CS 378 - Autonomous Vehicles in Traffic I
Week 11a - ROS parameters and Dynamic Reconfigure

Announcements

- Project Track - Literary review due this Wednesday
  ○ Send an email to cs378-spr12-submit with your review
  ○ The review should be in PDF format

- Assignment Track - Assignment 5 due Wednesday
  ○ Usual instructions apply

- Scheduled PRC Visit next Monday 2/9 (No class) - We'll be collecting data required for most projects. Please let me know what you'll need.

- This is also the last week of formal lectures!

roscore Revisted

- What does the roscore command do?
  ○ Start a ROS Master for matchmaking
  ○ Start's a rosnout node - collects logs from all ros nodes.
    ▪ All ROS_INFO etc. macros are recorded using rosnout
  ○ Starts a Parameter Server

- The Parameter Server is a central dictionary that can be used to look up variables
  ○ A single parameter server is shared by all the nodes; and data is transmitted over the network
  ○ Mainly used for configuration data, but not real-time transfer of information

rosparam (and rosservice)

- These are the last 2 commands in the ROS Cheatsheet that we have not seen.
- Let's take a look at rosservice with our example from last week
- We'll see how to use the parameters set by rosparam in today's slides
- More detail on these commands is available in the relevant tutorial here
ROS naming convention

- You can read up more on the ROS naming convention at the [wiki page](#).
- The general idea is that you can group nodes appropriately together in the same namespace. For instance:
  - `/marvin/node1`
  - `/marvin/node2`
  - What is the advantage of such a naming convention?
- There are 3 types of names in ROS
  - `relative/name` -> `/marvin/relative/name`
  - `/global/name` -> `/global/name`
  - `~private` -> `/marvin/node1/private`
- All the above examples assume resolution by the node `/marvin/node1`

Alright, back to using parameters

- Reading these parameters in code
  - `ros::param::get("/global_name", global_name)`
  - `ros::param::get("/relative_name", relative_name)`
  - `ros::param::get("~private_name", param);`
  - `ros::param::param<std::string>("default_param", default_param, "default_value");`
- Setting these parameters in code
  - `ros::param::set("/global_param", 5);`
  - `ros::param::set("relative_param", "my_string");`
- Checking parameter existence
  - `ros::param::has("my_param")`
- Deleting parameters
  - `ros::param::del("my_param");`

Setting parameters at command line

- Some of you may have used this command call while testing the opencv example package for programming assignment 3
  - `rosrun intro_to_opencv opencv_example --method`  
  - `flood_image_transport:=compressed usb_cam/image_raw:=center_front/camera/image_raw`
  - In the above example, the third line is a topic remapping.
  - The second line is a private parameter value being provided
- You can only set private parameters this way
- You can read more about command line remapping at [http://www.ros.org/wiki/Remapping%20Arguments](http://www.ros.org/wiki/Remapping%20Arguments)

Parameters in Launch File

- Example usage
  - `<param name="publish_frequency" type="double" value="10.0"/>
- You can put this inside a node tag to set a private parameter
  - `<node pkg="intro_to_opencv" type="opencv_example" name="vis">
  - `<param name="image_transport" value="compressed"></param>`
  - `</node>`
Changing parameters at runtime?

- In a number of situations, it is beneficial to be able to change a parameter value **while** your code is running.
- How did you guys select parameters in Programming Assignment 4?
- The ability to change OpenCV parameters at run-time can greatly increase the speed of finding the best parameters for the vision problem.
- You could have your code poll the parameter server periodically using **cached parameters**, but this increases the load on the ROS Master.

Enter **dynamic_reconfigure**

- Dynamic reconfigure allows for dynamic reconfiguration of a node's parameters.
  - [http://www.ros.org/wiki/dynamic_reconfigure](http://www.ros.org/wiki/dynamic_reconfigure)
- A good set of tutorials are provided:
- You need to take a few steps to use dynamic_reconfigure:
  - Create a **configuration file** defining the parameters that you want to reconfigure.
  - Change **CMakeLists** to generate the necessary classes using the configuration file.
  - Add a dependency to dynamic_reconfigure in the **manifest** file.
- Change your code to run a reconfigure server.

