TUM Institute for Cognitive Systems (ICS)
Multi-Sensory Based Robot Dynamic Manipulation

Session 10: Hands on Robot UR-10 & Robot Skin

Dr. Emmanuel Dean
Florian Bergner
Ana Martin

January 10, 2019

1 Safety Instructions

Before operating the robot, you ought to read the user manual of the robot (*user_manual_en_UR10_Global.pdf*)[1], provided in the template material of this tutorial, with special emphasis on Chapter 3 *"Safety"*.

Besides the instructions specified in the user manual, the following rules need to be follow in order to operate the robot in the Lab.

1. Do not operate the robot **ALONE**.
2. **ALWAYS** keep the robot's workspace clear and do not **INVADE** it specially during *Initialization*.
3. During operation, there should be **ALWAYS** one person holding the *teach-pendant*, see Fig. 3.1 c). This person should keep the eyes on the robot and one hand on the *Emergency Button*, see Fig. 2.2 a). Please, don't be distracted during robot operation!!!
4. **ALWAYS** move the robot to HOME position before shutting it down. This is to avoid permanent damage on the mechanical brakes.
5. **TEST** everything in the simulation **BEFORE** trying on the real robot.
6. **NEVER** leave the robot **ON/ACTIVE** unattended. Even if you need to leave the lab for a few seconds, move the robot to *Home* position and turn it off (see Fig. 5.1).
7. Keep the **DOOR** of the lab always **CLOSED** when the robot is in operation.
8. If you have any **DOUBTS**, **ASK** your **ADVISOR** before doing anything.

The access to the robot will be revoked to the person who doesn't follow these instructions.

2 Starting The Robot

These are the steps needed to start the robot.

1. Take the robot's *teach-pendant* and push the start button, see Fig.2.1 a). The button will be "on" and the system will start to boot (see Fig.). Please make sure that you are not inside the robot's workspace before turning on the robot and during the initialization phase.

2. Wait until the robot finish the booting process and the Main screen is shown in the *teach-pendant*, see Fig.2.2 b).

3. If the robot was properly shutdown, then the main screen will show a message "Emergency Stopped". Release the emergency button (red button see Fig. 2.2) by pulling it.

4. After releasing the emergency button, the robot will request to start the initialization. Press "OK" and the Initialization will be shown in the *teach-pendant* (see Fig. 2.3).

5. The initialization process is as follows:
   - a) First power on the motor drives. Press on the button "On" (see Fig.2.3 a)).
   - b) Then, start the controllers by pressing the "Start" button (see Fig.2.3 b)).
   - c) Initialize the joints. Keep pressing the "Auto" button until all the joint indicators turn green (see Fig.2.3 c)).
   - d) Finally, press the "Ok" button to exit the Initialization screen. This will take you back to the main screen.

6. In the main screen press "PROGRAM Robot" button (see Fig.2.4 a)). This will open a new screen to create a new program for the robot. In this screen, you can easily create a script to move the robot, e.g. "pick and place" motion, read the user manual [1] for more information. In our case, we are not going to use this form of robot programming, instead we are going to command the robot directly from the workstation, see Section 4 for more details on this.

7. We will only use this "PROGRAM Robot" mode to move the robot to its *Home* position. To this end, press the tab "Move" (see Fig.2.4 b)). This will open the Move screen.

8. In this screen, press the "Home" button (see Fig.2.5 a)). A new screen will be launched (see Fig.2.5 b)). To move the robot to *Home* keep pressing the button "Auto" until the button in the bottom right corner of the screen changes from "Cancel" to "OK".

9. When we externally control the robot using a workstation, we upload a script into the robot controller. The output information of this script is printed in the "Log" screen. To move to this screen, you just need to press on the tab "Log" (see Fig.2.6). In order to see more easily this information, is a good idea to clear the screen using the "Clear" button. Now you are ready to control the robot using an external workstation.

