ArcosWiki

ARCOS-Lab Humanoid Robot Simulator

Main Developer: Dr. rer. nat Federico Ruiz Ugalde

Maintainer: Daniel García Vaglio

Contributors: Javier Peralta

Description

This is a robot simulator for the ARCOS-Lab Robot. It was originally developed for TUM-Rosie robot. It simulates its two arms and fingers, force and torque sensing. It also simulates friction between an object, a table and a robot finger.

Installation instructions

- You can install this directly on your system if you have Debian Jessie stable.
- Configure xstow for local installations Using xstow for local installations
- Install software needed
- * For Debian Jessie:

```
sudo apt-get install python-numpy python-scipy python-opengl python-pygame python-matplotlib python-sip python-sip-dev python-qt4-dev python-qt4 python-gtk2 python-gtk2-dev python-vtk python-pyvtk python-gtkglext1 libeigen3-dev python-yaml python-setuptools python-future python-colorlog
```

- * For Debian Stretch and newer:

```
sudo apt install python-numpy python-scipy python-opengl python-pygame python-matplotlib python-sip python-sip-dev python-qt4-dev python-qt4 python-gtk2 python-gtk2-dev python-vtk6 python-pyvtk python-gtkglext1 libeigen3-dev python-yaml python-setuptools python-future python-colorlog
```

- Install Yarp Installing YARP in debian
- Install orocos-kdl Installing orocos kdl in debian
- Download the following script and execute it. It will install the simulator for you.

simulator_installer.sh

```
      #install avispy
      echo "install avispy"
      cd $HOME/local/src
      git clone https://gitlab.com/arcoslab/avispy.git
      cd avispy
      make xstow_install

      #install roboviewm
      echo "install roboview"
      cd $HOME/local/src
      git clone https://gitlab.com/arcoslab/roboview.git
      cd roboview
      make xstow_install

      #install cmoc
      echo "install cmoc"
      cd $HOME/local/src
      git clone https://gitlab.com/arcoslab/cmoc.git
      cd cmoc
      make xstow_install

      #install pyrovito
      echo "install pyrovito"
      cd $HOME/local/src
      git clone https://gitlab.com/arcoslab/pyrovito.git
      cd pyrovito
      make xstow_install

      #install vfl
      echo "vfl"
      cd $HOME/local/src
      git clone https://gitlab.com/arcoslab/vfl.git
      cd vfl
      make xstow_install

      #install vfclick
      echo "install vfclick"
      cd $HOME/local/src
      git clone https://gitlab.com/arcoslab/vfclik.git
      cd vfclik
      make xstow_install

      #install kdb cart cmd
      echo "install kdb cart cmd"
      cd $HOME/local/src
      git clone https://gitlab.com/arcoslab/kbd-cart-cmd.git
      cd kbd-cart-cmd
      make xstow_install

      #install arcospyu
      echo "install arcospyu"
      cd $HOME/local/src
      git clone https://gitlab.com/arcoslab/arcospyu.git
      cd arcospyu
      git checkout 0.1.2
      make xstow_install

      #robot_descriptions
      echo "robot_descriptions"
      cd $HOME/local/src
      git clone https://gitlab.com/arcoslab/robot_descriptions.git
```

Test the simulator

- Execute Yarp in one terminal:

```
yarpserver --write
```

- In another terminal execute vfclik for right arm:

```
cd ~/local/src
vfclik -i lwr -i right -d robot_descriptions/arcosbot/kinematics/lwr/ -s
```

- In another console execute vfclik for left arm:

```
cd ~/local/src
vfclik -i lwr -i left -d robot_descriptions/arcosbot/kinematics/lwr/ -s
```

- In another console execute hands simulator:

```
cd ~/local/src
sahand_yarp_sim -s -d -n -f robot_descriptions/arcosbot/kinematics/sahand/hands_kin.py
```

- In another console execute roboviewer visualizator:

```
cd ~/local/src
pyrovito -r lwr --arm_right --arm_left --hand_right --hand_left -a robot_descriptions/arcosbot/kinematics/lwr/ -d robot_descriptions/arcosbot/kinematics/sahand/
```

- In another console execute torque simulator (only working with left arm/hand):

```
cd ~/local/src
torque_sim -r -s -a robot_descriptions/arcosbot/kinematics/lwr/ -c robot_descriptions/arcosbot/kinematics/sahand/calibration_data/finger_calibration_data.py -f robot_descriptions/arcosbot/kinematics/sahand/hands_kin.py
```

- To test the system, in another console send forces to finger tips:

```
rlwrap yarp write ... /0/torque_sim/force_in
(1 -2. 0. 0.)
(1 0. 0. 0.)
```

- With the first force you should see the arm and finger moving away from the set point.
- With the second force you should see the arm and finger return to the set point.

