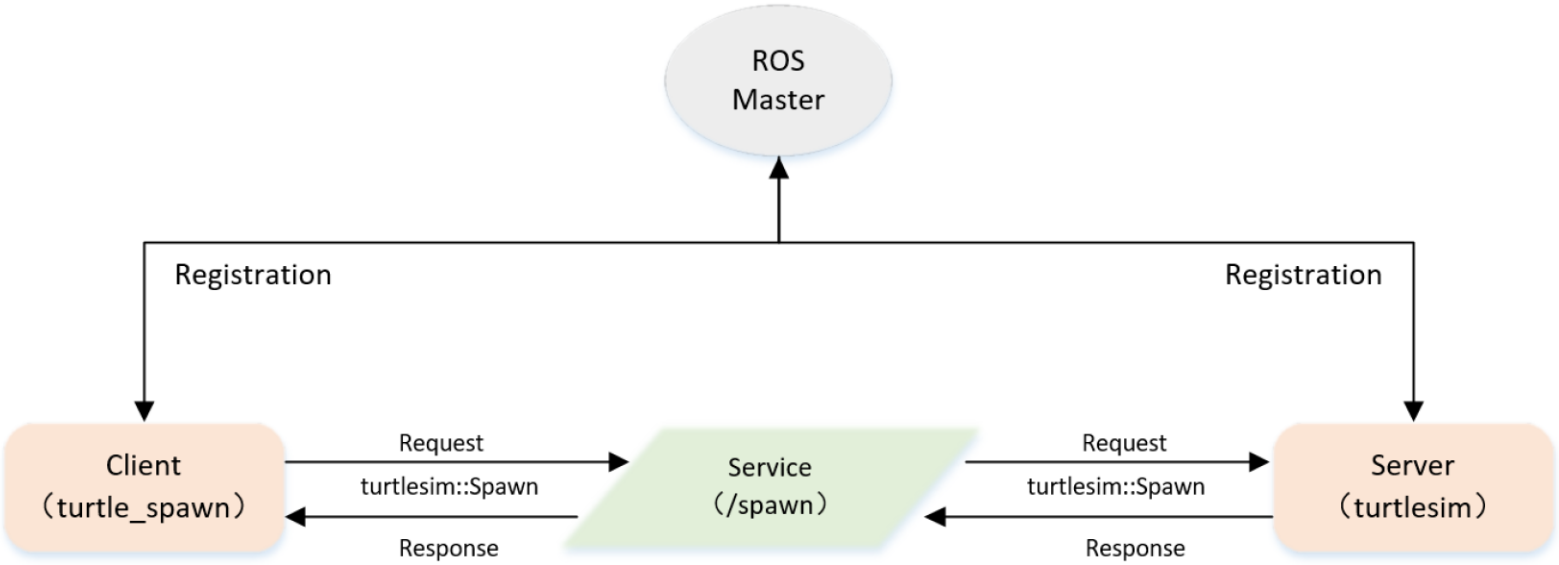
ECE 4703
Mobile Autonomous Robots

Jenny Zhen Yu
zhenyu@cpp.edu
Department of Electrical and Computer Engineering
California State Polytechnic University, Pomona

Lecture 8: Client Program

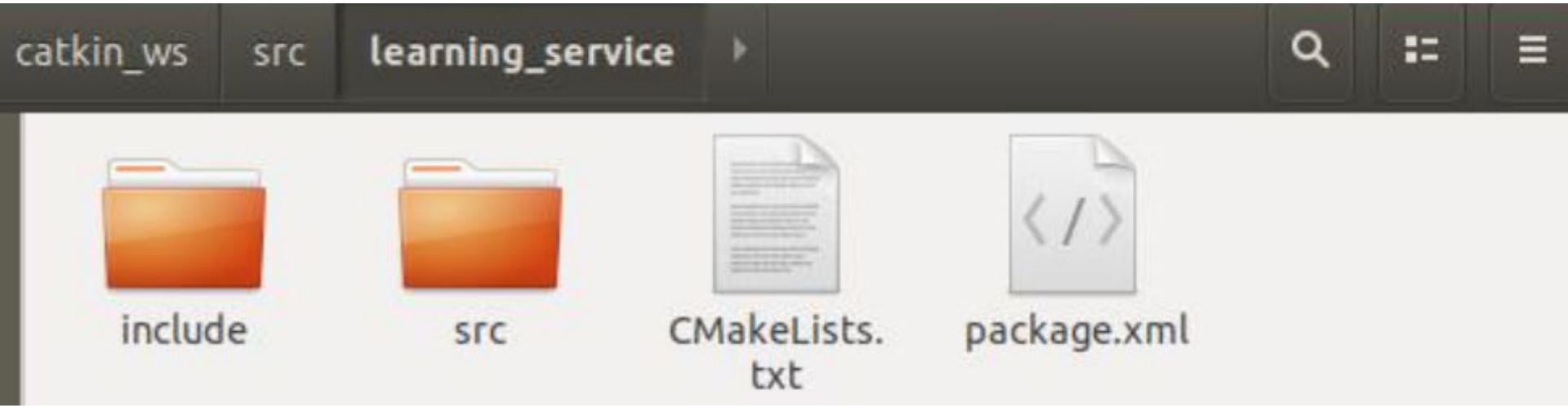
Client Model



Build Package

$ cd ~/catkin_ws/src

$ catkin_create_pkg learning_service roscpp rospy std_msgs geometry_msgs turtlesim



C++ Code

Open

```cpp
#include <ros/ros.h>
#include <turtlesim/Spawn.h>

int main(int argc, char** argv)
{

        ros::init(argc, argv, "turtle_spawn");


        ros::NodeHandle node;


        ros::service::waitForService("/spawn");
        ros::ServiceClient add_turtle = node.serviceClient<turtlesim::Spawn>("/spawn");


        turtlesim::Spawn srv;
        srv.request.x = 2.0;
        srv.request.y = 2.0;
        srv.request.name = "turtle2";


        ROS_INFO("Call service to spwan turtle[x:%0.6f, y:%0.6f, name:%s]",
                        srv.request.x, srv.request.y, srv.request.name.c_str());

        add_turtle.call(srv);

        ROS_INFO("Spwan turtle successfully [name:%s]", srv.response.name.c_str());

        return 0;
};
```

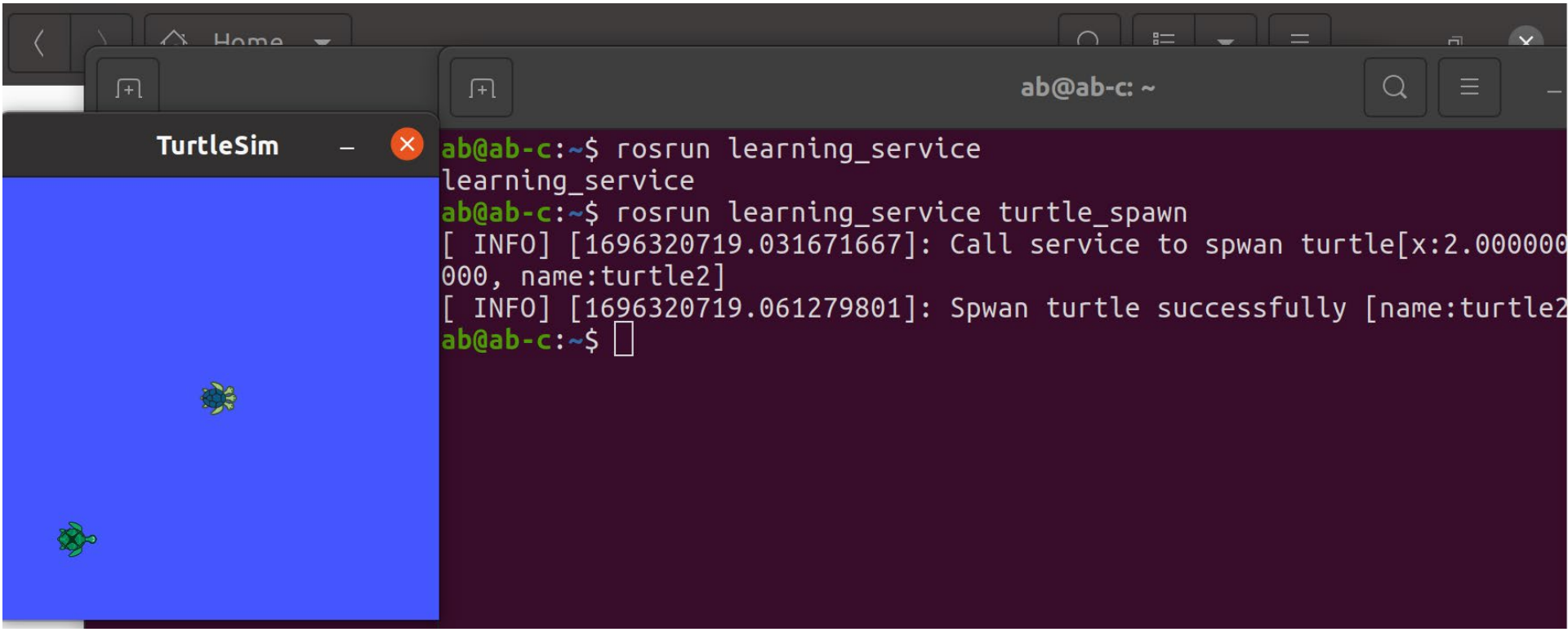
Compiling Code

```
## Declare a C++ executable
## With catkin_make all packages are built within a single CMake context
## The recommended prefix ensures that target names across packages don't collide
# add_executable(${PROJECT_NAME}_node src/learning_service_node.cpp)

## Specify libraries to link a library or executable target against
# target_link_libraries(${PROJECT_NAME}_node
#   ${catkin_LIBRARIES}
# )

add_executable(turtle_spawn src/turtle_spawn.cpp)
target_link_libraries(turtle_spawn ${catkin_LIBRARIES})
```

