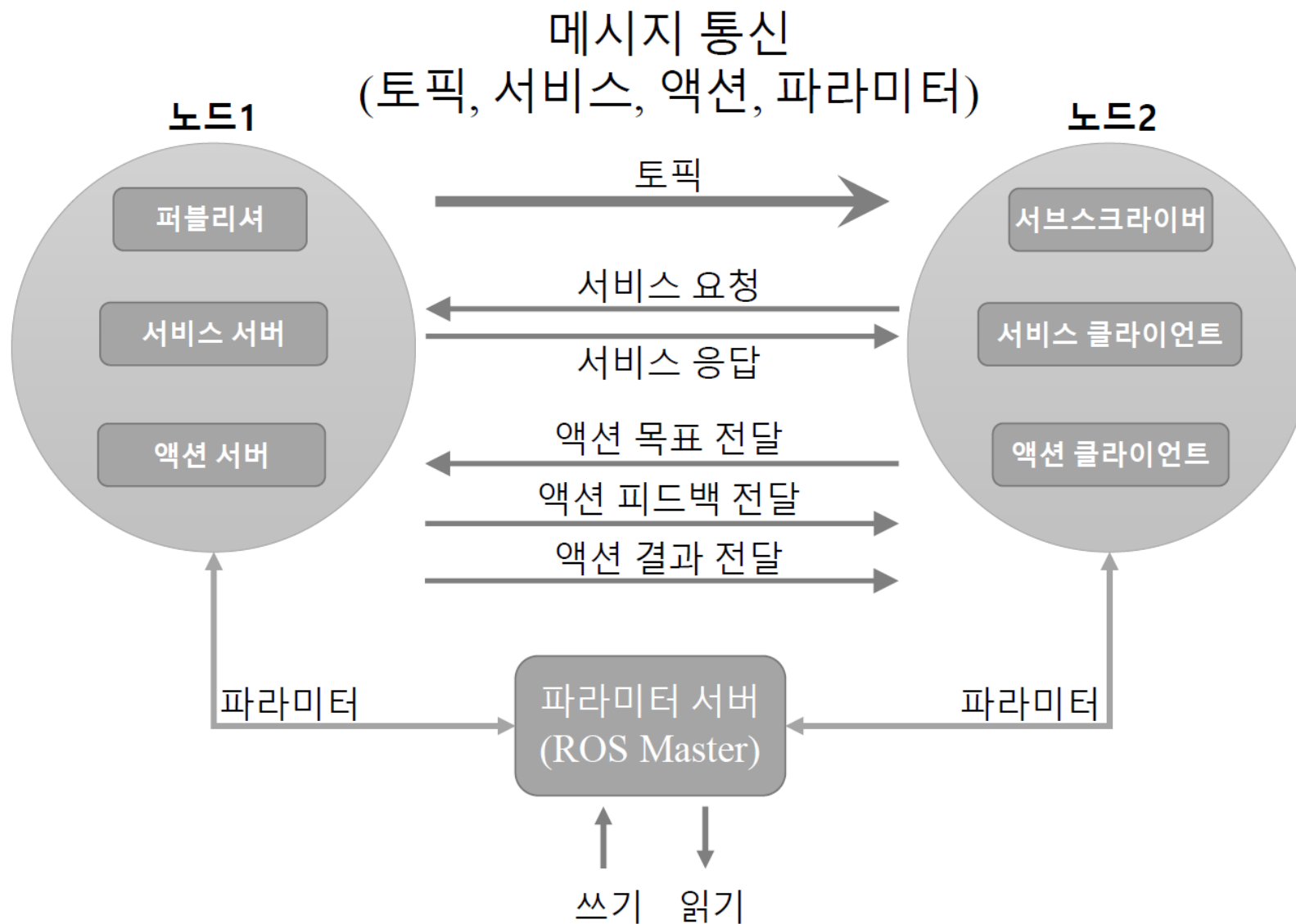

6. ROS Message, Service, Param

Overview

- ROS Message 소개, 실습
- ROS Service 소개, 실습
- ROS Parameter 소개, 실습



ROS Message



ROS Message

- 메시지는 노드 간에 데이터를 주고받을 때 사용하는 데이터의 형태
 - 토픽, 서비스, 액션은 모두 메시지를 사용
 - 단순 자료형
 - 예) 정수(integer), 부동소수점(floating point), 불(boolean)
 - 메시지 안에 메시지를 품고있는 간단한 데이터 구조
 - 예) geometry_msgs/PoseStamped
 - 메시지들이 나열된 배열과 같은 구조
 - 예) float32[] ranges
 - 예) sensor_msgs/LaserScan



ROS Message

- ROS 메시지(예: geometry_msgs/Twist)



[geometry_msgs/Twist]

Vector3 linear
Vector3 angular

[geometry_msgs/Vector3]

float64 x
float64 y
float64 z

[geometry_msgs/Vector3]

float64 x
float64 y
float64 z



ROS Message

- ROS Message

- Node가 Publish하는 데이터.
- .msg 파일에 데이터의 구조가 정의 되어 있음
- 데이터의 구조는 primitive type(e.g. bool, int8, float32...)이나 정의된 msg type(e.g. sensor_msgs/Pointcloud2)을 포함할 수 있음
- 필요에 따라 직접 정의된 .msg파일을 사용할 수 있음

< 구조 예시 >

```
fieldtype1 fieldname1  
fieldtype2 fieldname2  
fieldtype3 fieldname3
```