- You can also test the system with the program kbd-cart-cmd. You can run it for the right arm with:

```
run_right.sh /0
```

- Now you can use several keys to control the arm

Sliding model and control

Cancel all the programs run in the previous instructions.

Sliding model and control without robot

- In one console execute:

```
cd ~/local/src
pyrovito -r lwr -a robot_descriptions/tum-rosie/kinematics/lwr/ -d robot_descriptions/tum-rosie/kinematics/sahand/
```

- In another console:

```
planar_sliding_simple
```

- In another console:

```
xfinger_feeder
```

- Now you can move a "virtual" finger tip against the box and try sliding it.
- Try to move the box to the origin (0,0,0) where the coordinate frame is and orient the box in a at 0 degrees. Is it difficult?
- Now try to move the box to the origin with 0 degrees of orientation in one single shot. Is it difficult?
- Restart planar_sliding_simple (ctrl-c and the run it again)
- Now cancel xfinger_feeder and execute:

```
slider_control_simple
```

- Now the computer should move the box by itself to the "goal".

Sliding controller with the whole robot

- Cancel all the programs run on the last instructions.
- Run the following commands in different consoles from the directory ~/local/src/

```
vfclik -i lwr -i right -d robot_descriptions/tum-rosie/kinematics/lwr/ -s
vfclik -i lwr -i left -d robot_descriptions/tum-rosie/kinematics/lwr/ -s
sahand_yarp_sim -s -d -n -f robot_descriptions/tum-rosie/kinematics/sahand/hands_kin.py
```

- Wait until they are fully running. Then execute this one:

```
pyrovito -r lwr --arm_right --arm_left --hand_right --hand_left -a robot_descriptions/tum-rosie/kinematics/lwr/ -d robot_descriptions/tum-rosie/kinematics/sahand/
```

- Wait until it is fully running and execute:

```
torque_sim -s -a robot_descriptions/tum-rosie/kinematics/lwr/ -c robot_descriptions/tum-rosie/kinematics/sahand/calibration_data/finger_calibration_data.py -f robot_descriptions/tum-rosie/kinematics/sahand/hands_kin.py
```

- Wait until it is fully running and execute:

```
planar_sliding -o cmoc/objects/sliding/objects/ice_tea_params.py
```

- Wait until it is fully running and execute:

```
slider_control  -s -f robot_descriptions/tum-rosie/kinematics/sahand/hands_kin.py -o cmoc/objects/sliding/objects/ice_tea_params.py
```

- This will move the box to the goal by the whole robot!

DLR HIT II

WARNING: `planar_sliding` and `slider_control` will fail because of some changes that were introduced for the dual-cmap project. And it will probably remain deprecated because **OMS** is replacing **CMOC**.

Now if you want to run the simulation using the new hand (DLR HIT II):

```
vfclik -i lwr -i right -d robot_descriptions/arcosbot/kinematics/lwr/ -s
vfclik -i lwr -i left -d robot_descriptions/arcosbot/kinematics/lwr/ -s
sahand_yarp_sim -s -d -n -f robot_descriptions/arcosbot/kinematics/sahand/hands_kin.py
pyrovito -r lwr --arm_right --arm_left --hand_right --hand_left -a robot_descriptions/arcosbot/kinematics/lwr/ -d robot_descriptions/arcosbot/kinematics/sahand/
torque_sim -s -a robot_descriptions/arcosbot/kinematics/lwr/ -c robot_descriptions/arcosbot/kinematics/sahand/calibration_data/finger_calibration_data.py -f robot_descriptions/arcosbot/kinematics/sahand/hands_kin.py
planar_sliding -o cmoc/objects/sliding/objects/ice_tea_params.py
slider_control  -s -f robot_descriptions/arcosbot/kinematics/sahand/hands_kin.py -o cmoc/objects/sliding/objects/ice_tea_params.py
```

Real robot

FRI robot communication

Installation

KRC

Check that you have FRI properly installed. In case that there are network communication problems some times there is a problem with the network card installation. The KRC has two network cards, one for Windows and one for VXworks. To be sure that the one for VXWorks is properly installed, go to control-panel→System→Hardware→Device Manager, and check that the PCI network controller is listed under "Realtime OS Devices" and not under "Network Adapters". If it is under "Network Adapters" it means windows has "claimed" it. To reassign it to VXWorks, insert the FRI installation CD and go to INTERNAT\KRCUPD directory and run "KsNetCfg.exe". This will install again the VXWorks network adapter card. This is usually a problem with the Intel Pro1000 GT card. Reboot and reconfigure FRI ini files.

