

# “hello world!”

2016. 06. 18

**ROBOTIS**

Open Source Team

Yoonseok Pyo

# Index

---

I. 무조건 따라하기 “hello world!”

II. ROS 파일 시스템

III. ROS 빌드 시스템

# Index

---

I. 무조건 따라하기 “hello world!”

II. ROS 파일 시스템

III. ROS 빌드 시스템

# 무조건 따라하기 “hello world!”

---

- ROS판 “hello world!” 작성
  - 메시지 통신 중 토픽을 이용하는 퍼블리셔 노드 작성
- 1) 패키지 생성
  - 사용자가 패키지를 작성할 때 캐킨 빌드 시스템에 꼭 필요한 CMakeLists.txt 와 package.xml을 포함한 패키지 폴더를 생성한다.

```
$ catkin_create_pkg [패키지이름] [의존하는패키지1] [의존하는패키지n]
```

```
$ cd ~/catkin_ws/src  
$ catkin_create_pkg my_first_ros_pkg std_msgs roscpp  
$ cd ~/catkin_ws/src/my_first_ros_pkg
```

# 무조건 따라하기 “hello world!”

- 2) 패키지 설정 파일(package.xml) 수정
  - 패키지 이름, 저작자, 라이선스, 의존성 패키지 등의 패키지 관련 정보

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

```
$ gedit ./package.xml
```

```
<?xml version="1.0"?>
<package>
  <name>my_first_ros_pkg</name>
  <version>0.0.1</version>
  <description>The my_first_ros_pkg package</description>
  <maintainer email="pyo@robotis.com">Yoonseok Pyo</maintainer>
  <license>BSD</license>
  <url type="website">http://robotis.com</url>
  <url type="repository">https://github.com/oroca/oroca_ros_tutorials.git</url>
  <author email="pyo@robotis.com">Yoonseok Pyo</author>
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>roscpp</build_depend>
  <run_depend>std_msgs</run_depend>
  <run_depend>roscpp</run_depend>
  <export></export>
</package>
```

# 무조건 따라하기 “hello world!”

- 3) 빌드 설정 파일(CMakeLists.txt) 수정
  - 실행 파일 생성, 의존성 패키지 우선 빌드, 링크 생성 등의 빌드 환경 정보

```
$ cd ~/catkin_ws/src/my_first_ros_pkg  
$ gedit ./CMakeLists.txt
```

```
cmake_minimum_required(VERSION 2.8.3)  
project(my_first_ros_pkg)  
find_package(catkin REQUIRED COMPONENTS roscpp std_msgs)  
catkin_package(  
  INCLUDE_DIRS include  
  CATKIN_DEPENDS roscpp std_msgs  
  DEPENDS system_lib  
)  
include_directories(${catkin_INCLUDE_DIRS})  
add_executable(hello_world_node src/hello_world_node.cpp)  
add_dependencies(hello_world_node my_first_ros_pkg_generate_messages_cpp)  
target_link_libraries(hello_world_node ${catkin_LIBRARIES})
```

# 무조건 따라하기 “hello world!”

---

- 4) 소스 코드 작성

```
$ cd ~/catkin_ws/src/my_first_ros_pkg/src  
$ gedit ./hello_world_node.cpp
```

```
#include <ros/ros.h>  
#include <std_msgs/String.h>  
#include <sstream>  
  
int main(int argc, char **argv)  
{  
    ros::init(argc, argv, "hello_world_node");  
    ros::NodeHandle nh;  
    ros::Publisher chatter_pub = nh.advertise<std_msgs::String>("say_hello_world", 1000);  
    ros::Rate loop_rate(10);  
    int count = 0;
```

# 무조건 따라하기 “hello world!”

```
while (ros::ok())
{
    std_msgs::String msg;
    std::stringstream ss;
    ss << "hello world!" << count;
    msg.data = ss.str();
    ROS_INFO("%s", msg.data.c_str());
    chatter_pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    ++count;
}

return 0;

}
```



