Speeding Up ROS Training For New Lab's Members

Even if we would like to, engineering students do not receive proper ROS training during their undergraduate period.

This is a problem when students get engaged in the development of the Msc Thesis inside one of the labs of the University, or want to start their PhD. **The students must dedicate a long time to get up to speed in ROS**, before they can really use the code that is there already.

Typical option for the lab is to provide to the student with a computer, and a link to the ROS Wiki tutorials. Hence the student will pass the days and weeks trying to get the most of it.

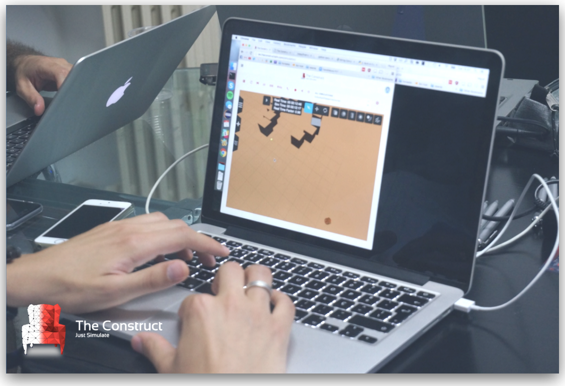
Here **we propose a smooth learning path for your interns in order to maximize their learning speed**. This path is inspired by the book The first 20 hours, and basically consists of four steps:

1. DECONSTRUCT ROS
2. REMOVE STUFF
3. LEARN ROS
4. PRACTICE (A LOT)

1. DECONSTRUCT ROS

Deconstructing ROS means to identify the different parts that compose ROS. This work has to be done by somebody that already knows ROS. We have done that work and identified the following main parts:

1. Installation and setup
2. Basic organization of the development environment with ROS (catkin workspaces, compilers issues, CMakeLists, IDE configuration)
3. Basic subjects: packages, roscore, rosparam server
4. Topics: publishers, subscribers, messages, how to create your own messages
5. Services: clients, servers, service messages, how to create your own service messages
6. Actions: clients, servers, service messages, how to create your own action messages
7. Debugging tools: rviz, rqt_console, rqt_graph, rosbags
8. TF
9. How to make a robot navigate: mapping, localization, path planning and obstacle avoidance
10. How to make a robot perceive: blob detection with OpenCV, object recognition, point cloud usage, people detection and recognition
11. Gazebo simulations
12. URDF robot creation
13. Robot Control
14. How to make a robot manipulate objects: MoveIt! usage, combining perception with manipulation, grasping



The previous points cover a global knowledge of ROS. This does not mean that your students will have to learn it all. Those points just express the most common subjects required for the creation of an autonomous robot (like for example the ones used in the Robocup@home competition).

2. REMOVE STUFF

That is, we eliminate as many things as possible, things that are not really necessary right now. The idea is to get the 80% of the results with the 20% of the effort.

C++ OR PYTHON?

For the moment, we eliminate the necessity to use C++ for ROS. **Using C++ in ROS introduces three problems**:

1. First, the C++ version of ROS includes a lot more of concepts, like the node_handle, the callback_queue or having to deal with the threads of each callback (if you want them in parallel). Python handles all that by itself.
2. Second, by using Python, the student will have to know just the minimum of CMake handling. Making the proper CMakeLists.txt in C++ is a nightmare. Instead, in Python you almost no need to touch the default one.
3. Using C++ for ANYTHING, makes the development a lot harder and by hence slower. And here speed is the key. The faster you can close the loop of doing something and experiencing the result, the faster the brain of the student will make the connection that makes him learn the concept. With C++, compilation problems are of the first order.

I know you must be thinking: but C++ is our development language!! They need to learn in that language.

I understand

However, **starting their development in C++ is a bad idea, because it makes the pill to swallow a lot bigger.** Even if you need the student to use ROS with C+, I recommend you to start with Python (even if the student doesn't know about Python!!!)

Unless the student is a master of C++, the following path:

> Learning Python -> Learning ROS with Python -> Learning ROS with C++

is faster than the path:

> Learning ROS with C++

Even if the student already know C++, the additional amount of knowledge required to get ROS using that language makes the evolution very slow.

INSTALLATION

The student doesn't need to know about how to install. Starting with the installation is a waste of time, since at that point the student knows nothing about ROS and may have trouble about the installation. ROS installation is a stupid step, that has nothing to do with ROS or intelligence or knowledge. It is just an experience, that the student will be able to do faster when he already knows ROS.