Linux Installation

Download and compile fri stanford library. Remember to update yarp. This uses the latest API. You will have to recompile and reinstall yarp.

```
cd ~/local/src
git clone ssh://gitolite3@arcoslab.eie.ucr.ac.cr/fri_stanford
cd fri_stanford/Linux
mkdir -p x64/release/obj
mkdir -p x64/release/lib
mkdir -p x64/release/bin
make all_release_x64
```

If you want to build for debugging, first follow the normal steps and then

```
sudo apt-get install g++-multilib
```

Give suid permissions to LWR_yarp_arcos fri-yarp bridge to allow it to run using preemptive realtime scheduling:

```
cd ~/local/src/fri-stanford/Linux/x64/release/bin
sudo chown root: LWR_yarp_arcos
sudo chmod u+s LWR_yarp_arcos
```

Gravity

Please check that the gravity vector is correctly configured. After you have selected the current payload (configure→set_tool/base→tool_no/base_no), go to monitor→variable→single and introduce in "name": $torque_tcp_est. Press shift-enter to have realtime variable updates. They should show small values No more than 5 Newtons in x,y or z. If they are big, this usually means, that either the payload is not properly selected, or that the gravity vector is incorrect.

To check the current gravitation vector monitor→variable→single in "name": $gravitation[]. To permanently change the gravitation values, edit file :

```
c:\krc\roboter\krc\steu\mada\$custom.dat
```

At the beginning of the file you will find the variable values.

Running

- Configure your network:

The KRC has two network interfaces. One is connected to the windows operating system and the other one is connected to the QNX virtual machine for the LWR controller. In ARCOS-Lab the windows interface has the ip 192.168.3.10, the QNX interface has the ip 192.168.2.250.

If this configuration is wrong in the KRC you can fix it by:

- Edit file C:\windows\vxwin.ini

```
[Boot]
Bootline=olPci(0,1)pc:vxworks h=192.0.1.2 b=192.0.1.1 e=192.168.2.250 u=target pw=vxworks
```

- Edit file C:\krc\roboter\ini\DIrrc.ini

```
[DLRRC]
TIMEOUT=25
IMMEDIATE_STARTUP=1
FRIHOST=192.168.2.113
FRISOCK=49938,0
FRIKEY=[use provided key)
```

We had troubles using an Intel Giga network controller. The 3Com one works correctly. To check that windows is properly detecting and using the FRI network controller, go to the windows "Device Manager" and check that the 3Com network controller is assigned as part of a "Realtime OS Devices" section and not inside "Network adapters" section.

Connecting your computer to the FRI network

The host computer (your computer or the computer running the FRI-yarp bridge) must have the following ip address: 192.168.2.113. This is configured in some .ini file in the KRC.

- Connect the host computer (your computer) to the same physical network as the KRC.
- Run the host computer (your computer):

```
sudo ifconfig eth0 up
sudo ifconfig eth0:0 192.168.2.113 netmask 255.255.255.0
```

- Test the connection to the robot:

```
ping 192.168.2.250
```

- It must responde with packages with low latency
- You can use the following bash script for configuring and testing FRI connectivity:

```
#!/bin/bash

if [ $1 = up ]
then
  echo "Disabling previous wired network configuration"
  sudo ifdown eth0
  sleep 2
  sudo ifdown eth0
  sleep 2
  echo "Turning wired network on"
  sudo ifconfig eth0 0.0.0.0 up
  echo "Kuka FRI network"
  sudo ifconfig eth0:0 192.168.2.113 netmask 255.255.255.0
  echo "Kuka KRC windows network"
  sudo ifconfig eth0:1 192.168.3.113 netmask 255.255.255.0
  echo "Wessling robotics Hand network"
  sudo ifconfig eth0:2 192.168.200.10 netmask 255.255.255.0
  echo "Checking communication with robot parts"
  ping -i 0.3 -w 3 -c 5 192.168.2.250
  if [ $? != 0 ]
  then
      echo "Comunication error with Kuka FRI, check FRI network"
      exit 1
  fi
  ping -i 0.3 -w 3 -c 5 192.168.3.10
  if [ $? != 0 ]
  then
      echo "Comunication error with Kuka KRC windows, check Kuka KRC windows network"
      exit 1
  fi
  ping -i 0.3 -w 3 -c 5 192.168.200.1
  if [ $? != 0 ]
  then
      echo "Comunication error with Wessling robotics Hand, check Hand network"
      exit 1
  fi

fi
if [ $1 = down ]
then
  echo "Turning all robot networks down"
  echo "Wessling robotics Hand network"
  sudo ifconfig eth0:2 down
  echo "Kuka KRC windows network"
  sudo ifconfig eth0:1 down
  echo "Kuka FRI network"
  sudo ifconfig eth0:0 down
  echo "Turning wired network off"
  sudo ifconfig eth0 down
fi
```