# 무조건 따라하기 “hello world!”

---

- 5) 패키지 빌드

```
$ cd ~/catkin_ws
```

```
$ catkin_make
```

(혹은 다음과 같이 함께 사용하여 1라인으로 처리)

```
$ cd ~/catkin_ws && catkin_make
```

- 6) roscore 실행

```
$ roscore
```

(이미 실행되어 있다면 생략)

- 7) 노드 실행

```
$ rospack profile
```

```
$ rosrun my_first_ros_pkg hello_world_node
```

```
[INFO] [1423443540.131775283]: hello world! 0
```

```
[INFO] [1423443540.231826916]: hello world! 1
```

```
[INFO] [1423443540.331798085]: hello world! 2
```

```
[INFO] [1423443540.431796634]: hello world! 3
```

```
...
```

# 무조건 따라하기 “hello world!”

- rqt\_graph

```
$ rosrun rqt_graph rqt_graph
```

- rostopic

```
$ rostopic list
```

```
/rosout
```

```
/rosout_agg
```

```
/say_hello_world
```

```
$ rostopic info /say_hello_world
```

```
Type: std_msgs/String
```

```
Publishers:
```

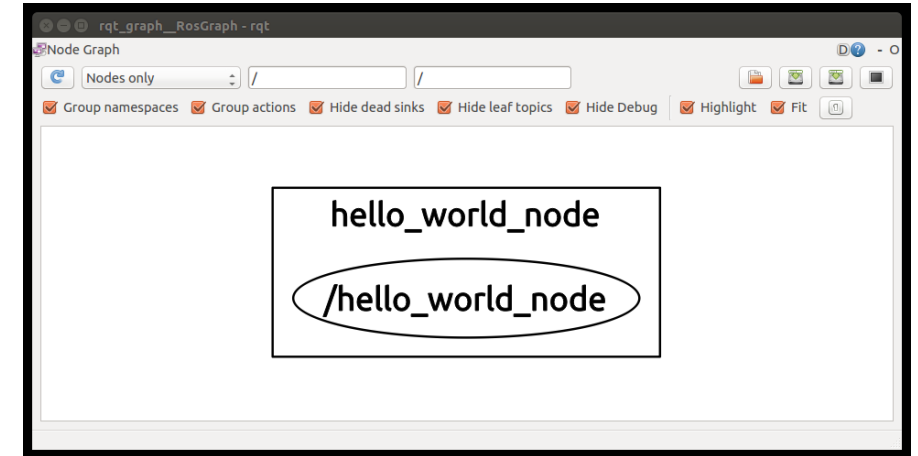
```
* /hello_world_node (http://localhost:51083/)
```

```
Subscribers: None
```

```
$ rostopic hz /say_hello_world
```

```
subscribed to [/say_hello_world]
```

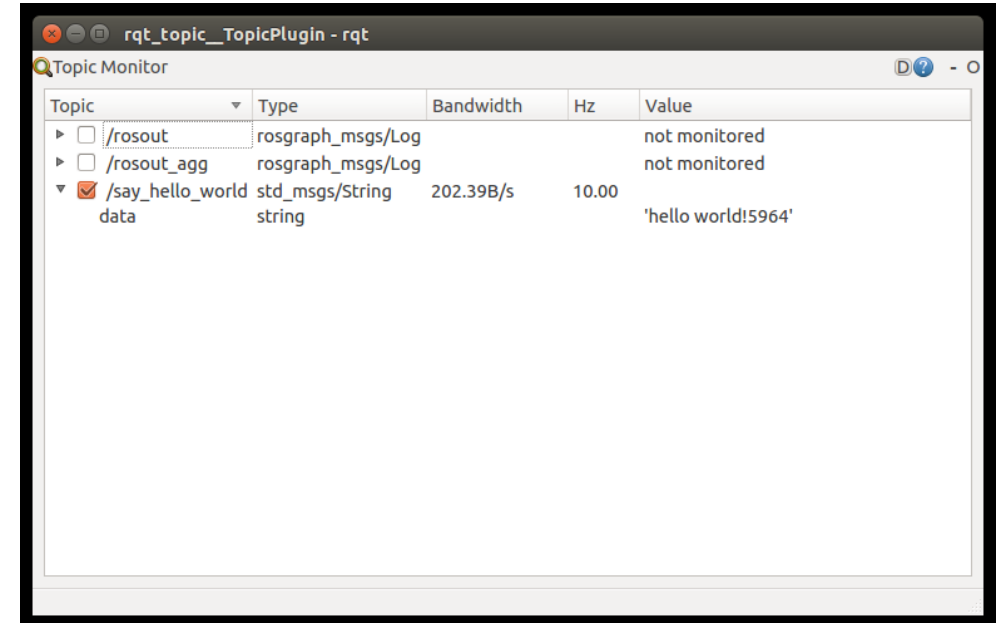
```
average rate: 9.999 min: 0.100s max: 0.100s std dev: 0.00001s window: 10
```