< .msg 예시 >

```
Header header  
sensor_msgs/Pointcloud2  
int32 x  
int32 y
```



ROS Message

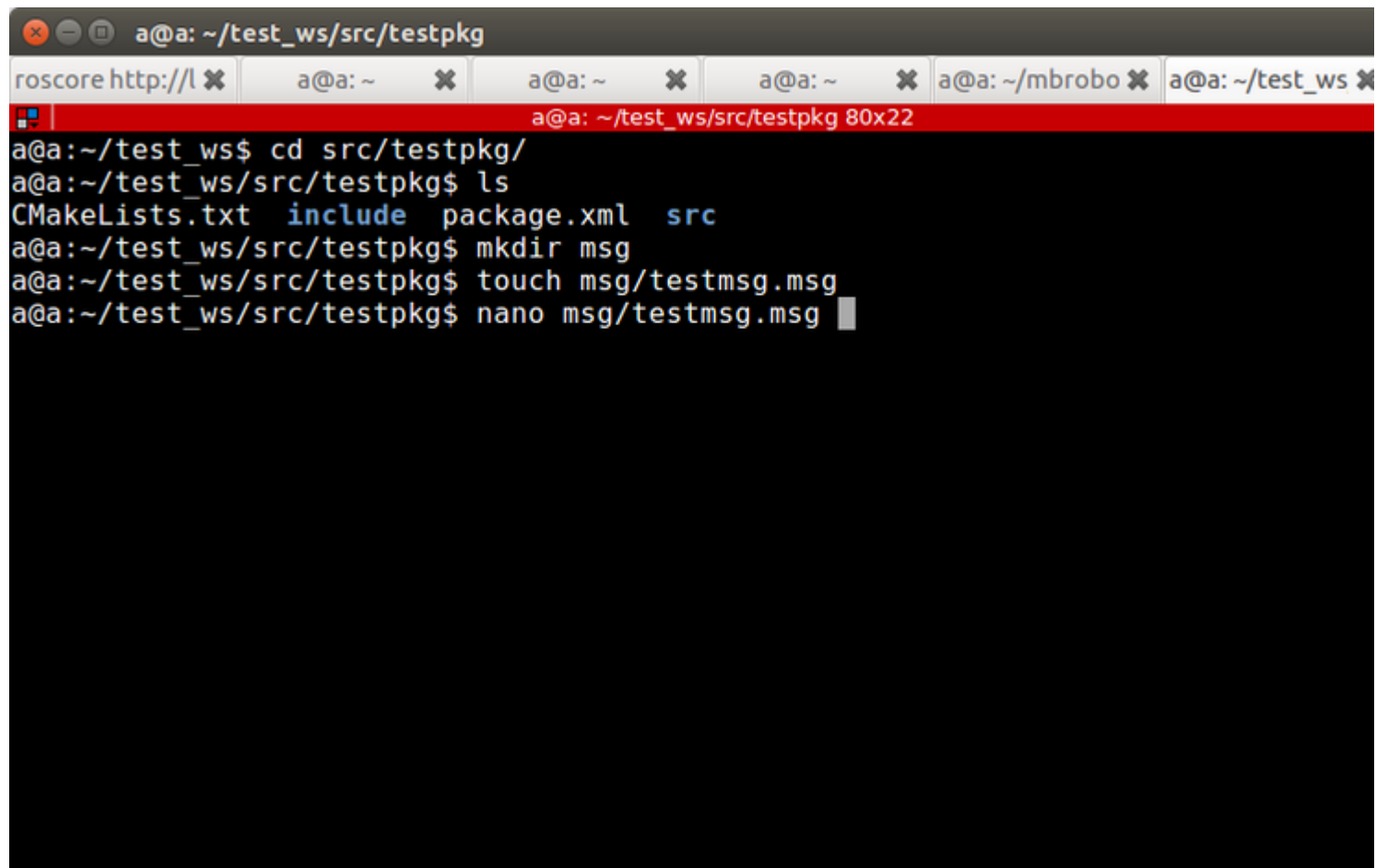
- Create msg

1. Testpkg에 새로운 폴더 생성,
testmsg.msg 파일 생성

```
$ mkdir msg  
$ touch testmsg.msg
```

2. 다음과 같이 testmsg.msg작성

```
Header header  
int32 x  
int32 y  
int32[] testarray
```



```
a@a: ~/test_ws/src/testpkg  
roscore http://l ✖ a@a: ~ ✖ a@a: ~ ✖ a@a: ~ ✖ a@a: ~/mbrobo ✖ a@a: ~/test_ws ✖  
a@a: ~/test_ws$ cd src/testpkg/  
a@a: ~/test_ws/src/testpkg$ ls  
CMakeLists.txt include package.xml src  
a@a: ~/test_ws/src/testpkg$ mkdir msg  
a@a: ~/test_ws/src/testpkg$ touch msg/testmsg.msg  
a@a: ~/test_ws/src/testpkg$ nano msg/testmsg.msg
```

ROS Message

- Create msg

3. Package.xml 파일에 다음 3줄 추가
(message generation dependency 추가)

```
<build_depend>message_generation</build_depend>  
<build_export_depend>message_generation</build_export_depend>  
<exec_depend>message_generation</exec_depend>
```

```
<buildtool_depend>catkin</buildtool_depend>  
  <build_depend>roscpp</build_depend>  
  <build_depend>rospy</build_depend>  
  <build_depend>std_msgs</build_depend>  
  <build_export_depend>roscpp</build_export_depend>  
  <build_export_depend>rospy</build_export_depend>  
  <build_export_depend>std_msgs</build_export_depend>  
  <exec_depend>roscpp</exec_depend>  
  <exec_depend>rospy</exec_depend>  
  <exec_depend>std_msgs</exec_depend>  
  
  <build_depend>message_generation</build_depend>  
  <build_export_depend>message_generation</build_export_depend>  
  <exec_depend>message_generation</exec_depend>
```



ROS Message

- Create msg

4. CMakeList.txt find_package를 다음과 같이 수정

```
## Compile as C++11, supported in ROS Kinetic and newer
# add_compile_options(-std=c++11)
```

```
## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  message_generation
)
```

```
## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)
```



ROS Message

- Create msg

5. add_message_files 부분을 주석처리 해제하고 수정

```
add_message_files(  
  FILES  
  testmsg.msg  
)
```

```
## * uncomment the add_*_files sections below as needed  
##   and list every .msg/.srv/.action file to be processed  
## * uncomment the generate_messages entry below  
## * add every package in MSG_DEP_SET to  
generate_messages(DEPENDENCIES ...)  
  
## Generate messages in the 'msg' folder  
add_message_files(  
  FILES  
  testmsg.msg  
  radiusArea.msg  
)
```