Compiling Code

```
$ cd ~/catkin_ws
$ catkin_make
$ source devel/setup.bash
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun learning_service turtle_spawn
```

Compiling Code



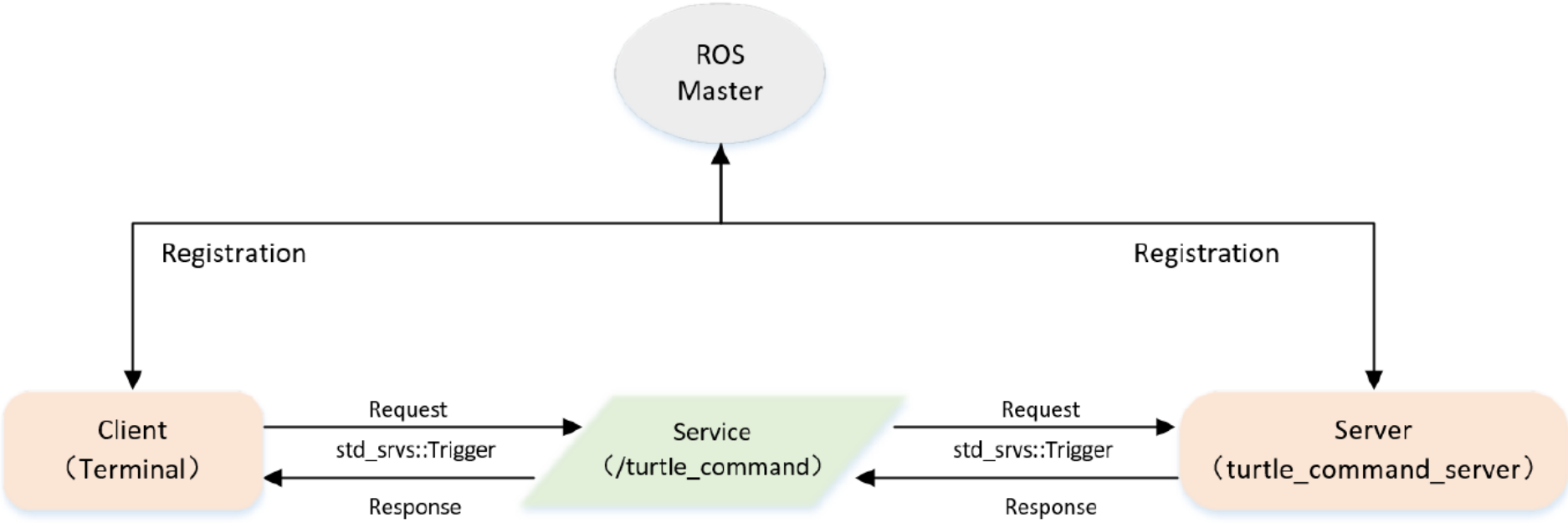
Python Code

Open

```
1 import sys
2 import rospy
3 from turtlesim.srv import Spawn
4
5 def turtle_spawn():
6
7     rospy.init_node('turtle_spawn')
8
9
10    rospy.wait_for_service('/spawn')
11    try:
12        add_turtle = rospy.ServiceProxy('/spawn', Spawn)
13
14
15        response = add_turtle(2.0, 2.0, 0.0, "turtle2")
16        return response.name
17    except rospy.ServiceException, e:
18        print "Service call failed: %s"%e
19
20 if __name__ == "__main__":
21
22    print "Spwan turtle successfully [name:%s]" %(turtle_spawn())
23
24
```

Lecture 9: Server Program

Server Model



```cpp
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include <std_srvs/Trigger.h>

ros::Publisher turtle_vel_pub;
bool pubCommand = false;

bool commandCallback(std_srvs::Trigger::Request  &req,
                     std_srvs::Trigger::Response &res)
{
        pubCommand = !pubCommand;


    ROS_INFO("Publish turtle velocity command [%s]", pubCommand==true?"Yes":"No");

        res.success = true;
        res.message = "Change turtle command state!";

    return true;
}

int main(int argc, char **argv)
{

    ros::init(argc, argv, "turtle_command_server");

    ros::NodeHandle n;

    ros::ServiceServer command_service = n.advertiseService("/turtle_command", commandCallback);

        turtle_vel_pub = n.advertise<geometry_msgs::Twist>("/turtle1/cmd_vel", 10);


    ROS_INFO("Ready to receive turtle command.");

        ros::Rate loop_rate(10);

        while(ros::ok())
        {

        ros::spinOnce();

                if(pubCommand)
                {
                        geometry_msgs::Twist vel_msg;
                        vel_msg.linear.x = 0.5;
                        vel_msg.angular.z = 0.2;
                        turtle_vel_pub.publish(vel_msg);
                }

            loop_rate.sleep();
        }

    return 0;
}
```

Compiling Code

```
## Declare a C++ executable
## With catkin_make all packages are built within a single CMake context
## The recommended prefix ensures that target names across packages don't collide
# add_executable(${PROJECT_NAME}_node src/learning_service_node.cpp)

## Specify libraries to link a library or executable target against
# target_link_libraries(${PROJECT_NAME}_node
#   ${catkin_LIBRARIES}
# )

add_executable(turtle_spawn src/turtle_spawn.cpp)
target_link_libraries(turtle_spawn ${catkin_LIBRARIES})

add_executable(turtle_command_server src/turtle_command_server.cpp)
target_link_libraries(turtle_command_server ${catkin_LIBRARIES})
```

Compiling Code

```
$ cd ~/catkin_ws
$ catkin_make
$ source devel/setup.bash
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun learning_service turtle_command_server
$ rosservice call /turtle_command "{}"
```

Double click Tab

Rosrun

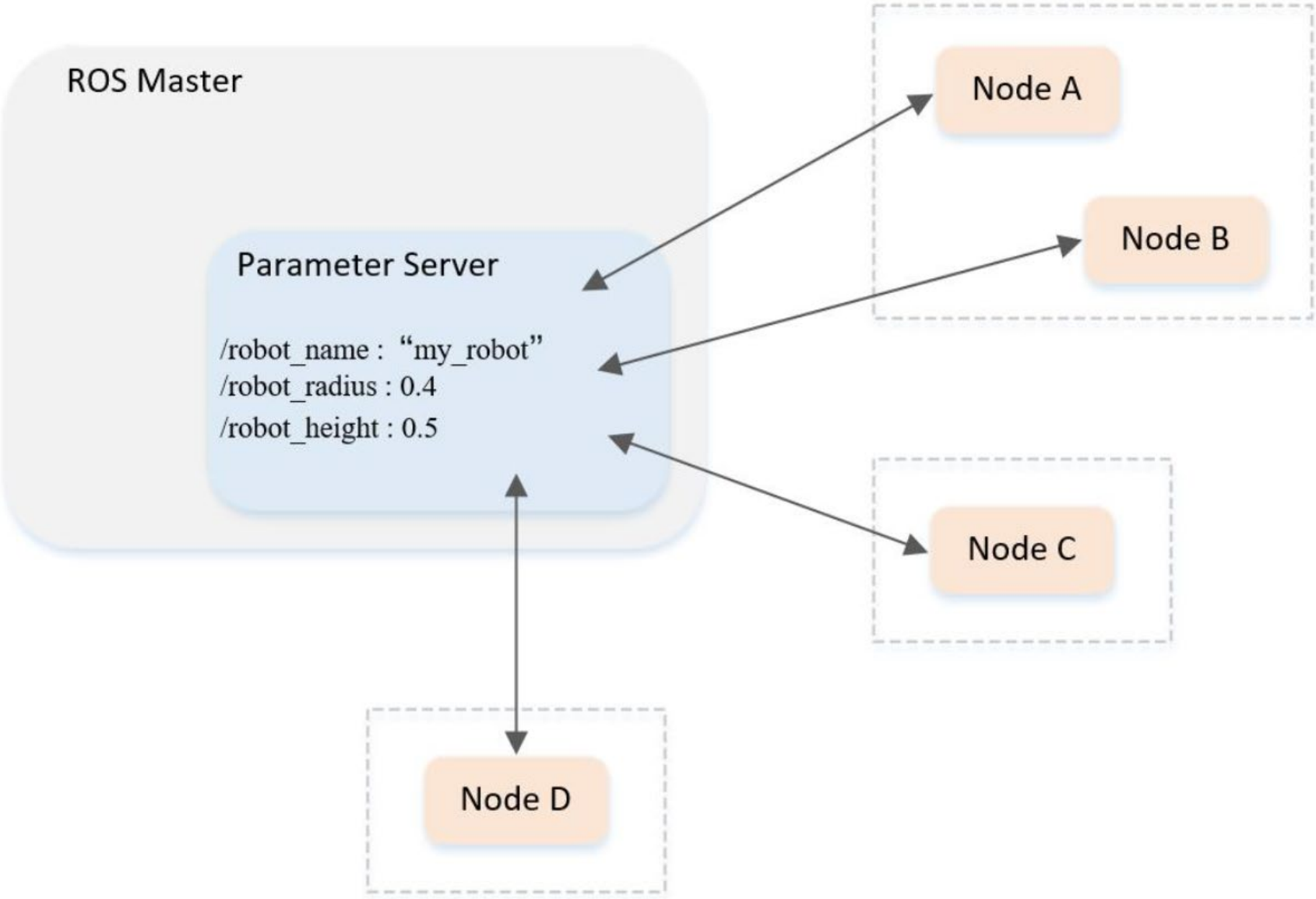

ab@ab-c: ~
ab@ab-c:~$ rosservice call /turtle_command "{}"
success: True
message: "Change turtle command state!"
ab@ab-c:~$
TurtleSim