# 무조건 따라하기 “hello world!”

- rqt\_topic

```
$ rosrun rqt_topic rqt_topic
```

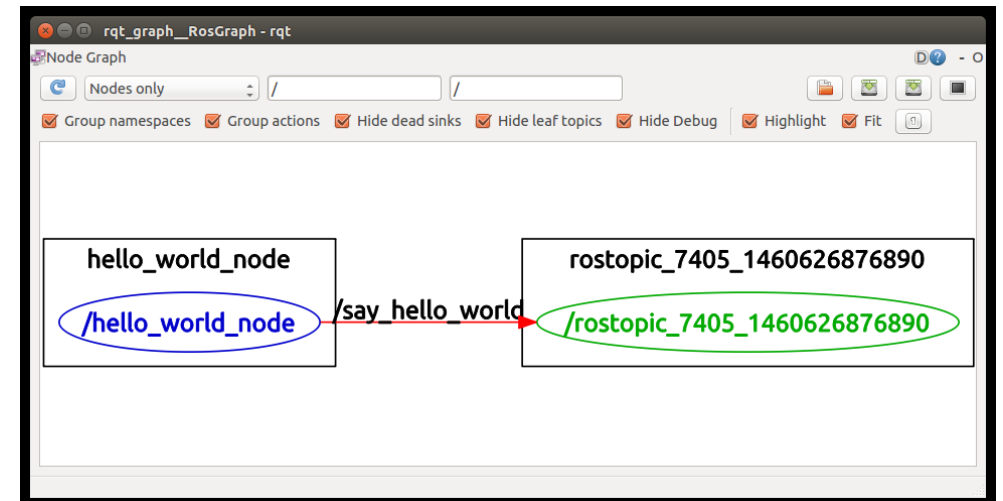


- 송/수신 테스트

```
$ rostopic echo /say_hello_world  
data: hello world!109  
---  
data: hello world!110  
---  
data: hello world!111  
---  
...
```

```
$ rosrun rqt_graph rqt_graph
```

```
ctrl + c
```



# Index

---

I. 무조건 따라하기 “hello world!”

II. ROS 파일 시스템

III. ROS 빌드 시스템

# ROS 파일 시스템

---

## 1. ROS 설치 폴더

- /opt/ros/[버전이름]
- /opt/ros/kinetic (ROS kinetic 버전의 경우)

## 2. 사용자 작업 폴더

- ~/catkin\_ws/ ('~/ '은 리눅스에서 '/home/사용자명'에 해당하는 폴더)
- /home/oroca/catkin\_ws/  
(사용자명이 oroca의 경우, 사용자 작업 폴더명을 catkin\_ws라고 지정한 경우)