Connecting your computer with the KRC windows operating system

You can also connect to the same network of the KRC windows OS. This may be useful for editing or copying files:

- Configure your network interface to be also in the same network of this windows computer:

```
sudo ifconfig eth0:1 192.168.3.113 netmask 255.255.255.0
```

- Test the connection to the windows OS:

```
ping 192.168.3.10
```

Access the KRC windows files in your computer

- Create a mount directory for the KRC windows files:

```
mkdir -p /mnt/krc
```

- mount the windows files in a local host directory (password: user):

```
sudo mount -t cifs //192.168.3.10/krc /mnt/krc/ -o user=user,vers=1.0
```

Now you can access the KRC windows files in your linux computer.

Running the FRI-yarp bridge

- If you configured LWR_yarp_arcos with suid:

```
cd ~/local/src/fri_stanford/Linux/x64/release/bin
./LWR_yarp_arcos
```

- If you didn't use suid:

```
cd ~/local/src/fri_stanford/Linux/x64/release/bin
sudo su
cd
export HOME=/home/my_user
. /home/my_user/.bashrc
export HOME=/root/
cd -
./LWR_yarp_arcos
```

Running the FRI KRC KRL client controller code

- Turn on the KRC box (big black switch CW 90 degrees)
- In the Kuka Pendant:
  - Rotate the key on top of the Kuka Pendant to the spiral without dot position.
  - Select Position Control with LWR button
  - Press the Button next to the key (the one with a vertical bar)
  - Configure→Set tool/base→ Tool no: 1(empty) 2(hand) 3(L link), base no: 1
  - With the file manager, go to: R1/Program/FRIDemo/
  - Select file FRIControl (this file is in the KRL/ARCOS-Lab_KRC1/FRIDemo/ repository directory. You must internally check for the gravity vector and the tool number to be correct)
  - Select Position Control with LWR button
  - Monitor→Variable→Single→Name: Storque_tcp_est (check that torques and forces don't exceed a value of 2)
  - Close Monitor window
  - Press green + button several times until no more advancing happens in the code
  - Press enter in the LWR_yarp_arcos console
  - Press green + button more times until no more advancing happens in the pendant code

Restarting in case of bad communication quality

- Acklnowled all messages in Pendant
- Press black button with circle to denergize the drives
- Press the white button with vertical bar next to the key to reenergize the drives
- Press the green button with "+" several times until FRI switches to joint impedance control. During this the arm moves to the starting position and then to the last commanded position. Beware!!!!

Starting the vfclik system for the real robot (AAMAS 2019 experiments). Only runs on the right arm/hand

- Remember to start the FRI yarp bridge first (look above)
- Remember to check to following repositories to the tag "robio_2019_right_arm_hand_forces"
- gitlab:

```
arcospyu avispy cmoc kbd-cart-cmd orocos-kdl pyrovito robot_descriptions vfclik vfl
```

- git.arcoslab.org:

```
fri-stanford hand-code
```

- For yarp use the following git commit:

```
ced5964df3e730fe5d578e044277f69665b95a1f
```

Running the ARM software

- VFCLIK:

```
cd ~/local/src
vfclik -n /arcosbot-real -r lwr -i right -d robot_descriptions/arcosbot/kinematics/lwr/
```

- Visualization (pyrovito):

```
cd ~/local/src
pyrovito -n /arcosbot-real -r lwr --arm_right --hand_right -a robot_descriptions/arcosbot/kinematics/lwr/ -d robot_descriptions/arcosbot/kinematics/sahand/
```

- Joint limits visualization:

```
cd ~/local/src
bar_vis
yarp connect /arcosbot-real/lwr/right/debug/qdist /bar_vis/bar/in
```

- Keyboard control:

```
run_right.sh "/arcosbot-real"
```

- To control nullspace movement:

```
yarp write ...  /arcosbot-real/lwr/right/nullspace/control
```

In this console you can write nullspace speed movements. Try small numbers like 0.1 first.

Running the HAND acquisition and force estimation

- Start the ardnet system, and the ardnet/yarp bridge (hand_yarp):

```
cd ~/local/src/hand-code/hand_yarp
./python/wessling_hand/start_ard.py
```

- Disable and enable the hand fingers (using the hand emergency stop button)
- Start the real/sim hand bridge:

```
cd ~/local/src
sahand_yarp_sim -b /arcosbot-real -r -d -n -f robot_descriptions/arcosbot/kinematics/sahand/hands_kin.py
```

- Torque fingertip estimation:

```
cd ~/local/src
torque_sim -b /arcosbot-real -r -a robot_descriptions/arcosbot/kinematics/lwr/ -c robot_descriptions/arcosbot/kinematics/sahand/calibration_data/finger_calibration_data.py -f robot_descriptions/arcosbot/kinematics/sahand/hands_kin.py
```

- Now you should see yellow arrows representing the forces on the fingertips.