Python Code

```python
import rospy
import thread,time
from geometry_msgs.msg import Twist
from std_srvs.srv import Trigger, TriggerResponse

pubCommand = False;
turtle_vel_pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)

def command_thread():
        while True:
                if pubCommand:
                        vel_msg = Twist()
                        vel_msg.linear.x = 0.5
                        vel_msg.angular.z = 0.2
                        turtle_vel_pub.publish(vel_msg)

                time.sleep(0.1)

def commandCallback(req):
        global pubCommand
        pubCommand = bool(1-pubCommand)


        rospy.loginfo("Publish turtle velocity command![%d]", pubCommand)


        return TriggerResponse(1, "Change turtle command state!")

def turtle_command_server():

    rospy.init_node('turtle_command_server')


    s = rospy.Service('/turtle_command', Trigger, commandCallback)


    print "Ready to receive turtle command."

    thread.start_new_thread(command_thread, ())
    rospy.spin()

if __name__ == "__main__":
    turtle_command_server()
```

Lecture 11: Parameter Server

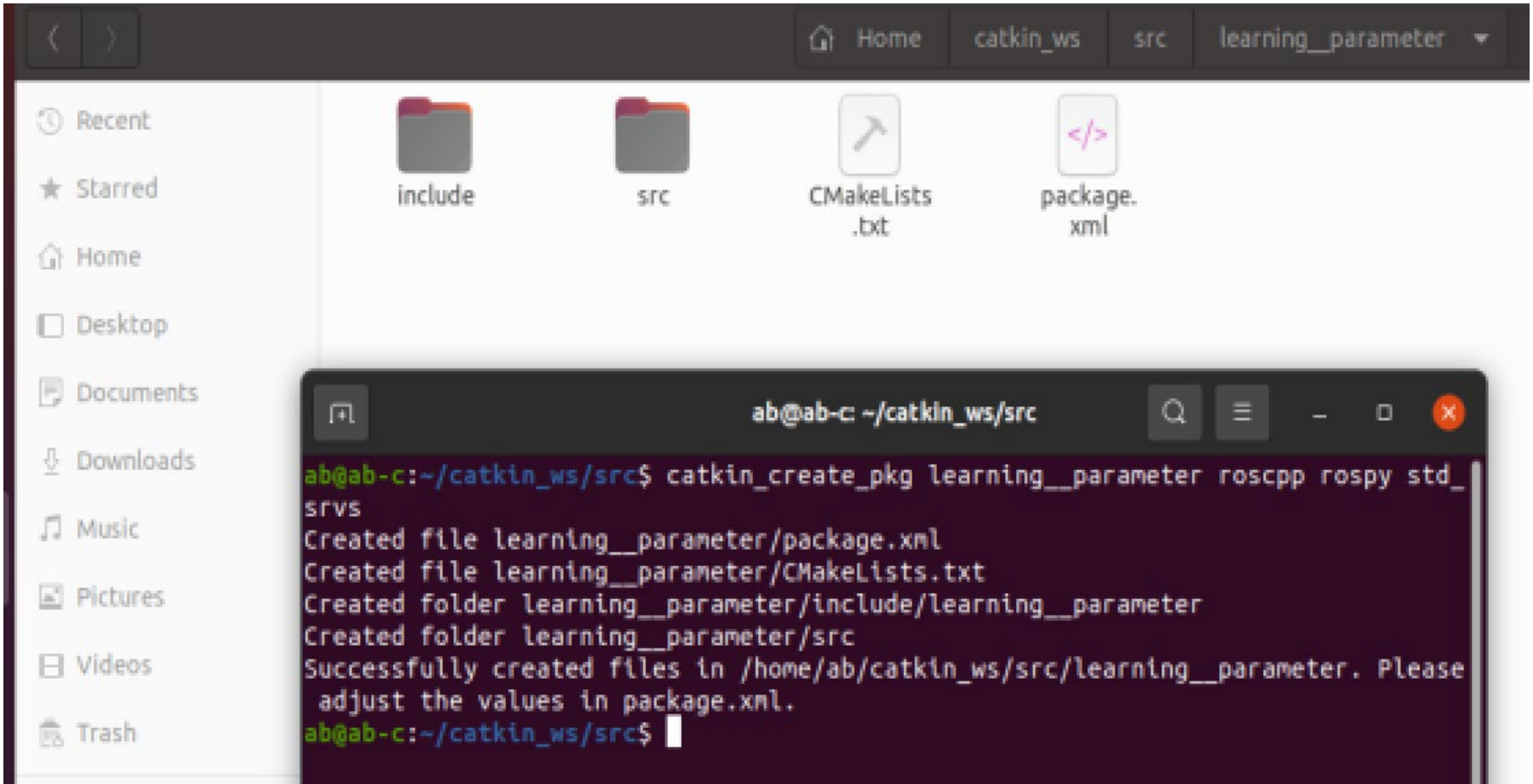
Parameter Server Model



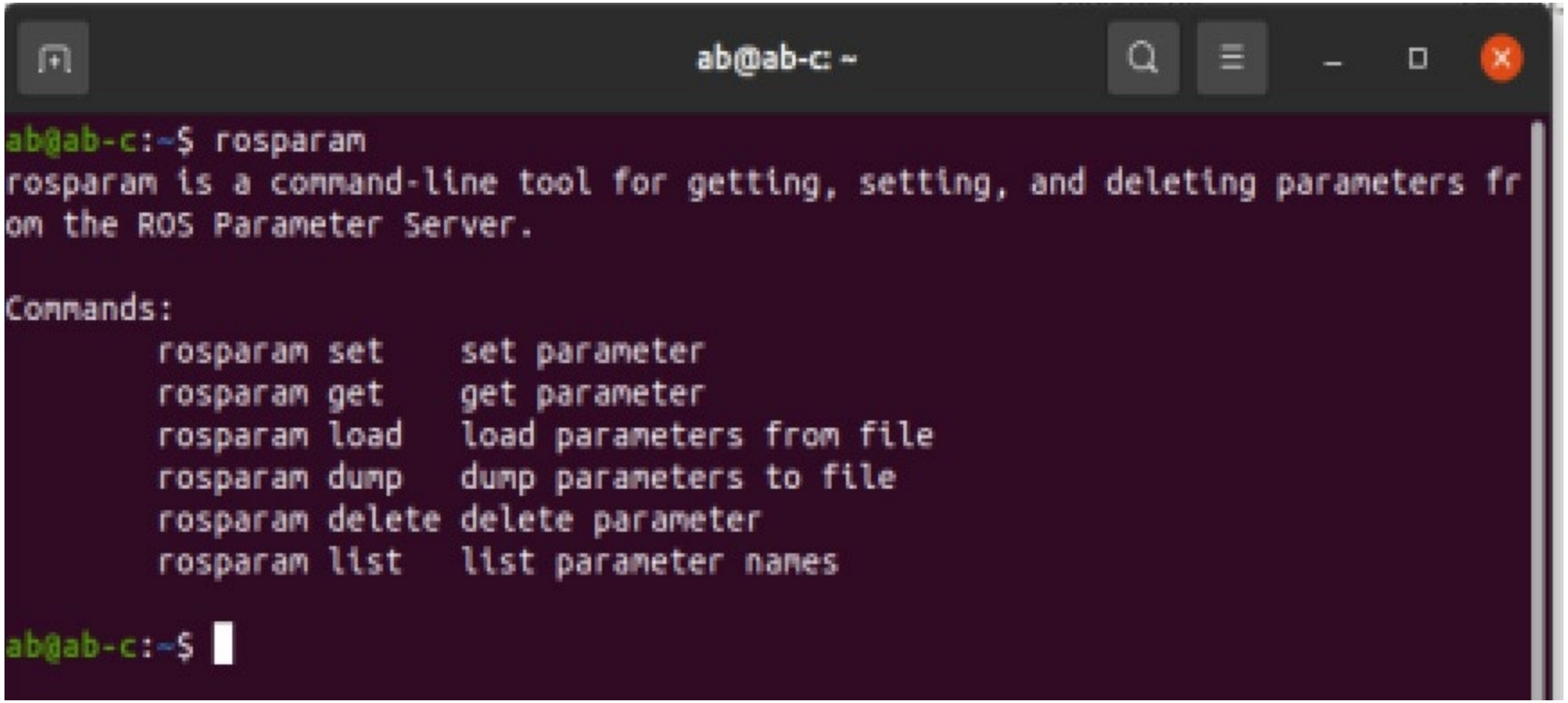
Create Package

$ cd ~/catkin_ws/src

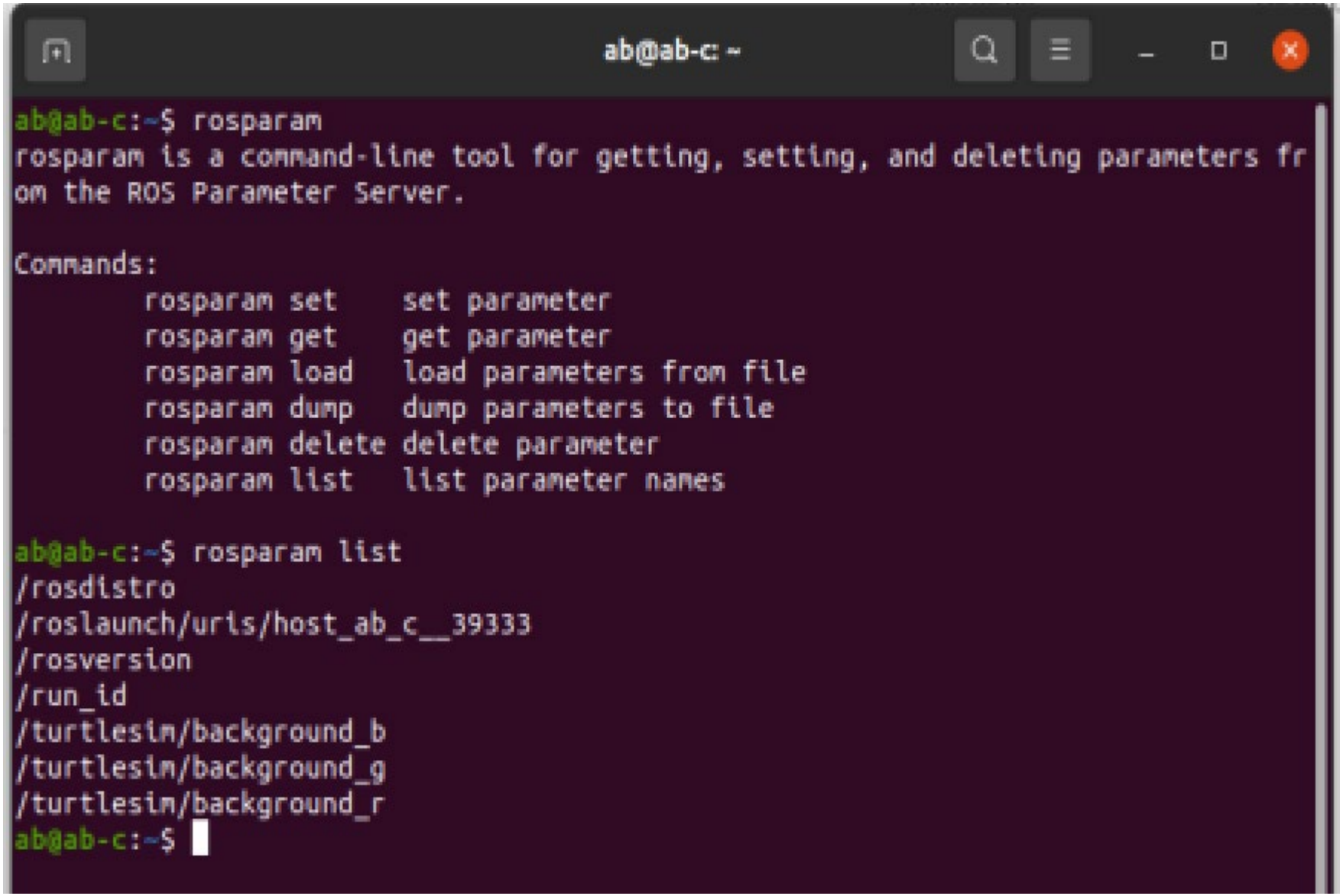
$ catkin_create_pkg learning_parameter roscpp rospy std_srvs



rosparam


ab@ab-c: ~
ab@ab-c:~$ rosparam
rosparam is a command-line tool for getting, setting, and deleting parameters fr
om the ROS Parameter Server.
Commands:
rosparam set set parameter
rosparam get get parameter
rosparam load load parameters from file
rosparam dump dump parameters to file
rosparam delete delete parameter
rosparam list list parameter names
ab@ab-c:~$

Rosparam: list parameter



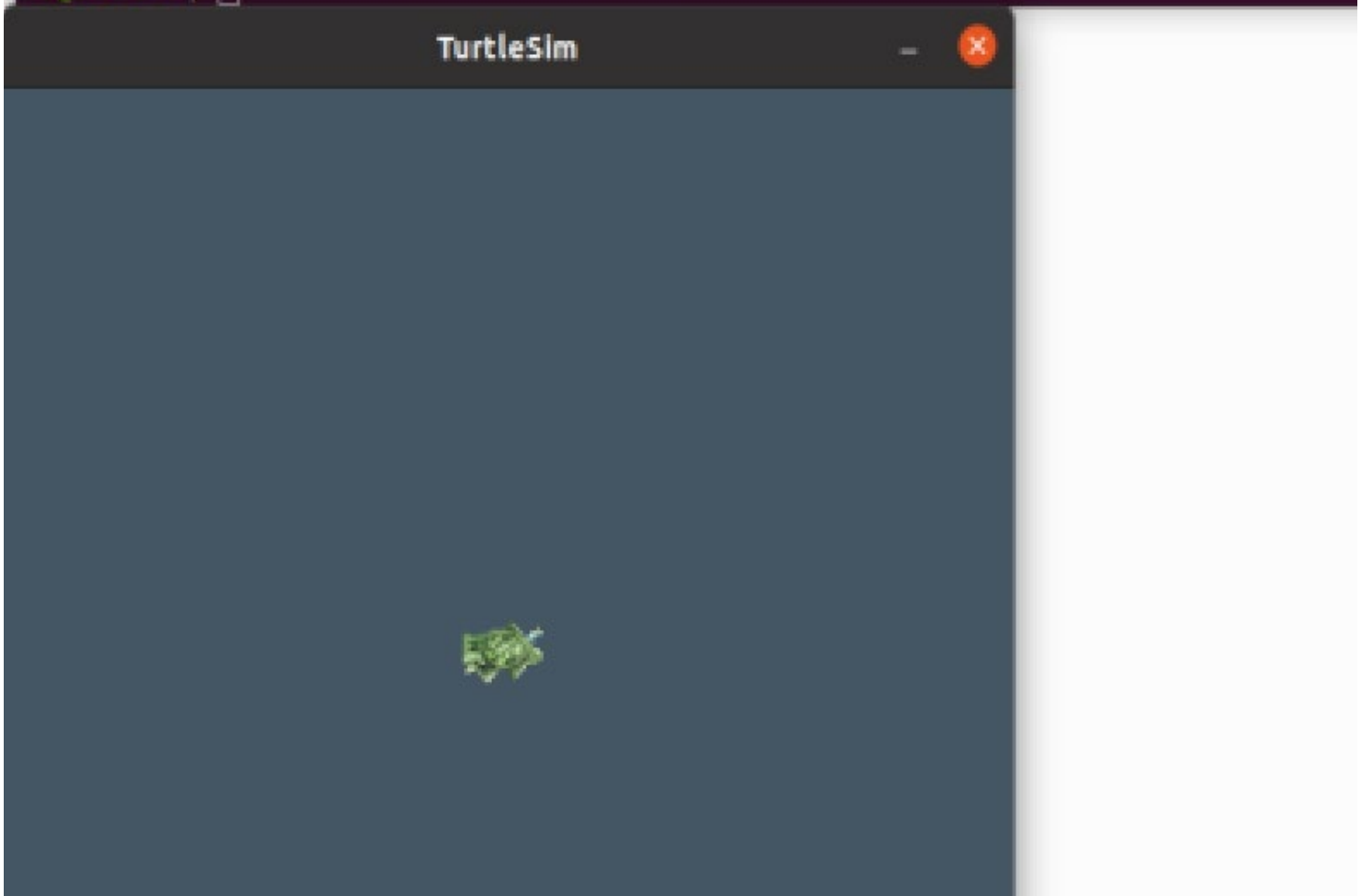
Rosparam: display parameter

```
ab@ab-c:~$ rosparam get /turtlesim/background_b
255
ab@ab-c:~$ rosparam get /turtlesim/background_r
69
```

```
ab@ab-c:~$ rosparam get /rosversion
'1.16.0
```

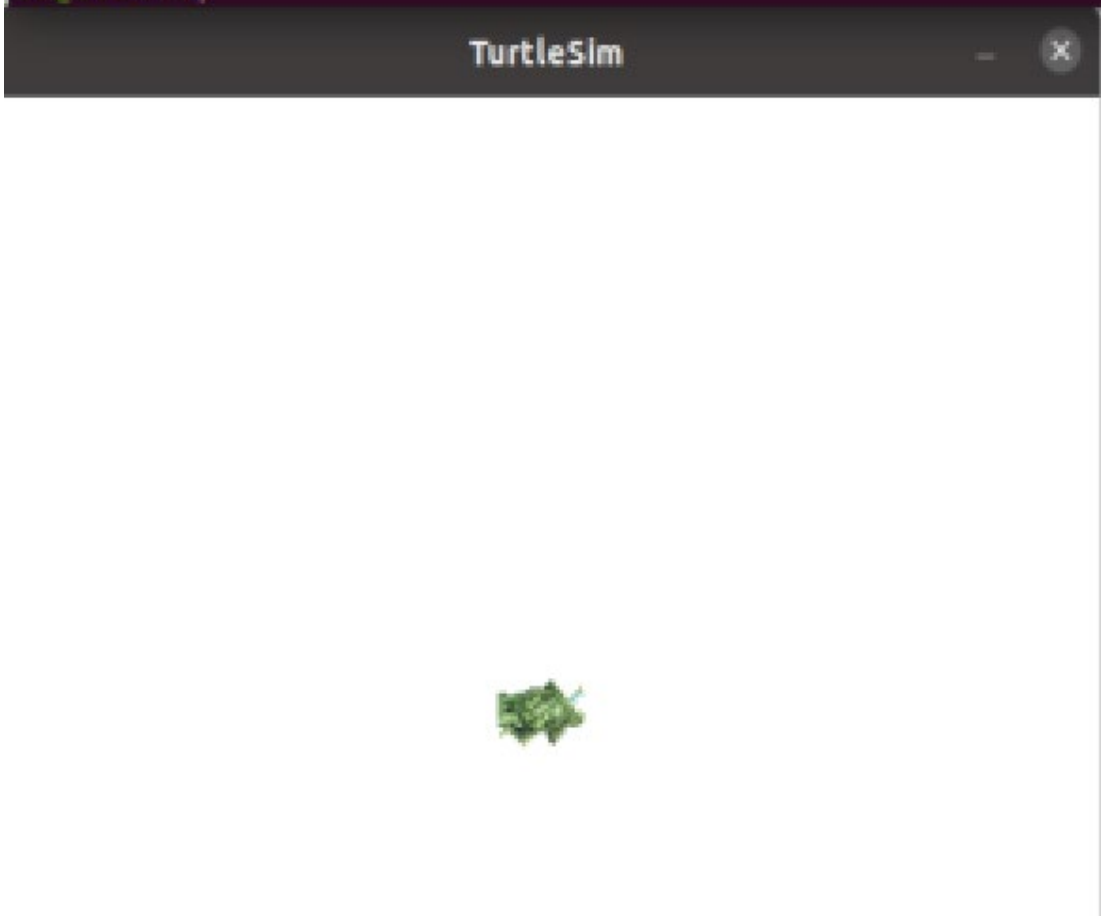
Rosparam: set parameter

```
ab@ab-c:~$ rosparam set /turtlesim/background_b 100
ab@ab-c:~$ rosparam get /turtlesim/background_b
100
ab@ab-c:~$
ab@ab-c:~$ rosservice call /clear "{}"

ab@ab-c:~$
```