## 3. 사용자 패키지 폴더

- /home/oroca/catkin\_ws/src
- 사용자 작업 폴더내의 하나의 패키지
- 0(?)개 이상의 노드로 구성됨

# ROS 파일 시스템

## 1. ROS 설치 폴더(/opt/ros/kinetic)

- /bin 실행 가능한 바이너리 파일
- /etc ROS 및 catkin 관련 설정 파일
- /include 헤더 파일
- /lib 라이브러리 파일
- /share ROS 패키지
- env.\* 환경 설정 파일
- setup.\* 환경 설정 파일

| Name           | Size      | Type    | Modified |
|----------------|-----------|---------|----------|
| bin            | 69 items  | Folder  | Aug 21   |
| etc            | 2 items   | Folder  | May 22   |
| include        | 156 items | Folder  | Aug 21   |
| lib            | 308 items | Folder  | Aug 21   |
| share          | 327 items | Folder  | Aug 21   |
| env.sh         | 506 bytes | Program | Jul 11   |
| setup.bash     | 260 bytes | Program | Jul 11   |
| setup.sh       | 2.4 kB    | Program | Jul 11   |
| setup.zsh      | 251 bytes | Program | Jul 11   |
| _setup_util.py | 12.2 kB   | Text    | Jul 11   |
| .catkin        | 0 bytes   | Text    | Jul 11   |
| .rosinstall    | 55 bytes  | Text    | Jul 11   |

# ROS 파일 시스템

---

## 2. 사용자 작업 폴더(/home/oroca/catkin\_ws/)

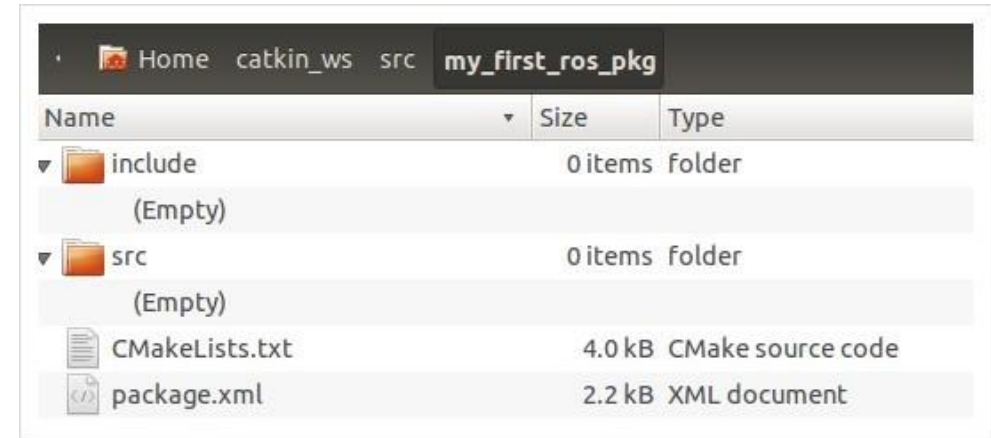
- /build 빌드 관련 파일
- /devel msg, srv 헤더 파일과 사용자 패키지 라이브러리, 실행 파일
- /src 사용자 패키지 폴더

|             |                 |
|-------------|-----------------|
| ▼ catkin_ws | 3 items folder  |
| ▶ build     | 11 items folder |
| ▶ devel     | 11 items folder |
| ▶ src       | 2 items folder  |