ROS Message

- Create msg

6. generate_messages 부분을 주석처리 해제하고 수정

```
generate_messages(  
  DEPENDENCIES  
  std_msgs  
)
```

```
## Generate actions in the 'action' folder  
# add_action_files(  
#   FILES  
#   Action1.action  
#   Action2.action  
# )
```

```
## Generate added messages and services with any dependencies listed  
here
```

```
generate_messages(  
  DEPENDENCIES  
  std_msgs  
)
```

```
#####  
## Declare ROS dynamic reconfigure parameters ##  
#####
```



ROS Message

- Use custom msg

- Custom msg를 publish/subscribe.
- Custommsg.cpp와 custommsg_sub.cpp 생성
- 정의된 msg의 내용을 구조체처럼 접근가능

```
//custommsg.cpp
#include "ros/ros.h"
#include "testpkg/testmsg.h"
#include <stdlib.h>
#include <iostream>
#include <vector>
std::vector<int> storedVector;
int main(int argc, char **argv)
{
    ros::init(argc, argv, "custommsg_pub");
    ros::NodeHandle n;
    ros::Publisher chatter_pub = n.advertise<testpkg::testmsg>("custommsg", 1);
    ros::Rate loop_rate(10); //hz
    int count = 0; for(int i = 0; i < 5; i++)
        storedVector.push_back(i);
    while (ros::ok())
    {
        testpkg::testmsg pub_data;
        pub_data.header.frame_id = "/map";
        pub_data.header.seq = count;
        pub_data.header.stamp = ros::Time::now();
        pub_data.x = 10;
        pub_data.y = 20;
        storedVector.push_back(count*2);
        storedVector.erase(storedVector.begin());
        pub_data.testarray = storedVector;
        chatter_pub.publish(pub_data);
        std::cout << "pub!" << std::endl;
        ros::spinOnce();
        loop_rate.sleep();
        ++count;
    }
    return 0;
}
```



ROS Message

- Use custom msg

```
#include "testpkg/testmsg.h"
```

→ 생성한 msg파일을 포함 시켜줌

```
ros::Publisher chatter_pub = n.advertise<testpkg::testmsg>("custommsg", 1);
```

→ Publisher 생성. `n.advertise<패키지명::메시지명>("토픽명", 1);`

```
testpkg::testmsg pub_data;  
pub_data.header.frame_id = "/map";  
pub_data.header.seq = count;  
pub_data.header.stamp = ros::Time::now();  
pub_data.x = 10;  
pub_data.y = 20;
```

→ Msg 파일이 포함하고 있는 멤버에 '로 접근

ROS Message

- Use custom msg
 - custommsg_sub.cpp 생성

```
#include "ros/ros.h"
#include "testpkg/testmsg.h"
#include <iostream>
void msgCallback(const testpkg::testmsg::ConstPtr& msg)
{
    std::cout << "callback!\n";
    std::cout << "msg->header.frame_id " << msg->header.frame_id << "\n";
    std::cout << "msg->header.seq " << msg->header.seq << "\n";
    std::cout << "msg->header.stamp " << msg->header.stamp << "\n";
    std::cout << "msg->x = " << msg->x << "\n";
    std::cout << "msg->y = " << msg->y << "\n";
    std::cout << "msg->testarray.push_back ";
    for (int i = 0; i < msg->testarray.size();i++)
    {
        std::cout << " " << msg->testarray.at(i);
    }
    std::cout << std::endl;
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "custommsg_sub");
    ros::NodeHandle n;
    ros::Subscriber sub = n.subscribe("custommsg", 1000, msgCallback);

    ros::spin();

    return 0;
}
```



ROS Message

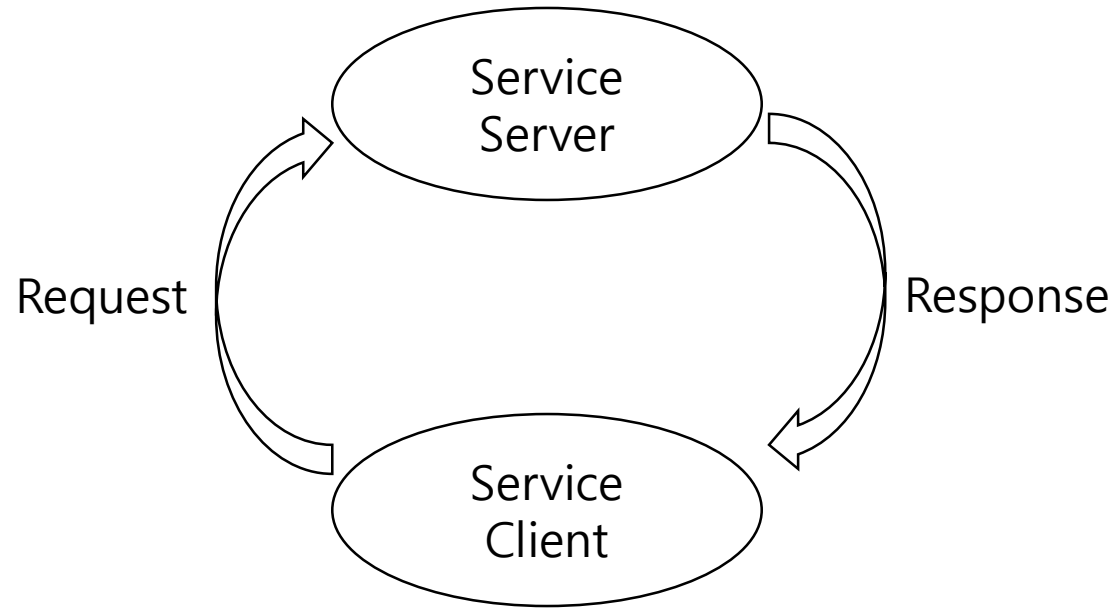
- 실행결과
 - CMakeList.txt에 두 노드를 추가, 빌드 후 실행

```
a@a: ~/mbrobotLecture/mobilerobot_ws
a@a: ~/mbrobotLecture/mobilerobot_ws 93x24
msg->y = 20
msg->testarray.push_back 336 338 340 342 344
callback!
msg->header.frame_id /map
msg->header.seq 173
msg->header.stamp 1589179517.776844984
msg->x = 10
msg->y = 20
msg->testarray.push_back 338 340 342 344 346
callback!
msg->header.frame_id /map
msg->header.seq 174
msg->header.stamp 1589179517.877018896
msg->x = 10
msg->y = 20
msg->testarray.push_back 340 342 344 346 348
callback!
msg->header.frame_id /map
msg->header.seq 175
msg->header.stamp 1589179517.976842299
msg->x = 10
msg->y = 20
msg->testarray.push_back 342 344 346 348 350
```