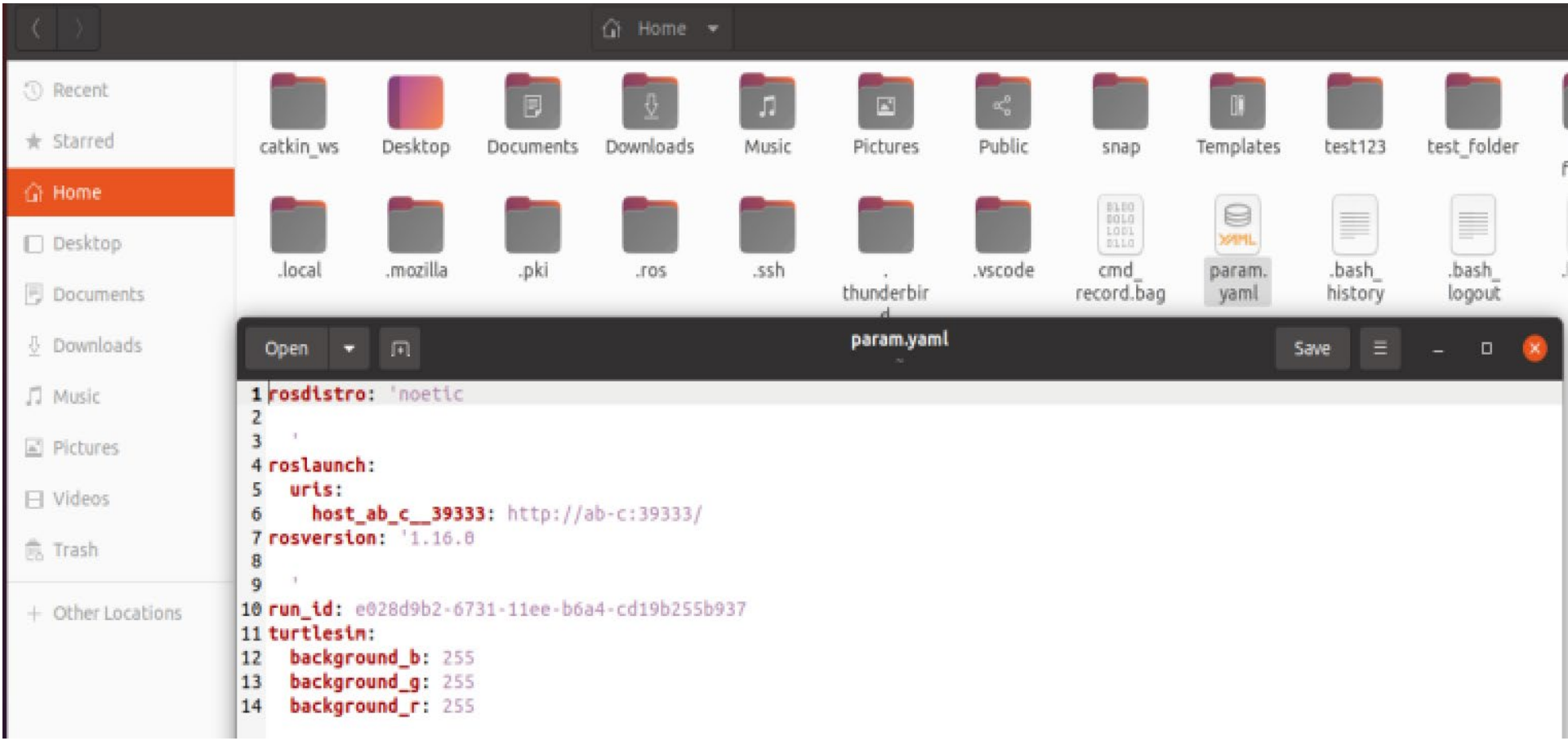
Rosparam: Change Turtle Color

```
ab@ab-c:~$ rosparam set /turtlesim/background_b 255
ab@ab-c:~$ rosparam set /turtlesim/background_r 255
ab@ab-c:~$ rosparam set /turtlesim/background_g 255
ab@ab-c:~$ rosservice call /clear "{}"

ab@ab-c:~$
```



Rosparam: save parameter

```
ab@ab-c:~$ rosparam dump param.yaml
ab@ab-c:~$
```



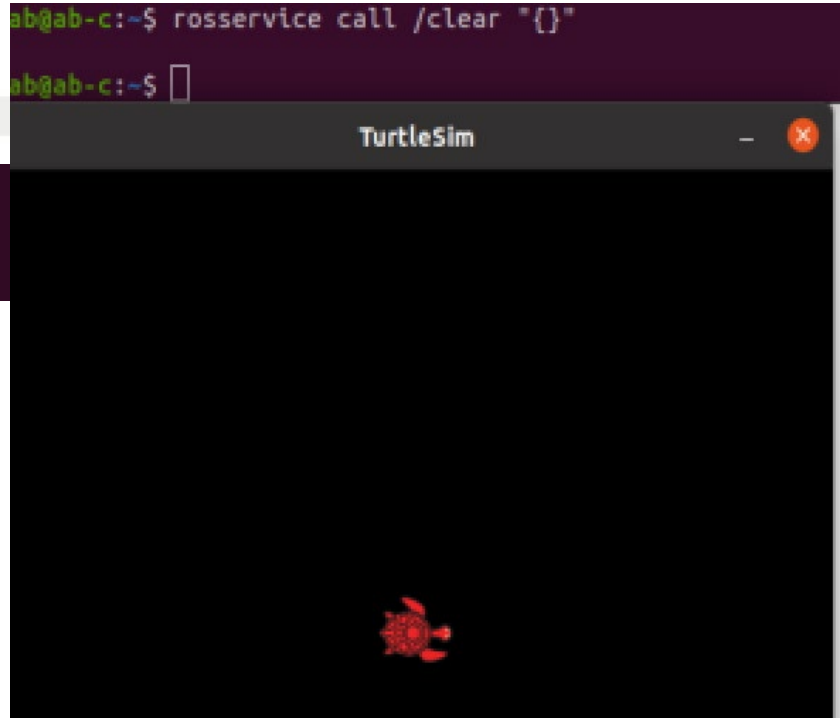
Rosparam: load parameter

Open param.yaml

```
1 rosdistro: 'noetic
2
3   '
4 roslaunch:
5   uris:
6     host_ab_c__39333: http://ab-c:39333/
7 rosversion: '1.16.0
8
9   '
10 run_id: e028d9b2-6731-11ee-b6a4-cd19b255b937
11 turtlesim:
12   background_b: 0
13   background_g: 0
14   background_r: 0
```

```
ab@ab-c:~$ rosparam load param.yaml
ab@ab-c:~$ rosparam get /turtlesim/background_b
0
```

```
ab@ab-c:~$ rosservice call /clear "{}"

ab@ab-c:~$
```



Rosparam: delete parameter

```
ab@ab-c:~$ rosparam delete /turtlesim/background_g
ab@ab-c:~$
ab@ab-c:~$ rosparam list
/rosdistro
/roslaunch/uris/host_ab_c__39333
/rosversion
/run_id
/turtlesim/background_b
/turtlesim/background_r
ab@ab-c:~$
ab@ab-c:~$ rosservice call /clear "{}"

ab@ab-c:~$
```

Create Parameter Code C++

Open | parameter_config.cpp — ~/catkin_ws/src/learning_parameter/src | Save

```
1 #include <string>
2 #include <ros/ros.h>
3 #include <std_srvs/Empty.h>
4
5 int main(int argc, char **argv)
6 {
7         int red, green, blue;
8
9
10     ros::init(argc, argv, "parameter_config");
11
12
13     ros::NodeHandle node;
14
15
16         ros::param::get("/turtlesim/background_r", red);
17         ros::param::get("/turtlesim/background_g", green);
18         ros::param::get("/turtlesim/background_b", blue);
19
20         ROS_INFO("Get Backgroud Color[%d, %d, %d]", red, green, blue);
21
22
23         ros::param::set("/turtlesim/background_r", 255);
24         ros::param::set("/turtlesim/background_g", 255);
25         ros::param::set("/turtlesim/background_b", 255);
26
27         ROS_INFO("Set Backgroud Color[255, 255, 255]");
28
29
30         ros::param::get("/turtlesim/background_r", red);
31         ros::param::get("/turtlesim/background_g", green);
32         ros::param::get("/turtlesim/background_b", blue);
33
34         ROS_INFO("Re-get Backgroud Color[%d, %d, %d]", red, green, blue);
35
36
37         ros::service::waitForService("/clear");
38         ros::ServiceClient clear_background = node.serviceClient<std_srvs::Empty>("/clear");
39         std_srvs::Empty srv;
40         clear_background.call(srv);
41
42         sleep(1);
43
44     return 0;
45 }
```

Compiling Code

add_executable(parameter_config src/parameter_config.cpp)