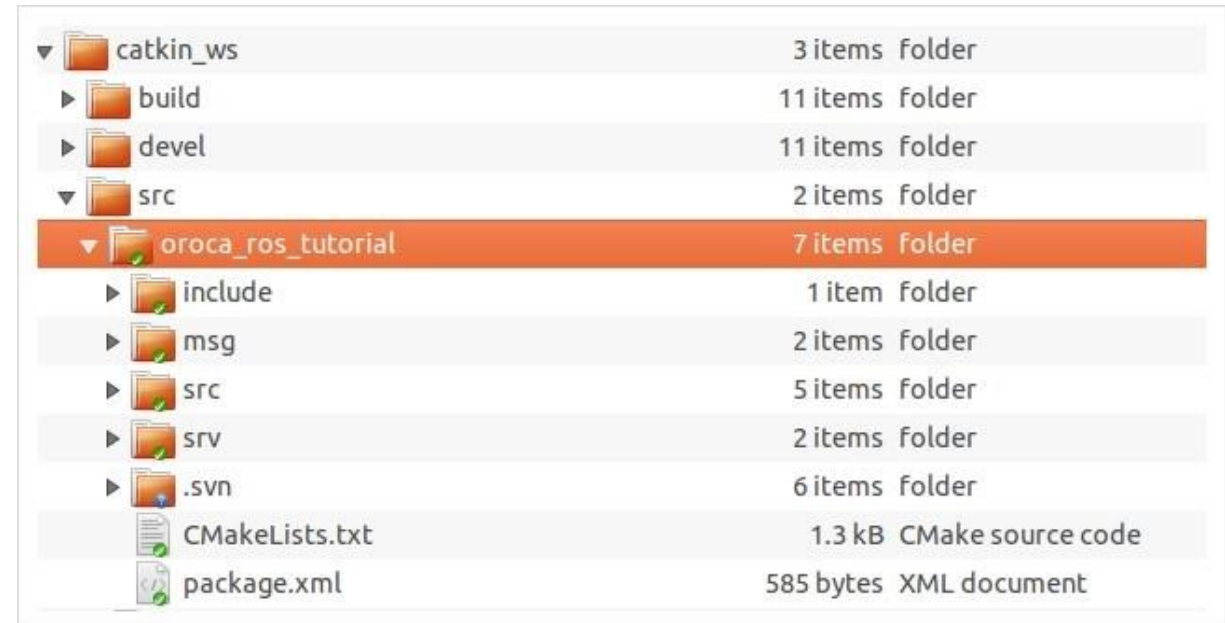
# ROS 파일 시스템

## 3. 사용자 패키지 폴더 (/home/oroca/catkin\_ws/src)

- /include 헤더파일
- /launch roslaunch에 사용되는 launch 파일
- /node rospy용 스크립트
- /msg 메시지 파일
- /src 코드 소스 파일
- /srv 서비스 파일
- CMakeLists.txt 빌드 설정 파일
- package.xml 패키지 설정 파일



| Name           | Size    | Type              |
|----------------|---------|-------------------|
| include        | 0 items | folder            |
| (Empty)        |         |                   |
| src            | 0 items | folder            |
| (Empty)        |         |                   |
| CMakeLists.txt | 4.0 kB  | CMake source code |
| package.xml    | 2.2 kB  | XML document      |



|                    |           |                   |
|--------------------|-----------|-------------------|
| catkin_ws          | 3 items   | folder            |
| build              | 11 items  | folder            |
| devel              | 11 items  | folder            |
| src                | 2 items   | folder            |
| oroca_ros_tutorial | 7 items   | folder            |
| include            | 1 item    | folder            |
| msg                | 2 items   | folder            |
| src                | 5 items   | folder            |
| srv                | 2 items   | folder            |
| .svn               | 6 items   | folder            |
| CMakeLists.txt     | 1.3 kB    | CMake source code |
| package.xml        | 585 bytes | XML document      |



# Index

---

I. 무조건 따라하기 “hello world!”

II. ROS 파일 시스템

III. ROS 빌드 시스템

# ROS 빌드 시스템

---

- CMake(Cross Platform Make)를 이용
- 빌드 환경은 패키지 폴더의 CMakeLists.txt 파일에 기술
- ROS 개발 환경에 맞도록 Cmake의 기능을 추가하여 ROS에 특화시킨 캐킨(catkin) 빌드 시스템을 제공
  - 캐킨 빌드 시스템은 ROS와 관련된 빌드, 패키지 관리, 패키지 간 의존관계 등을 편리하게 사용할 수 있도록 하고 있다.
  - ROS 패키지 설정 내용은 package.xml에 기재하고 있다.

