Introduction to ROS

ROS.org

ROS = Robot Operating System

- Framework for developing control software architecture for robots.
- Initially developed at Stanford, continued at Willow Garage, now supported and developed by the Open Source Robotics Foundation.
- Supports Linux, Mac OS X, Windows, Raspberry, QNX.
- Currently used in several research and industrial platforms.

Distribution:

- Available for Linux, Mac OS X, Windows, Raspberry, QNX.
  - Ubuntu (Debian-based)
  - Ubuntu LTS releases.
    - Current: 16.04
    - Previous: 14.04

- Current ROS version: Kinetic Kame
- Previous ROS version: Jade (EOL: 2017)

- Check Ubuntu installed version: `lsb_release –a`
- Check ROS installed version: `rosversion -d`

ROS Installation

ROS Installation guide:

- http://wiki.ros.org/ROS/Installation (for latest version)
- http://wiki.ros.org/kinetic/Installation/Ubuntu (for Kinetic)

Repository setup:

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

Keys setup:

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key
0xB01FA116
```

Packages update and Install:

```
sudo apt-get update
sudo apt-get install ros-kinetic-desktop-full
```

Check ROS installation:

```
rosversion -d
roscore
```

ROS Setup

Initialize rosdep:

```
sudo rosdep init
rosdep update
```

Environment setup:

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Getting rosinstall:

```
sudo apt-get install python-rosinstall
```

Check ROS installation:

```
rosversion -d
roscore
roscd
```

Install additional tools (text editor):

```
sudo apt-get install geany
```

ROS catkin workspace

Create catkin workspace:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
```

Compile and complete workspace structure:

```
cd ~/catkin_ws/
catkin_make
ls
roscd
```

Workspace Environment setup:

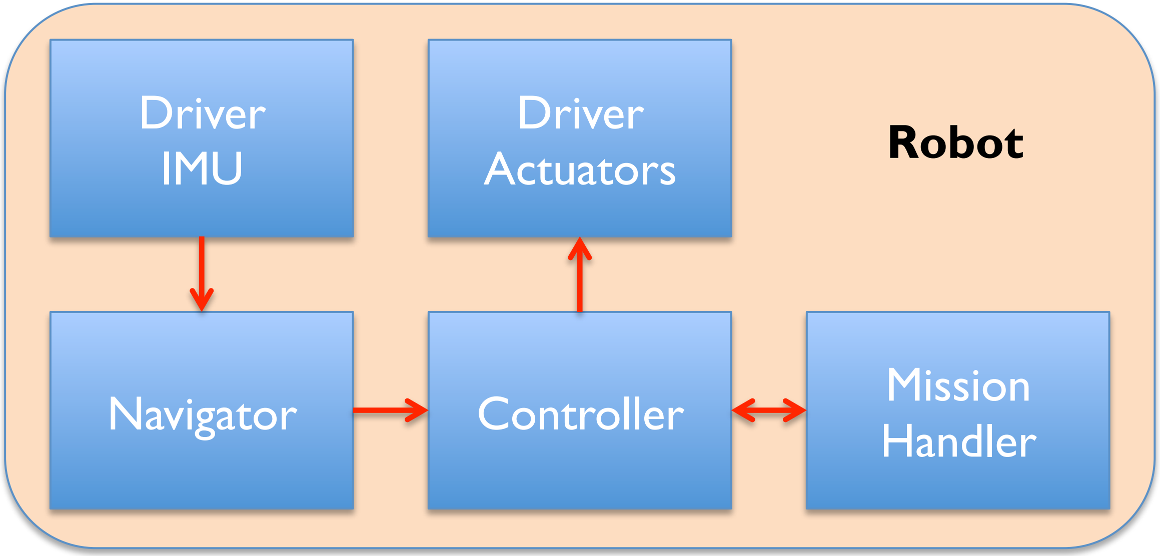
```
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

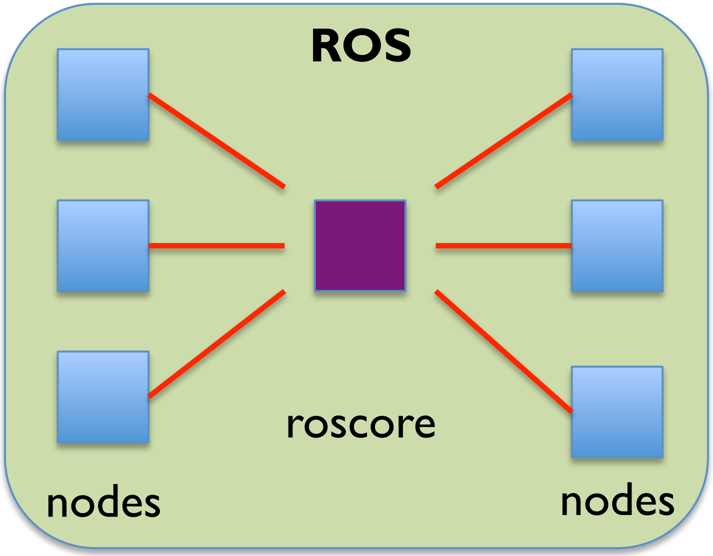
Workspace Environment setup:

```
roscd
cd ~/catkin_ws
ls
```

ROS Basics and Concepts (roscore and nodes)

- Modularization in ROS is achieved by separated operating system processes.
- A *node* is a process that uses ROS framework.
- *Nodes* can be executing from different machines.
- All *nodes* can get/send information from/to other *nodes* via *roscore.*
- *roscore* acts primarily as a name server.
- A *roscore* is always required for *nodes* communication.





ROS Basics and Concepts (roscore and nodes)

Executing roscore

Open a new terminal (`Ctrl+t`)

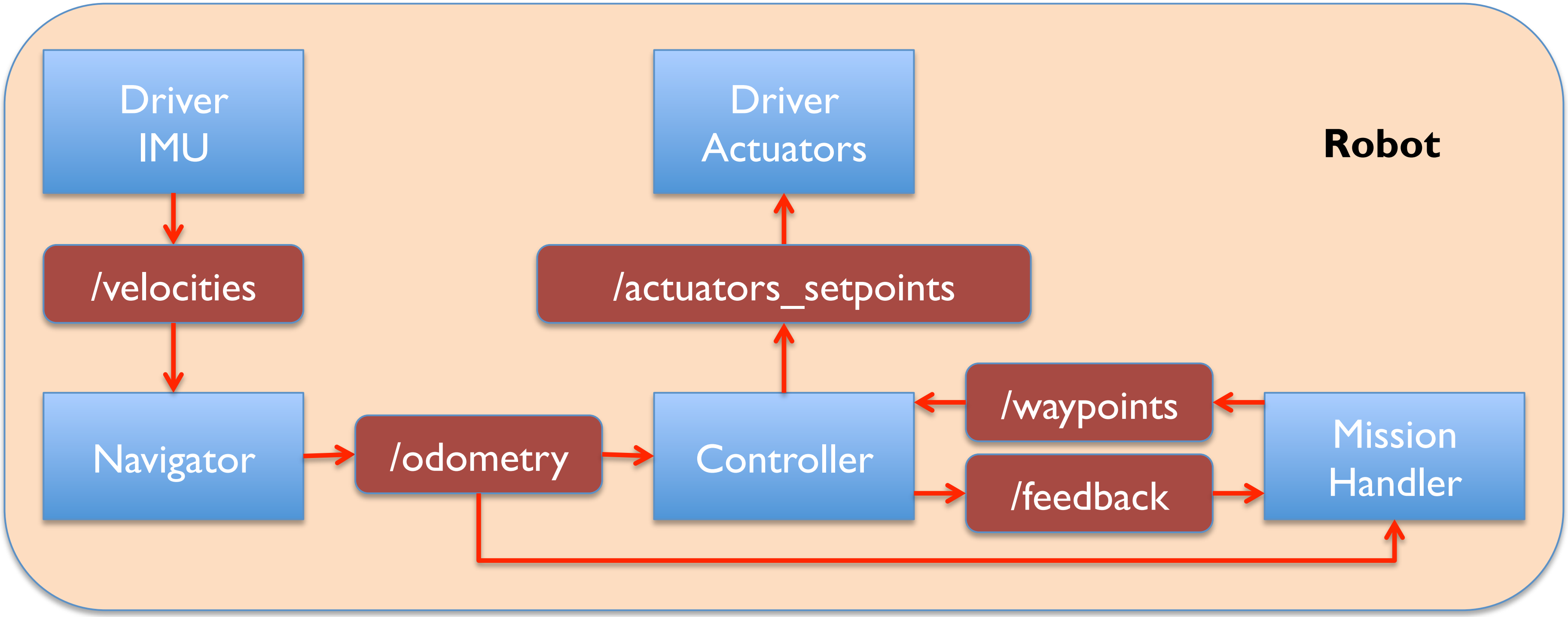
```
roscore
```

Open another terminal (`Ctrl+shift+t`)