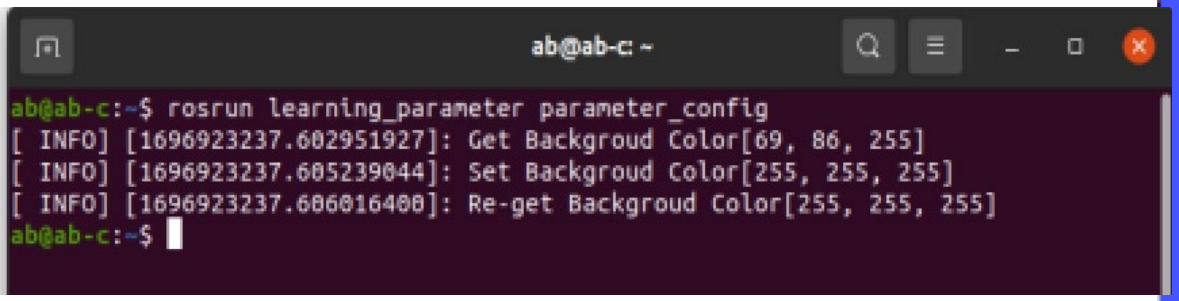
target_link_libraries(parameter_config ${catkin_LIBRARIES})

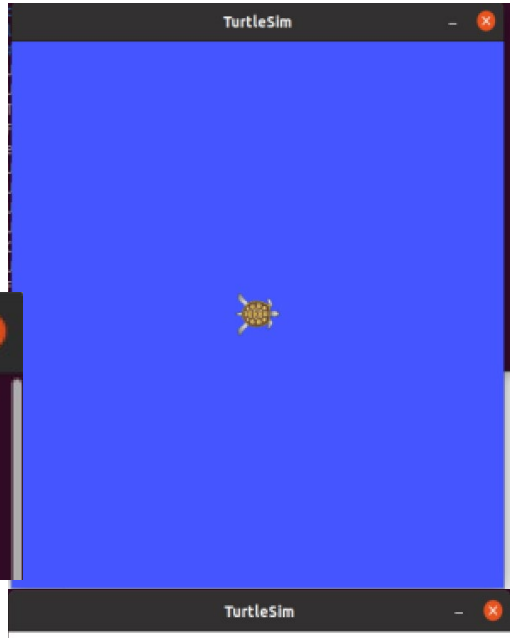
```
143
144 ## Add cmake target dependencies of the executable
145 ## same as for the library above
146 # add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
147
148 ## Specify libraries to link a library or executable target against
149 # target_link_libraries(${PROJECT_NAME}_node
150 #   ${catkin_LIBRARIES}
151 # )
152
153 add_executable(parameter_config src/parameter_config.cpp)
154 target_link_libraries(parameter_config ${catkin_LIBRARIES})
155
156 #############
157 ## Install ##
158 #############
```

Compiling Code

```
$ cd ~/catkin_ws
$ catkin_make
$ source devel/setup.bash
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun learning_parameter parameter_config
```

Rosrun


ab@ab-c: ~
ab@ab-c:~$ rosrun learning_parameter parameter_config
[INFO] [1696923237.602951927]: Get Backgroud Color[69, 86, 255]
[INFO] [1696923237.605239044]: Set Backgroud Color[255, 255, 255]
[INFO] [1696923237.606016400]: Re-get Backgroud Color[255, 255, 255]
ab@ab-c:~$


TurtleSim
TurtleSim

Create Parameter Code Python

Text Editor

Open

parameter_config.py
~/catkin_ws/scripts

```
1  import sys
2  import rospy
3  from std_srvs.srv import Empty
4
5  def parameter_config():
6
7      rospy.init_node('parameter_config', anonymous=True)
8
9
10     red   = rospy.get_param('/background_r')
11     green = rospy.get_param('/background_g')
12     blue  = rospy.get_param('/background_b')
13
14     rospy.loginfo("Get Backgroud Color[%d, %d, %d]", red, green, blue)
15
16
17     rospy.set_param("/background_r", 255);
18     rospy.set_param("/background_g", 255);
19     rospy.set_param("/background_b", 255);
20
21     rospy.loginfo("Set Backgroud Color[255, 255, 255]");
22
23
24     red   = rospy.get_param('/background_r')
25     green = rospy.get_param('/background_g')
26     blue  = rospy.get_param('/background_b')
27
28     rospy.loginfo("Get Backgroud Color[%d, %d, %d]", red, green, blue)
29
30
31     rospy.wait_for_service('/clear')
32     try:
33         clear_background = rospy.ServiceProxy('/clear', Empty)
34
35
36         response = clear_background()
37         return response
38     except rospy.ServiceException, e:
39         print "Service call failed: %s"%e
40
41 if __name__ == "__main__":
42     parameter_config()
```

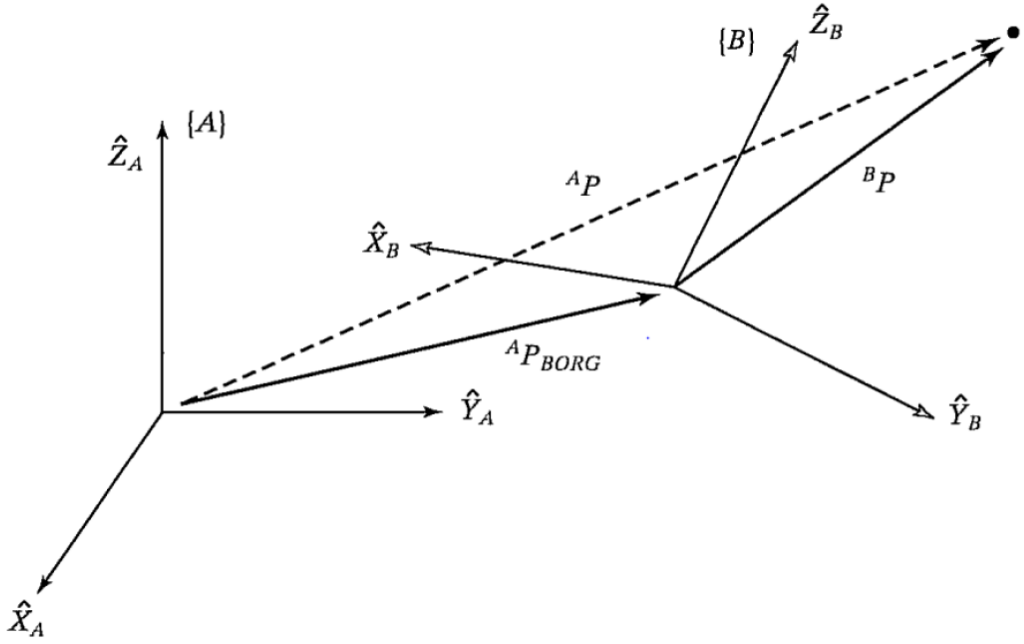
Lecture 12: ROS Coordinate System

Coordinate Transformation

$$^{A}P = {}^{A}_{B}R\ {}^{B}P + {}^{A}P_{BORG}.$$

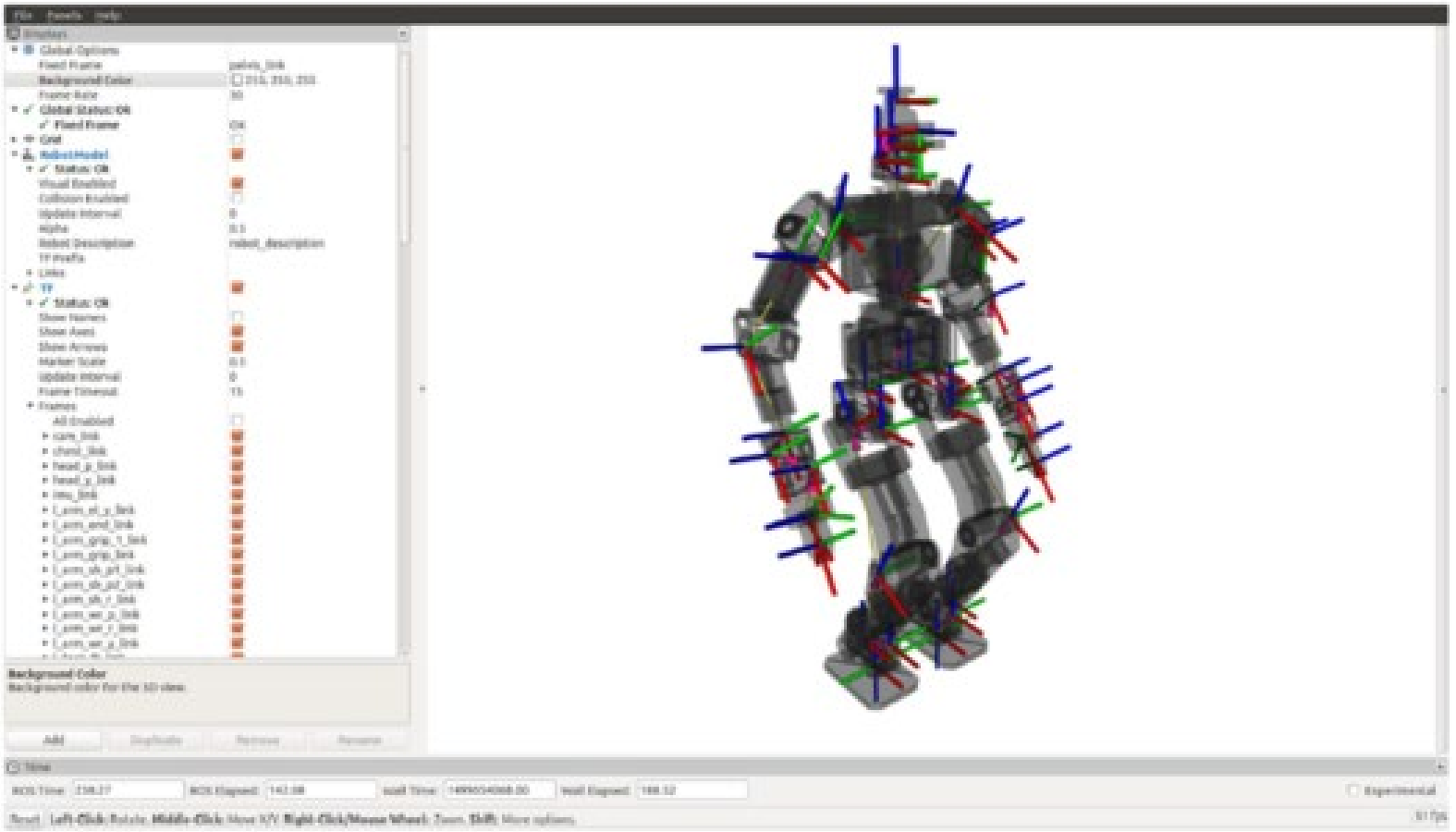
$$^{A}P = {}^{A}_{B}T\ {}^{B}P.$$

$$\begin{bmatrix} {}^{A}P \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} & {}^{A}_{B}R & & {}^{A}P_{BORG} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} {}^{B}P \\ 1 \end{bmatrix}.$$