# 패키지 설정 파일(package.xml)

- `<?xml>` 문서 문법을 정의하는 문구로 아래의 내용은 xml 버전 1.0을 따르고 있다는 것을 알린다.
- `<package>` 이 구문부터 `</package>`까지가 ROS 패키지 설정 부분이다.
- `<name>` 패키지의 이름이다. 패키지를 생성할 때 입력한 패키지 이름이 사용된다. 다른 옵션도 마찬가지로 이의 사용자가 원할 때 언제든지 변경할 수 있다.
- `<version>` 패키지의 버전이다. 자유롭게 지정할 수 있다.
- `<description>` 패키지의 간단한 설명이다.
- `<maintainer>` 패키지 관리자의 연락처를 기재한다.
- `<license>` 라이선스를 기재한다. BSD, MIT, GPLv3, LGPLv3 등을 기재하면 된다.
- `<url>` 패키지를 설명하는 웹 페이지 또는 버그 관리, 저장소 등의 주소를 기재한다. 이 종류에 따라 type에 website, bugtracker, repository를 대입하면 된다.
- `<author>` 패키지 개발에 참여한 개발자를 적는다. 복수의 개발자가 참여한 경우에는 바로 다음 줄에 `<author>` 태그를 이용하여 추가로 넣어주면 된다.
- `<buildtool_depend>` 빌드 시스템의 의존성을 기술한다. 캐킨 빌드 시스템을 이용하고 있으므로 catkin을 입력한다.
- `<build_depend>` 패키지를 빌드할 때 의존하는 패키지 이름을 적는다.
- `<run_depend>` 패키지를 실행할 때 의존하는 패키지 이름을 적는다.
- `<test_depend>` 패키지를 테스트할 때 의존하는 패키지 이름을 적는다.
- `<export>` ROS에서 명시하지 않은 태그명을 사용할 때 쓰인다. 일반적인 경우 쓸 일이 없다.
- `<metapackage>` export 태그 안에서 사용하는 공식적인 태그로 현재의 패키지가 메타 패키지이면 이를 선언한다.

# 빌드 설정 파일(CMakeLists.txt)

---

- `cmake_minimum_required(VERSION 2.8.3)`
  - 운영체제에 설치된 cmake의 최소 요구 버전
- `project(my_first_ros_pkg)`
  - 패키지의 이름 (package.xml에서 입력한 패키지 이름을 그대로 사용하자.)
- `find_package(catkin REQUIRED COMPONENTS roscpp std_msgs)`
  - 캐킨 빌드를 할 때 요구되는 구성요소 패키지
- `find_package(Boost REQUIRED COMPONENTS system)`
  - ROS 이외의 패키지가 요구될 때 명시하는 방법
- `catkin_python_setup()`
  - 파이썬을 사용할 때 설정하는 옵션이다. 파이썬 설치 프로세스인 setup.py를 부르는 역할을 한다.

# 빌드 설정 파일(CMakeLists.txt)

---

- `add_message_files(FILES Message1.msg Message2.msg)`
  - 노드에서 사용하는 메시지 파일을 추가하는 옵션 (msg 폴더의 .msg 파일)
- `add_service_files(FILES Service1.srv Service2.srv)`
  - 노드에서 사용하는 서비스 파일을 추가하는 옵션 (srv 폴더의 .srv 파일)
- `generate_messages(DEPENDENCIES std_msgs)`
  - 의존하는 메시지를 설정하는 옵션
- `catkin_package(`  
    `INCLUDE_DIRS include`  
    `LIBRARIES my_first_ros_pkg`  
    `CATKIN_DEPENDS roscpp std_msgs`  
    `DEPENDS system_lib`  
    `)`
  - 해당 폴더의 헤더 파일을 사용
  - 해당 패키지의 라이브러리를 사용
  - 의존하는 ROS 패키지를 설정
  - 의존하는 시스템 라이브러리를 설정
  - 캐킨 빌드 옵션

# 빌드 설정 파일(CMakeLists.txt)

---

- `include_directories(${catkin_INCLUDE_DIRS})`
  - 인클루드 폴더를 지정
- `add_library(my_first_ros_pkg src/${PROJECT_NAME}/my_first_ros_pkg.cpp)`
  - 빌드 후 생성할 라이브러리를 지정
- `add_executable(my_first_ros_pkg_node src/my_first_ros_pkg_node.cpp)`
  - 소스 코드 파일 지정, 빌드 후 생성할 실행 파일명을 지정
- `add_dependencies(my_first_ros_pkg_node my_first_ros_pkg_generate_messages_cpp)`
  - 패키지를 빌드하기에 앞서 생성해야 할 메시지 헤더 파일이 있으면 빌드 전에 우선으로 메시지를 생성하라는 설정
- `target_link_libraries(my_first_ros_pkg_node ${catkin_LIBRARIES})`
  - `my_first_ros_pkg_node`를 생성하기에 앞서 링크해야 하는 라이브러리와 실행 파일을 링크해주는 옵션

# ROS 로봇 프로그래밍

을 위한 선수 학습 완료!