```
rosnode list
```

ROS Basics and Concepts (topics)

- *Topic* is a mechanism to send messages from a *node* to other *nodes*.
- *Topics* use a publisher-subscriber approach:
  - *Node(s)* **publish(es)** a message to a *topic* (sending info).
  - *Node(s)* **subscribe(s)** to a *topic* (receiving info).
- Published messages are broadcasted to all subscribers.



ROS Packages

ROS Package:

- Self-contained directory containing source, makefiles, etc.
- There are several ROS packages available: sensor drivers, simulators, controllers, planners, image processing, etc.
- Programming languages:
  - C++ (compiled using catkin).
  - Python.

Create catkin ROS package:

- User packages must be in a catkin workspace.
- In a terminal type:

```
cd ~/catkin_ws/src
catkin_create_pkg hello_world_pkg std_msgs rospy roscpp
cd ~/catkin_ws/
catkin_make
```

ROS, hello world node (C++)

Create a C++ ROS node (User packages must be in a catkin workspace).

In the same terminal type:

```
roscd hello_world_pkg
cd ./src
geany hello_world_node.cpp
```

```
#include <iostream>
#include <string>
#include <csignal>
//ROS
#include <ros/ros.h>

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

    ros::Rate loop_rate(1);
    while(ros::ok()) {
        ROS_INFO("%s: hello world (C++)", ros::this_node::getName().c_str());
        loop_rate.sleep();
    }
    return (0);
}
```

ROS, hello world node (C++)

Modify CMakeLists.txt:

```
roscd hello_world_pkg
geany CMakeLists.txt
```

```
…
## Your package locations should be listed before other locations
# include_directories(include)
include_directories(
  ${catkin_INCLUDE_DIRS}
)
…
## Declare a C++ executable
add_executable(hello_world_node src/hello_world_node.cpp)
…
## Specify libraries to link a library or executable target against
target_link_libraries(hello_world_node
  ${catkin_LIBRARIES}
)
```

Compile a C++ ROS node (compilation is done from the workspace root):

```
cd ~/catkin_ws/
catkin_make
```

ROS, hello world node (C++)

Execute a C++ ROS node (always execute one roscore):

In one terminal execute:

```
roscore
```

In another terminal (`Ctrl+shift+t`) execute:

```
rosrun hello_world_pkg hello_world_node
```

In a third terminal (`Ctrl+shift+t`) execute:

```
rosnode list
```

ROS, hello world node (Python)

Create a Python ROS node (User packages must be in a catkin workspace).
In the same terminal type:

```
roscd hello_world_pkg
cd ./src
geany hello_world_node.py
```

```
#!/usr/bin/env python

# ROS imports
import roslib; roslib.load_manifest('hello_world_pkg')
import rospy

if __name__ == '__main__':
    rospy.init_node('hello_world_node_py', log_level=rospy.INFO)

    loop_rate = rospy.Rate(1) # 1hz
    while not rospy.is_shutdown():
        rospy.loginfo("%s: hello world (Python)", rospy.get_name())
        loop_rate.sleep()
```

ROS, hello world node (Python)

Execute a Python ROS node (always execute one roscore):

Make the Python node executable:

```
roscd hello_world_pkg/src
chmod +x hello_world_node.py
```

In one terminal execute:

```
roscore
```

In another terminal execute:

```
rosrun hello_world_pkg hello_world_node.py
```

In a third terminal execute:

```
rosnode list
```

ROS, hello world nodes (C++ and Python)

Execute Python and C++ ROS nodes (always execute one roscore):

In one terminal execute:

```
roscore
```

In another terminal execute:

```
rosrun hello_world_pkg hello_world_node
```

In another terminal execute:

```
rosrun hello_world_pkg hello_world_node.py
```

In another terminal execute:

```
rosnode list
```

ROS, nodes and topics example (C++)

Create a new catkin ROS package:

```
cd ~/catkin_ws/src
catkin_create_pkg robot_example_pkg std_msgs geometry_msgs rospy roscpp
roscd robot_example_pkg/src

geany Navigator.h
...
geany Navigator.cpp
...
geany Controller.h
...
geany Controller.cpp
```

ROS, nodes and topics example (C++)

```
//Navigator.h
#ifndef NAVIGATOR_H_
#define NAVIGATOR_H_

#include <iostream>
#include <string>
//ROS
#include <ros/ros.h>
#include <geometry_msgs/Point.h>

class Navigator {
private:
   ros::NodeHandle nh_;
   ros::Publisher pub_;

public:
   Navigator();
   void start();
};

#endif /*NAVIGATOR_H_*/
```

ROS, nodes and topics example (C++)

```cpp
//Navigator.cpp
#include "Navigator.h"

Navigator::Navigator() {
    pub_ = nh_.advertise<geometry_msgs::Point>("/odometry", 1);
}

void Navigator::start() {
    //calculate position
    geometry_msgs::Point pos_msg;
    pos_msg.x = 1.0; pos_msg.y = 2.0; pos_msg.z = 5.0;//example
    ros::Rate loop_rate(1);
    while(ros::ok()) {
        pub_.publish(pos_msg);
        loop_rate.sleep();
    }
}

int main(int argc, char **argv) {
    ros::init(argc, argv, "navigator_cpp");
    ROS_INFO("Starting navigator node");
    Navigator navigator;
    navigator.start();
    return (0);
}
```

ROS, nodes and topics example (C++)

```cpp
//Controller.h
#ifndef CONTROLLER_H_
#define CONTROLLER_H_

#include <iostream>
#include <string>
//ROS
#include <ros/ros.h>
#include <geometry_msgs/Point.h>
#include <geometry_msgs/Vector3.h>

class Controller {
private:
   ros::NodeHandle nh_;
   ros::Publisher pub_;
   ros::Subscriber sub_;

public:
   Controller();
   void start();
   void odoCallback(geometry_msgs::Point pos_msg);

   double pos_x_, pos_y_, pos_z_, k_;
};

#endif /*CONTROLLER_H_*/
```

ROS, nodes and topics example (C++)

```cpp
//Controller.cpp
#include "Controller.h"

