Readme for Textbased Command and Control using Socket of a ROS RVIZ Visualization of a Motoman

Michaloski, John L. (Fed)

11/17/2016 5:13:00 PM

This document presents a Command and Control (C&C) text based interface for communicating with Robot Operating System (ROS) package thjat interfaces to an ROS RVIZ visualation of a robot (currently only Motoma SI20D). This implementation offers robot controllers that are not ROS based the ability to simulate robot motion that is displayed in RVIZ. It uses the Unified Robot Description Format (URDF) to provide the ROS parameter "robot\_description" used by RVIZ to draw and control the robot.

The tested version information for the ROS implementation is:

* ROS indigo
* OS - Ubuntu 12.04 (64-bit)
* Boost ASIO
* Package versions in Appendix I

The text based C&C interface appear at the end of the document. Note many of the packages don't appear to be used but are included as a dependency by other packages.

# Software Architecture

The ROS software is minimalistic as possible. You will need ROS, RVIZ, joint\_state\_publisher, and robot\_transform packages installed, but these are part of the main distribution so this should not be an issue. RVIZ reads the robot\_description ROS parameter to determine how and where to display a robot. Using this robot\_description ROS parameter the cmdinterpreter package understands the joint configuration of the robot. The command and control interface to the RVIZ robot is only through joints. (However, the ability to send RVIZ markers which required poses (position and orientation) is available, but is not documented at this time.)

The cmdinterpreter package advertises joint value updates to the /nist\_controller/robot/joint\_states topic which is read by the joint\_state\_publisher package.



The cmdinterpreter package advertises updates to the /nist\_controller/robot/joint\_states topic which is read by the joint\_state\_publisher package. This communication is enabled in the launch file by the following snippet:

<node name="joint\_state\_publisher" pkg="joint\_state\_publisher" type="joint\_state\_publisher">

<param name="/use\_gui" value="true"/>

<rosparam param="/source\_list">[nist\_controller/robot/joint\_states]</rosparam>

</node>

In roslaunch, a robot description for the motoman sia20d is loaded and used by RVIZ.

<param name="robot\_description" command="$(find xacro)/xacro.py $(find motoman\_sia20d\_support)/urdf/ motoman\_sia20d.xacro" />

<node name="rviz" pkg="rviz" type="rviz">

# Installation

The Motoman Command and Control Visualization assumes ROS indigo has been installed on your platform. It should work under Jade or Kinetic versions of ROS but has not been tested. There is not much source, and you will have to compile, so

## Download & Build from Source

First, build your workspace:

> mkdir -p tcp\_c2\_rviz/src

> cd tcp\_c2\_rviz/src

Then clone the motoman visualizaton code from the github repository.

# clones motoman rvi packages into the src dirctory

> git clone <https://github.com/usnistgov/tcp_c2_rviz.git>

# commands are initiated from the home ws directory

> cd ..

# build where catkin is a set of CMake macros that simplify build and maintenance

> catkin build -DCMAKE\_BUILD\_TYPE=Debug

# update path environment variable for ROS

> source devel/setup.sh

# Testing

After compiling you will have 3 packages: cmdinterpreter, motoman\_sia20d\_support, and robotconfig.

* motoman\_sia20d\_support contains the urdf/xacro and robot description
* robotconfig configures the motoman xacro and has launch file
* cmdinterpreter contains the ROS interface to the joint publisher, which reads commands on a socket and publishes the joints to a ROS topic.

To run the motoman RVIZ command and control interfface, run he following roslaunch

> cd tcp\_c2\_rviz

> sourc devel/setup.bash

> roslaunch cmdinterpreter rviz.launch

You should see the following two screens: (1) the rviz motoman visualization; and (2) the joint\_state\_publisher