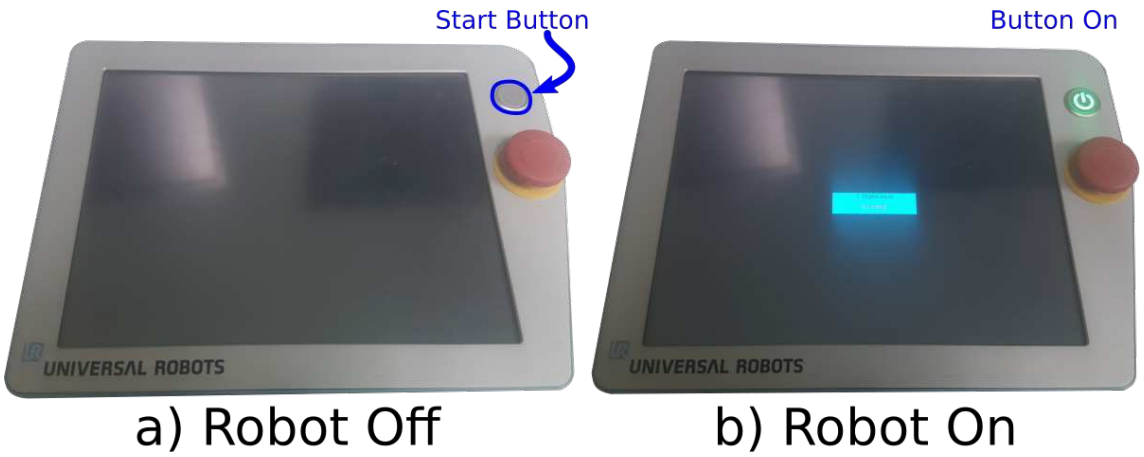


Figure 2.1: Turning the robot on.