Controller::Controller() {
	pub_ = nh_.advertise<geometry_msgs::Vector3>("/act_setpoint", 1);
	sub_ = nh_.subscribe("/odometry", 1, &Controller::odoCallback, this);
	k_ = 3.0;//controller constant, example
}

void Controller::start() {
	//calculate control action
	geometry_msgs::Vector3 force_msg;
	ros::Rate loop_rate(1);
	while(ros::ok()) {
		force_msg.x = k_*pos_x_; force_msg.y = k_*pos_y_; force_msg.z = k_*pos_z_;//example
		pub_.publish(force_msg);
		ros::spinOnce();
		loop_rate.sleep();
	}
}

void Controller::odoCallback(geometry_msgs::Point pos_msg) {
	pos_x_ = pos_msg.x;
	pos_y_ = pos_msg.y;
	pos_z_ = pos_msg.z;
}

int main(int argc, char **argv) {
	ros::init(argc, argv, "controller_cpp");
	ROS_INFO("Starting controller node");
	Controller controller;
	controller.start();
	return (0);
}
```

ROS, nodes and topics example (C++)

Modify CMakeLists.txt:

```
roscd robot_example_pkg
geany CMakeLists.txt
```

```
…
## Your package locations should be listed before other locations
# include_directories(include)
include_directories(
  ${catkin_INCLUDE_DIRS}
)
…
## Declare a C++ executable
add_executable(navigator src/Navigator.cpp)
add_dependencies(navigator robot_example_pkg_generate_messages_cpp)

add_executable(controller src/Controller.cpp)
add_dependencies(controller robot_example_pkg_generate_messages_cpp)
…
## Specify libraries to link a library or executable target against
target_link_libraries(navigator
  ${catkin_LIBRARIES}
)
target_link_libraries(controller
  ${catkin_LIBRARIES}
)
```

Compile a C++ ROS node (compilation is done from the workspace root):

```
cd ~/catkin_ws/
catkin_make
```

ROS, nodes and topics example (C++)

Execute C++ ROS nodes (always execute one roscore):

In one terminal execute:

```
roscore
```

In another terminal execute:

```
rosrun robot_example_pkg navigator
```

In another terminal execute:

```
rosrun robot_example_pkg controller
```

In another terminal execute:

```
rosnode list
rostopic echo /odometry
rostopic echo /act_setpoint
```

ROS, nodes and topics example (Python)

Now in Python:

```
roscd robot_example_pkg/src

geany Navigator.py
...
geany Controller.py
...
```

ROS, nodes and topics example (Python)

```python
#!/usr/bin/env python
# Navigator.py
# ROS imports
import roslib; roslib.load_manifest('robot_example_pkg')
import rospy
from geometry_msgs.msg import Point

class Navigator(object):
    def __init__(self):
        self.pub_ = rospy.Publisher("/odometry", Point, queue_size = 1)
        return

    def start(self):
        #calculate position
        pos_msg = Point(1.0, 2.0, 5.0)
        loop_rate = rospy.Rate(1) # 1hz
        while not rospy.is_shutdown():
            self.pub_.publish(pos_msg);
            loop_rate.sleep()
        return

if __name__ == '__main__':
    rospy.init_node('navigator_py', log_level=rospy.INFO)
    rospy.loginfo("%s: starting navigator node", rospy.get_name())
    navigator = Navigator()
    navigator.start()
```

ROS, nodes and topics example (Python)

```python
#!/usr/bin/env python
# Controller.py
# ROS imports
import roslib; roslib.load_manifest('robot_example_pkg')
import rospy
from geometry_msgs.msg import Point, Vector3

class Controller(object):
    def __init__(self):
       self.pub_ = rospy.Publisher("/act_setpoint", Vector3, queue_size = 1)
       self.sub_ = rospy.Subscriber("/odometry", Point, self.odoCallback, queue_size = 1)
       self.pos_x_ = 0.0; self.pos_y_ = 0.0; self.pos_z_ = 0.0
       self.k_ = 3.0 #controller constant, example
       return