다음에는 퍼블리셔, 서브스크라이버,  
서비스 서버, 서비스 클라이언트 작성에 들어갑니다.

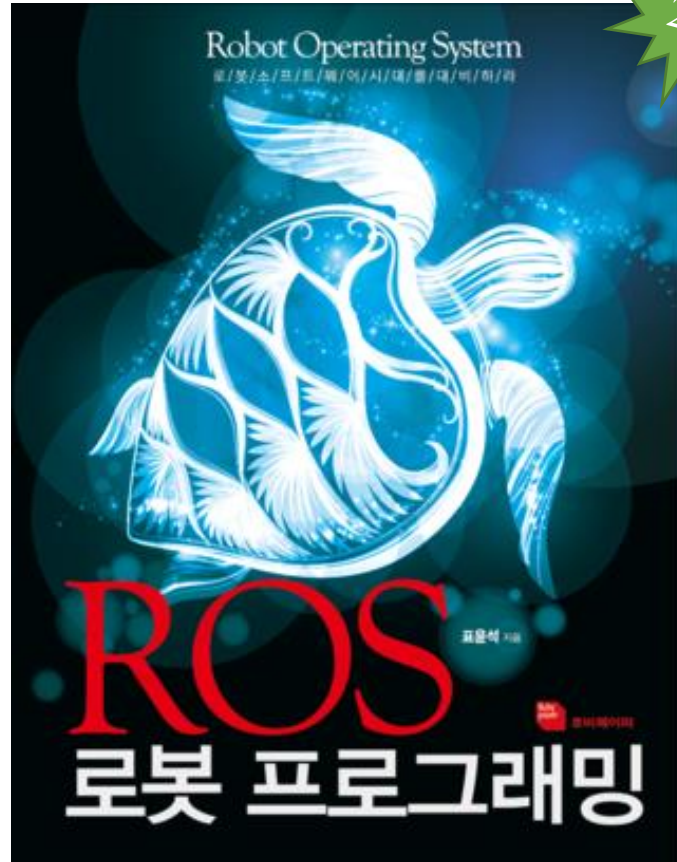
---

**질문  
대환영!**

---



여기서! 광고 하나 나가요~



국내 유일! 최초! ROS 책  
비 영어권 최고의 책  
인세 전액 기부

여기서! 광고 둘 나가요~



- 오로카
- [www.oroqa.org](http://www.oroqa.org)
- 오픈 로보틱스 지향
- 풀뿌리 로봇공학의 저변 활성화
- 공개 강좌, 세미나, 프로젝트 진행

- 로봇공학을 위한 열린 모임 (KOS-ROBOT)
- [www.facebook.com/groups/KoreanRobotics](https://www.facebook.com/groups/KoreanRobotics)
- 로봇공학 통합 커뮤니티 지향
- 일반인과 전문가가 어울러지는 한마당
- 로봇공학 소식 공유
- 연구자 간의 협력

혼자 하기에 답답하시다고요?  
커뮤니티에서 함께 해요~

# 끝.

---

표윤석

Yoonseok Pyo  
pyo@robotis.com  
www.robotpilot.net



[www.facebook.com/yoonseok.pyo](https://www.facebook.com/yoonseok.pyo)

# Thanks for your attention!

표윤석

Yoonseok Pyo  
pyo@robotis.com  
www.robotpilot.net

[www.facebook.com/yoonseok.pyo](https://www.facebook.com/yoonseok.pyo)