publisher pub_number = nh.advertise<std_msgs::Int16>("/test/topic", 10, true);
    → 퍼블리셔 선언. 메시지형, 토픽명, 토픽 큐사이즈가 담김

    ros::Rate loop_rate(1); → 루프 주기. "1"은 1Hz.
    std_msgs::Int16 count; → std_msgs의 Int16형으로 count라는 메시지 선언

    while(ros::ok()) → rosmaster가 살아있는 동안
    {
        pub_number.publish(count); → count라는 메시지를 publish한다.
        ROS_INFO("pub %d", count.data); → count.data를 표시한다.
        count.data++; → count.data 1씩 증가

        ros::spinOnce(); → 현재까지 요청된 콜백함수를 모두 호출하고 넘어감.
        loop_rate.sleep(); → 위의 정해진 주기에 따라 슬립에 들어간다.
    }
}
```

// 작성이 끝나면 esc → shift+; → wq로 저장한 뒤 vi 에디터를 닫는다.



## Publisher Node 코드

```
#include <ros/ros.h>
#include <std_msgs/Int16.h>

int main(int argc, char** argv)
{
    ros::init(argc, argv, "pub_node");
    ros::NodeHandle nh;

    ros::Publisher pub_number = nh.advertise<std_msgs::Int16>("/test/topic", 10, true);

    ros::Rate loop_rate(1);
    std_msgs::Int16 count;

    while(ros::ok())
    {
        pub_number.publish(count);
        ROS_INFO("pub %d", count.data);
        count.data++;

        ros::spinOnce();
        loop_rate.sleep();
    }
}
```



## Subscriber Node 생성하기

```
$ cd ~/catkin_ws/src/tutorial_package/src  
// tutorial_package의 src 디렉토리로 이동한다.
```

```
$ vi sub_node.cpp  
// vi 에디터로 sub_node 작성을 시작한다.
```



## Subscriber Node 생성하기

```
#include <ros/ros.h>
#include <std_msgs/Int16.h>

void NumberCallback(const std_msgs::Int16 &msg) → /test/topic에 대한 콜백함수
{
    ROS_INFO("sub %d", msg.data); → 터미널에 구독한 데이터 출력
}

int main(int argc, char** argv)
{
    ros::init(argc, argv, "sub_node");
    ros::NodeHandle nh;

    ros::Subscriber sub_number = nh.subscribe("/test/topic", 10, NumberCallback); → 서브스크라이버 선언.
    토픽명, 토픽 큐사이즈, 콜백함수가 담김

    ros::spin(); → 프로그램이 종료될 때까지 콜백함수를 끊임없이 처리
}
```

// 작성이 끝나면 esc → shift+; → wq로 저장한 뒤 vi 에디터를 닫는다.



# Subscriber Node 코드

```
#include <ros/ros.h>
#include <std_msgs/Int16.h>

void NumberCallback(const std_msgs::Int16 &msg)
{
    ROS_INFO("sub %d", msg.data);
}

int main(int argc, char** argv)
{
    ros::init(argc, argv, "sub_node");
    ros::NodeHandle nh;

    ros::Subscriber sub_number = nh.subscribe("/test/topic", 10, NumberCallback);

    ros::spin();
}
```

## CMakeLists.txt에 새 노드 등록하기

```
$ vi ~/catkin_ws/src/tutorial_package/CMakeLists.txt  
// tutorial_package의 CMakeLists.txt를 vi에디터로 열람한다.
```

아래와 같이 내용을 추가한다. pub\_node와 sub\_node를 실행파일로 등록하겠다는 의미이다.

```
add_executable(pub_node src/pub_node.cpp) → 소스코드를 빌드하여 실행파일 생성  
add_dependencies(pub_node ${PROJECT_NAME}_EXPORTED_TARGETS ${catkin_EXPORTED_TARGETS}) → 빌드할 파일 타겟에 의존성 추가  
target_link_libraries(pub_node ${catkin_LIBRARIES}) → 실행할 파일 타겟에 라이브러리 링크  
  
add_executable(sub_node src/sub_node.cpp)  
add_dependencies(sub_node ${PROJECT_NAME}_EXPORTED_TARGETS ${catkin_EXPORTED_TARGETS})  
target_link_libraries(sub_node ${catkin_LIBRARIES})
```

// 작성이 끝나면 esc → shift+; → wq로 저장한 뒤 vi 에디터를 닫는다.

## Package 빌드하기

```
$ cd ~/catkin_ws/
```

// cd : 디렉토리를 이동한다.

```
$ catkin_make
```

// catkin\_make : 현재 디렉토리의 src에 있는 모든 패키지를 빌드한다.

위와 같이 입력하거나 bashrc에 등록했던 단축키 'cm'을 사용하여 빌드할 수 있다.



## 노드 실행하기

```
$ roscore
```

// rosmaster를 실행한다.

```
$ rosrunc tutorial_package sub_node
```

//tutorial\_package의 sub\_node를 실행한다.

```
$ rosrunc tutorial_package pub_node
```

//tutorial\_package의 pub\_node를 실행한다.

\* 새 터미널 열기 : Ctrl + Shift + `

\* 터미널 분할하기 : Ctrl + Shift + 5



## 실행 결과

publisher에서 보낸 메시지를 subscriber가 받는 모습을 확인할 수 있다.

```

터미널
o chris20@DESKTOP-6906KK0:~/tutorial_ws/src$ roscore
... logging to /home/chris20/.ros/log/723ff70c-6a55-11ed-afe3-00155d662815/roslaunch-DESKTOP-6906KK0-1280.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://localhost:34191/
ros_comm version 1.15.14

SUMMARY
=====

PARAMETERS
* /rostdistro: noetic
* /rosversion: 1.15.14

NODES

auto-starting new master
process[master]: started with pid [1288]
ROS_MASTER_URI=http://localhost:11311/

setting /run_id to 723ff70c-6a55-11ed-afe3-00155d662815
process[rosout-1]: started with pid [1299]
started core service [/rosout]
[]

o chris20@DESKTOP-6906KK0:~/tutorial_ws/src$ rosrunc tutorial_pa
ckage sub_node
[/sub_node, 11:11:46.472470] sub 0
[/sub_node, 11:11:47.228965] sub 1
[/sub_node, 11:11:48.229074] sub 2
[/sub_node, 11:11:49.229098] sub 3
[/sub_node, 11:11:50.229392] sub 4
[/sub_node, 11:11:51.229307] sub 5
[/sub_node, 11:11:52.229075] sub 6
[/sub_node, 11:11:53.228974] sub 7
[/sub_node, 11:11:54.228948] sub 8
[/sub_node, 11:11:55.229001] sub 9
[/sub_node, 11:11:56.228949] sub 10
[/sub_node, 11:11:57.229092] sub 11
[/sub_node, 11:11:58.228962] sub 12
[/sub_node, 11:11:59.228955] sub 13
[/sub_node, 11:12:00.229029] sub 14
[/sub_node, 11:12:01.229131] sub 15
[/sub_node, 11:12:02.229187] sub 16
[/sub_node, 11:12:03.229025] sub 17
[/sub_node, 11:12:04.228941] sub 18
[/sub_node, 11:12:05.228915] sub 19
[/sub_node, 11:12:06.228973] sub 20
[/sub_node, 11:12:07.228923] sub 21
[/sub_node, 11:12:08.229117] sub 22
[/sub_node, 11:12:09.229367] sub 23
[]