ROS Service

- ROS Service

- 실시간으로 처리되지 않아도 되는 작업이거나 요청에 따라 한번만 실행해도 되는 작업의 경우, Client의 요청에 의해 실행되도록 하는 것이 효율적



ROS Service

- ROS Service

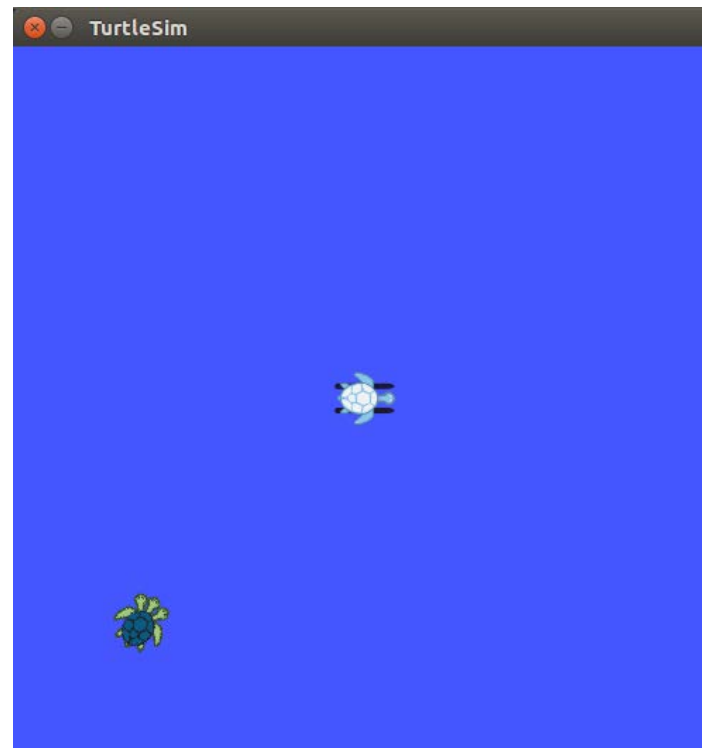
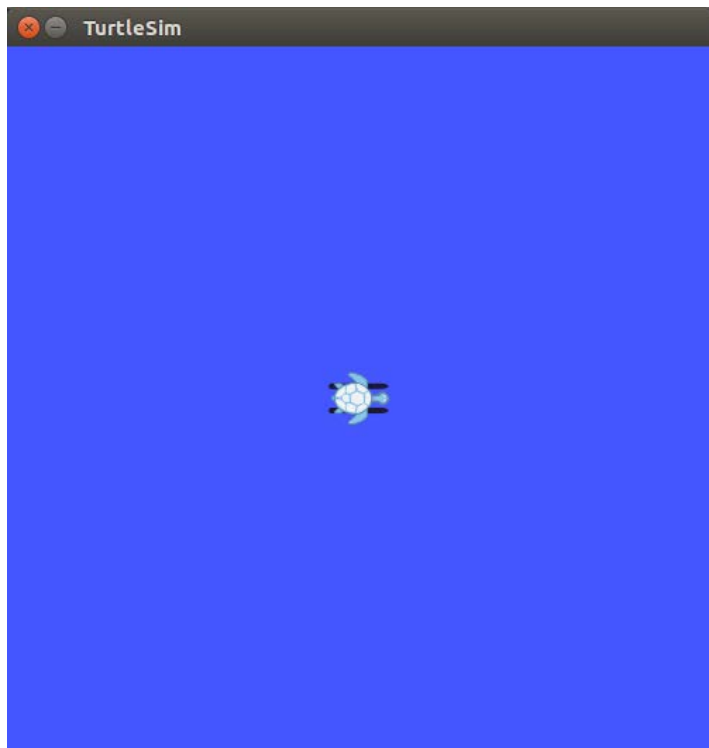
- Turtlesim에서의 예제 → Roservice 를 이용해 service 호출(새로운 turtle 생성)

```
$ rosrun turtlesim turtlesim_node
```

```
- Roservice command line
```

```
$ rosservice call /SERVICE_NAME arg1 arg2 ...
```

```
$ rosservice call /spawn 2 2 1 "myturtle"  
                        x y th name
```



ROS Service

- ROS Custom Service

- Testpkg에 srv폴더 생성, testsrv.srv 생성

```
$ mkdir msg  
$ touch testsrv.srv
```

- Testpkg에 srv폴더 생성, testsrv.srv 생성
- Srv에서 "---"으로 input/output 구분

```
# testsrv.srv  
int64 a          Type input1  
int64 b          Type input2  
---  
int64 sum        Type return1...
```

- Msg예제에서 추가했던 dependency는 이미 추가했으므로 생략.(package.xml, CMakeList findpackage)

```
<build_depend>message_generation</build_depend>  
<build_export_depend>message_generation</build_export_depend>  
<exec_depend>message_generation</exec_depend>
```

```
find_package(catkin REQUIRED COMPONENTS  
  roscpp  
  rospy  
  std_msgs  
  message_generation  
)
```



ROS Service

- ROS Custom Service

- CMakeList.txt add_service_files주석해제후 다음과 같이 작성

```
add_service_files(  
  FILES  
  testsrv.srv  
)
```