Enter **dynamic_reconfigure**

- For the next part of the slides, we'll refer to the the first two dynamic reconfigure tutorials.
  - [How to Write Your First .cfg File](#)
  - [Setting up Dynamic Reconfigure for a Node(cpp)](#)
  - I have checked in these tutorials in the **art_examples** stack under the package **reconfigure**
- In class we'll only go over the C++ code. A Python API for dynamic reconfigure is also available.

Tutorials.cfg

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

PACKAGE = "reconfigure"

import rosrerlib:roslib.load_manifest(PACKAGE)

from dynamic_reconfigure.parameter_generator import *

gen = ParameterGenerator()

gen.add(\"int_param\", int_t, 0, \"An Integer parameter\", 50, 0, 100)
gen.add(\"double_param\", double_t, 0, \"A double parameter\", .5, 0, 1)
gen.add(\"str_param\", str_t, 0, \"A string parameter\", \"Hello World\")
gen.add(\"bool_param\", bool_t, 0, \"A Boolean parameter\", True)

size_enum = gen.enum(\"Small\", int_t, 0, \"A small constant\"),
    gen.const(\"Medium\", int_t, 1, \"A medium constant\"),
    gen.const(\"Large\", int_t, 2, \"A large constant\"),
    gen.const(\"ExtraLarge\", int_t, 3, \"An extra large constant\",
        \"An enum to set size\")

size_enum = gen.enum(\"Small\", int_t, 0, \"A small parameter which is edited via an enum\",
    1, 0, 3, edit_method=\"size_enum\")
exit(gen.generate(PACKAGE, \"reconfigure\", \"Tutorials\"))
```

10

11

12
Arguments in add function

- name
- type
- level
- description
- default
- min
- max

Most of these parameters are self explanatory. level usage is totally up to you, and you can use it to decide how to handle the reconfiguration request.

Using the config file

- To generate the necessary data structures from the config file to use in your code, you need to do the following:
  - Add dynamic_reconfigure as a dependency in your manifest
  - Add the following lines to your CMakeLists.txt file
    - #add dynamic reconfigure api
    - rosbudl_find_ros_package(dynamic_reconfigure)
    - include(S{dynamic_reconfigure_PACKAGE_PATH}/cmake/cfgbuild.cmake)
    - genctg()
  - The config script needs to be executed, give it the necessary permissions
    - chmod a+x cfg/Tutorials.cfg
  - Run make

server.cpp

```cpp
#include <ros/ros.h>
#include <dynamic_reconfigure/server.h>
#include <reconfigure/TutorialsConfig.h>

void callback(dynamic_reconfigure::TutorialsConfig &config, uint32_t level) {
    ROS_INFO("Reconfigure Request: %d %s %d");
    config.int_param,
    config.double_param,
    config.str_param.c_str(),
    config.boolParam?"True":"False",
    config.size);
}

int main(int argc, char **argv) {
    ros::init(argc, argv, "reconfigure");
    dynamic_reconfigure::Server<dynamic_reconfigure::TutorialsConfig> server;
    dynamic_reconfigure::Server<dynamic_reconfigure::TutorialsConfig>::CallbackType f;
    f = boost::bind(&callback, _1, _2);
    server.setCallback(f);
    ROS_INFO("Spinning node");
    ros::spin();
    return 0;
}
```

Using the configuration in your node

- Once you create the node with the dynamic reconfigure server, you can run it as normal
  - roscore
  - rosrun reconfigure server
- To change the parameters, use the dynamic reconfigure gui:
  - rosrun dynamic_reconfigure reconfigure_gui
- Alternatively, you can use the command line dynparam utility to change these parameters as well
  - rosrun dynamic_reconfigure dynparam COMMAND
Saving/loading parameters

- Even the dynamic reconfigure parameters are stored on the Parameter server

- The *load* and *dump* commands can be extremely useful for saving these parameters to file and loading them back

- You can read more about these commands on the cheatsheet or on the wiki pages.