o chris20@DESKTOP-6906KK0:~/tutorial_ws/src$ rosrunc tutorial_p
ackage pub_node
[/pub_node, 11:11:46.228940] pub 0
[/pub_node, 11:11:47.228865] pub 1
[/pub_node, 11:11:48.228853] pub 2
[/pub_node, 11:11:49.228918] pub 3
[/pub_node, 11:11:50.228894] pub 4
[/pub_node, 11:11:51.228961] pub 5
[/pub_node, 11:11:52.228918] pub 6
[/pub_node, 11:11:53.228859] pub 7
[/pub_node, 11:11:54.228848] pub 8
[/pub_node, 11:11:55.228855] pub 9
[/pub_node, 11:11:56.228864] pub 10
[/pub_node, 11:11:57.228871] pub 11
[/pub_node, 11:11:58.228891] pub 12
[/pub_node, 11:11:59.228861] pub 13
[/pub_node, 11:12:00.228858] pub 14
[/pub_node, 11:12:01.228902] pub 15
[/pub_node, 11:12:02.229080] pub 16
[/pub_node, 11:12:03.228914] pub 17
[/pub_node, 11:12:04.228854] pub 18
[/pub_node, 11:12:05.228836] pub 19
[/pub_node, 11:12:06.228882] pub 20
[/pub_node, 11:12:07.228855] pub 21
[/pub_node, 11:12:08.228881] pub 22
[/pub_node, 11:12:09.229005] pub 23
[]

```

# pigpio 라이브러리

pigpio는 GPIO pin들을 제어할 수 있는 라즈베리파이(RPI)용 라이브러리이다.

pigpiod라는 데몬 형태로 사용할 수 있으며 로우레벨 엑세스를 지원함.

// 데몬(daemon) : 사용자가 직접적으로 제어하지 않고 백그라운드에서 돌면서 여러 작업을 하는 프로그램.

// 로우레벨 : 기계어 및 하드웨어에 가까운 단계를 의미함.

로우레벨 제어로 GPIO 자체에 대한 정교한 조작으로 Software PWM을 쉽게 이용 가능함.

// PWM은 펄스 폭 변조로 추후 설명.



# pigpio 설치하기

<http://abyz.me.uk/rpi/pigpio/download.html> 홈페이지 참조.

```
$ wget https://github.com/joan2937/pigpio/archive/master.zip
```

// wget : 웹에서 파일을 다운로드 할수 있는 리눅스 명령어.

```
$ unzip master.zip
```

// unzip : zip파일 압축을 푸는 리눅스 명령어.

```
$ cd pigpio-master
```

// pigpio-master로 이동한다.

```
$ make
```

// makefile에 기술된 Shell 명령어들을 순차적으로 실행하여 컴파일을 수행하는 리눅스 명령어.

```
$ sudo make install
```

// make로 만들어진 설치파일을 설치하는 리눅스 명령어



# igpio 실행 및 테스트하기

```
$ sudo pigpiod
```

// pigpio 데몬(pigpiod)을 실행한다. → 모터 노드를 실행하기 전에 1회 실행해 줄 것.

```
$ sudo ./x_pigpiod_if2
```

// pigpio 설치시 제대로 설치되었는지 확인한다.

```
$ sudo killall pigpiod
```

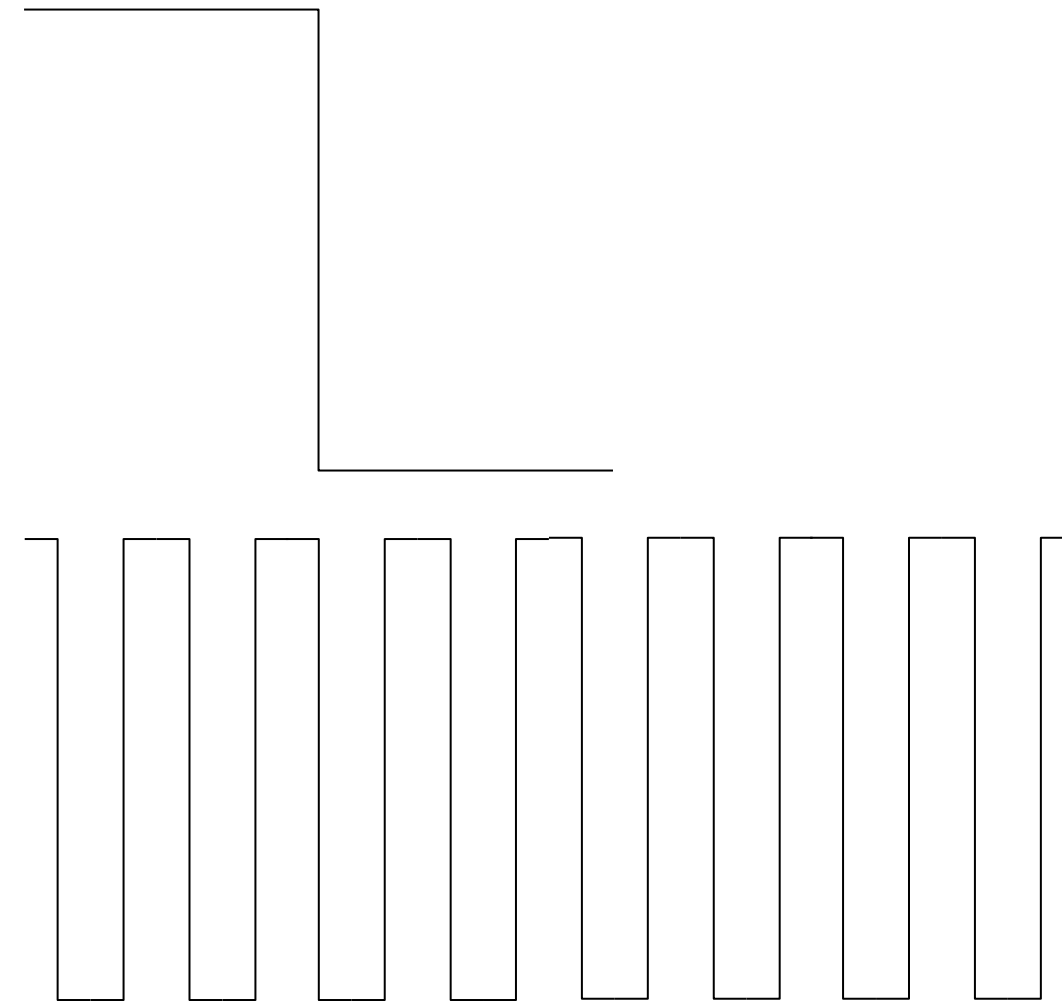
//pigpiod가 비정상일시, 멈추고 다시 시작할 때 사용한다.

# 디지털 신호로 모터의 출력을 제어하는 방법

0과 1로 구성된 디지털 신호로 어떻게 모터의 출력을 제어할 수 있을까?

0은 모터가 꺼짐. 1은 모터가 켜짐.

이것을 매우 짧은 시간동안 계속 반복한다면 어떻게 될까?



## PWM (펄스 폭 변조, Pulse Width Modulation)

그리고 그 신호의 폭을 조절한다면?

→ 이 제어 방법을 PWM이라고 한다.

PWM의 구성은 아래와 같다.

TON : 제어 대상의 출력이 1인 구간.

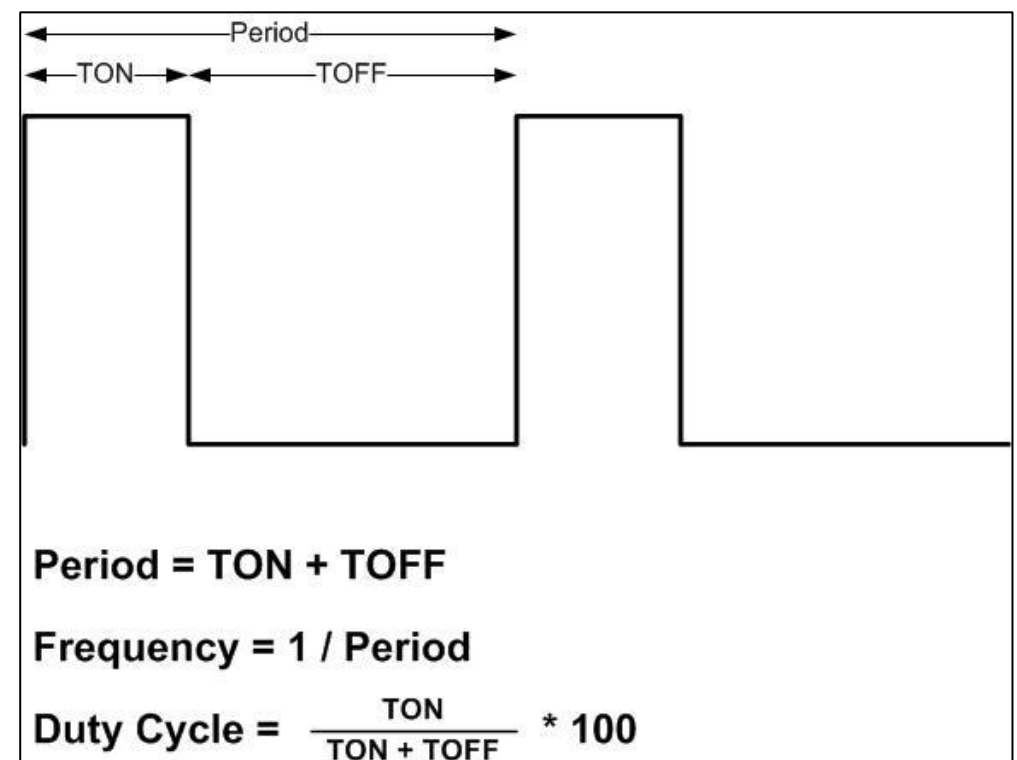
TOFF : 제어 대상의 출력이 0인 구간.

Period : TON과 TOFF를 나누는 주기.

Frequency : 단위 시간 동안의 주기 빈도. (주파수)

Duty Cycle : 주기 중 TON의 비율. (출력의 퍼센트)

D: 0%



## pwm\_test.cpp 둘러보기

모터가 PWM 출력이 제대로 되는지 확인해보는 코드

CheckPigpio 함수 : pigpio 데몬과 연결을 시도하고 연결 여부를 점검하는 함수

InitPwm 함수 : GPIO 핀 모드와 PWM의 dutycycle, frequency를 초기화하는 함수

SetPwmDutycycle 함수 : 모든 모터에 PWM 출력을 활성화하는 함수

main 함수 : pigpio 데몬과 연결하고 모든 모터를 10% dutycycle로 출력함



## pwm\_test.cpp 둘러보기

모터가 PWM 출력이 제대로 되는지 확인해보는 코드

CheckPigpio 함수 : pigpio 데몬과 연결을 시도하고 연결 여부를 점검하는 함수

InitPwm 함수 : GPIO 핀 모드와 PWM의 dutycycle, frequency를 초기화하는 함수

SetPwmDutycycle 함수 : 모든 모터에 PWM 출력을 활성화하는 함수

main 함수 : pigpio 데몬과 연결하고 모든 모터를 10% dutycycle로 출력함

## 라이브러리 및 상수, 변수 설정