Hand torque sensor calibration

- If you are running a previous sahand_yarp_sim and torque_sim please stop them.
- Check that your finger_calibration_data.py file is linking to finger_calibration_data_unit.py file
- Start sahand_yarp_sim and torque_sim again (but with no calibration data (unit file above):

```
cd ~/local/src
sahand_yarp_sim -b /arcosbot-real -r -d -n -f robot_descriptions/arcosbot/kinematics/sahand/hands_kin.py
```

```
cd ~/local/src
torque_sim -b /arcosbot-real -r -a robot_descriptions/arcosbot/kinematics/lwr/ -c robot_descriptions/arcosbot/kinematics/sahand/calibration_data/finger_calibration_data.py -f robot_descriptions/arcosbot/kinematics/sahand/hands_kin.py
```

- Adjust the desired fingers to calibrate in the finger variable in hand_calibration.py file
- Prepare a small "weight object" for calibration purposes with a previously measured weight. We recommend using a small bottle filled with water. Remember that the fingers can't lift something too heavy. Use 200grams.
- Run the hand_calibration.py script:

```
cd ~/local/src/hand-code/hand_yarp/python/wessling_hand
./hand_calibration.py -b /arcosbot-real -c ~/local/src/robot_descriptions/arcosbot/kinematics/sahand/calibration_data/finger_calibration_data.py -f ~/local/src/robot_descriptions/arcosbot/kinematics/sahand/hands_kin.py -a ~/local/src/robot_descriptions/arcosbot/kinematics/lwr/
```

- Follow the instructions of the script. Be sure to disconnect or "lift" the finger connected weight when the script is zeroing sensor's torques. Be sure to connect, to the corresponding finger, the bottle when indicated.
- When the script is "looping", ctrl-c it !!! (Stop it)
- Annotate the torque and angle calibration values.
- Create a new finger_calibration_data_something.py by copying the finger_calibration_data_unit.py file.
- Modify the new finger_calibration_data_something.py file with the new annotated torque and angle calibration values.
- Link finger_calibration_data.py to the new file.
- Run again sahand_yarp_sim and torque_sim. This will apply the new calibration values to the estimated forces.
- Remember to flex the finger joints to a prudent value to avoid a singularity and get better force estimations.
- Remember to zero the finger torques every time you reorient the hand. The finger impedance control doesn't have gravity weight compensation, therefore, the finger weight affects the measurement greatly.
- You can run hand_calibration.py again until the fingers are flexed and the torques are zeroed to test the new calibration values.

Camera calibration

The usual steps for setting up a system with a camera with marker detection consists:

- Install ROS modules (camera module, rectification module, marker detection module)
- Calibrate the camera intrinsics
- Measure the arm_base-to-camera_base transform. Since this is done totally manually, this is usually pretty wrong.
- Run a arm_base-to-camera_base calibration to obtain the corrected arm_base-to-camera_base transform.
- Set a launch file for running the camera module, together with a rectification module, together with an static transform representing the arm_base-to-camera_base transform.
- Use a ros to yarp bridge to send the marker poses to yarp.

Install ROS1 modules

- We use a realsense sensor. Our sensor has a full-HD RGB camera which we are using.
- We detected a problem with the brightness adjustment of the camera: the brightness oscillates erratically. We deal with this problem by stopping the ROS realsense module, running the realsense-viewer, deactivating all the auto-exposure and auto-brightness features, stopping the realsense-viewer and then running again the ROS realsense module.
- We have a launch file for our realsense camera in the following repository:

```
git clone git@git.arcoslab.org:humanoid-software/oms-cylinder.git
```

- You can run this configuration with:

```
roslaunch oms_launcher rs_camera.launch
```

- Check this launch file for relevant camera settings
- We use ROS1 with a module "ar_pose" to detect object's position and orientation. Follow typical ROS1 procedures to install this module (works in melodic and Ubuntu Bionic)

Calibrate camera intrinsics

- The purpose of this step is to rectify the image coming from the camera. Usually there is some degree of image distortion with any lenses. This is possible to counteract with image_proc ROS module.
- You can use the camera calibration toolset from ROS1:

http://wiki.ros.org/camera_calibration [http://wiki.ros.org/camera_calibration]

- After finding a way to deactivate the realsense module intrinsic internal rectification we ran a full camera calibration and found out that the included realsense calibration is quite good. This step is not necessary with this camera.
- If you have a camera that needs this calibration, you will need to configure the camera intrinsics in the camera module somehow (some modules can't do this, beware!). Then you will need to connect the camera output to image_proc. This will create an /image_rect topic with the rectified image.

