Readme for Simulated Agile and Collaborative Robot Applications using ROS

John Michaloski

0/0/0000 0:00:00 AM

This github site contains various ROS repositories that integrate various ROS and non-ROS robotic tools. The code was compile with gnu c++ 0x on a Ubuntu 12.04 with ROS installed and various additional ROS packages installed.

This document describes the Robot Operating System (ROS) pick/place applications for two arm and industrial robot trajectory generation. The genesis of this work was the delving into the existing pick/place functionality in ROS and dealing with the software complexity involved in making the robot perform such tasks. First, a gripper was required to grasp objects. To achieve this in ROS, you need a URDF description of the gripper. Robotiq gripper was chosen since SWRI had developed several gripper models. Although all I wanted was a prismatic open/close gripper, could not find one.

So far, besides using the core ROS tools, the following packages or code have been very useful:  
  
- David Lu arm\_kinematics – for clean integration into KDL  
- Dave Coleman rviz\_visual\_tools –   
- SWRI – host of robot URDF and visualizations  
  
Here is an animated gif of what is supposed to happen when the Fanuc LR Mate robot places gears in a gear tray:  
  
![Figure2](./images/fanuc-2.gif?raw=true)

# Background

The robot applications are coordinated two robots (Fanuc LR Mate 200id and a Motoman sia 20d) activity that visually simulates either arranging “bolts” into a bolt tray or playing checkers. The application is capable of showing collaborative robots (where the two robots work together) or agile robots (where one robot can perform the same functionality as the other with minimal change).

In either case, the fundamental robot functionality is pick and place. The simulated robot control is open loop, with no sensor feedback. At its simplest, motion planning is simple bang-bang control. In the bang-bang control, the robot is commanded to goal joint positions and the simulated robot “moves” there. If a goal position as a Cartesian pose (position and orientation), the goal pose is transformed into joint positions using an inverse kinematics function, and these joints are then used as the goal position. Intermixed with the motion control are robot commands to dwell (to delay), open/close gripper, and object handling. Highlights of the various ROS and non-ROS robotic tools include:

* The robot and gripper definition and visualization use open source ROS URDF definition for the Fanuc LR Mate 200id and a Motoman sia 20d and Robotiq C2 gripper.
* Rviz is a ROS visualization tool that integrates with URDF.
* Inverse kinematics uses existing ikfast ROS software to solve the forward and inverse kinematics. The Fanuc robot had used the Orocos KDL kinematics solver that was part of the ROS arm navigation C++ software of David Lu, but required hints (seeds) in order for the KDL IK solution to converge. The KDL IK solution did not work for the Motoman sia 20d even with hints. However, use of the ikfast routines assumes that the robot is situated at the world origin (0,0,0) so transformations from the base position of each robot to the world origin were done for each commanded Cartesian position.
* rviz\_visual\_tools was used to describe “markers” in the rviz smulation. In other words, if the robot scene contained checkers or a bolt tray, these objects were described using the rviz\_visual\_tools package and linked library. The rviz\_visual\_tools ROS communication was found to be problematic, so extra ROS “spins” and other programming hacks were used to guarantee communication to the maker visualization front-end and rviz.

# Robot and Gripper Description

Once the robot and gripper URDF is in place, definitions can become a macro with Xacro (ok programming with XML bad eye test)

<?xml version="1.0"?>

<robot name="fanuc\_lrmate200id" xmlns:xacro="http://ros.org/wiki/xacro">

<xacro:include filename="$(find fanuc\_lrmate200id\_support)/urdf/lrmate200id\_macro.xacro"/>

<xacro:include filename="$(find robotiq\_c2\_model\_visualization)/urdf/robotiq\_c2\_model\_macro.xacro"/>

<xacro:include filename="$(find motoman\_sia20d\_support)/urdf/sia20d\_macro.xacro"/>