```
#include <ros/ros.h>
#include <pigpiod_if2.h>

const int k_control_cycle = 10;
const int k_pins = 8;
const int k_pwm_range = 1000;
const int k_pwm_frequency = 20000;
const int k_pin_nums[k_pins] = {17,4,22,27,6,5,19,13};
int pi_num;
```

# CheckPigpio 함수

```
bool CheckPigpio()
{
    pi_num = pigpio_start(NULL, NULL);

    if (pi_num < 0)
    {
        ROS_ERROR("PI number is %d", pi_num);
        ROS_ERROR("PIGPIO connection failed.");
        return false;
    }

    ROS_INFO("Setup Finished.");
    return true;
}
```

## InitPwm 함수

```
void InitPwm()
{
    for (int i = 0; i < k_pins; i++)
    {
        set_mode(pi_num, k_pin_nums[i], PI_OUTPUT);
        set_PWM_range(pi_num, k_pin_nums[i], k_pwm_range);
        set_PWM_frequency(pi_num, k_pin_nums[i], k_pwn_frequency);
    }
}
```

# SetPwmDutycycle 함수

```
void SetPwmDutycycle(int rate)
{
    if(rate < 0 || rate > k_pwm_range)
    {
        ROS_WARN("Invalid Dutycycle.");
        for (int i = 0; i < k_pins; i++) set_PWM_dutycycle(pi_num, k_pin_nums[i], 0);
    }

    for (int i = 0; i < k_pins; i++) set_PWM_dutycycle(pi_num, k_pin_nums[i], rate);
}
```

## main 함수

```
int main(int argc, char **argv)
{
    ros::init(argc, argv, "pwm_test");
    ros::NodeHandle nh;
    ros::Rate loop_rate(k_control_cycle);

