ROS Readme for NIST Fanuc Operation

Michaloski, John L.

5/30/2016 2:00:00 PM

NistFanucReadme.docx

This document presents installation of necessary ROS components.

Table of Contents

[Installation Readme 2](#_Toc452733216)

[Set up ROS with Descartes and ROS Industrial 2](#_Toc452733217)

[Installing Xerces c with Ubuntu 3](#_Toc452733218)

[Installing CodeSynthesis XSD 4](#_Toc452733219)

[Optionally - download netbeans 4](#_Toc452733220)

[Setup ROS workspace 4](#_Toc452733221)

[Compile ROS packages 5](#_Toc452733222)

[Run ROS Simulator 5](#_Toc452733223)

[Install Java 1.08 from Oracle for Java CRCL Tool 5](#_Toc452733224)

[Install doxygen for documentation generation 6](#_Toc452733225)

[Fanuc LRMate 200iD Robot Operation Instructions 7](#_Toc452733226)

[Powerup: 7](#_Toc452733227)

[Powerdown: 7](#_Toc452733228)

[Fault Recovery 7](#_Toc452733229)

[Run ROS Fanuc demo 7](#_Toc452733230)

[Run RVIZ roslaunch with Fanuc LRMate 200id 7](#_Toc452733231)

[Launch file: 8](#_Toc452733232)

[Fanuc ROS installation tutorial - I 8](#_Toc452733233)

[Fanuc ROS installation tutorial - II 8](#_Toc452733234)

# Installation Readme

## Set up ROS with Descartes and ROS Industrial

Copy repositories from https://github.com/ros-industrial:

fanuc

fanuc experimental

motoman

ros industrial core

Repositories from ROS-I Consortium https://github.com/ros-industrial-consortium github site:

descartes

descartes\_tutorials

## 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.

Need include file path in CMakeLists.txt:

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

If you cannot run Ubuntu software centerto 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

## Optionally - download netbeans

https://netbeans.org/downloads/

cd to the directory where you downloaded netbeans:

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

## Setup ROS workspace

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

Let's create a catkin workspace:

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

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

$ catkin\_init\_workspace

Now you can add your packages to the ROS catkin workspace. For those packages you cloned from GITHUB, you will need to symbolically link the package directory to the github local directory))

$ mkdir -p /usr/local/github/ros-industrial

$ cd /usr/local/github/ros-industrial

$ git clone https://github.com/ros-industrial/fanuc\_experimental.git

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

$ ln -s /usr/local/github/ros-industrial/fanuc\_experimental fanuc\_experimental

And do for descartes, nist\_fanuc, etc.

## Compile ROS packages

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

$ catkin\_make -DCMAKE\_BUILD\_TYPE=Debug

## Run ROS Simulator

$ cd /usr/local/michalos/

$ cd nistfanuc\_ws/

$ source devel/setup.bash

$roslaunch fanuc\_lrmate200id\_moveit\_config moveit\_planning\_execution.launch sim:=true

## Install Java 1.08 from Oracle for Java CRCL Tool

To build java crcl tool one needs:

JDK 1.8+ (http://www.oracle.com/technetwork/java/javase/downloads/index.html) and

maven 3.0.5+ (https://maven.apache.org/download.cgi)

Install maven: $ sudo apt-get install maven

Use the command:

mvn package

If you see this message at the beginning, bummer:

Warning: JAVA\_HOME environment variable is not set.

You can check /usr/lib/jvm to see if a 1.8 Java Virtual Machine has been installed. If so, skip the installation step.

So you do not have Java installed. These are instructions for the less than sudo installers. Note, you need the Oracle Java JDK 1.8 version, not the 1.7 version of Ubuntu!!!

Download, unzip and copy to /usr/local/jdk1.8.0\_77 or whatever is the latest 1.8 version.

Change you will need to change .bashrc to set the PATH to know where the jdk is installed:

for dir in /usr/local/jdk1.8.0\_77/bin /usr/lib/jvm/java-[6,7,8]-\*/bin /usr/local/jdk\*/bin /usr/local/jdk\*/bin ; do

if [ -x $dir/java ] ; then

javadir=$dir

fi

done

if [ x"$javadir" = x ] ; then javadir=/usr/bin ; fi

# platform-specific environment vars

THISPLAT=`uname -s` ; export THISPLAT

case $THISPLAT in

Linux)

PATH=$javadir:

And make sure you source the .bashrc before you attempt run.sh in java crcl. The voodoo worked for me.