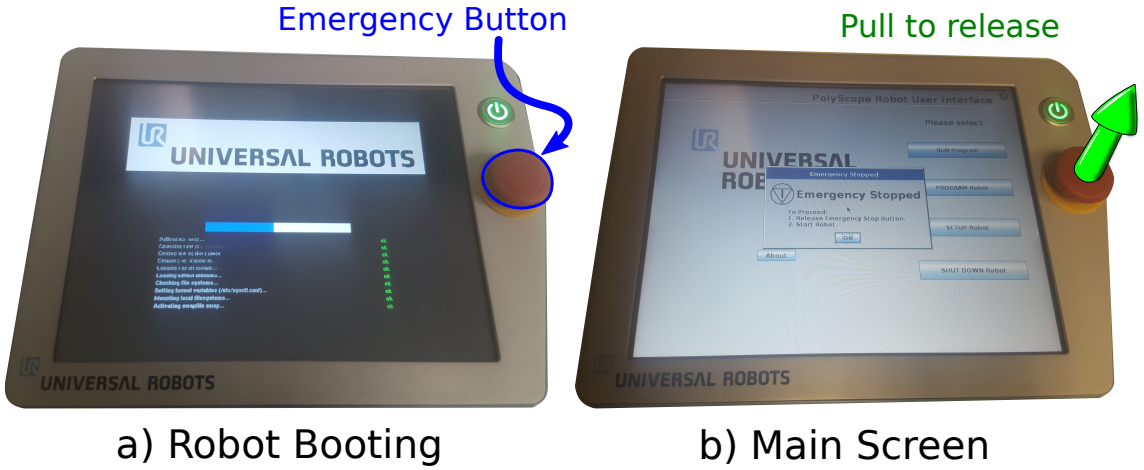


Figure 2.2: Main Screen

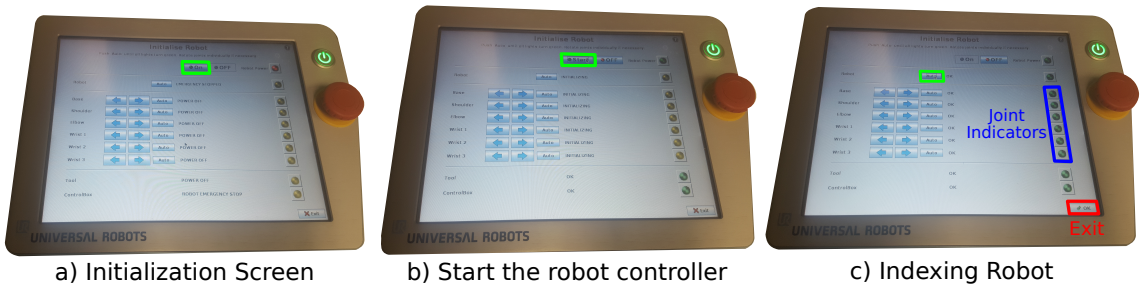
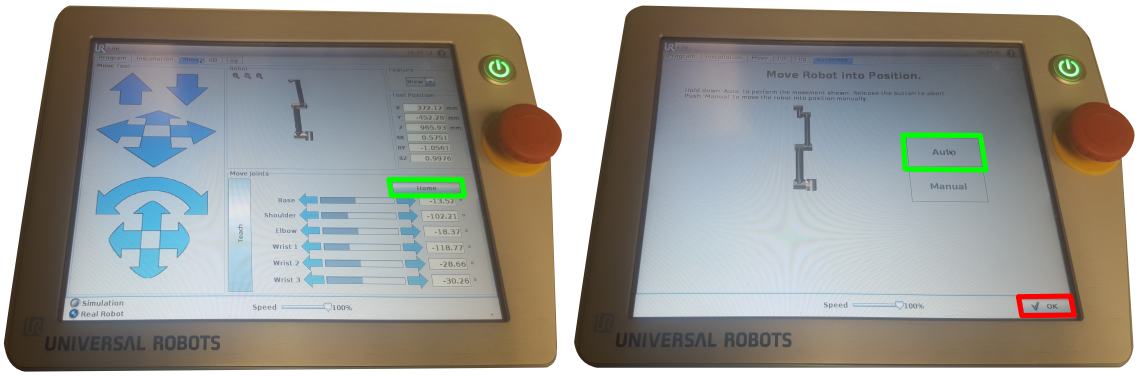


Figure 2.3: Initialization Screen.

a) Main Screen b) Robot Programm Screen

Figure 2.4: Robot Program Screen.



a) Move Screen b) Move robot to Home Position

Figure 2.5: Move Robot Screen.

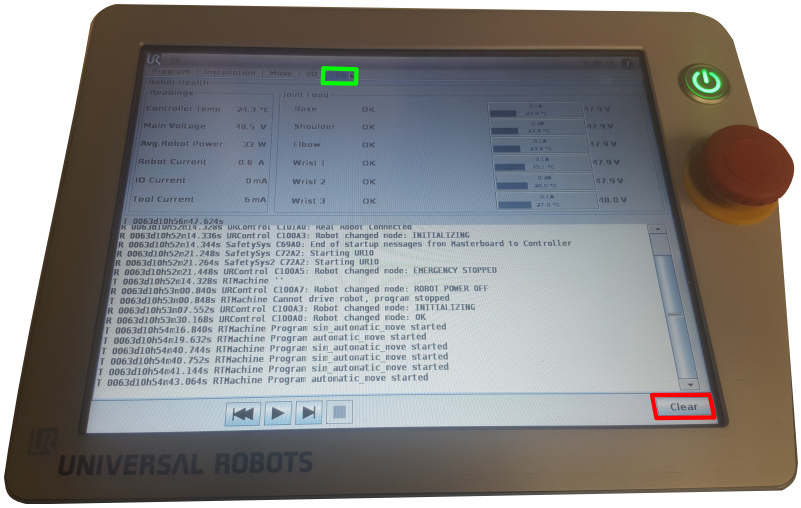
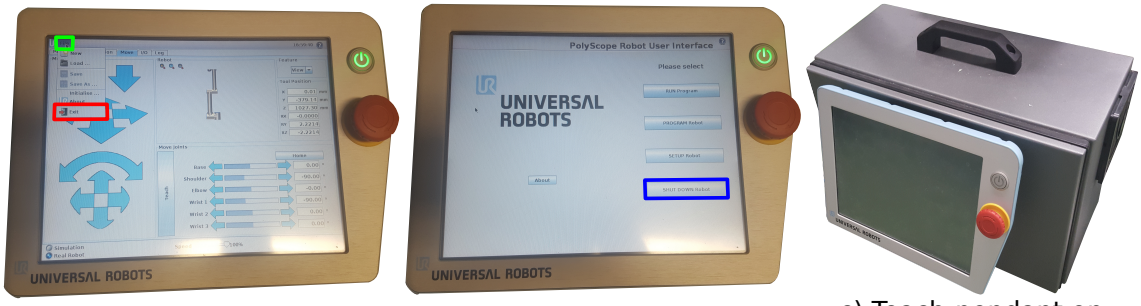


Figure 2.6: Log Screen.



a) File->Exit b) Shutdown Robot c) Teach-pendant on the control unit

Figure 3.1: Turning the robot off.