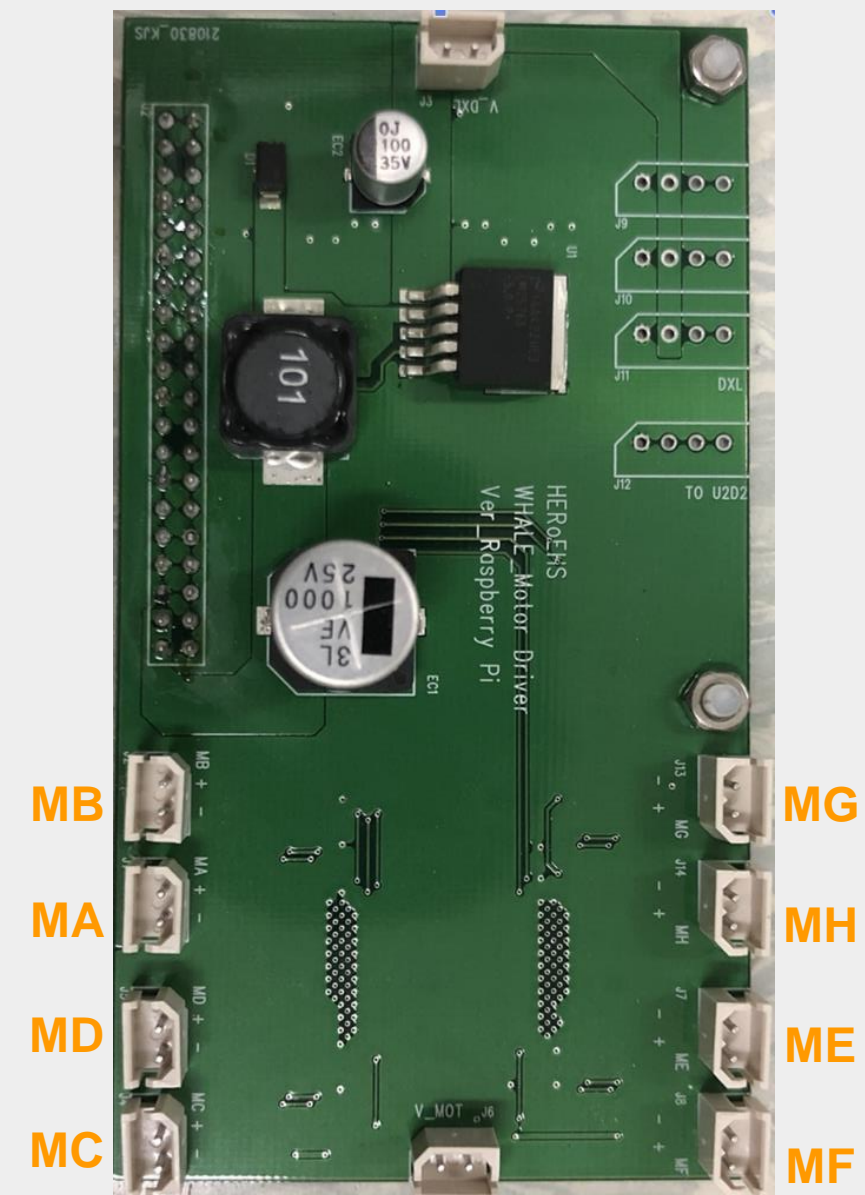
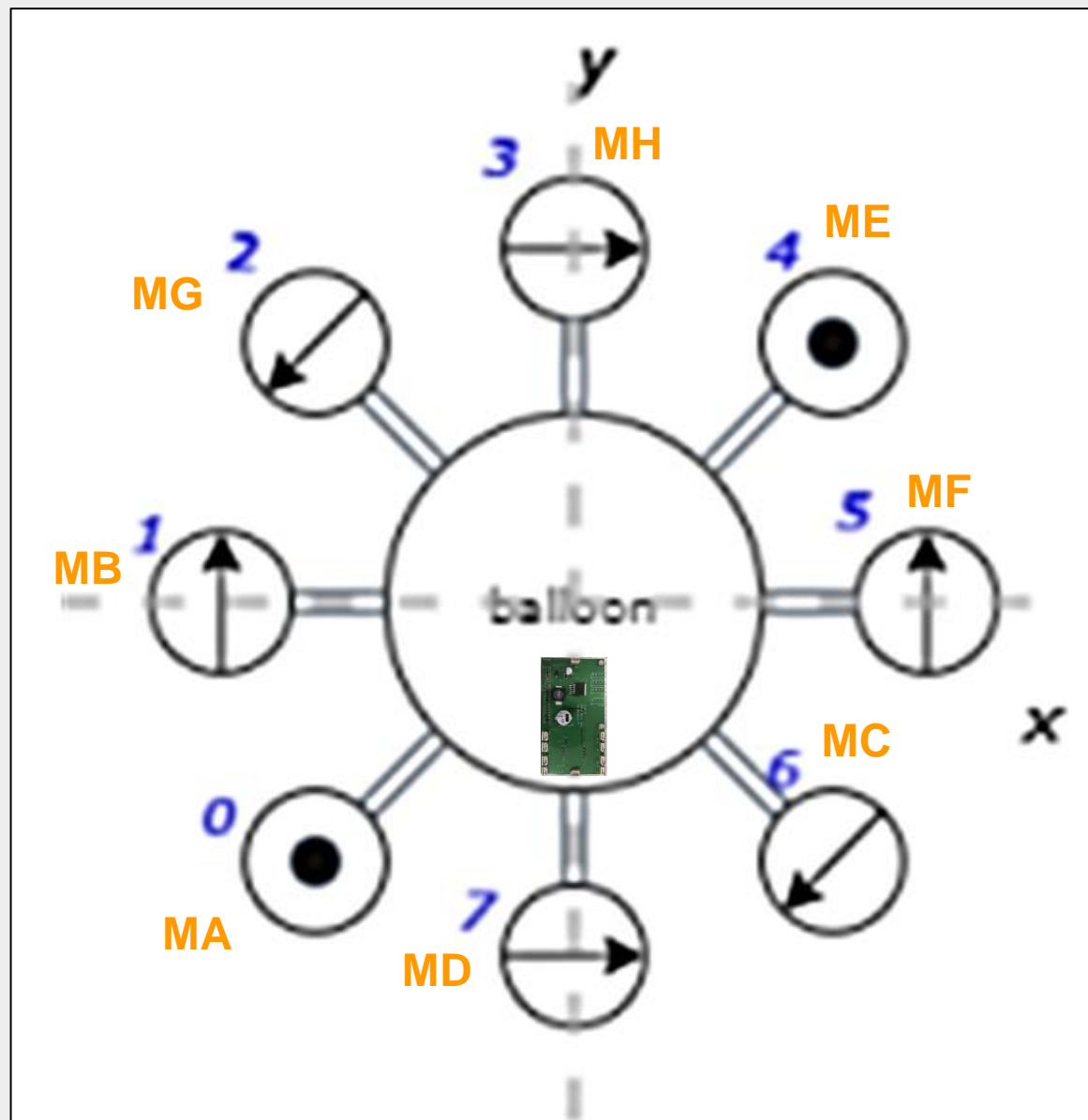
    if(!CheckPigpio()) return -1;
    InitPwm();

    while (ros::ok())
    {
        SetPwmDutycycle(100);
        loop_rate.sleep();
        ros::spinOnce();
    }

