This document will display:# Written by Chenhan Zhang, Jaspor Jiang, and Heyun Wang. Mar.17

1. How to setup dVRK environments on Robotorium computer

2. (Optional) On your PC

3. How to run and call GUI

4. How to physically control dVRK arm: PSM3

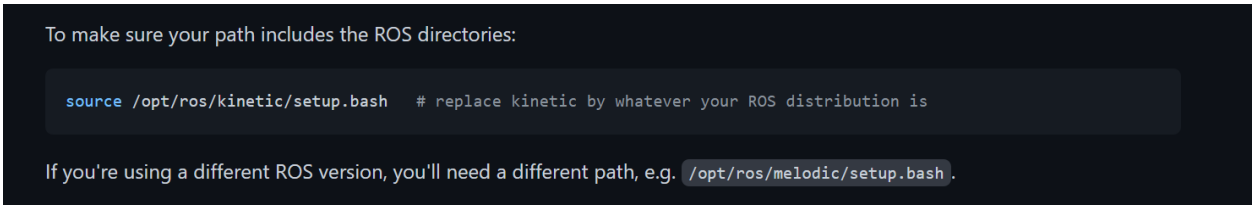
1. Section1： How to setup dVRK ROS at Robatarium

First, click this link and follow the instructions.

https://github.com/jhu-cisst/cisst/wiki/Compiling-cisst-and-SAW-with-CMake#setting-up-your-environment-variables-for-ros

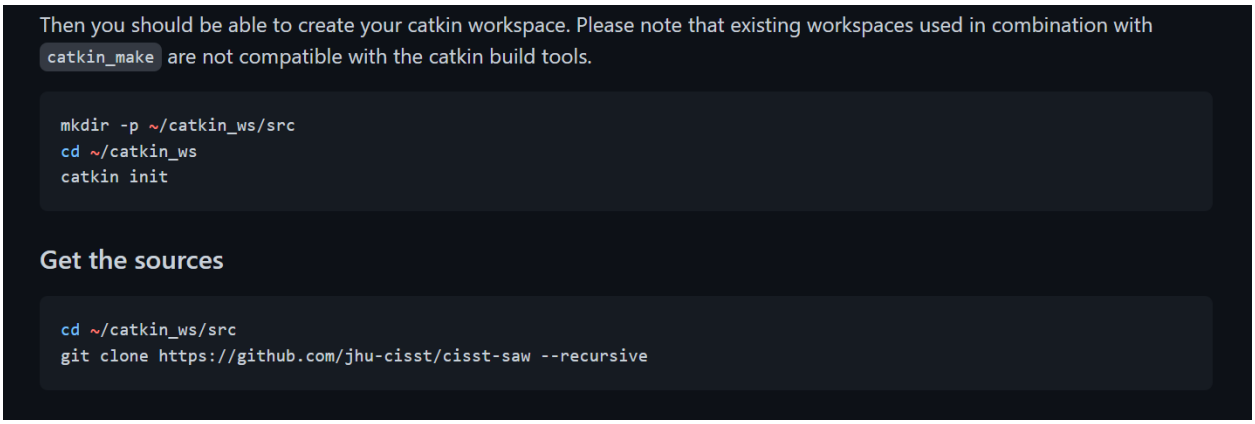
The computer at Robatarium has already installed the ROS 20.04 Noetic, so we should skip ROS install start with installing CISST-SAW library:

1). Find 1**. Building using catkin build tools for ROS, open a terminal via"ctrl+alt+t" and copy following codes from the webpage.**



Remember to replace the "kinetic" by "noetic" which is our ROS version.

2). Then create catkin workspace :



3).

Configure and build

For (significantly) better performances, you should compile your code in CMake Release mode. Using the catkin build tools, this can be done using profile configuration. Then you can build and update your environment variables:

```
# make sure we are in the right place
cd ~/catkin_ws
# make sure you have the proper ROS environment variables
source /opt/ros/kinetic/setup.bash  # or whatever your ROS distribution is
# set default CMake build type
catkin config --cmake-args -DCMAKE_BUILD_TYPE=Release
# build
catkin build --summary
# set environment variables - MAKE SURE YOU SOURCE THE RIGHT FILE
source devel/setup.bash
```

4). Use code:" gedit ~/.bashrc" to open that file and copy these codes to the end of that file.