- Service 요청이 오면 연산할 Node생성.
- Src에 testsrv_server.cpp 생성

Srv의 이름 "add_two_ints"

```
// testsrv_server.cpp
```

```
#include "ros/ros.h"
```

```
#include "testpkg/testsrv.h"
```

```
bool add(testpkg::testsrv::Request &req, testpkg::testsrv::Response &res)  
{  
  res.sum = req.a + req.b;  
  ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);  
  ROS_INFO("sending back response: [%ld]", (long int)res.sum);  
  return true;  
}
```

```
int main(int argc, char **argv)  
{  
  ros::init(argc, argv, "testsrv_server");  
  ros::NodeHandle n;  
  ros::ServiceServer service = n.advertiseService("add_two_ints", add);  
  ROS_INFO("Ready to add two ints.");  
  ros::spin();  
  return 0;  
}
```



ROS Service

- ROS Custom Service

- Testsrv_client.cpp 작성
- Main arguments 로 더할 두숫자를 입력
- Srv이름 설정
- Srv 호출

```
//testsrv_client.cpp
#include "ros/ros.h"
#include "testpkg/testsrv.h"
#include <cstdlib>

int main(int argc, char **argv)
{
    ros::init(argc, argv, "testsrv_client");
    if (argc != 3)
    {
        ROS_INFO("usage: add_two_ints_client X Y");
        return 1;
    }
    ros::NodeHandle n;
    ros::ServiceClient client = n.serviceClient<testpkg::testsrv>("add_two_ints");
    testpkg::testsrv srv;

    srv.request.a = atoll(argv[1]);
    srv.request.b = atoll(argv[2]);

    if (client.call(srv))
    {
        ROS_INFO("Sum: %ld", (long int)srv.response.sum);
    }
    else
    {
        ROS_ERROR("Failed to call service add_two_ints");
        return 1;
    }
    return 0;
}
```

ROS Service

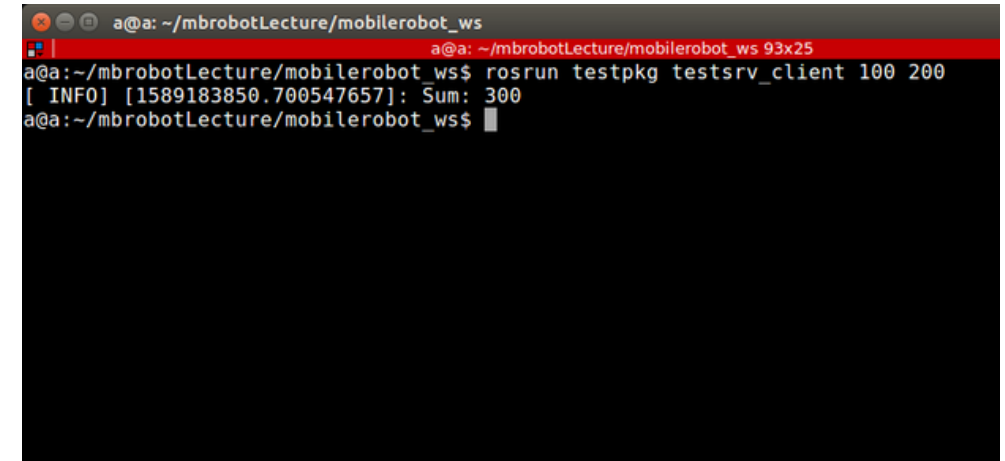
- ROS Custom Service

- CMakeList에 server와 client 빌드설정

```
add_executable(testsrv_server src/testsrv_server.cpp)
target_link_libraries(testsrv_server ${catkin_LIBRARIES})
add_dependencies(testsrv_server testpkg)
```

```
add_executable(testsrv_client src/testsrv_client.cpp)
target_link_libraries(testsrv_client ${catkin_LIBRARIES})
add_dependencies(testsrv_client testpkg)
```

→ Server가 실행중이면 service호출 가능!

A terminal window with a black background and white text. The title bar shows 'a@a: ~/mbrobotLecture/mobilerobot_ws' and 'a@a: ~/mbrobotLecture/mobilerobot_ws 93x25'. The terminal content shows a user running 'roslaunch testpkg testsrv_client 100 200' in the directory '~/mbrobotLecture/mobilerobot_ws'. The output is '[INFO] [1589183850.700547657]: Sum: 300'.

ROS Param

- ROS Param

- Topic, service와 다르게 parameter의 저장에 적합한 parameter server를 이용.
- Parameter를 .yaml파일에 저장한후 실행할 때 불러올 수 있기 때문에 source file내에서 parameter를 수정하지 않는 장점.

```
source_frame: "/map"
child_frame: "/base_link"
map_topic: "/map"
method: "tf" # tf or odom only #TODO: combined method
odom_topic: "/odom"
cost_map: true
pointcloud_topic: "/pcl"
global_map_static: false
map_size: [5.0, 3.0] #x,y (meter)
tolerance: 0.3
cost: 0.2 # meter
update_frequency: 10 # Hz
```