    SetPwmDutycycle(0);
    return 0;
}
```

### 비행로봇 모터 배선

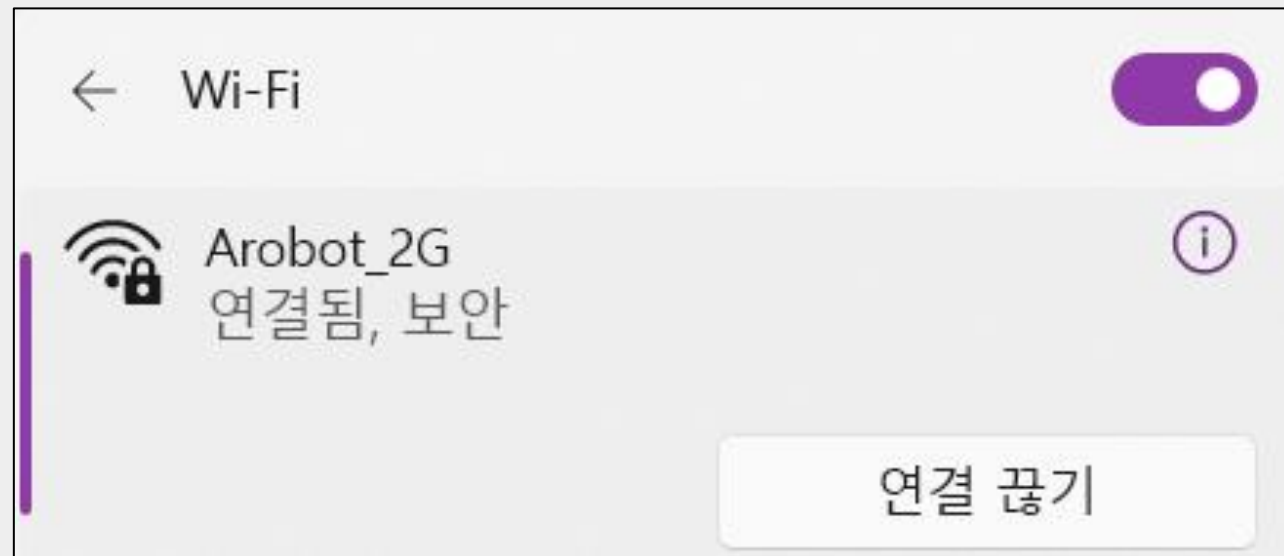
프로펠러의 방향을 잘 보고 모터선을 모터 드라이버에 연결하자.



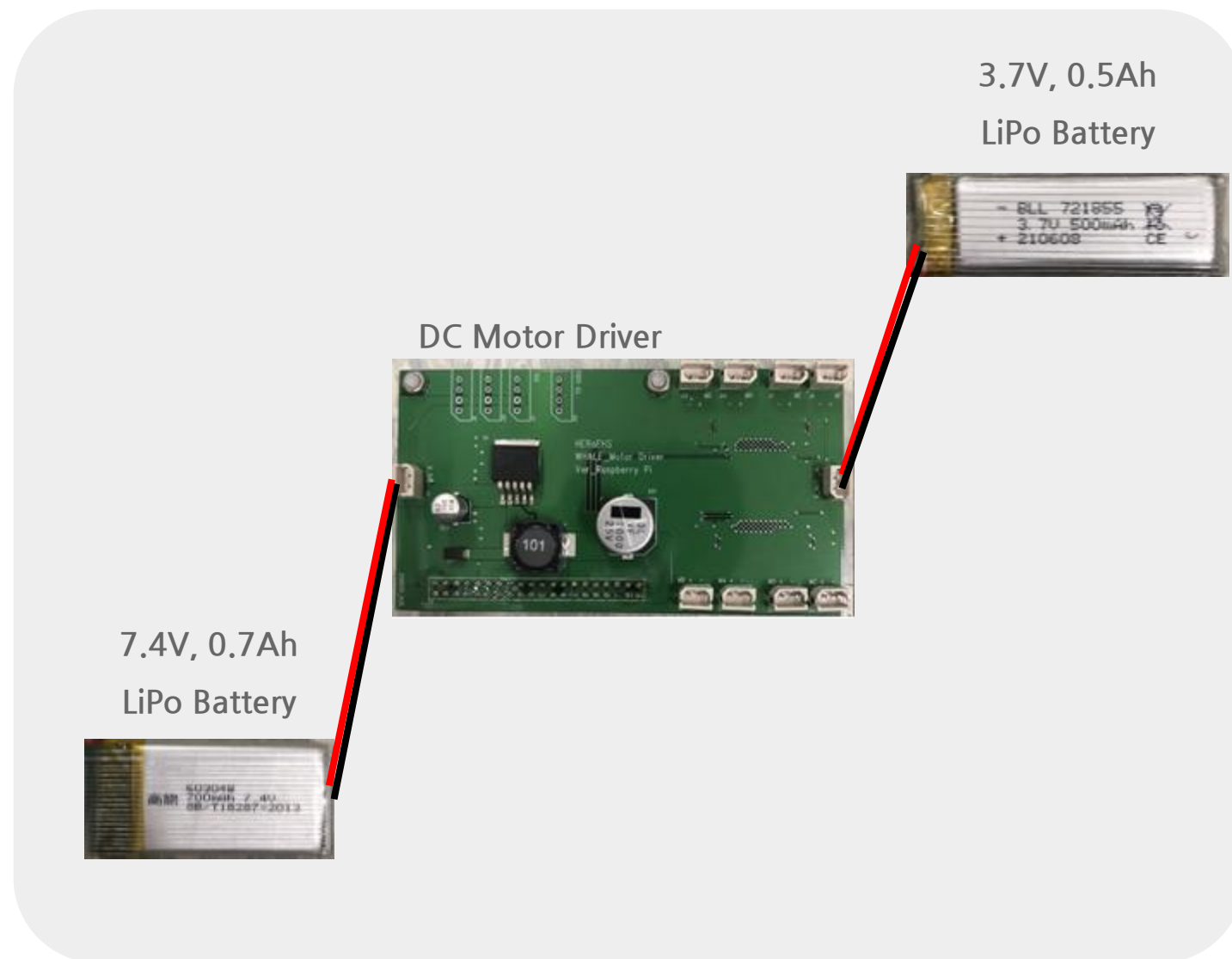
〈풍선형 비행 로봇의 이동 방향(좌), 비행 로봇용 모터 드라이버 보드(우)〉

## 내부 네트워크로 연결

- 로컬 PC와 RPI를 같은 네트워크에 연결하기 위한 작업
- 로컬 PC에서 **Arobot\_2G / humansociety1!**로 Wi-Fi를 잡을 것. (인터넷은 지원하지 않음.)
- RPI는 자동으로 연결되게끔 설정되어 있음.
- RPI가 연결되지 않으면 RPI에 직접 접속하여 설정을 수정해야 함.







## 비행로봇의 배터리 연결

- 7.4V, 0.7Ah 배터리는 RPI 전원
- 3.7V 0.5Ah 배터리는 DC모터 전원
- **안전상 RPI 배터리를 먼저 꽂고 몇 초 뒤에 DC모터 배터리를 꽂을 것.**
- RPI가 부팅하면서 초기화되지 않은 GPIO 핀의 영향으로 모터가 움직일 수 있음.

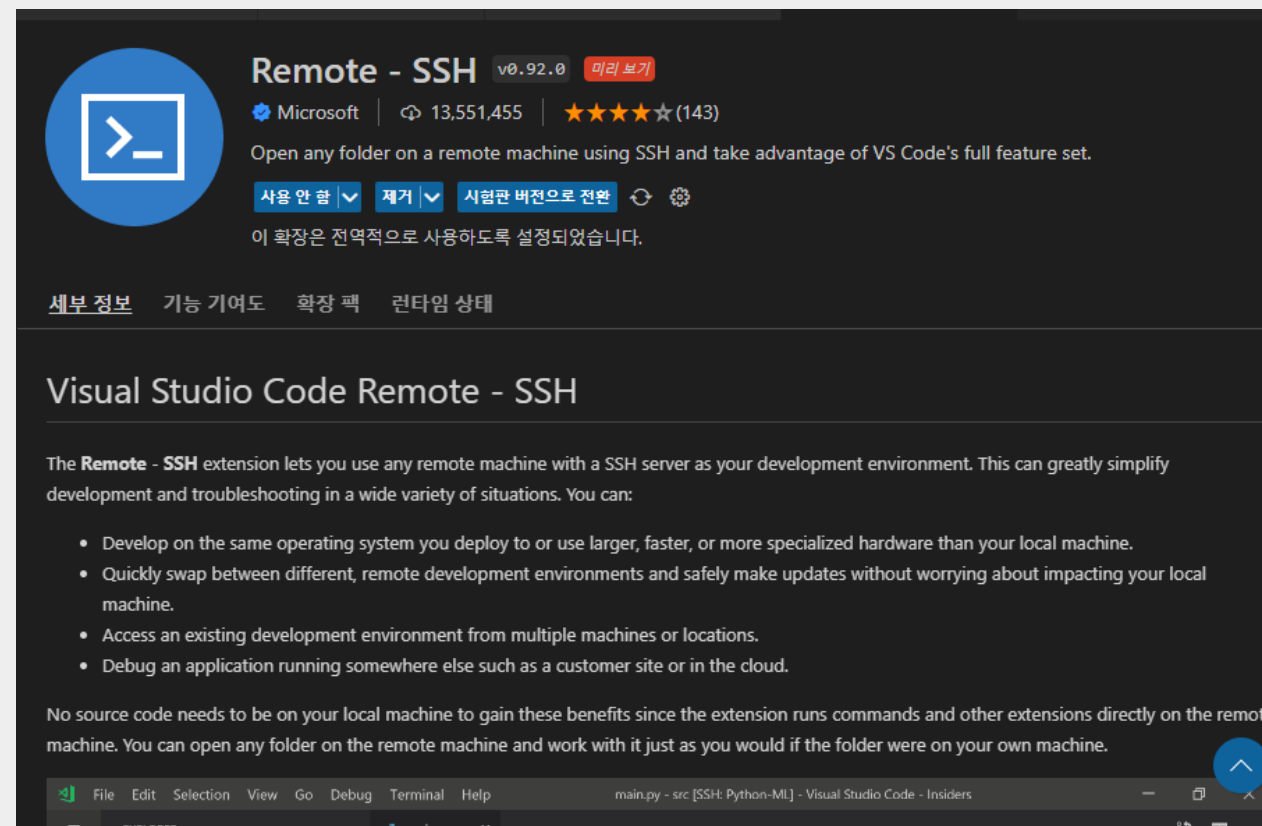
## 로컬 PC 코드 이동하기

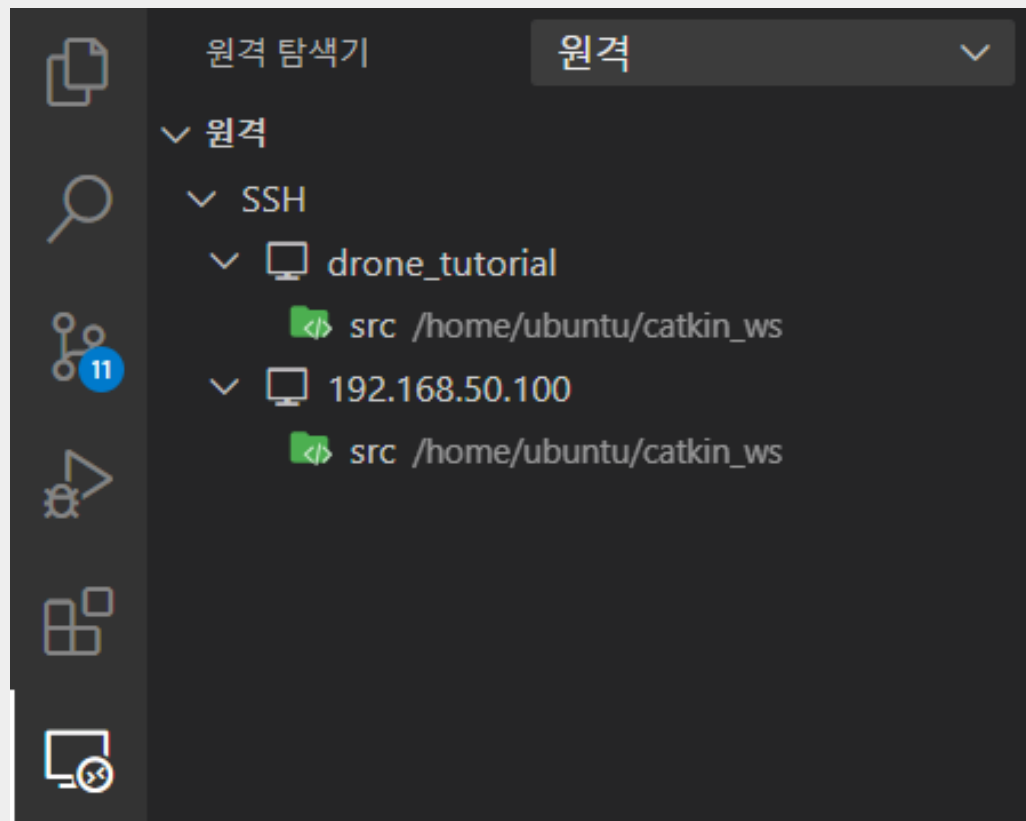
```
$ scp -r ~/catkin_ws/src/drone_tutorial-main/ ubuntu@192.168.50.XXX:~/catkin_ws/src