Setting up your environment variables for ROS

If you have a single catkin workspace, you might want to automatically `source` the `setup.bash` when you log in. To do so, you should add the following lines at the end of your `~/.bashrc` (hidden file in your home directory):

```
# for ROS
if [ -f ~/catkin_ws/devel/setup.bash ]; then
  . ~/catkin_ws/devel/setup.bash
fi
# for cisst (optional)
if [ -f ~/catkin_ws/devel/cisstvars.sh ]; then
  . ~/catkin_ws/devel/cisstvars.sh
fi
```

If you're new to Unix, you can launch the text editor `gedit` with:

```
gedit ~/.bashrc
```

Save that file and back to our terminal.

5). this part is optional, it's not important for our project.

Maintaining multiple Catkin profiles (optional)

For **advanced users**, you can also create multiple profiles and have different compilation options in each profile. To create a release profile, use the following sequence of commands:

```
# make sure we are in the right place
cd ~/catkin_ws
# make sure you have the proper ROS environment variables
source /opt/ros/kinetic/setup.bash  # or whatever your ROS distribution is
# create a profile named release with _release extension
catkin config --profile release -x _release
# switch to newly created release profile
catkin profile set release
# set default CMake build type
catkin config --cmake-args -DCMAKE_BUILD_TYPE=Release
# build
catkin build --summary
# set environment variables - MAKE SURE YOU SOURCE THE RIGHT FILE (debug vs. release)
source devel_release/setup.bash
```

To create another profile (e.g. debug), the safest way is to '''open a new shell''' without any environment variables specific to the current profile and then:

```
# make sure we are in the right place
cd ~/catkin_ws
# make sure you have the proper ROS environment variables
source /opt/ros/kinetic/setup.bash
# create a profile named debug with _debug extension
catkin config --profile debug -x _debug
# switch to newly created debug profile
catkin profile set debug
# set default CMake build type
catkin config --cmake-args -DCMAKE_BUILD_TYPE=Debug
# build
catkin build --summary
# set environment variables
source devel_debug/setup.bash
```

Every time you toggle between profiles, make sure you change two things:

- Set active profile, e.g. to switch to release: `catkin profile set release`
- Set environment variables, e.g. to switch to release: `source ~/catkin_ws/devel_release/setup.bash`

6).skip 2.1, run two examples from 2.2:

2.2. CMakeLists.txt examples

Simple example

```
# in CMake we will have to set cisst_DIR to <cisst-build-tree>/cisst
find_package (cisst)

# make sure cisst is found
if (cisst_FOUND)

  # this will set the include and link directories
  # using all cisst settings
  include (${CISST_USE_FILE})

  # add a library
  add_library (myLibrary libFile.cpp libFile.h)
  # add a program
  add_executable (myExecutable main.cpp)
  # usual CMake link
  target_link_libraries (myExecutable myLibrary)

  # using cisst libraries and settings
  # if any of these has not be compiled/set, you will get a CMake error
  cisst_target_link_libraries (myExecutable cisstCommon cisstVector cisstQt)

else (cisst_FOUND)
  message (SEND_ERROR "Oops, cisst not found")
endif (cisst_FOUND)
```

Using `REQUIRED/COMPONENTS`

In this example we demonstrate how to be a bit more specific and specify which libraries are needed.

```
# make a list of libraries and settings we will need
set (REQUIRED_CISST_LIBRARIES cisstCommon cisstVector cisstQt)

# in CMake we will have to set cisst_DIR to <cisst-build-tree>/cisst
find_package (cisst REQUIRED ${REQUIRED_CISST_LIBRARIES})

# make sure cisst is found AS REQUIRED
# this checks that all required components are found
if (cisst_FOUND_AS_REQUIRED)

  # this will set the include and link directories
  # using only the settings for cisstCommon, cisstVector and cisstQt
  include (${CISST_USE_FILE})

  # same as above
  add_library (myLibrary libFile.cpp libFile.h)
  add_executable (myExecutable main.cpp)
  target_link_libraries (myExecutable myLibrary)

  # using cisst libraries and settings as defined before
  cisst_target_link_libraries (myExecutable ${REQUIRED_CISST_LIBRARIES})

else (cisst_FOUND_AS_REQUIRED)
  message (SEND_ERROR "Oops, cisst ${REQUIRED_CISST_LIBRARIES} not found")
endif (cisst_FOUND_AS_REQUIRED)
```

Now, you are all set!, try our codes on section 3 to go further!