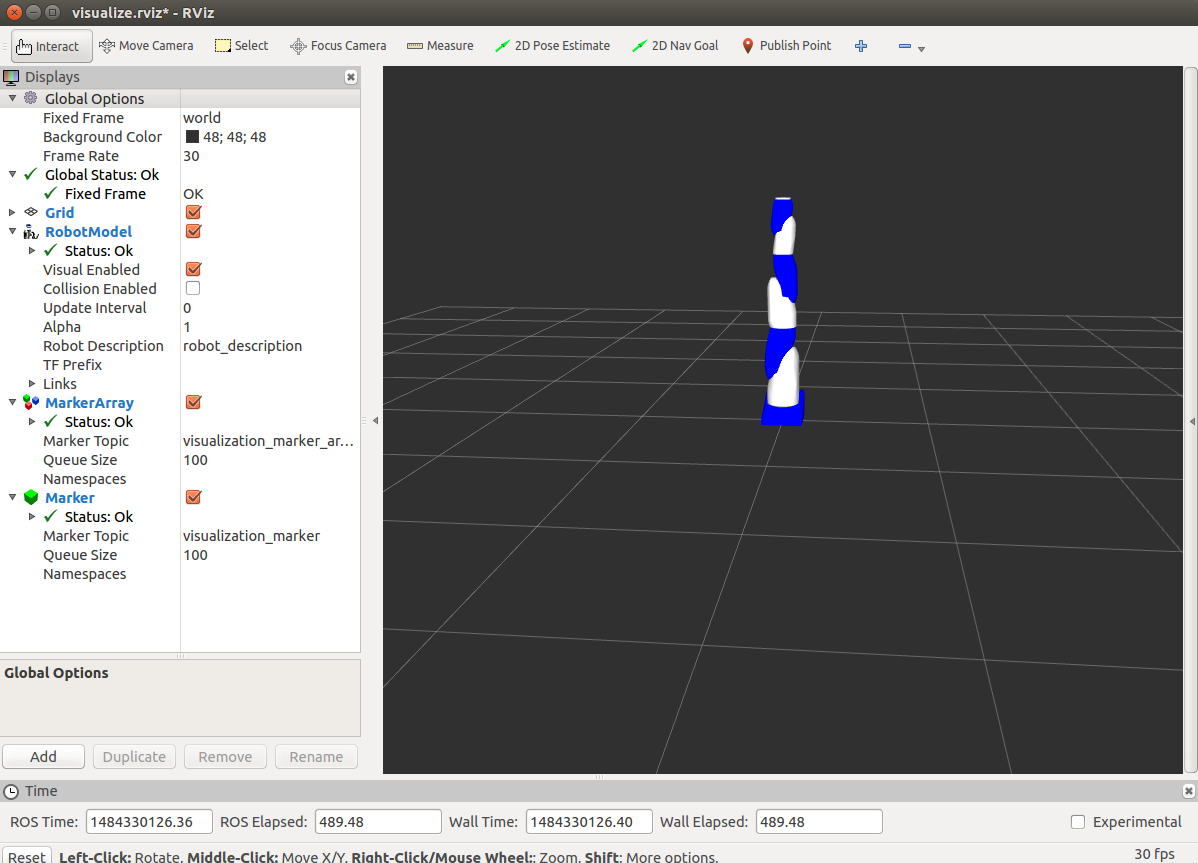


Figure 2 Motoman RVIZ Visualization

Now you can test the RVIZ motoman command and control. Open a separate terminal and change to the tcp\_c2\_rviz directory and under it the python directory in the cmdinterpreter src package.

cd xxxx/tcp\_c2\_rviz/src/cmdinterpreter/python

Now run the python application that will accepts text based command and control joint values for the motoma:

> python socketcmdline.py

You should see the following sequence of socket information be displayed, as the python program is connecting vial socket 31000 to the cmdinterpreter ROS package:

Socket Created  
Socket Connected

>

You can now run some test commands that should be displayed in the RVIZ motoman display. Below are some sample commands with comments following the "#" pound sign:

> rand # randonly move joints  
> home # home joints, i.e., move joints to all zero positions  
> degrees # use degrees as input values, not radians  
> j 45,45,45,45,45,45,45 # move ALL joints to all given positions

# joints and j are identical commands  
> sleep 10.0 # sleep 10.0 seconds (double to specify seconds)  
> move 0 90 # move n,m p,q : move joints n,m to positions p,q  
> move 0 -90 # ditto

There is not much error checking. So buyer beware. There is also the ability to read and process a test file of RVIZ motoman commands.

> file motoman.txt  
subtokens ['sleep', '5']  
subtokens ['degrees']  
subtokens ['home']  
subtokens ['rand']  
subtokens ['sleep', '5']  
subtokens ['move', '0', '90.0']  
subtokens ['sleep', '3']  
subtokens ['move', '0', '-90.0']  
subtokens ['sleep', '3']  
subtokens ['j', '45,45,45,45,45,45,45']  
subtokens ['sleep', '3']  
subtokens ['joints', '0,0,0,0,0,0,0']  
subtokens ['sleep', '3']  
subtokens ['move', '0,1', '90,90']  
subtokens ['sleep', '4.0']  
subtokens ['move', '0,1', '-90,-90']  
>

If you want to quit the Python command line interface program hit ^C and the program will exit. If you want to exit both the Python command line interface program and the RVIZ motoman visualization, enter the command "quit".