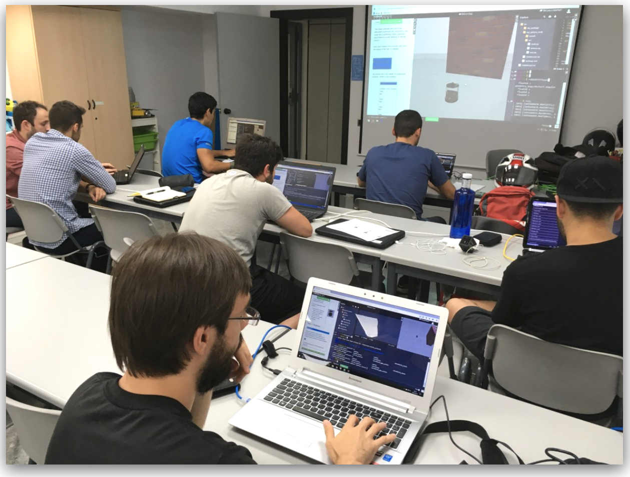
So we suggest you avoid that step to your students.

HOW TO CREATE A CATKIN_WS

Again, this is a concept that is difficult to grasp if you don't know ROS and understand the problems of programming with it. Trying to make the student understand why he has to create a catkin_ws, how to do it and where, is a waste of time. The student will not understand and keep coming to this point once and every time, slowing the progress speed.

OTHER POINTS TO REMOVE

Other points to avoid are, configuring IDEs for programming with ROS (you must provide it done), what is ROS (who cares), how to setup Gazebo (provide it). Basically, you have to remove barriers, physical, emotional and mental, that make the work of learning ROS a lot harder.



So in this point what we propose is that you have a system already set up and working for the student, and that he concentrates on learning ROS for Python.

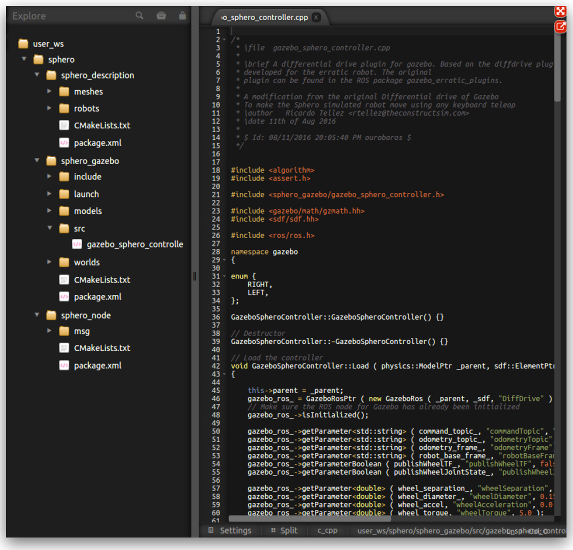
3. LEARN ROS

Here is were the student has to dedicate the time to learn. But **the learning has to be done intelligently,** that means, **following a proper order and with the**

option of self-correcting at each step. The faster the self correction is produced, the faster the learning will be.

We propose you to provide to your students with the following sequence of tutorials of ROS (from wiki.ros.org), which we have found to be optimal for faster learning:

1. Creating a ROS package
2. Understanding rosnodes
3. Understanding topics
4. Writing a publisher and a subscriber
5. Using rqt_console and roslaunch
6. Understanding services and parameters
7. Writing a service and a client
8. Creating ROS msg and srv
9. Writing a simple action client
10. Writing a simple action server
11. Defining custom messages
12. Recording and playing back data
13. Visualization tutorials
14. TF
15. Gazebo
16. URDF
17. ROS control
18. ROS Navigation
19. PCL with ROS
20. MoveIt!



```
/*
 * \file  gazebo_sphero_controller.cpp
 *
 * \brief A differential drive plugin for gazebo. Based on the diffdrive plug
 * developed for the erratic robot. The original
 * plugin can be found in the ROS package gazebo_erratic_plugins.
 *
 * A modification from the original Differential drive of Gazebo
 * To make the Sphero simulated robot move using any keyboard teleop
 * \author   Ricardo Tellez <rtellez@theconstructsim.com>
 * \date 11th of Aug 2016
 *
 * $ Id: 08/11/2016 20:05:40 PM ouroboros $
 */


#include <algorithm>
#include <assert.h>

#include <sphero_gazebo/gazebo_sphero_controller.h>

#include <gazebo/math/gzmath.hh>
#include <sdf/sdf.hh>

#include <ros/ros.h>

namespace gazebo
{

enum {
    RIGHT,
    LEFT,
};

GazeboSpheroController::GazeboSpheroController() {}

// Destructor
GazeboSpheroController::~GazeboSpheroController() {}

// Load the controller
void GazeboSpheroController::Load ( physics::ModelPtr _parent, sdf::ElementPtr
{

    this->parent = _parent;
    gazebo_ros_ = GazeboRosPtr ( new GazeboRos ( _parent, _sdf, "DiffDrive" )
    // Make sure the ROS node for Gazebo has already been initialized
    gazebo_ros_->isInitialized();

    gazebo_ros_->getParameter<std::string> ( command_topic_, "commandTopic",
    gazebo_ros_->getParameter<std::string> ( odometry_topic_, "odometryTopic"
    gazebo_ros_->getParameter<std::string> ( odometry_frame_, "odometryFrame"
    gazebo_ros_->getParameter<std::string> ( robot_base_frame_, "robotBaseFra
    gazebo_ros_->getParameterBoolean ( publishWheelTF_, "publishWheelTF", fal
    gazebo_ros_->getParameterBoolean ( publishWheelJointState_, "publishWheel

    gazebo_ros_->getParameter<double> ( wheel_separation_, "wheelSeparation",
    gazebo_ros_->getParameter<double> ( wheel_diameter_, "wheelDiameter", 0.1
    gazebo_ros_->getParameter<double> ( wheel_accel, "wheelAcceleration", 0.0
    gazebo_ros_->getParameter<double> ( wheel_torque, "wheelTorque", 5.0 );
```