## **Install doxygen for documentation generation**

$ sudo apt-get install doxygen

# Fanuc LRMate 200iD Robot Operation Instructions

## Powerup:

1. Turn on power on front of controller (keyed)
2. If auto mode, make sure teach pendant upper left corner knob is **OFF**
3. If in fault- hold deadman switch halfway, Hold [**Shift**], press [**Reset**] key
4. reset to local mode Menu -> [32] Remote/Local/... [F4] Local [Enter]
5. Start ROS programs[Teach][Select] => scroll down to ROS, number 39 hit [Enter] (starts 2 programs)
6. Cycle startGreen Auto button on front controller panel, press/release, green Light should go on.

## Powerdown:

1. Kill ROS programs - DO TWICE - 2 programs running [FCNT] -> 1 -> [ENTER]  
   [FCNT] -> 1 -> [ENTER]

## Fault Recovery

If fanuc controller faulted, you have to manually reset the joint(s) to a "safe" position:

1. Turn controller box to teach pendant from auto
2. hold deadman switch half-on, [**SHIFT**] Hold, hit [**Reset**]
3. Now move robot - +/- joint key or xyz key
4. Note to increase traversal- feedoverride in green xx% field in upper right corner

## Run ROS Fanuc demo

roslaunch fanuc\_lrmate200id\_moveit\_config moveit\_planning\_execution.launch

sim:=false robot\_ip:=129.6.78.111

## Run RVIZ roslaunch with Fanuc LRMate 200id

**FIXME**

#!/bin/bash

source .../catkin\_ws/devel/setup.bash

roslaunch nist\_fanuc lrmate200id\_sim.launch

sleep 100

### Launch file:

<?xml version="1.0"?>

<launch>

<arg name="sim" default="true" />

<include file="$(find fanuc\_lrmate200id\_moveit\_config)/launch/moveit\_planning\_execution.launch">

<arg name="sim" value="$(arg sim)"/>

</include>

</launch>

## Fanuc ROS installation tutorial - I

Web stop #1:

http://wiki.ros.org/fanuc?distro=indigo

## Fanuc ROS installation tutorial - II

Web stop #2:

http://wiki.ros.org/fanuc/Tutorials/hydro/Installation

The “Next” link at the bottom takes you to the next in the series.

# Canonical Robot Control Language

## How does the CrclSession work?

in the main cpp program we declare the controller session thread (with default cycle time).

// This thread handles new XML messages received from crcl asio socket.

RCS::ControllerSession session(DEFAULT\_LOOP\_CYCLE);

session.Start(); // start the thread

It uses the Thread template to run cyclically.

class Thread

void Cycle ( )

{

Init( ); <--- calls ControllerSession::Init()

\_timer.sync( );

while ( \_bThread )

{

Action( );

\_timer.wait( );

}

Cleanup( );

}

void ControllerSession::Init() {

std::string info;

crclinterface= boost::shared\_ptr<Crcl::CrclDelegateInterface>(

new Crcl::CrclDelegateInterface() );

\_kinematics = boost::shared\_ptr<IKinematics>(new DummyKinematics());

std::string sStatus = DumpHeader(",") + "\n";

CsvLogging.Timestamping() = false;

CsvLogging.LogMessage("Timestamp," + sStatus);

}

## How to publish joints to ROS

http://answers.ros.org/question/43157/trying-to-use-get-joint-state-with-my-urdf/

## rostopic to send commands to rviz

rostopic pub <topic-name> <topic-type> [data...]

rostopic pub -1 /joint\_states sensor\_msgs/JointState '{header: auto, name: ['joint\_1', 'joint\_2', 'joint\_3', 'joint\_4', 'joint\_5', 'joint\_6'], position: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ], velocity: [], effort: []}'

rostopic pub -1 /joint\_states sensor\_msgs/JointState '{header: auto, name: ['joint\_1', 'joint\_2', 'joint\_3', 'joint\_4', 'joint\_5', 'joint\_6'], position: [0.097, 0.007, -0.590, -0.172, 0.604, -0.142 ], velocity: [], effort: []}'

rostopic pub /joint\_states sensor\_msgs/JointState '{header: auto, name: ['joint\_1', 'joint\_2', 'joint\_3', 'joint\_4', 'joint\_5', 'joint\_6'], position: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ], velocity: [], effort: []}'