Run a arm_base-to-camera_base calibration

- For this type of calibration we used the ROS1 module: robot_cal_tools

```
https://github.com/Jmeyer1292/robot_cal_tools
```

- This module is robot independent and is not too complicated to use
- In our oms-cylinder ROS module take a look at the static_calibration.launch and its internal configuration files
- These are the main steps to follow to execute a camera_base calibration:
  - Mount rigidly a the calibration panel to the arms end-effector
  - Mount rigidly the camera to the robot (this can't change after calibration!)
  - Write and run a script that moves the end-effector in the picture space. Be sure to navigate to very disperse 6D areas of the picture space. Try strong rotations but still camera visible. Try this rotations if far away x,y,z points. Also try the same in some center areas of the picture. It is possible to calibration the camera_base with positions. This script has to move the end-effector to these positions and then capture an image in each of them. Be sure to stop the arm completely, if not, the image can be blurred and the calibration table can be incorrectly detected. You need to store the corresponding image file and a yaml file that points to the corresponding image. This yaml file includes the end-effector position.
  - With the images and end-effector positions now you can run the robot_cal_tools calibrator. If it converges it will give an estimated camera_base transform. You will need this transform for the next main step.

Set a launch file for running the camera module, together with a rectification module, together with an static transform representing the arm_base-to-camera_base transform.

- In oms-cylinder repository use the following two launch files to incorporate the camera_to_arm_base transform found in the previous step:

```
oms_launcher_marker90.launch
ar_oms_cylinder_marker90.launch
```

- You can use this launch file to run the system once the calibration is satisfactory

Ros to yarp bridge

- The ROS oms-cylinder module has a ros_to_yarp script to send the markers poses to yarp. Find it in:

```
cd ar_pose_marker_yarp_bridge/scripts
./ros_to_yarp_box_pose.py -w ~/local/src/robot_descriptions/arcosbot/perception/webcam_calibration_values.py -o ~/local/src/cmoc/objects/sliding/objects/object_params.py
```

- The webcam_calibration_values.py file is for now ignored. You will need to put a camera_pose homomatrix array (identity) though. This file was used in case you didn't use a static_transform in the camera launch modules for the camera_base transform.
- The object_params.py file contains self.markers_transformations dictionary. There you have to configure an static transformation for you object of interest (a box has the marker attached on one side, you can add a static transform to set the markers yarp data to that position).
- This yarp module will publish marker data in the /arcosbot-real/marker/object_pos:o port
- You can use the -f option to do a "finger_testing" run. This will disable the objects transforms and it will give a pure marker position. This will be useful for the the next tutorial part.

Finger tip pushing calibration

Once you followed the #hand_torque_sensor_calibration, you will have one or more fingers for exerting forces against objects. Now we will assume that you will use one finger to push an object. Once you selected the particular finger you will need to find a particular hand orientation to push the object to avoid table crashes or other parts of the hand to crash against the object itself. A trick to get this easier is to glue a marker to the fingertip of the selected pushing finger. Then orient the hand (using run_right.sh "/arcosbot-real") in the desired pushing orientation. Command the fingers to a finger pushing configuration. Glue the marker to the finger tip. Adjust the marker such that it is vertical in orientation (to match the markers that are glued to other objects). Get the current marker pose (To_m), get the current finger global pose (To_f), calculate the fingertip-to-marker homo transform (Tt_m=((To_t)^-1)*To_m). Use this transform in the robot_descriptions/arcosbot/perception/webcam_calibration_values.py file with the rel_pose_marker_finger variable.

Step-by-step instructions:

- Run the ros_to_yarp marker bridge without any object transform:

```
cd ar_pose_marker_yarp_bridge/scripts
./ros_to_yarp_box_pose.py -w ~/local/src/robot_descriptions/arcosbot/perception/webcam_calibration_values.py -o ~/local/src/cmoc/objects/sliding/objects/object_params.py -f
```

- This will give the pure marker pose with no extra object transforms.
- Edit the webcam_calibration_values.py file. Set rel_pose_marker_finger transform to an identity homomatrix.
- Set the finger_pushing_pos joint angles in the exploration.py file to select your desired finger pushing configuration.
- Glue a marker to an object of interest.
- Position the object on a reachable pose on top of a table.
- Run exploration.py

```
cd local/src/cmoc/objects/sliding/scripts/
./exploration.py -n /arcosbot-real -c ~/local/src/robot_descriptions/arcosbot/kinematics/sahand/calibration_data/finger_calibration_data.py -o ~/local/src/cmoc/objects/sliding/objects/object_params.py -f ~/local/src/robot_descriptions/arcosbot/kinematics/sahand/hands_kin.py -w ~/local/src/robot_descriptions/arcosbot/perception/webcam_calibration_values.py
```