```

// scp : 현재 PC의 파일을 원격 PC로 복사한다. -r은 디렉토리 째 보내는 경우 사용하는 옵션

## Remote - SSH의 설치

- Visual Studio Code(VS Code)에서 같은 네트워크 상의 외부기기에 원격 접속할 수 있는 도구
- VS Code의 확장(Ctrl + Shift + X)에서 'Remote - SSH'를 검색하여 설치



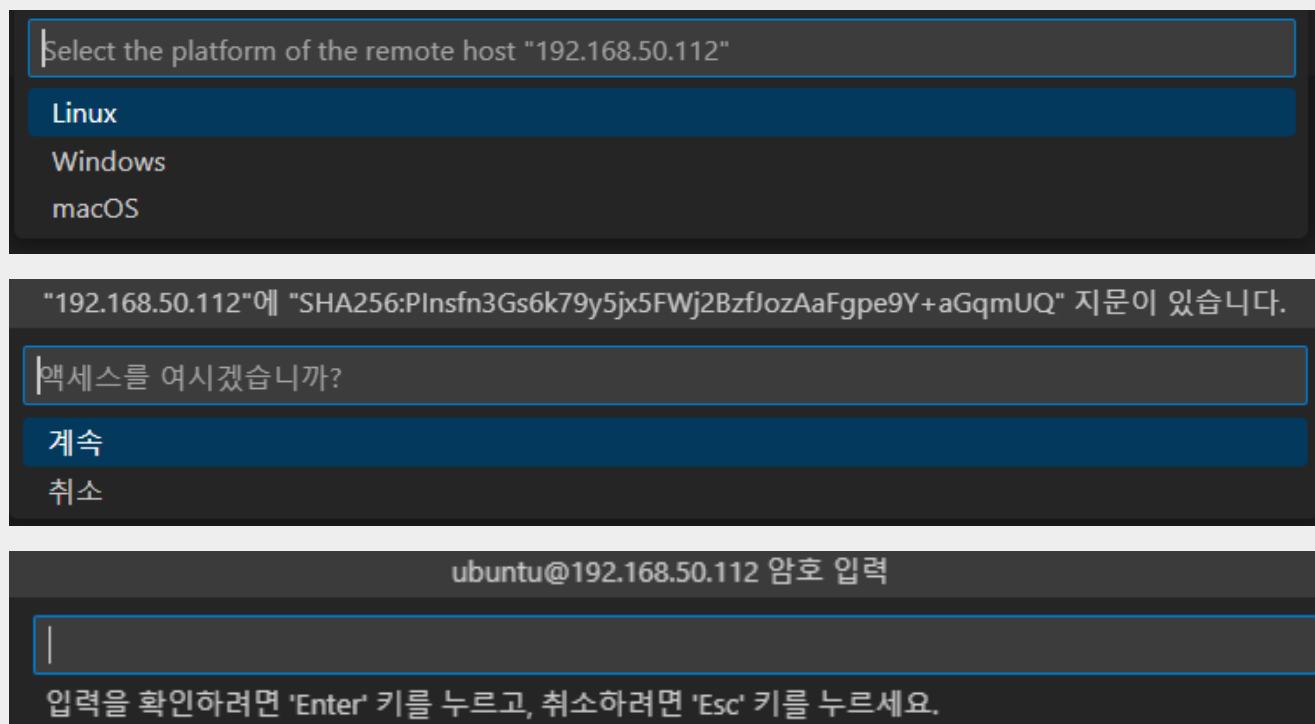


## Remote - SSH를 통한 RPI 접속(1)

- VS Code 왼쪽 탭에 원격탐색기를 클릭
- 상단 탭을 '원격'으로 변경
- SSH 우측의 '+ 새 원격' 버튼을 눌러서 새 원격 추가
- 연결 명령 '**ssh ubuntu@192.168.50.XXX**' 입력
- 원격 탭 새로 고침
- SSH 탭에서 아이피를 찾아 '새 창에서 연결' 클릭

## Remote - SSH를 통한 RPI 접속(2)

- 새 VS Code 창에서 'Linux' 선택
- '계속' 클릭
- RPI 암호 입력 (암호 : 12341234)
- VS Code에서 RPI 접속 완료



만약에 접속에 실패한다면 로컬 PC 터미널에서  
**\$ ssh ubuntu@192.168.50.XXX**로 접속할 것

# RPI에서 **pwm\_test** 노드 실행

SSH로 접속한 RPI VS Code 창에서 새 터미널 실행(Ctrl + Shift + `)