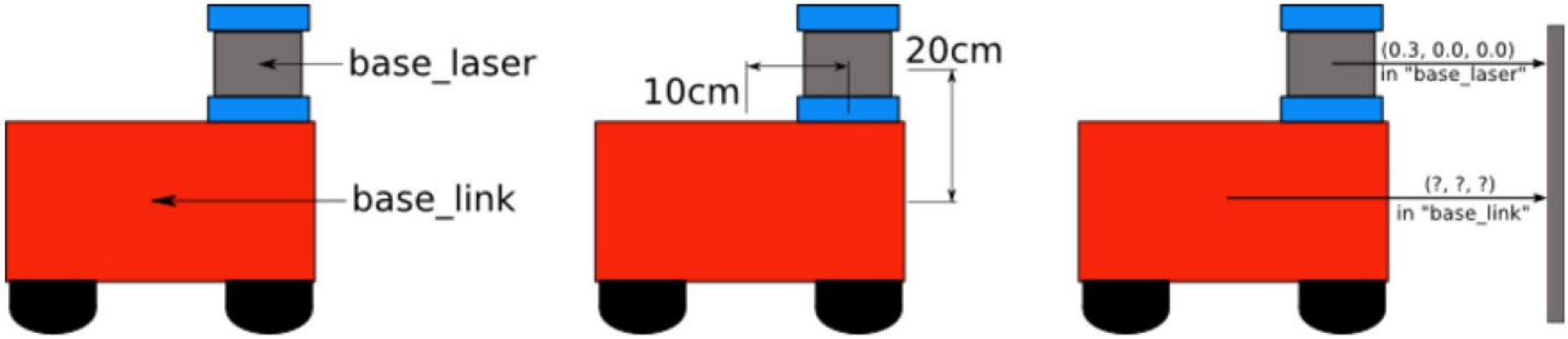


Coordinate transformation(TF, transform)

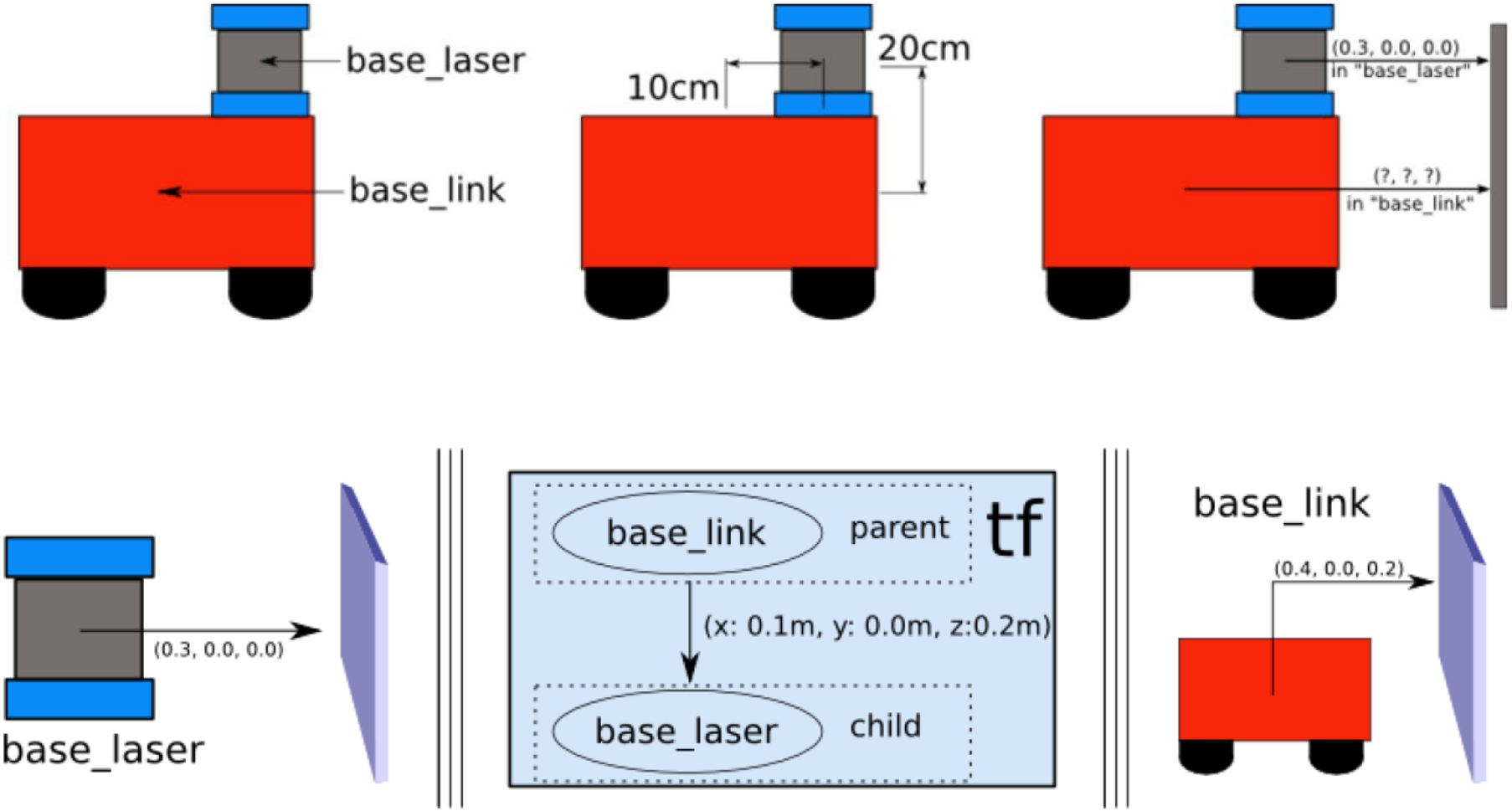
- Relative coordinate transformation of each joint
  - Indicates the relationship between joints in the form of tree structure



Coordinate transformation(TF, transform)



Coordinate transformation(TF, transform)



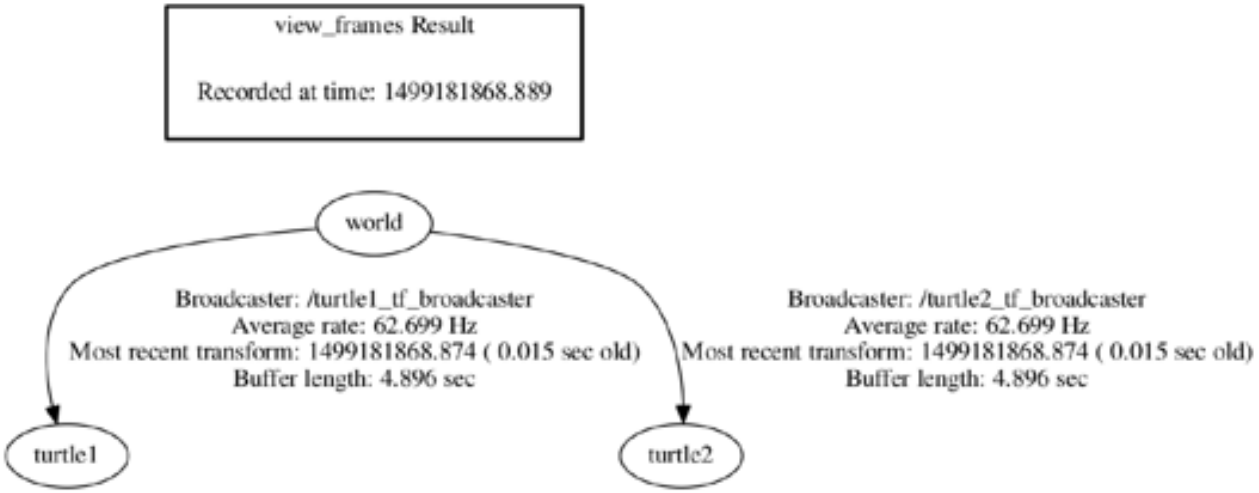
Coordinate transformation(TF, transform)

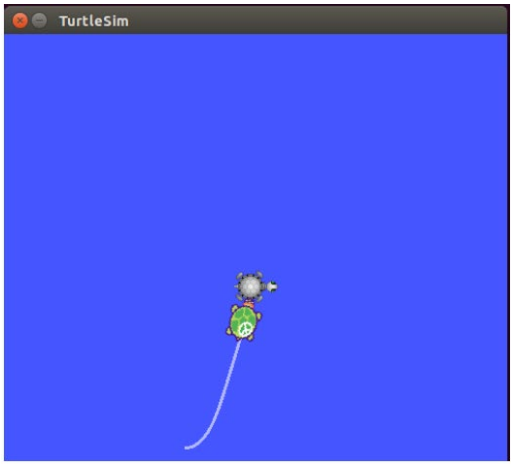
```
ab@ab-c:~$ sudo apt-get install ros-noetic-tutle-tf
```

```
ab@ab-c:~$ sudo apt install python-is-python3
```

```
ab@ab-c:~$ roslaunch turtle_tf turtle_tf_demo.launch
```