2. Section2: (Optional) On PC

3. Section3: How to run and call GUI on ROS:

1). First, open the terminal on desktop via ctrl+alt+t (or mouse right click, but not recommended).

2). Right click on the terminal and “Split Vertically”.

3). Type in following command below, so that you should run and call the GUI of PSM3 for physical control.

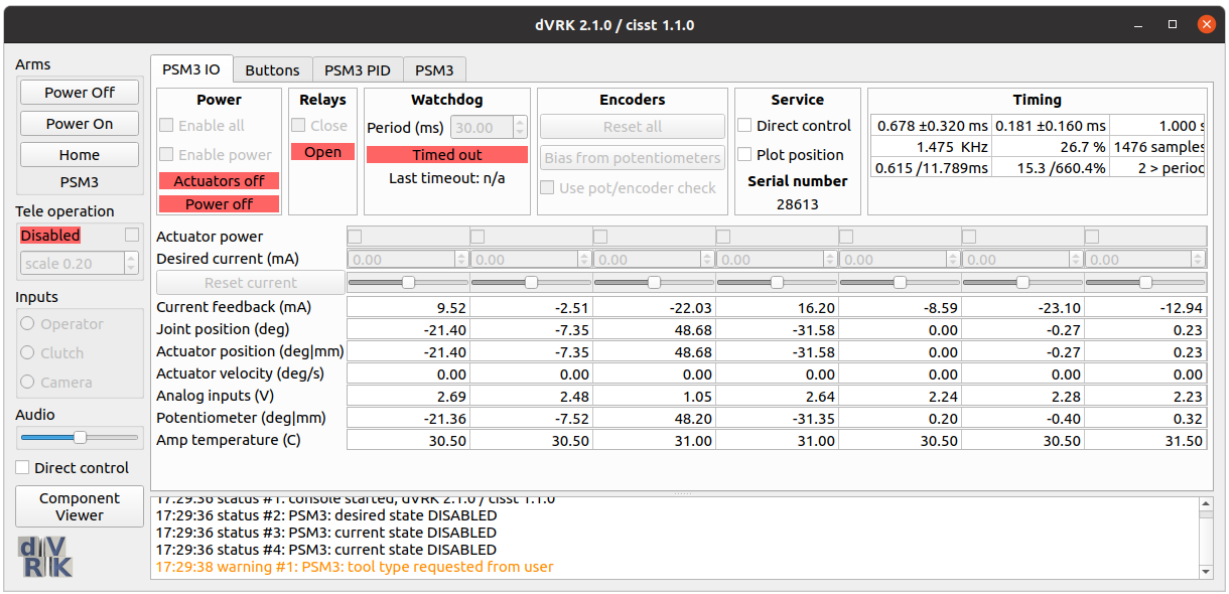
roscore # on one side teminal. And copy following code on another terminal. **Please use “crtl+c” to turn # off the roscore before logging out!!!!!**

cd ~/ # return to root.

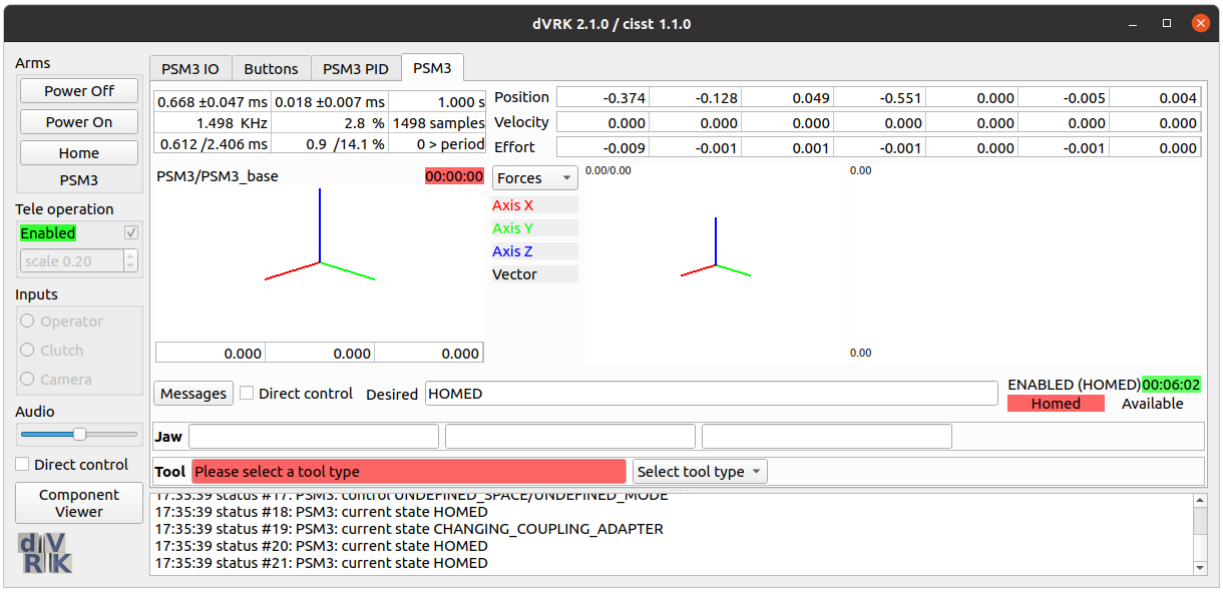
cd catkin_ws/src/cisst-saw/sawIntuitiveResearchKit/share/jhu-daVinci/ # Jump to the target folder.

