CS 378 - Autonomous Vehicles in Traffic I
Week 10b - ROS Messages and Services

Today

- We'll take a look at creating custom messages
- We'll take a look at generating service messages
- We'll take a look at writing a simple service server and client

ROS Messages

- Till now, we have only used predefined ROS messages
- It is also possible to create messages as necessary.
- You can also take a look at the tutorial on creating messages and services on the ROS wiki:

ROS Messages

- Messages can contain the following types:
  - int8, int16, int32, int64 (plus uint*)
  - float32, float64
  - string
  - time, duration
  - other msg files
  - variable-length array[] and fixed-length array[C]

- time and duration convert to ros::Time and ros::Duration
- Arrays get converted to std::vector. You can see the reference for std::vector [here](http://).
ROS Messages

- Message files are stored in the msg directory in the package
- The C++ and Python interfaces to these messages are generated automatically
  - To enable generation, open up CMakeLists.txt and uncomment the line:
    - #roscpp_genmsg()
- Generated python files are placed in src/<package-name>/msg inside the package
- Generate cpp files are in msg_gen/cpp/include inside the package. This folder is automatically included when you compile your code, as a result the following line works:
  - #include<std_msgs/Float32.h>

ROS Messages

- If you are generating messages that will be used across multiple packages, it is a good idea to store the messages in a separate package to prevent unnecessary dependencies
- We store our messages in the following packages:
  - Car-related messages are in a package called art_msgs in the art_vehicle stack.
  - Velodyne-related messages are in a package called velodyne_msgs in the velodyne stack.
- Let's see a couple of examples
  - These examples are in the messages package in the art_examples stack.

ROS Messages

Simple.msg

```plaintext
int32 var_int
float32 var_float
string var_string
time some_time
```
Complex.msg

std_msgs/Header header
int16[10] vars_int
messages/Simple[] vars_simple

ROS Services

- Name your variables better!
- Let's take a look at the generated C++ messages
  - roscd messages
  - make
- Lets take a look at some of the messages in art_msgs

ROS Services

- Sometimes you want an external node to provide you with information - similar to a remote procedure call (RPC)
  - for instance "What is 2+2?" or "What is 3+4?"
- We can do this using the existing framework already -
  - You can potentially use 2 messages for this, one for the request and one for the response.
  - The external node could subscribe to your message and send you back a response.
  - Why is this not the best way of doing things?
- ROS Services are designed for this style of communication

ROS Services

- Today we'll see how to define a service, and use that service within the ROS ecosystem.
- For this, we'll also use go through Tutorial 15 of the ROS introductory tutorials.
  - Writing a service client
- This tutorial along with an example service has been checked in the services package in the art_examples stack.
- To enable service generation, open up CMakeLists.txt and uncomment the line:
  - #rosbuild_gensrv()
AddTwoInts.srv

```c
int64 a
int64 b
---
int64 sum
```

add_two_ints_server.cpp

```c
#include <ros/ros.h>
#include <services/AddTwoInts.h>

bool add(services::AddTwoInts::Request &req,
          services::AddTwoInts::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, "add_two_ints_server");
    ros::NodeHandle n;
    ros::ServiceServer service = n.advertiseService("add_two_ints", add);
    ROS_INFO("Ready to add two ints.");
    ros::spin();
    return 0;
}
```

add_two_ints_client.cpp

```c
#include <cstdlib>
#include <ros/ros.h>
#include <services/AddTwoInts.h>

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

Running the code

- Update your copy of art_examples
  - `rosd art_examples`
  - `svn up`

- Build the code
  - `rosd services`
  - `make`

- Run the code
  - `roscore`
  - `rosrun services add_two_ints_server`
  - `rosrun services add_two_ints_client`
Programming Assignment 5

- Programming assignment 5 is now up and due in a week!
- Let's take a look.