```
$ cm
```

//catkin\_make의 단축키. 작업공간 내의 소스코드들을 빌드한다.

//만약 clock skew 에러로 빌드에 실패할 경우, sudo date -s "2023-01-26 tt:mm"을 입력.

```
$ roscore
```

```
$ rosrn drone_tutorial pwm_test
```

// 비행로봇의 8개 모터가 10% dutycycle로 움직인다.



# drone\_teleop.cpp 둘러보기

키보드 키를 통해 비행 로봇을 원격조종 할 수 있도록 토픽을 보내는 코드

KeyboardReader 클래스 : 키보드의 입력에 관한 함수들이 모여 있는 클래스

DroneTeleop 클래스 : 비행로봇의 원격조종 키입력과 토픽 발행에 대한 클래스

main 함수 : DroneTeleop 클래스로 객체를 생성하여 원격조종을 실행하는 함수



# 라이브러리 및 상수, 변수 설정

```
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include <signal.h>
#include <stdio.h>
#include <termios.h>
#include <unistd.h>

#define KEYCODE_RIGHT 0x43
#define KEYCODE_LEFT 0x44
#define KEYCODE_UP 0x41
#define KEYCODE_DOWN 0x42
#define KEYCODE_B 0x62
#define KEYCODE_Q 0x71
#define KEYCODE_S 0x73
#define KEYCODE_Z 0x7a
#define KEYCODE_X 0x78
#define KEYCODE_C 0x63
#define KEYCODE_SPACE 0x20
const int k_control_cycle = 100;
```





## KeyboardReader 클래스

```
class KeyboardReader
{
public:
    KeyboardReader(): kfd(0)
    {
        // get the console in raw mode
        tcgetattr(kfd, &cooked);
        struct termios raw;
        memcpy(&raw, &cooked, sizeof(struct termios));
        raw.c_lflag &=~ (ICANON | ECHO);
        // Setting a new line, then end of file
        raw.c_cc[VEOL] = 1;
        raw.c_cc[VEOF] = 2;
        tcsetattr(kfd, TCSANOW, &raw);
    }
}
```



## KeyboardReader 클래스(2)

```
void readOne(char * c)
{
    int rc = read(kfd, c, 1);
    if (rc < 0)
    {
        throw std::runtime_error("read failed");
    }
}

void shutdown()
{
    tcsetattr(kfd, TCSANOW, &cooked);
}

private:
    int kfd;
    struct termios cooked;
};

KeyboardReader input;
```



# DronePWM 구조체

```
struct DronePWM
{
    double l_x;
    double l_y;
    double l_z;
    double a_w;
    double l_scale;
    double a_scale;
};
```



# DroneTeleop 클래스(1)

```
class DroneTeleop
{
public:
    DroneTeleop():
        velocity_x(0),
        velocity_y(0),
        velocity_z(0),
        velocity_w(0),
        l_scale_(1000.0),
        a_scale_(200.0)
    {
        nh_.param("scale_angular", a_scale_, a_scale_);
        nh_.param("scale_linear", l_scale_, l_scale_);

        twist_pub_ = nh_.advertise<geometry_msgs::Twist>("drone_teleop_pwm", 1);
    }
}
```

## DroneTeleop 클래스(2)

```
void keyLoop()
{
    char c;
    bool dirty=false;
    ros::Rate loop_rate(k_control_cycle);

    puts("Reading from keyboard");
    puts("-----");
    puts("Use keys to move the drone.");
    puts("↑ : foward      ↓ : backward");
    puts("← : left        → : right");
    puts("z : turn left   x : turn right");
    puts("space : up      c : down");
    puts("b : all        s : stop          q : quit");
}
```



## DroneTeleop 클래스(3)

```
while (ros::ok())
{
    // get the next event from the keyboard
    try
    {
        input.readOne(&c);
    }
    catch (const std::runtime_error &)
    {
        perror("read()");
        return;
    }
    ROS_DEBUG("value: 0x%02X\n", c);
```



# DroneTeleop 클래스(4)

```
switch(c)
{
    case KEYCODE_LEFT:
        ROS_DEBUG("LEFT");
        velocity_y += 0.01;
        dirty = true;
        break;
    case KEYCODE_RIGHT:
        ROS_DEBUG("RIGHT");
        velocity_y -= 0.01;
        dirty = true;
        break;
    ...
    case KEYCODE_Q:
        ROS_DEBUG("quit");
        return;
}
```

## DroneTeleop 클래스(5)

```
if(velocity_x > 1.0)velocity_x = 1.0;
else if(velocity_x < -1.0)velocity_x = -1.0;
if(velocity_y > 1.0)velocity_y = 1.0;
else if(velocity_y < -1.0)velocity_y = -1.0;
if(velocity_z > 1.0)velocity_z = 1.0;
else if(velocity_z < 0.0)velocity_z = 0.0;
if(velocity_w > 1.0)velocity_w = 1.0;
else if(velocity_w < -1.0)velocity_w = -1.0;

geometry_msgs::Twist twist;
twist.linear.x = l_scale*velocity_x;
twist.linear.y = l_scale*velocity_y;
twist.linear.z = l_scale*velocity_z;
twist.angular.x = a_scale*velocity_w;
```



## DroneTeleop 클래스(6)

```
    if(dirty ==true)
    {
        dirty=false;
        twist_pub_.publish(twist);
    }

    loop_rate.sleep();
    ros::spinOnce();
}
return;
}

private:

ros::NodeHandle nh_;
double velocity_x, velocity_y, velocity_z, velocity_w, l_scale_, a_scale_;
ros::Publisher twist_pub_;
};
```



# quit 함수

```
void quit(int sig)
{
    (void)sig;
    input.shutdown();
    ros::shutdown();
    exit(0);
}
```



## main 함수

```
int main(int argc, char** argv)
{
    ros::init(argc, argv, "drone_teleop");
    DroneTeleop drone_teleop;

    signal(SIGINT, quit);

    drone_teleop.keyLoop();
    quit(0);

    return 0;
}
```



## 로컬 PC에서 **drone\_teleop** 노드 실행

로컬 PC의 VS Code 창에서 새 터미널 실행(Ctrl + Shift + `)