<xacro:fanuc\_lrmate200id prefix="fanuc\_"/>

<xacro:robotiq\_c2\_model prefix="fanuc\_"/>

<xacro:motoman\_sia20d prefix="motoman\_"/>

<xacro:robotiq\_c2\_model prefix="motoman\_"/>

Place the robots w/ ROS Xacro/URDF – robots positioned +/-.5 in y

<link name="world" />

<!-- Dummy Link -->

<link name="link0" />

<joint name="world\_joint" type="fixed">

<parent link="world" />

<child link="link0" />

<axis xyz="0 0 1"/>

</joint>

<!-- First Robot FANUC -->

<joint name="fanuc\_joint\_0" type="fixed">

<parent link="link0" />

<child link="fanuc\_base\_link" />

<origin xyz="0 -0.5 0" rpy="0 0 0"/>

</joint>

<!-- Second Robot MOTOMAN -->

<joint name="motoman\_joint\_0" type="fixed">

<parent link="link0" />

<child link="motoman\_base\_link" />

<origin xyz="0 0.5 0" rpy="0 0 0"/>

</joint>

Connect the grippers to the end link of the robots:

<joint name="fanuc\_joint\_6-tool0" type="fixed">

<origin xyz="0 0 0" rpy="0 0 0" />

<parent link="fanuc\_link\_6" />

<child link="fanuc\_robotiq\_85\_adapter\_link" />

<axis xyz="0 0 0"/>

</joint>

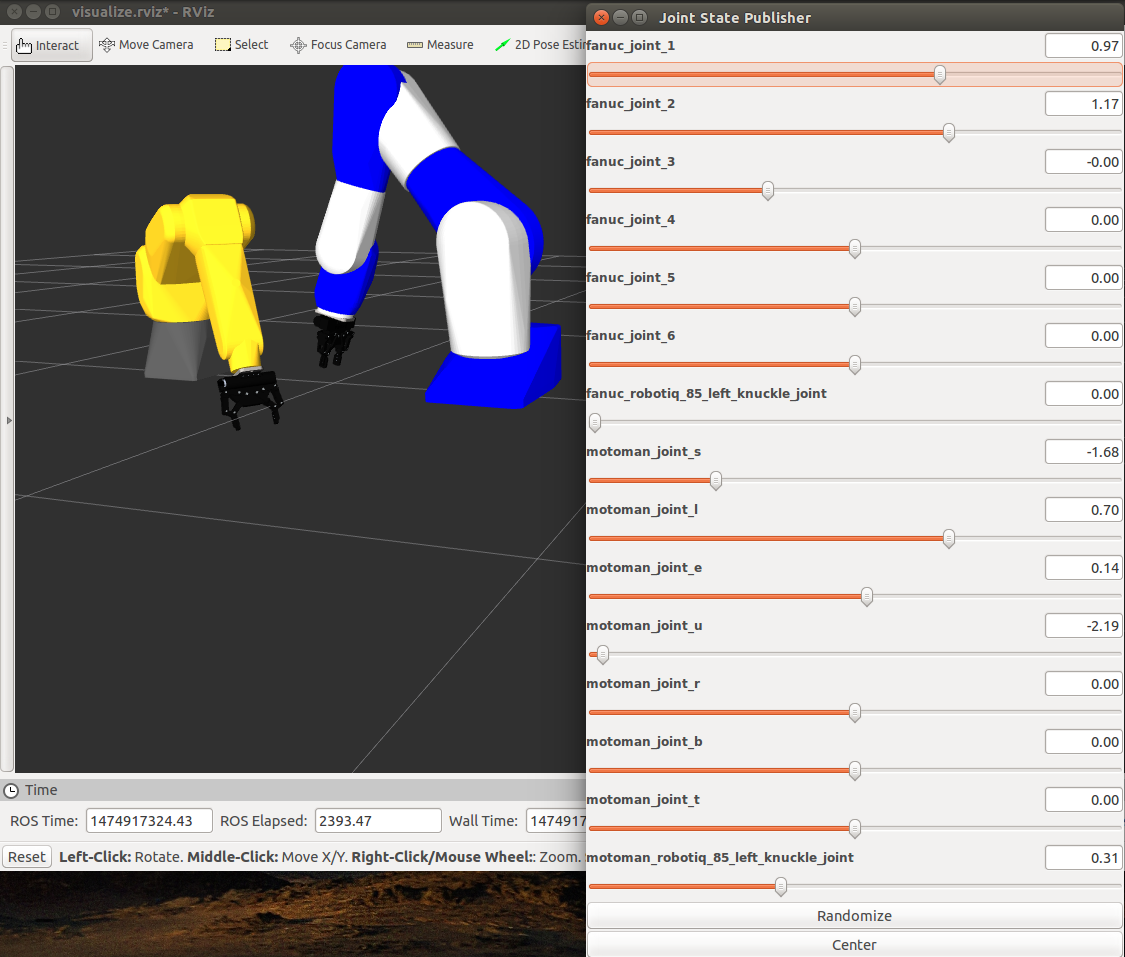
<joint name="motoman\_link\_t-tool0" type="fixed" >