rosrun dvrk_robot dvrk_console_json -j console-PSM3.json # Run and call the GUI

4). Then take following operation on GUI:

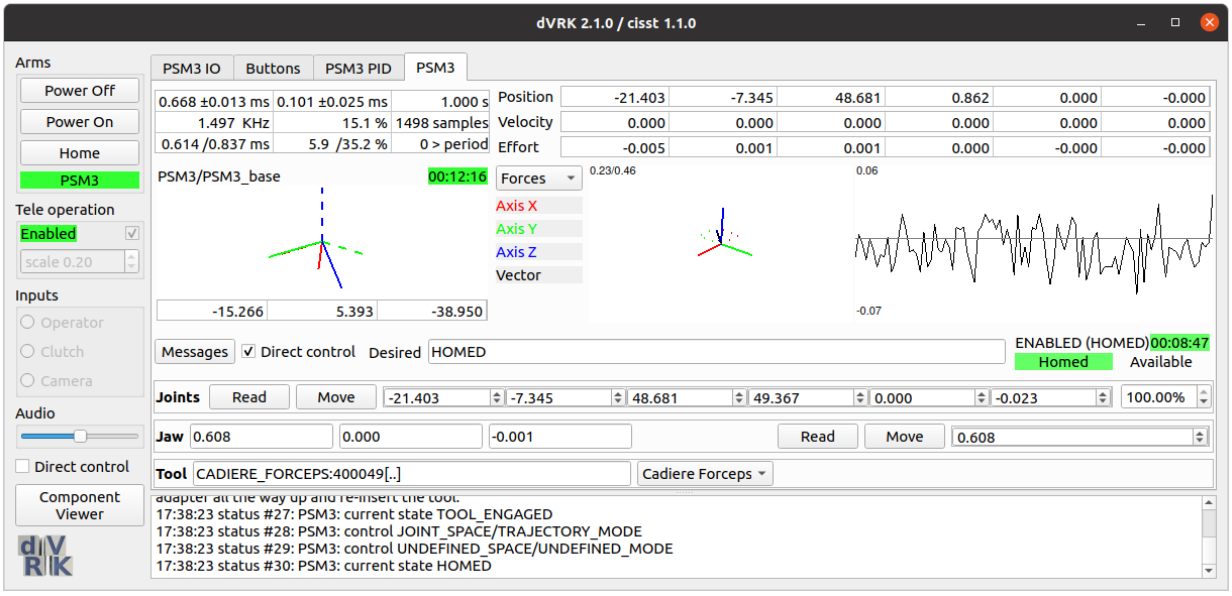


a. Click power on to power on the dVRM arm. **Please Remember click Power off or turn off GUI before you log out the system!!!**
b. Then click “Home”, “Disabled”, and “PSM3”, you will see:



c. Then click”Select tool type” choose 400049 as instance, for other tools, you can find their code on its shell. You will see a Warning, just click ”Yes”.

d. Click ”Direct control”, you will see:



E. Now it’s all set, you can do physical control on PSM3 now! Try to change the values of joints and read their values.

F. To quit the GUI, you can use “ctrl+c ” on terminal and remember quit roscore on terminal via”ctrl+c” as well.