```
$ cm
```

```
$ roscore
```

```
$ rosrun drone_tutorial drone_teleop_key
```

```
○ chris20@ChrisChun:~$ rosrun drone_tutorial drone_teleop
Reading from keyboard
-----
Use keys to move the drone.
↑ : foward      ↓ : backward
← : left        → : right
z : turn left   x : turn right
space : up      c : down
b : all         s : stop      q : quit
```



## drone\_controller.cpp 둘러보기

키보드 키를 통해 비행 로봇을 원격조종 할 수 있도록 토픽을 보내는 코드

TeleopCallback 함수 : 키보드 명령을 받고 해당하는 PWM 출력을 결정하는 함수

SetPwmDutycycle : 정해진 수치대로 PWM값을 조절하는 함수.

main 함수 : 원격조종 명령대로 비행로봇을 움직이는 함수

## 라이브러리 및 상수, 변수 설정

```
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include <pigpiod_if2.h>

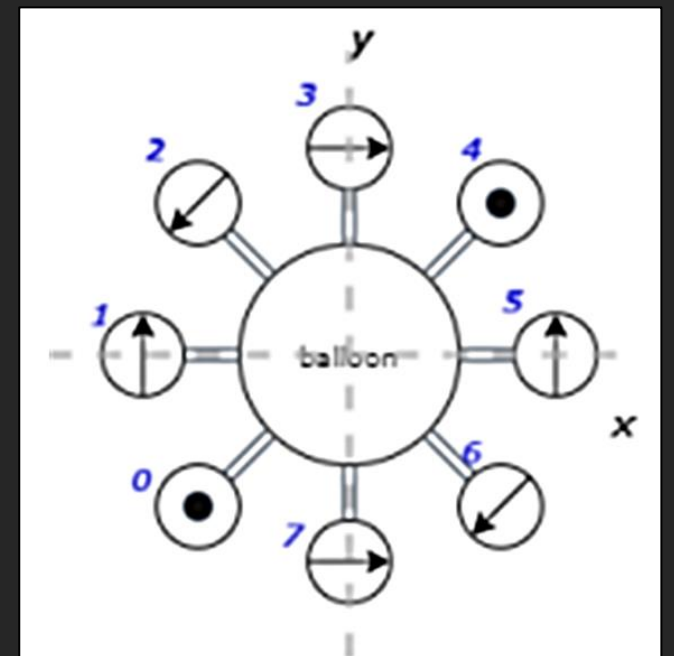
#define constrain(amt, low, high) ((amt) < (low) ? (low) : ((amt) >
(high) ? (high) : (amt)))
const int k_control_cycle = 10;
const int k_pins = 8;
const int k_pwm_range = 1000;
const int k_pwm_frequency = 20000;
const int k_pin_nums[k_pins] = {17,4,22,27,6,5,19,13};
int pi_num;
int target_pwm[k_pins] = {};
```

# TeleopCallback 함수

```
void TeleopCallback(const geometry_msgs::Twist &msg)
{
    double X_pos = (msg.linear.x >= 0) ? msg.linear.x : 0;
    double Y_pos = (msg.linear.y >= 0) ? msg.linear.y : 0;
    double Z_pos = (msg.linear.z >= 0) ? msg.linear.z : 0;
    double W_pos = (msg.angular.x > 0) ? constrain(msg.angular.x,15,250) : 0;
    double X_neg = (msg.linear.x < 0) ? msg.linear.x : 0;
    double Y_neg = (msg.linear.y < 0) ? msg.linear.y : 0;
    double W_neg = (msg.angular.x < 0) ? constrain(msg.angular.x,-250,-15) : 0;

    double vh = sqrt(2) * 0.5;

    target_pwm[0] = 0 + 0 + Z_pos + 0;
    target_pwm[1] = -X_neg * vh + Y_pos + 0 + W_pos;
    target_pwm[2] = -X_neg - Y_neg + 0 - W_neg;
    target_pwm[3] = X_pos - Y_neg * vh + 0 + W_pos;
    target_pwm[4] = 0 + 0 + Z_pos + 0;
    target_pwm[5] = -X_neg * vh + Y_pos + 0 - W_neg;
    target_pwm[6] = -X_neg - Y_neg + 0 + W_pos;
    target_pwm[7] = X_pos - Y_neg * vh + 0 - W_neg;
}
```



## CheckRpi 함수

```
bool CheckPigpio()  
{  
    pi_num = pigpio_start(NULL, NULL);  
  
    if (pi_num < 0)  
    {  
        ROS_ERROR("PI number is %d", pi_num);  
        ROS_ERROR("PIGPIO connection failed.");  
        return false;  
    }  
  
    ROS_INFO("Setup Finished.");  
    return true;  
}
```





## InitPwm 함수

```
void InitPwm()
{
    for (int i = 0; i < k_pins; i++)
    {
        set_mode(pi_num, k_pin_nums[i], PI_OUTPUT);
        set_PWM_range(pi_num, k_pin_nums[i], k_pwm_range);
        set_PWM_frequency(pi_num, k_pin_nums[i], k_pwn_frequency);
    }
}
```



# SetPwmDutycycle 함수(1)

```
void SetPwmDutycycle(int rate)
{
    if(rate < 0 || rate > k_pwm_range)
    {
        ROS_WARN("Invalid Dutycycle.");
        for (int i = 0; i < k_pins; i++) set_PWM_dutycycle(pi_num, k_pin_nums[i], 0);
    }

    for (int i = 0; i < k_pins; i++) set_PWM_dutycycle(pi_num, k_pin_nums[i], rate);
}
```

## SetPwmDutycycle 함수(2)

```
void SetPwmDutycycle()
{
    for (int i = 0; i < k_pins; i++)
    {
        if(target_pwm[i] < 0 || target_pwm[i] > k_pwm_range)
        {
            ROS_WARN("Invalid Dutycycle.");
            set_PWM_dutycycle(pi_num, k_pin_nums[i], 0);
        }
        set_PWM_dutycycle(pi_num, k_pin_nums[i], target_pwm[i]);
    }
}
```

## main 함수

```
int main(int argc, char **argv)
{
    ros::init(argc, argv, "drone_controller");
    ros::NodeHandle nh;
    ros::Rate loop_rate(k_control_cycle);
    ros::Subscriber sub_teleop = nh.subscribe("/drone_teleop_pwm", 5, TeleopCallback);

    if(!CheckPigpio()) return -1;
    InitPwm();

    while (ros::ok())
    {
        SetPwmDutycycle();
        loop_rate.sleep();
        ros::spinOnce();
    }

    SetPwmDutycycle(0);
    return 0;
}
```

# SSH에서 **drone\_controller** 노드 실행

SSH로 접속한 RPI VS Code 창에서 새 터미널 실행(Ctrl + Shift + `)

```
$ cm
```

```
$ roscore
```

```
$ rosrunc drone_tutorial drone_teleop
```

```
$ rosrunc drone_tutorial drone_controller
```

// 키보드를 통해 비행로봇을 원격 조종할 수 있다.

### 비행로봇 최종 데모

비행로봇을 조종하여 목적지까지 이동하여 보자.

비행로봇의 풍선에 헬륨을 충전하여 준비한다.

비행로봇을 섬세하게 조종하여 이동시킨다.

풍선은 관성의 영향을 많이 받으므로 관성을 고려하여 조종한다.

배터리를 완전 방전 시키지 않도록 유의한다.

다른 비행로봇, 벽이나 장애물에 충돌하지 않도록 유의한다.

비행로봇을 안전하게 목적지까지 이동시켜보자.



감사합니다