<origin xyz="0 0 0.0" rpy="0 -1.57 0"/>

<parent link="motoman\_link\_t" />

<child link="motoman\_robotiq\_85\_adapter\_link" />

</joint>



# Installation

The current implementation is based on the ROS indigo version, which supports the Ubuntu 14.04 distribution and employs the catkin beta build system (<http://catkin-tools.readthedocs.io/en/latest/>) . Later a section on using the Netbeans C++ IDE will be discussed.

The github site contains a couple ROS workspaces: primarily checkers\_ws, nistfanuc\_ws and crcl\_ws. Included in these workspaces are ROS packages found in repositories at https://github.com/ros-industrial:

* fanuc - for some URDF and Xacro definitions, which is part of the main ROS distribution
* fanuc\_lrmate200id\_support package - inside fanuc\_experimental repository, other packages are not required. Used for URDF and visualization.
* motoman\_sia20d\_support package - inside motoman repository, other packages are not required. Used for URDF and visualization.
* robotiq\_c2\_model\_visualization package inside the robotiq repository - other packages are not required. Used for URDF and visualization.

## CRCL Installation Requirements

CRCL is a standalone workspace, with Python testing programs. The nist\_crcl ROS package is integrated into the checkers\_ws and nistfanuc\_ws workspaces. The nist\_crcl package handles socket communication with a CRCL client. The client will send CRCL commands and expect CRCL status in XML. The XML commands are enocded into ROS messages and are then communicated via a ROS topic (i.e., crcl\_command).

The CRCL package requires the following code installations

1. Xerces
2. CodeSythesis

## Installing Xerces c with Ubuntu

<https://www.daniweb.com/hardware-and-software/linux-and-unix/threads/409769/ubuntu-11-10-xerces-c> As far as I'm aware libxerces is the same as pretty much any other library in Debian based systems. It should be available in the repositories (the exact version will depend on which version of Ubuntu you're running).

You can use apt-get to install the packages for the library and the dev files. Then to use them in your C/C++ programs you simply #include the appropriate headers and link with the library when compiling/linking.

sudo apt-get update

apt-cache search libxerces

sudo apt-get install libxerces-c3.1 libxerces-c-dev

Need include file path CMakeLists.txt:

include\_directories(/usr/include/xercesc)

Link library in CMakeLists.txt:

link\_directories(/usr/lib/x86\_64-linux-gnu/)

Need to link against libxerces.a in CMakeLists.txt:

target\_link\_libraries(nist\_fanuc

libxerces-c.a

${catkin\_LIBRARIES}

${Boost\_LIBRARIES}

)

## Installing CodeSynthesis XSD

<http://www.codesynthesis.com/products/xsd/download.xhtml>

1. Chose the linux deb install file that matches your computer (below 64 bit amd).
2. Download xsd\_4.0.0-1\_amd64.deb and it will say open with Ubuntu Software Center
3. Click to install, authenticate and add /usr/include/xsd/cxx/xml as include path.