    def start(self): #calculate control action
       force_msg = Vector3()
       loop_rate = rospy.Rate(1) # 1hz
       while not rospy.is_shutdown():
          force_msg.x = self.k_*self.pos_x_; force_msg.y = self.k_*self.pos_y_;
          force_msg.z = self.k_*self.pos_z_;#example
          self.pub_.publish(force_msg)
          loop_rate.sleep()
       return

    def odoCallback(self, pos_msg):
       self.pos_x_ = pos_msg.x; self.pos_y_ = pos_msg.y; self.pos_z_ = pos_msg.z
       return

if __name__ == '__main__':
    rospy.init_node('controller_py', log_level=rospy.INFO)
    rospy.loginfo("%s: starting controller node", rospy.get_name())
    controller = Controller()
    controller.start()
```

ROS, nodes and topics example (Python)

Execute Python ROS nodes (always execute one roscore):

Make the Python nodes executable:

```
roscd robot_example_pkg/src
chmod +x Navigator.py
chmod +x Controller.py
```

In one terminal execute:

```
roscore
```

In another terminal execute:

```
rosrun robot_example_pkg Navigator.py
```

In another terminal execute:

```
rosrun robot_example_pkg Controller.py
```

In another terminal execute:

```
rosnode list
rostopic echo /odometry
rostopic echo /act_setpoint
```

ROS, nodes and topics example (C++ and Python)

Execute C++ and Python ROS nodes:

In one terminal execute:

```
roscore
```

In another terminal execute:

```
rosrun robot_example_pkg navigator
```

In another terminal execute:

```
rosrun robot_example_pkg Controller.py
```

In another terminal execute:

```
rosnode list
rostopic echo /odometry
rostopic echo /act_setpoint
```

ROS, Defining Custom Messages

Create the messages folder:

```
roscd robot_example_pkg
mkdir -p msg
```

Define a custom message:

```
roscd robot_example_pkg/msg
geany RobotStatus.msg
```

```
## Robot status message
geometry_msgs/Point position
geometry_msgs/Vector3 force
```

Modify CMakeLists.txt:

```
roscd robot_example_pkg
geany CMakeLists.txt
```

```
…
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  geometry_msgs
  roscpp
  rospy
  std_msgs
  message_generation
)
…
## Generate messages in the 'msg' folder
add_message_files(
  FILES
  RobotStatus.msg
)
…
## Generate added messages and services with any dependencies listed here
generate_messages(
  DEPENDENCIES
  geometry_msgs
)
```

Compile messages:

```
cd ~/catkin_ws/
catkin_make
```

ROS, Using Custom Messages (C++)

Edit the controller node (C++):

```
cd ~/catkin_ws/src

geany Controller.h
...
geany Controller.cpp
```

ROS, Using Custom Messages (C++)

```cpp
//Controller.h
#ifndef CONTROLLER_H_
#define CONTROLLER_H_

#include <iostream>
#include <string>
//ROS
#include <ros/ros.h>
#include <geometry_msgs/Point.h>
#include <geometry_msgs/Vector3.h>
#include <robot_example_pkg/RobotStatus.h>

class Controller {
private:
   ros::NodeHandle nh_;
   ros::Publisher pub_, pub_status_;
   ros::Subscriber sub_;

public:
   Controller();
   void start();
   void odoCallback(geometry_msgs::Point pos_msg);

   double pos_x_, pos_y_, pos_z_, k_;
};

#endif /*CONTROLLER_H_*/
```

ROS, Using Custom Messages (C++)

```
//Controller.cpp
#include "Controller.h"

Controller::Controller() {
    pub_ = nh_.advertise<geometry_msgs::Vector3>("/act_setpoint", 1);
    sub_ = nh_.subscribe("/odometry", 1, &Controller::odoCallback, this);
    pub_status_ = nh_.advertise<robot_example_pkg::RobotStatus>("/robot_status", 1);
    k_ = 3.0;//controller constant, example
}

void Controller::start() {
    //calculate control action
    geometry_msgs::Vector3 force_msg;
    //status msg
    robot_example_pkg::RobotStatus status_msg;

    ros::Rate loop_rate(1);
    while(ros::ok()) {
        force_msg.x = k_*pos_x_; force_msg.y = k_*pos_y_; force_msg.z = k_*pos_z_;//example
        pub_.publish(force_msg);

        status_msg.position.x = pos_x_; status_msg.position.y = pos_y_; status_msg.position.z = pos_z_;
        status_msg.force = force_msg;
        pub_status_.publish(status_msg);

