Introduction to ROS: Basics, Motion, and Vision

INNOPOLIS UNIVERSITY

ROS Debugging

1. In general, we can use **gdb** for debugging. Also, memory leaks can be checked with **valgrind**. To see the graphical graph representation and performance analysis, **callgrind** and **KCachegrind**.
2. You can not use **gdb rosrun ros_debugging gdb_hello_world** for debugging the code. However, there are two ways you can debug your code

to debug with gdb

```
cd devel/lib/ros_debugging
gdb gdb_hello_world
> (gdb) r
```

However, this way we can not load parameters and other config files.

ROS Debugging

1. Second option is to define the gdb options under launch-prefix the launch file

to debug with gdb

```
<launch>
<node pkg="ros_debugging" type="gdb_hello_world" name="gdb_hello_world"
 output="screen" launch-prefix="xterm -e gdb –args"/>
</launch>
```

Depending on the situation, you can use different keys to control your program execution flow. r for running the program, then bt for backtracing if you got into some problems.

ROS Debugging: Memory Handling

1. If there is problems with memory leaks or with performance, valgraid can be used

to debug with valgrind

```
<launch>
<node pkg="ros_debugging" type="rqt_logging" name="rqt_logging"
 output="screen" launch-prefix="valgrind"/> </launch>
```

ROS Debugging: Callgrind + Kcachegrind

1. Callgrind is a profiling tool that records the call history among functions in UNIX process

to debug with valgrind

```
<node pkg="ros_debugging" type="rqt_logging" name="rqt_logging"
 output="screen" launch-prefix="valgrind –tool=callgrind
–callgrind-out-file='callgrind.example1.%p'"/> </launch>
```

2. The profiling log is saved in the /.ros directory
3. kcachegrind /.ros/callgrind.example1.xxxx for visualizing the profile

ROS Logging

1. Logging usually performance-wise expensive
2. However, log4cxx, which a port of log4j logger library, has a null footprint on performance
3. Logging has several LEVELs: DEBUG, INFO, WARN, ERROR, and FATAL
4. Different between ERROR and FATAL is that if there is an error, but program can still run, logging should be defined as ERROR otherwise FATAL

to import logging functionality

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

ROS Logging

1. How can we do logging?

to define logging

ROS_<LEVEL>[_<OTHER>], e.g., ROS_INFO("INFO message");
ROS_<LEVEL>[_STREAM]_NAMED, e.g.,
ROS_INFO_STREAM_NAMED("class_name", "INFO stream message; val = " « val);
ROS_<LEVEL>[_STREAM]_COND[_NAMED], e.g.,
ROS_INFO_STREAM_COND(val < 0., "INFO stream message; val (" « val « ") < 0");
ROS_INFO_STREAM_ONCE("INFO stream message");

ROS Logging

to manipulate with logging messages

rosrun rqt_console rqt_console