Writing joint values to rviz and then moving robot and "waiting" until rviz reaches goal

# CJointWriter

CJointWriter is a class that handles updating the joint value to be displayed by RVIZ, or if a real robot is running, update the robot position.

class CJointWriter {

public:

CJointWriter(ros::NodeHandle &nh);

// Position only for now

bool JointTrajectoryWrite(std::vector<sensor\_msgs::JointState>);

bool JointTrajectoryPositionWrite(sensor\_msgs::JointState joint);

////////////////////////////////

ros::Publisher traj\_pub;

std::vector<std::string> jointnames;

static boost::mutex \_writer\_mutex;

};

CJointWriter constructor requires the ROS node handle, to advertise that it will be publishing joint values. In order to publish joint values, it must have the names of the joints that it will be updating. Thus, the command geParam with the parameter name controllerjointnames retrieves a list of the jointnames. Then, the constructor advertises to ROS that it will be publishing to the topic “jointpathcommand”.

CJointWriter::CJointWriter(ros::NodeHandle &nh)

{

nh.getParam("controller\_joint\_names", jointnames);

// Trajectory publisher

traj\_pub = nh.advertise<trajectory\_msgs::JointTrajectory>("joint\_path\_command", 1);

}

The JointTrajectoryPositionWrite method publishes updated joint values that the ROS system will publish to all listeners (which of interest in our case is RVIZ). It accepts a sensormsg JointState structure containing the updated joint values. In theory, the jointpath\_command topic could accept many points to display, however, only 1 point at a time is written to the topic.

bool CJointWriter::JointTrajectoryPositionWrite(sensor\_msgs::JointState joint) {

ActionGoal traj\_goal;

trajectory\_msgs::JointTrajectory traj;

std::vector<trajectory\_msgs::JointTrajectoryPoint> points;

size\_t n\_joints=joint.position.size();

// Where we are going

trajectory\_msgs::JointTrajectoryPoint point;

point.positions.resize(n\_joints);

point.positions=joint.position;

point.velocities.resize(n\_joints, 0.0);

point.accelerations.resize(n\_joints, 0.0);

traj.joint\_names = jointnames;

traj.points.resize(1, point);

// Send trajectory

traj.header.stamp = ros::Time(0); // Start immediately

traj\_pub.publish(traj);

return true;

}

The JointTrajectoryWrite method to updates a vector of joint values. (Unclear if this is useful.) It accepts a std vector of sensor\_msg JointState values, and will update and then write each value (containing a vector of joint positions).

bool CJointWriter::JointTrajectoryWrite(std::vector<sensor\_msgs::JointState> joints ) {

for (size\_t i = 0; i < joints.size(); i++) {

JointTrajectoryPositionWrite(joints[i]);

}

return true;

}

The package.xml manifest contains all the following Moveit! entries, although it is unclear which ones are necessary.

<build\_depend>moveit\_core</build\_depend>

<build\_depend>moveit\_ros\_planning\_interface</build\_depend>

<build\_depend>moveit\_ros\_move\_group</build\_depend>

<build\_depend>moveit\_ros\_planning</build\_depend>

<build\_depend>moveit\_ros\_manipulation</build\_depend>

<run\_depend>moveit\_core</run\_depend>

run\_depend>moveit\_ros\_planning\_interface</run\_depend>

<run\_depend>moveit\_ros\_move\_group</run\_depend>

<run\_depend>moveit\_ros\_planning</run\_depend>

<run\_depend>moveit\_ros\_manipulation</run\_depend>

Likewise the CMakeLists.txt contains the following moveit entries. Of note, the trajectory\_msgs

## Find catkin macros and libraries

## if COMPONENTS list like find\_package(catkin REQUIRED COMPONENTS xyz)

## is used, also find other catkin packages

find\_package(catkin REQUIRED COMPONENTS

moveit\_core

roscpp

cmake\_modules

trajectory\_msgs

sensor\_msgs

moveit\_ros\_planning\_interface

moveit\_ros\_move\_group

moveit\_ros\_planning

moveit\_ros\_manipulation

)

catkin\_package(

INCLUDE\_DIRS

include

LIBRARIES

CATKIN\_DEPENDS

roscpp

moveit\_core

sensor\_msgs

moveit\_ros\_planning\_interface

moveit\_ros\_move\_group

moveit\_ros\_planning

moveit\_ros\_manipulation

DEPENDS

Boost

Eigen