Rosparam set을 위한 yaml예제



Intelligent Robots Lab.
Chungbuk National University

ROS Param

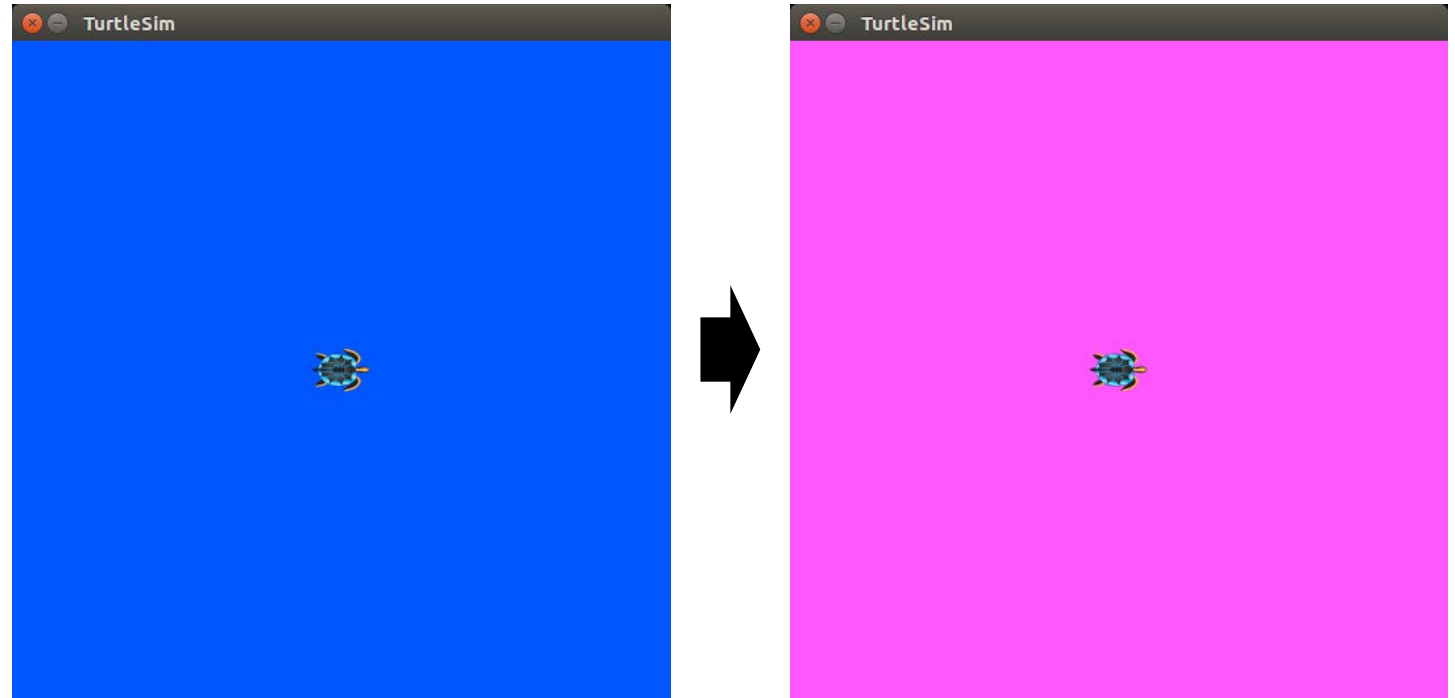
- ROS Param example with turtlesim

- Turtlesim node 실행

```
roslaunch turtlesim turtlesim_node
```

- 아래 명령어로 param list확인,
param의 값 변경

```
rosparam list  
rosparam set /background_r 255  
rosservice call /clear
```



ROS Param

- ROS Param

- Topic, service와 다르게 parameter의 저장에 적합한 parameter server를 이용.
- Parameter를 .yaml파일에 저장한후 실행할 때 불러올 수 있기 때문에 source file내에서 parameter를 수정하지 않는 장점.

```
source_frame: "/map"
child_frame: "/base_link"
map_topic: "/map"
method: "tf" # tf or odom only #TODO: combined method
odom_topic: "/odom"
cost_map: true
pointcloud_topic: "/pcl"
global_map_static: false
map_size: [5.0, 3.0] #x,y (meter)
tolerance: 0.3
cost: 0.2 # meter
update_frequency: 10 # Hz
```