Whenever you use CodeSynthesis in ROS, you need to include file path by adding the following to CMakeLists.txt:

include\_directories(/usr/include/xsd/cxx/xml)

If you cannot run Ubuntu software center to install CodeSynthesis, you can download the source and install it. You need to go to the web page: <http://www.codesynthesis.com/products/xsd/download.xhtml> and select:

xsd-4.0.0-x86\_64-linux-gnu.tar.bz2

It will be saved into /usr/local/downloads, but you can save it anywhere. Then cd to where you saved it, and do this:

tar --bzip2 -xvf xsd-4.0.0-x86\_64-linux-gnu.tar.bz2 (dash-dash bzip2, dash-xvf)

It will create a directory xsd-4.0.0-x86\_64-linux-gnu.

Make a symbolic link:

ln -s <path/to/xsd-4.0.0-x86\_64-linux-gnu/libxsd/xsd /usr/local/include/xsd

e.g., ln -s /usr/local/xsd-4.0.0-x86\_64-linux-gnu/libxsd/xsd /usr/local/include/xsd

# ROS Workspace

We assume that ROS has been installed.

## Setting up the ROS workspace

$ source /opt/ros/indigo/setup.bash

Create the ROS checkers\_ws workspace:

$ mkdir -p /home/local/michalos/checkers\_ws/src

$ cd /home/local/michalos/checkers\_ws/src

$ catkin init

Now add the github packages to the ROS checkers\_ws catkin workspace. In theory you can git clone under the directory /usr/local/michalos and the workspace will be created.

## Compile ROS packages

$ cd /home/local/michalos/checkers\_ws

$ catkin build -DCMAKE\_BUILD\_TYPE=Debug

# Netbeans

Netbeans was used as the IDE for debugging instead of Gnu Emacs. To install Netbeans, navigate to: <https://netbeans.org/downloads/>

Change directory (i.e., cd) to the directory where you downloaded netbeans:

./netbeans-8.1-cpp-linux-x64.sh

There may be better ways to incorporate Netbeans into ROS. The following hard coding works.

Before you call any ROS you must set up the environment to match the ROS command :

source setup.bash

Without the proper shell environment variables, ROS will fail. So the following is an example of hard coding the environment setup

setenv("ROS\_ROOT", "/opt/ros/indigo/share/ros", true);

setenv("ROS\_PACKAGE\_PATH",

"/usr/local/michalos/nistfanuc\_ws/src/fanuc\_lrmate200id\_support:"

"/usr/local/michalos/nistfanuc\_ws/src/nist\_fanuc:"

"/usr/local/michalos/nistfanuc\_ws/src/nistcrcl:"

"/opt/ros/indigo/share:/opt/ros/indigo/stacks", true);

setenv("ROS\_MASTER\_URI", "http://localhost:11311", true);

setenv("ROS\_DISTRO", "indigo", true);

setenv("ROS\_ETC\_DIR", "/opt/ros/indigo/etc/ros", true);

setenv("PYTHONPATH", "/usr/local/michalos/nistfanuc\_ws/devel/lib/python2.7/dist-packages:"

"/usr/local/michalos/nistcrcl\_ws/devel/lib/python2.7/dist-packages:"

"/opt/ros/indigo/lib/python2.7/dist-packages:"

"/home/isd/michalos/el-robotics-core/nist\_kitting/src", true);

setenv("PATH", "/usr/local/michalos/nistfanuc\_ws/devel/bin:/usr/local/michalos/nistcrcl\_ws/devel/bin:/opt/ros/indigo/bin:/usr/local/jdk1.8.0\_60/bin:/bin:/usr/bin:/usr/local/bin:/sbin:/usr/sbin:/usr/local/sbin:/usr/X11R6/bin:/usr/local/ulapi/bin:/usr/local/gomotion/bin:/home/isd/michalos/bin", true);

BEFORE:

// Initialize ROS

ros::init(argc, argv, "myrospackage");

ros::NodeHandle nh;

ros::Rate r(50);

It is suggested you perform a source setup.bash, and then do a

env | grep ROS

to understand what the environment variables are set to. PATH and PYTHONPATH do not contain ROS so you will have to examine them in env.

The setenv() function is part of C++ and is included with the command: #include <stdlib.h>. setenv() adds the variable name to the environment with the value value, if name does not already exist. If name does exist in the environment, then its value is changed to value if overwrite is nonzero; if overwrite is zero, then the value of name is not changed.