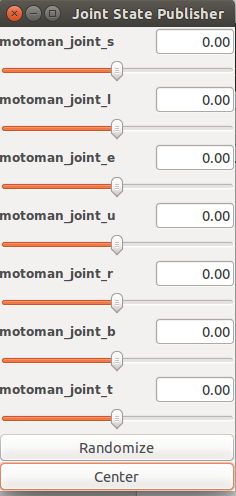


Figure 3 joint\_state\_publisher ROS GUI

Command and Control Language

Most of the command and control is for the ROS package intervace. Some commands instruct the Python test interpreter and the Description field will denote this use.

|  |  |  |
| --- | --- | --- |
| Command | Parameters | Description |
| rand |  | Moves to randomly generated joint values |
| home |  | home joints, i.e., move joints to all zero positions |
| degrees |  | Use degrees for joint angle values |
| radians |  | Use radians for joint angle values |
| joints | Val0,..., valn | move ALL joints to all given positions, ALL joint values must be specified, from 0 to number of robot joints. |
| j | Synomym for joints |  |
| move | {i,..., j} vali,..., valj | move joints n,m to joint values p,q |
| file | filename | Python interpreter will read the file and then send each command to the ROS C&C cmdinterpreter. |
| sleep | seconds | Python sleeps for seconds in double |

# Version Dependencies

|  |  |
| --- | --- |
| **Package** | **Version** |
| actionlib | 1.11.6 |
| actionlib\_msgs | 1.11.9 |
| catkin | 0.6.18 |
| class\_loader | 0.3.4 |
| cpp\_common | 0.5.7 |
| eigen\_conversions | 1.11.8 |
| eigen\_stl\_containers | 0.1.4 |
| fcl | 0.3.3 |
| gencpp | 0.5.5 |
| genlisp | 0.4.15 |
| genmsg | 0.5.7 |
| genpy | 0.5.10 |
| geometric\_shapes | 0.4.4 |
| geometry\_msgs | 1.11.9 |
| graph\_msgs | 0.1.0 |
| kdl\_conversions | 1.11.8 |
| kdl\_parser | 1.11.11 |
| libccd | 1.5.0 |
| message\_filters | 1.11.20 |
| message\_generation | 0.2.10 |
| message\_runtime | 0.4.12 |
| moveit\_core | 0.7.2 |
| moveit\_msgs | 0.7.4 |
| object\_recognition\_msgs | 0.4.1 |
| octomap | 1.6.9 |
| octomap\_msgs | 0.3.3 |
| orocos\_kdl | 1.3.1 |
| pluginlib | 1.10.3 |
| python\_orocos\_kdl | 1.3.1 |
| random\_numbers | 0.3.0 |
| resource\_retriever | 1.11.6 |
| rosbag | 1.11.20 |
| rosbag\_migration\_rule | 1.0.0 |
| rosbag\_storage | 1.11.20 |
| rosbuild | 1.11.13 |
| rosclean | 1.11.13 |
| rosconsole | 1.11.20 |
| rosconsole\_bridge | 0.4.4 |
| roscpp | 1.11.20 |
| roscpp\_serialization | 0.5.7 |
| roscpp\_traits | 0.5.7 |
| rosgraph | 1.11.20 |
| rosgraph\_msgs | 1.11.2 |
| roslaunch | 1.11.20 |
| roslib | 1.11.13 |
| roslz4 | 1.11.20 |
| rosmaster | 1.11.20 |
| rosmsg | 1.11.20 |
| rosnode | 1.11.20 |
| rosout | 1.11.20 |
| rospack | 2.2.7 |
| rosparam | 1.11.20 |
| rospy | 1.11.20 |
| rosservice | 1.11.20 |
| rostest | 1.11.20 |
| rostime | 0.5.7 |
| rostopic | 1.11.20 |
| rosunit | 1.11.13 |
| roswtf | 1.11.20 |
| rviz\_visual\_tools | 2.2.1 |
| sensor\_msgs | 1.11.9 |
| shape\_msgs | 1.11.9 |
| srdfdom | 0.3.1 |
| std\_msgs | 0.5.10 |
| tf | 1.11.8 |
| tf2 | 0.5.13 |
| tf2\_msgs | 0.5.13 |
| tf2\_py | 0.5.13 |
| tf2\_ros | 0.5.13 |
| tf\_conversions | 1.11.8 |
| topic\_tools | 1.11.20 |
| trajectory\_msgs | 1.11.9 |
| urdf | 1.11.11 |
| urdf\_parser\_plugin | 1.11.11 |
| urdfdom\_py | 0.3.1 |
| visualization\_msgs | 1.11.9 |
| xmlrpcpp | 1.11.20 |