Rosparam set을 위한 yaml 예시

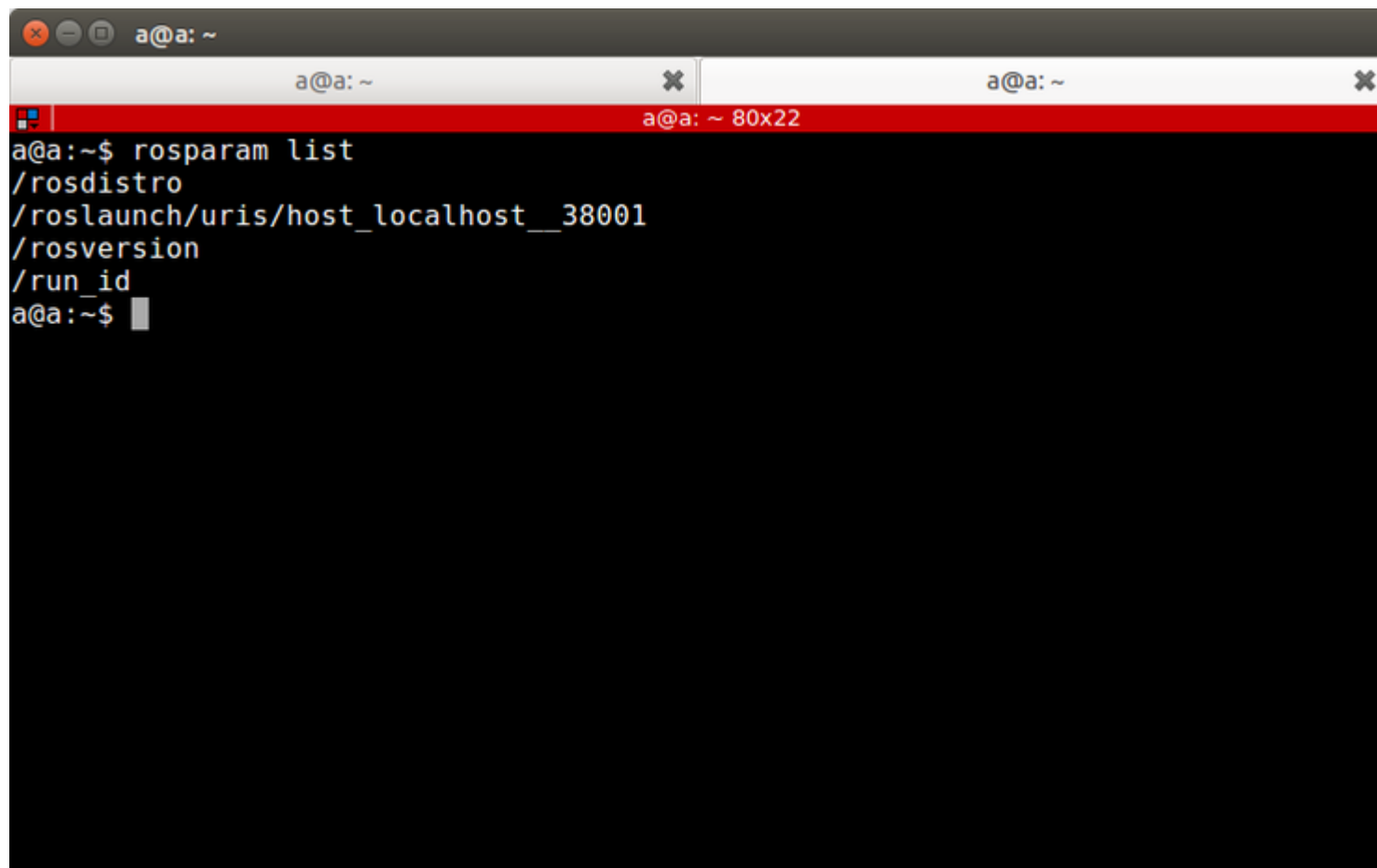


Intelligent Robots Lab.
Chungbuk National University

ROS Param

- ROS Param
 - Rosparam list로 현재 param확인

\$ rosparam list



```
a@a:~$ rosparam list
/rosdistro
/roslaunch/uris/host_localhost__38001
/rosversion
/run_id
a@a:~$
```

The image shows a terminal window with a dark background and a red title bar. The title bar contains the text 'a@a: ~ 80x22'. The terminal displays the command 'rosparam list' and its output, which lists four ROS parameters: '/rosdistro', '/roslaunch/uris/host_localhost__38001', '/rosversion', and '/run_id'. The prompt 'a@a:~\$' is visible at the bottom of the terminal.

ROS Param

- ROS Param

- Param의 정보를 갖는 yaml파일 생성

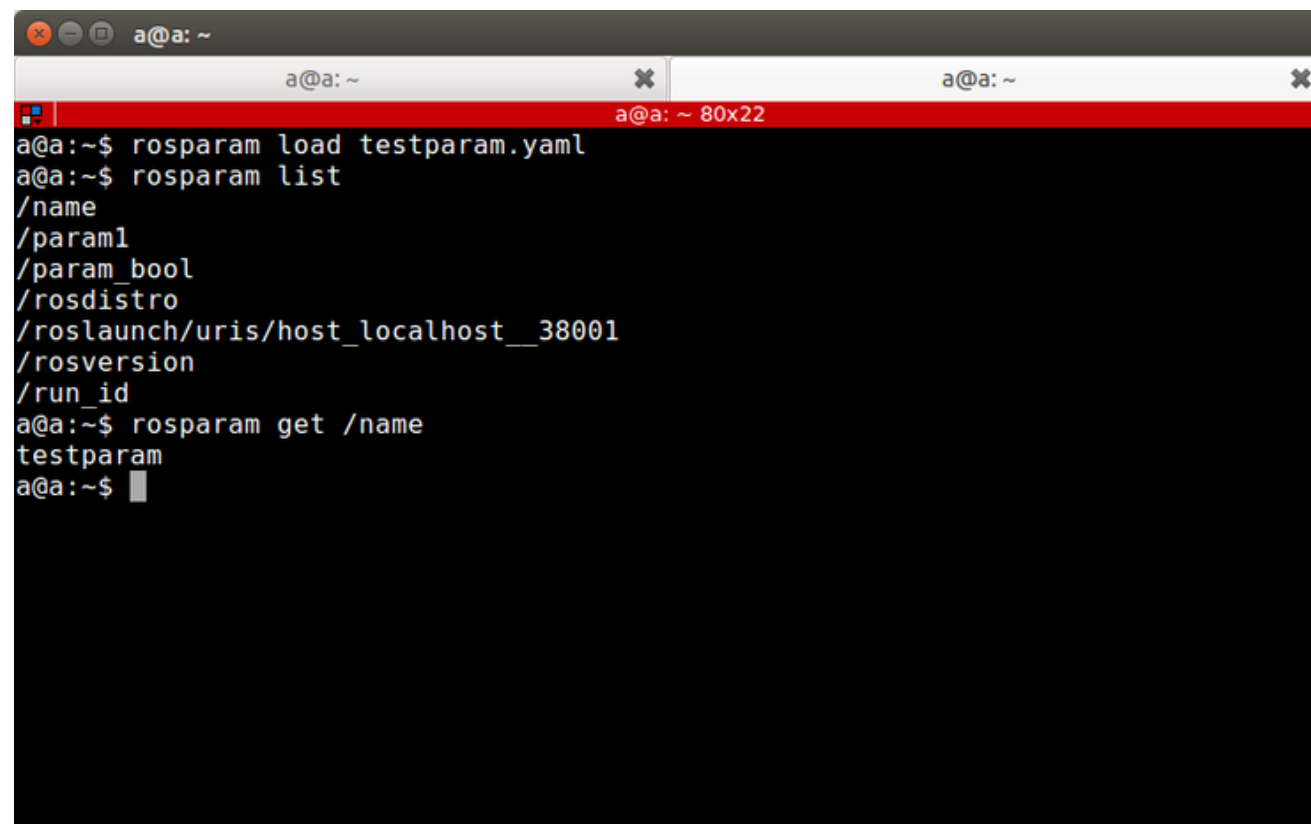
```
# testparam.yaml
param1: 1.5
name: "testparam"
param_bool: true
```