4. PRACTICE (A LOT)

In order to learn fast, the student has to practice. Copy + paste of the code of the wiki does not count as practice. Practice means, providing the student with exercises to be solved (without solution provided). The exercises have to be tied to a simulation, so the student can see the results of his efforts quickly and with a meaning. Providing just a number example, for example a topic publishing a text, is not good enough to get the student engaged. And engagement here is key if we want the student to learn quickly. The student will be a lot more motivated if he can see the result of his efforts in a robot. We recommend to provide simulated robots, since they provide the possibility of testing quickly.

Practice should include an exam, if possible. Students learn the most under the stress situation of an evaluation. Sorry, but it works like that (I did not make the rules).

Super important: in order to apply the method and speed up learning, the student has to be practicing at least 20 hours in a row. Dedicating 20 hour focused in the important points will create a momentum that will increase the speed of learning.

SUMMARY

So how can you have your students up to speed with ROS fast? This is a summary of what we have said above:

1. Provide a fully setup environment. Complete ROS + Gazebo + development IDE installed computer.
2. Provide the student with the optimal list of tutorials to follow (either the one above of your own), but do not just point the student to the ROS wiki.
3. Provide a full list of exercises that the student has to solve, without providing the solution.
4. I know that all that is a lot of work but if you want to invest on that once, you will speed up the process of integration of new students in your lab, and your results will raise.

Another option that you have is to use the services of Robot Ignite Academy. In our academy we provide everything already done for your student, organised in the proposed manner, including development environments, robot simulations, exercises and exams. Everything already working, and requiring only a web browser. No installation required, any computer will work. Give it a try!

ROBOT IGNITE ACADEMY

www.robotigniteacademy.com
Robot Creation with URDF ROS
Learn how to create URDF files by creating three different robots
Details

Robot Creation with URDF

Advancing ROS

Learn how to create the URDF files to control your robot with ROS

www.robotignitacademy.com
ROS Autonomous Vehicles 101
Introduction to Autonomous Vehicles in the ROS ecosystem
Details

ROS Autonomous Vehicles 101

Advancing ROS

Introduction to Autonomous Vehicles in the ROS ecosystem

www.robotigniteacademy.com
ROS-Industrial 101
Introduction to some basic tools for start working with ROS-Industrial
Details

ROS-Industrial 101

Advancing ROS

Introduction of some basic ROS tools to control industrial robots with ROS

Learn what is needed to be able use Open AI-Gym in your next project
OpenAI Gym for Robotics 101 course
www.robotignitacademy.com
Details

OpenAI Gym for Robotics 101

Advancing ROS

Learn what is needed to be able to use OpenAI-Gym in your next project

RGB Navigation with ROS 101
Learn how to use the rtabmap_ros package for performing RGB-D SLAM
www.robotignitacademy.com
Details

RTAB-Map in ROS 101

Advancing ROS

Learn how to use the rtabmap_ros package for performing RGB-D SLAM

www.robotigniteacademy.com
ROS Control 101
Learn how to connect your simulated robot with the ros_control packages.
Details

ROS Control 101

Advancing ROS

Learn how to ROSify the control of your robot

robotignite ACADEMY

Academy of The Construct