- Follow the program instructions until this appears:

```
Getting initial object position
Box global pose [[ 0.99983308  0.00634678 -0.04358436  0.63681302]
 [-0.00841706  0.99883239 -0.04757109  0.25734383]
 [ 0.04315164  0.04789128  0.99792002  0.85763204]
 [ 0.          0.          0.          1.        ]]
If right, press "y"
```

- If the position looks file press y and enter. The robot should move the finger behind the object with the fingertip aligned to the marker of the object (more or less).
- Remove the object. (hide this marker)
- Glue a marker to the finger tip.
- Ctrl-C (cancel) exploration.py
- Using run_right.sh /arcosbot-real rotate the end-effector until the pushing orientation is found. Adjust the glued marker such that the orientation of the marker is the some as the orientation that the object previously had.
- Get the finger_tip pose and the marker pose:
  - In one console run (To_m):

```
yarp read ... /arcosbot-real/marker/object_pos:o
```

- In another console run (To_f):

```
yarp read ... /arcosbot-real/lwr/right/vectorField/pose
```

- Anotate both homomatrix. Calculate Tf_m =((To_f)^-1)*To_m
- Use Tf_m in webcam_calibration_values.py file with the rel_pose_marker_finger variable.
- Run exploration.py again. Check that the pushing orientation is the desired one.

Exploring for the object parameters

./exploration.py -n /arcosbot-real -c ~/local/src/robot_descriptions/arcosbot/kinematics/sahand/calibration_data/finger_calibration_data.py -o ~/local/src/cmoc/objects/sliding/objects/object_params.py -f ~/local/src/robot_descriptions/arcosbot/kinematics/sahand/hands_kin.py -w ~/local/src/robot_descriptions/arcosbot/perception/webcam_calibration_values.py

Running in simulation the same conf as in real life

This part of the tutorial is useful when you want to run the simulator using the same hardware and configuration that is available right now in "real life". Make sure to use the correct tag as stated in the **Starting the vfclik system for the real robot (ROBIO 2019 experiments). Only runs on the right arm/hand** section. Each line of the following code block should be run in a separate terminal at `~/local/src/`.

```
vfclik -s -n /arcosbot-real -i lwr -i right -d robot_descriptions/arcosbot/kinematics/lwr/
pyrovito -n /arcosbot-real -r lwr --arm_right --hand_right -a robot_descriptions/arcosbot/kinematics/lwr/ -d robot_descriptions/arcosbot/kinematics/sahand/
sahand_yarp_sim -b /arcosbot-real -r -d -n -f robot_descriptions/arcosbot/kinematics/sahand/hands_kin.py -s
torque_sim -b /arcosbot-real -r -a robot_descriptions/arcosbot/kinematics/lwr/ -c robot_descriptions/arcosbot/kinematics/sahand/calibration_data/finger_calibration_data.py -f robot_descriptions/arcosbot/kinematics/sahand/hands_kin.py -s
run_right.sh "/arcosbot-real"
```

Sending commands to the Arcosbot server

If the arcosbot server is running the system for using the robot, then it is possible to just send commands to the server using YARP.

- Make sure that you are in the same physical network as the humanoid robot
- Detect arcosbot's yarp server

```
yarp detect --write
```

Now YARP will use the remote server. You can send basic commands to the robot using kbd commands.

```
run_right.sh "/arcosbot-real"
```

Common Problems

If you encounter an X server error while executing the simulator, check the following:

- If DISPLAY environment variable of the chroot is the same as the host machine
- If the graphics driver of the chroot and the host machine are the same
- If the host Xserver allows indirect rendering

If you encounter errors with **multiprocessing**. Add the following line to /etc/fstab

```
none /dev/shm tmpfs rw,nosuid,nodev,noexec 0 0
```

And then execute:

```
sudo mount /dev/shm
```

Running the Dual Capability Map System

* Open a terminal and execute Yarp:

```
yarpserver start
```

* In another console execute vfclik for right arm:

```
cd ~/local/src/
vfclik -i lwr -i right -d robot_descriptions/arcosbot/kinematics/lwr/ -s
```

* In another console execute vfclik for left arm:

```
cd ~/local/src/
vfclik -i lwr -i left -d robot_descriptions/arcosbot/kinematics/lwr/ -s
```