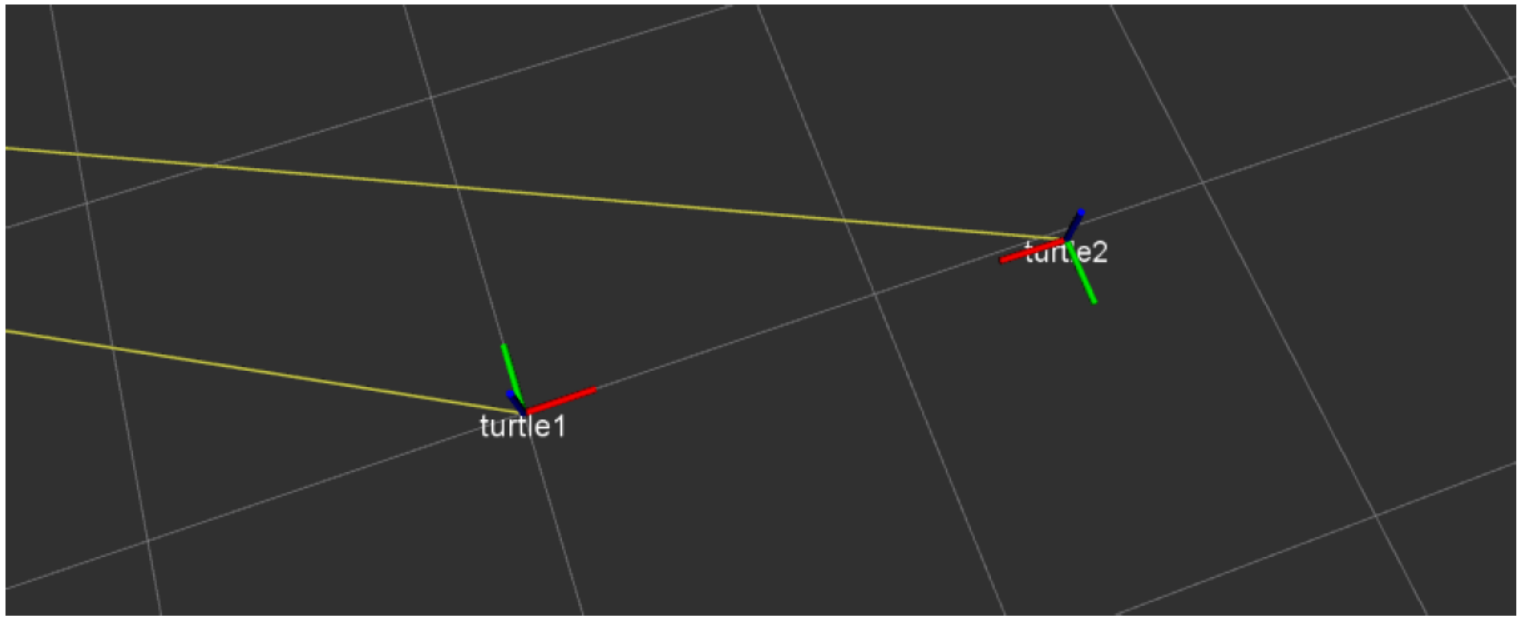
```
ab@ab-c:~$ rosrun turtlesim turtle_teleop_key
```





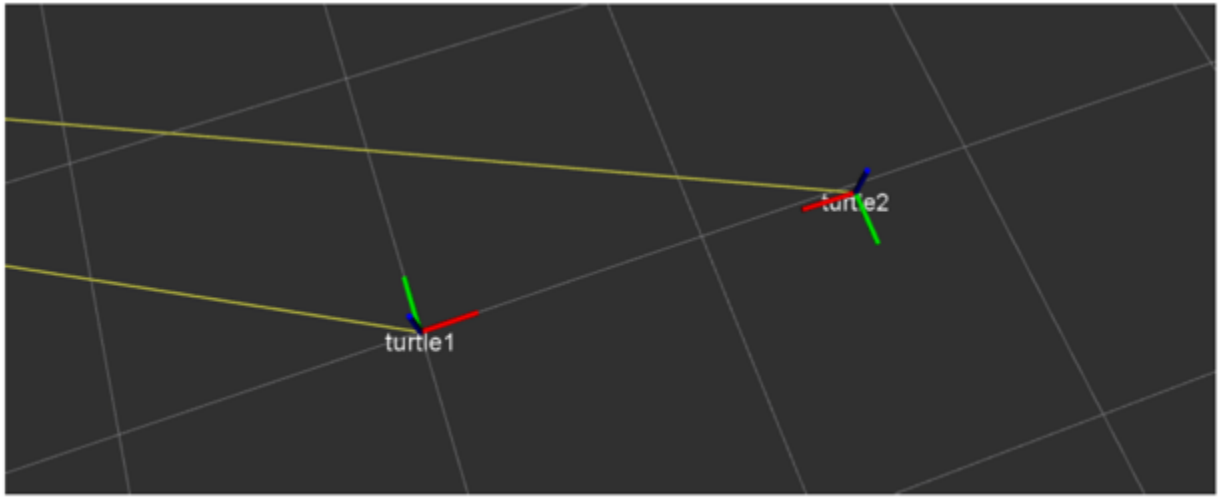
Coordinate transformation(TF, transform)

```
➜  ~ rosrun tf tf_echo turtle1 turtle2
At time 1504942486.329
- Translation: [0.000, 0.000, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.311, 0.950]
            in RPY (radian) [0.000, -0.000, 0.633]
            in RPY (degree) [0.000, -0.000, 36.290]
At time 1504942487.018
- Translation: [0.000, 0.000, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.311, 0.950]
            in RPY (radian) [0.000, -0.000, 0.633]
            in RPY (degree) [0.000, -0.000, 36.290]
```

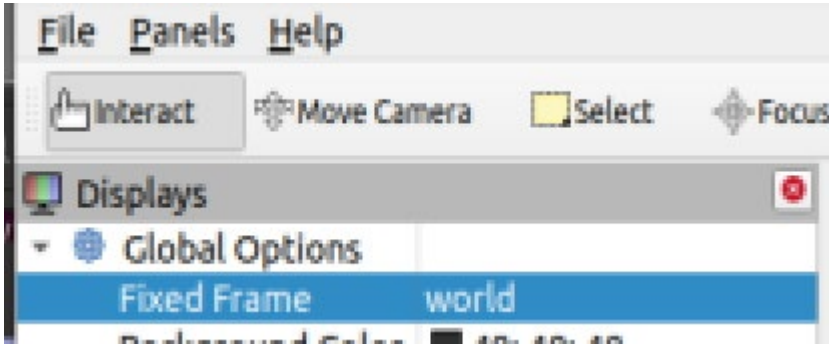


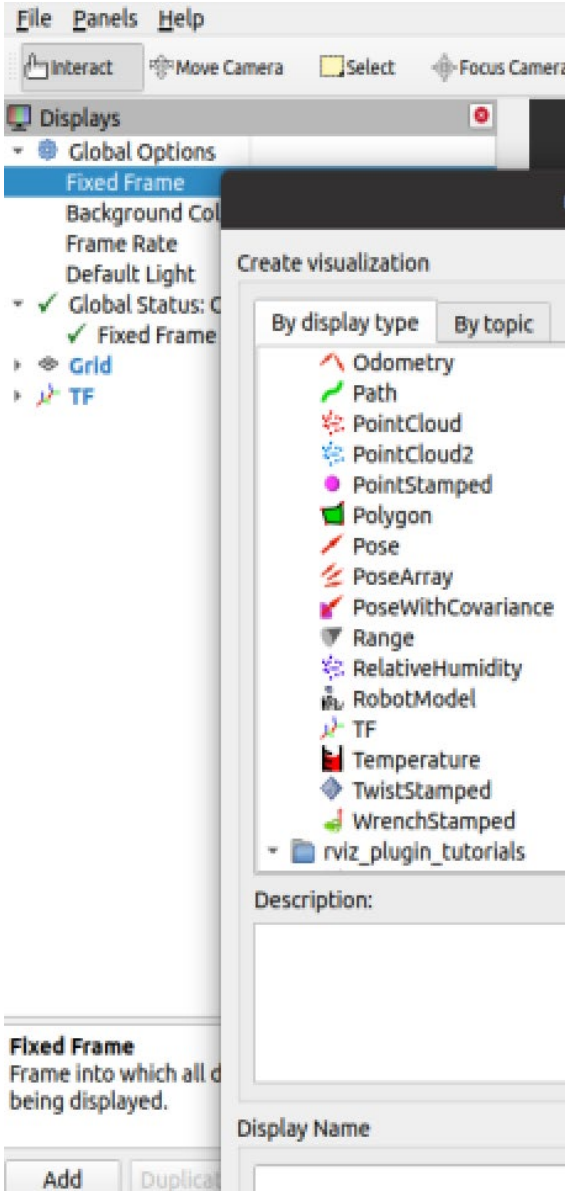
$ rosrun rviz rviz -d `rospack find turtle_tf` /rviz/turtle_rviz.rviz

Coordinate transformation(TF, transform)



$ rosrun rviz rviz -d `rospack find turtle_tf` /rviz/turtle_rviz.rviz





Reference

✓ Download link

✓ Language:

English, Chinese, Japanese, Korean

"ROS Robot Programming"

A Handbook is written by TurtleBot3 Developers

Reference

- R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza. Introduction to Autonomous Mobile Robots. MIT Press, 2nd Edition, 2011, ISBN-10: 0262015358.
- Y. Pyo, H. Cho, R. Jung, and T. Lim, ROS Robot Programming, ROBOTIS Co., Ltd., 2017, ISBN 979-11-962307-1-5
- J. O'Kane, A Gentle Introduction to ROS, CreateSpace Independent Publishing Platform, 2013, ISBN-13: 978-1492143239