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

ROS, Using Custom Messages (C++)

Compile changes in C++ ROS nodes:

```
cd ~/catkin_ws/
catkin_make
```

Execute C++ ROS nodes (always execute one roscore):

In one terminal execute:

```
roscore
```

In another terminal execute:

```
rosrun robot_example_pkg navigator
```

In another terminal execute:

```
rosrun robot_example_pkg controller
```

In another terminal execute:

```
rosnode list
rostopic echo /odometry
rostopic echo /act_setpoint
rostopic echo /robot_status
```

ROS, Using Custom Messages (Python)

Edit the controller node (Python):

```
cd ~/catkin_ws/src

geany Controller.py
```

ROS, Using Custom Messages (Python)

```
#!/usr/bin/env python
# Controller.py
# ROS imports
import roslib; roslib.load_manifest('robot_example_pkg')
import rospy
from geometry_msgs.msg import Point, Vector3
from robot_example_pkg.msg import RobotStatus

class Controller(object):
    def __init__(self):
        self.pub_ = rospy.Publisher("/act_setpoint", Vector3, queue_size = 1)
        self.pub_status_ = rospy.Publisher("/robot_status", RobotStatus, queue_size = 1)
        self.sub_ = rospy.Subscriber("/odometry", Point, self.odoCallback, queue_size = 1)
        self.pos_x_ = 0.0; self.pos_y_ = 0.0; self.pos_z_ = 0.0
        self.k_ = 3.0 #controller constant, example
        return

    def start(self): #calculate control action
        force_msg = Vector3()
        status_msg = RobotStatus()
        loop_rate = rospy.Rate(1) # 1hz
        while not rospy.is_shutdown():
            force_msg.x = self.k_*self.pos_x_; force_msg.y = self.k_*self.pos_y_;
            force_msg.z = self.k_*self.pos_z_;#example
            self.pub_.publish(force_msg)

            status_msg.position.x = self.pos_x_;
            status_msg.position.y = self.pos_y_;
            status_msg.position.z = self.pos_z_;
            status_msg.force = force_msg
            self.pub_status_.publish(status_msg)
            loop_rate.sleep()
        return
…
```

ROS, Using Custom Messages (C++ and Python)

Execute C++ and Python ROS nodes:

In one terminal execute:

```
roscore
```

In another terminal execute:

```
rosrun robot_example_pkg navigator
```

In another terminal execute:

```
rosrun robot_example_pkg Controller.py
```

In another terminal execute:

```
rosnode list
rostopic echo /odometry
rostopic echo /act_setpoint
rostopic echo /robot_status
```

ROS, Defining Services

Create the messages folder:

```
roscd robot_example_pkg
mkdir -p srv
```

Define a custom message:

```
roscd robot_example_pkg/srv
geany ChangeConstControl.srv
```

```
## Service
float32 new_k
---
float32 previous_k
```

Modify CMakeLists.txt:

```
roscd robot_example_pkg
geany CMakeLists.txt
```

```
…
## Generate services in the 'srv' folder
add_service_files(
  FILES
  ChangeConstControl.srv
)
…
```