* In another console execute hands simulator:

```
cd ~/local/src/
sahand_yarp_sim -s -d -n -f robot_descriptions/arcosbot/kinematics/sahand/hands_kin.py
```

* In another console execute roboviewer visualizator:

```
cd ~/local/src/
pyrovito -r lwr --arm_right --arm_left --hand_right --hand_left -a robot_descriptions/arcosbot/kinematics/lwr/ -d robot_descriptions/arcosbot/kinematics/sahand/
```

Yarp port descriptions

Module: Bridge

weights port

- Name: /lwr/right/bridge/weights
- Description: This is used to select which controller controls the robot, one can select between vfclik, nullscace, jointcontroller, mechanism, xtra1, xtra2
- Usage example: 1.0 1.0 0.0 0.0 0.0 0.0 0.0 Selects vfclik and nullspace controllers
- Usage example: 0.0 0.0 1.0 0.0 0.0 0.0 0.0 Selects joint controller

Module: vectorField

weight port

- Name: /lwr/right/vectorField/weight
- Description: This is used to adjust the importance of a joint or task space dimension during the jacobian calculation for cartesian movement.
- Usage example: j 1 1 1 1 1 1 1 Uses all 7 joints as much as possible
- Usage example: t 1 1 1 1 1 1 Uses all 7 task space dimensions as equally important

Module: Object feeder

object port

- Name: /lwr/right/ofeeder/object
- Description: This port is used to feed objects to the vfclik system. It can accept obstacles and goals.
- Usage example: set goal (-0.995363 0.001243 0.096186 0.979951 0.062618 0.767426 0.638073 -0.348856 -0.073022 0.641137 -0.763945 0.861143 0 0 0 1)

Module: joint controller

reference port

- Name: /lwr/left/jpctrl/ref
- Description: This port accepts joint positions.
- Usage example: -0.4 0.3 0.1 0.5 0.5 0.5 0.5

Module: Distance monitor

Distance out port

- Name: /lwr/right/dmonitor/distOut
- Description: This port outputs the distance to current goals/obstacles
- example: (object/goal number, linear distance (meters), angular distance (degrees)

Module: Debug module

Joint Distance port

- Name: /lwr/right/debug/qdist
- Description: This port outputs distances in percentage to joint limits

- Usage example: 19.6106992983374 60.6037843402149 14.0171939079065 73.3064145340412 31.6837274879154 -53.8251156968477 59.6518223463255

Module: Nullspace

Control port

- Name: /lwr/right/nullspace/control
- Description: This port accepts a list of nullspace speeds for each nullspace variable
- Usage example: 0.0 : No movement in nullspace, 0.5 : Positive movement in nullspace, 0.1 0.3: Two nullspaces used

Changing robot base position/orientation

Changes in KRC

- Physically relocate the robot in the new designated place.
- Annotate or find out the new orientation information. (Translation/Rotation in X, Y and Z) (remember to annotate the order of this rotations)
- Gravitation vector points away from the earth
- Check, with cartesian jogging, the direction of the robot base axis. This will help to correctly project the gravitation vector onto those axis.
- Update the gravitation vector in file

```
c:\krc\roboter\krc\steu\mada\$custom.dat
```

- Restart and check \$torque_tcp_est variable. No value should be bigger than 3 Newtons
- Consider changing the hand mounting orientation.
- If hand mounting orientation changed, update hand mass data:
- In the KRC, change to administrator privileges.
- Setup→Tool→Payload data→Tool no: 2 (hand tool), Continue
- Adjust corresponding values (mass, x, y, z)
- Restart and check \$torque_tcp_est variable. No value should be bigger than 3 Newtons
- Move the robot in joint space to an appropriate initial pose. (away from table, away from joint limits, easy for table reaching)
- Open \R1\Program\FRIDemo\FRIControl.src file.
- With the cursor move to line before “PTP P6” line
- Select: Commands→Motion→PTP
- Select the correct base and hand tool frames.
- This will add the new ARM starting position.
- Erase the previous “PTP P6” line: Put cursor on this line, then: Program→Delete
- Restart KRC
- Start the FRIControl file as you normally do. Check that the initial position is as desired

Changes in Linux software

Review:

- Change fri interface last two joint limits (in case hand mounting position changed)
- Change VFClik initial joint and frame positions
- Change VFClik last two joint limits (in case hand mounting position changed)
- Update arm kinematic chain (arm base and hand base)

VFClik initial joint and frame positions

- Edit file:

```
robot_descriptions/arcosbot/kinematics/lwr/config-lwr-right.py
```

- Adjust initial_joint_pos (use the ones from the fri KRC initial joint positions)
-