- Rosparam command를 이용한 param load

```
$ rosparam load testparam.yaml
```

- Rosparam command를 이용한 param 확인

```
$ rosparam get /PARAM_NAME
```

A terminal window with a dark background and a red title bar. The title bar contains the text 'a@a: ~' and 'a@a: ~ 80x22'. The terminal shows the following commands and output:

```
a@a:~$ rosparam load testparam.yaml
a@a:~$ rosparam list
/name
/param1
/param_bool
/rosdistro
/roslaunch/uris/host_localhost__38001
/rosversion
/run_id
a@a:~$ rosparam get /name
testparam
a@a:~$
```

ROS Param

- ROS Param – NODE에서 Param 사용
 - 실습을 위한 paramtest.cpp 생성
 - `n.getParam("/파라미터이름", 저장할 변수);`
 - CMakeList에 추가, 빌드 후 실행

```
#include "ros/ros.h"
#include <iostream>
#include <string>

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

    double param1;
    std::string name;
    bool paramBool;
    n.getParam("/param1", param1);
    n.getParam("/name", name);
    n.getParam("/param_bool", paramBool);

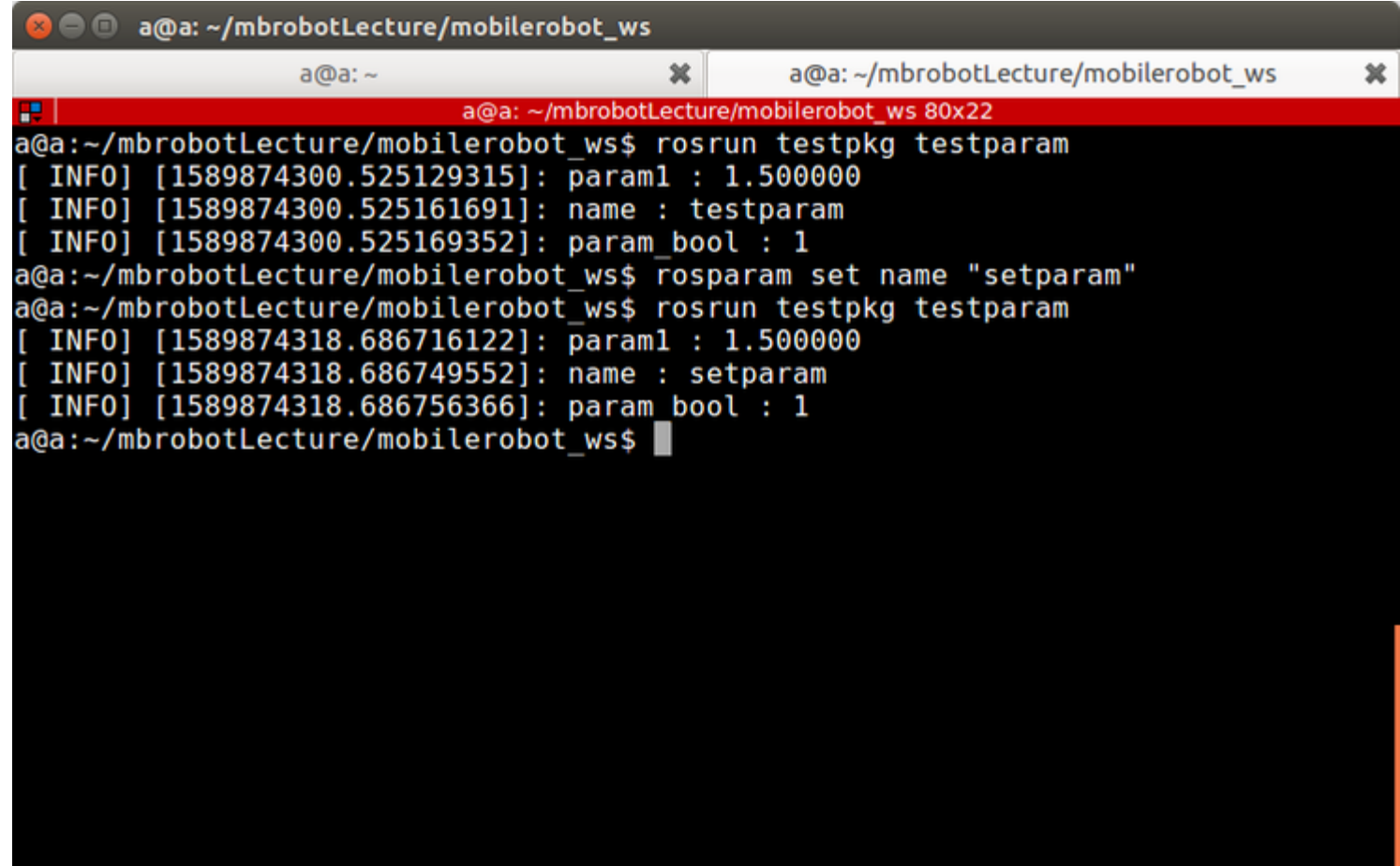
    ROS_INFO("param1 : %lf", param1);
    ROS_INFO("name : %s", name.c_str());
    ROS_INFO("param_bool : %d", paramBool);

    return 0;
}
```

ROS Param

- ROS Param – NODE에서 Param 사용

- Rosparam set을 이용한 parameter 변경
- rosparam set param명 param값
- 다시 빌드할 필요없이 사용가능
\$ rosparam set name "setparam"



```
a@a: ~/mbrobotLecture/mobilerobot_ws
a@a: ~
a@a: ~/mbrobotLecture/mobilerobot_ws 80x22
a@a:~/mbrobotLecture/mobilerobot_ws$ rosrun testpkg testparam
[ INFO] [1589874300.525129315]: param1 : 1.500000
[ INFO] [1589874300.525161691]: name : testparam
[ INFO] [1589874300.525169352]: param_bool : 1
a@a:~/mbrobotLecture/mobilerobot_ws$ rosparam set name "setparam"
a@a:~/mbrobotLecture/mobilerobot_ws$ rosrun testpkg testparam
[ INFO] [1589874318.686716122]: param1 : 1.500000
[ INFO] [1589874318.686749552]: name : setparam
[ INFO] [1589874318.686756366]: param bool : 1
a@a:~/mbrobotLecture/mobilerobot_ws$
```