Compile messages:

```
cd ~/catkin_ws/
catkin_make
```

ROS, Implementing Services (C++)

```
//Controller.h
#ifndef CONTROLLER_H_
#define CONTROLLER_H_

#include <iostream>
#include <string>
//ROS
#include <ros/ros.h>
#include <geometry_msgs/Point.h>
#include <geometry_msgs/Vector3.h>
#include <robot_example_pkg/RobotStatus.h>
#include <robot_example_pkg/ChangeConstControl.h>

class Controller {
private:
   ros::NodeHandle nh_;
   ros::Publisher pub_, pub_status_;
   ros::Subscriber sub_;
   ros::ServiceServer service_;

public:
   Controller();
   void start();
   void odoCallback(geometry_msgs::Point pos_msg);
   bool changeContConst(robot_example_pkg::ChangeConstControl::Request  &req,
                        robot_example_pkg::ChangeConstControl::Response &res);

   double pos_x_, pos_y_, pos_z_, k_;
};

#endif /*CONTROLLER_H_*/
```

ROS, Implementing Services (C++)

```
//Controller.cpp
#include "Controller.h"

Controller::Controller() {
      pub_ = nh_.advertise<geometry_msgs::Vector3>("/act_setpoint", 1);
      sub_ = nh_.subscribe("/odometry", 1, &Controller::odoCallback, this);
      pub_status_ = nh_.advertise<robot_example_pkg::RobotStatus>("/robot_status", 1);
      service_ = nh_.advertiseService("/controller/change_control_const",
                                      &Controller::changeContConst, this);
      k_ = 3.0;//controller constant, example
}
…
bool Controller::changeContConst(robot_example_pkg::ChangeConstControl::Request  &req,
                                 robot_example_pkg::ChangeConstControl::Response &res){
      res.previous_k = k_;
      k_ = req.new_k;
}
…
```

ROS, Using Services (C++)

Compile changes in C++ ROS nodes:

```
cd ~/catkin_ws/
catkin_make
```

Execute C++ ROS nodes (always execute one roscore):

```
roscore
```

In another terminal execute:

```
rosrun robot_example_pkg navigator
```

In another terminal execute:

```
rosrun robot_example_pkg Controller.py
```

In another terminal execute:

```
rosnode list
rostopic echo /odometry
rostopic echo /act_setpoint
rostopic echo /robot_status
```

In another terminal execute:

```
rosservice call /controller/change_control_const "new_k: 5.0"
```

ROS, Implementing Services (Python)

Edit the controller node (Python):

```
cd ~/catkin_ws/src

geany Controller.py
```

ROS, Implementing Services (Python)

```
#!/usr/bin/env python
# Controller.py
# ROS imports
import roslib; roslib.load_manifest('robot_example_pkg')
import rospy
from geometry_msgs.msg import Point, Vector3
from robot_example_pkg.msg import RobotStatus

class Controller(object):
    def __init__(self):
       self.pub_ = rospy.Publisher("/act_setpoint", Vector3, queue_size = 1)
       self.pub_status_ = rospy.Publisher("/robot_status", RobotStatus, queue_size = 1)
       self.sub_ = rospy.Subscriber("/odometry", Point, self.odoCallback, queue_size = 1)
       self.serv_ = rospy.Service('/controller/change_control_const_py',
                                  ChangeConstControl,
                                  self.changeContConst)
       self.pos_x_ = 0.0; self.pos_y_ = 0.0; self.pos_z_ = 0.0
       self.k_ = 3.0 #controller constant, example
       return
…
    def changeContConst(self, req):
       previou_k = self.k_
       self.k_ = req.new_k
       return ChangeConstControlResponse(previou_k)
…
```

ROS, Implementing Services (C++ and Python)

Execute C++ and Python ROS nodes:

In one terminal execute:

```
roscore
```

In another terminal execute:

```
rosrun robot_example_pkg navigator
```

In another terminal execute:

```
rosrun robot_example_pkg Controller.py
```

In another terminal execute:

```
rosnode list
rostopic echo /odometry
rostopic echo /act_setpoint
rostopic echo /robot_status
```

In another terminal execute:

```
rosservice call /controller/change_control_const_py "new_k: 5.0"
```