3 Shutting Down The Robot

The following sequence describes the shutting down process of the robot.

1. First make sure that the robot is not moving and any script is running in the background.
2. Go to the "Move" tab and send the robot back to *Home* (steps 7 and 8 from Sec. 2).
3. Next, press the "File" button at the top left corner of the screen (see Fig.3.1 a)) and press "Exit". This will take you to the Main screen, where you can press the "SHUT DOWN" button (see Fig.3.1 b)).
4. Wait until the screen is completely off and the "On" button in the teach-pendant is off.
5. Put the teach-pendant back to its holder (see Fig.3.1 c)).

4 ROS Robot Control Interface

You will need 4 terminals: a) ROS core, b) Rviz visualization, c) Control test node and d) Script Manager node. Remember to always source your workspace before running the ros commands, i.e. `$source devel/setup.bash`

In the **Terminal 1** run the ros core:

```
$rosclean purge -y
$roscore
```

In the **Terminal 2** launch the UR10 visualization:

```
$roslaunch tum_ics_ur10_description bringUR10.launch
```

4.1 Simulation Mode

First verify that the configuration file is in simulation mode. To this end, open the **Terminal 3** and launch the ros editor:

```
$cd /home/<user>/ros/workspaces/ur10_tutorial_ws/
$source devel/setup.bash
$rosed tum_ics_ur10_controller_tutorial configUR10_FD.ini
```

This will open the default ros editor. Please verify the variable **ROBOT_TYPE** in the configuration file. It should be defined as:
`ROBOT_TYPE=sim;`

In the same Terminal run the controller test in simulation mode:
`$roslaunch tum_ics_ur10_controller_tutorial testSimpleEffortCtrl.launch`

Don't worry if you get the following message on the screen:

`"sched\_setscheduler: Operation not permitted mlockall failed: Cannot allocate memory"`

It is basically a super-user access rights, but the robot is still operational.

In order to change the parameters of the controller, e.g. gains or joint position goal, edit the following yaml file.
`$rosed tum_ics_ur10_controller_tutorial simpleEffortCtrl.yaml`

For example, you can change the goal by modifing the line:
`goal: [0, -90, 0, -90, 0, 0, 4.0]` or `goal: [45, -45, 45, -45, 45, 45, 5.0]`

The first 6 values of this parameter are the joint desired positions, and the last value is the total time to reach the goal.

Please verify that your controller works perfectly in simulation before running the real robot. Also, verify in simulation that the trajectory and workspace are correct (e.g. collision-free, singular-free, etc.) before commanding the real robot.

IMPORTANT!!: DO NOT MODIFY ANY VARIABLES IN THE YAML FILE "**pidInt.yaml**". THIS IS FOR INTERNAL USE ONLY.

4.2 Real Robot Mode

IMPORTANT!!!: From this moment on, you should keep the robot always on sight and your hand over the emergency button (red button in the teach-pendant).

NOTE: Please verify that the network is connected to the static connection **RobotNet**. In order to verify the correct connection with the robot, you can try to ping the robot
`$ping 192.168.1.10`

Verify again the configuration file. In this case, it should be in *real mode*. Open the **Terminal 3** and edit the configuration file:
```
$cd /home/user/ros/workspaces/ur10_tutorial_ws/
$source devel/setup.bash
$rosed tum_ics_ur10_controller_tutorial configUR10_FD.ini
```
This will open the default editor. Please verify the value of the variable **ROBOT_TYPE**. It should be defined as:
`ROBOT_TYPE=real;`

In **Terminal 4**, Launch the robot script manager. (**Where is your hand?**)
`$roslaunch tum_ics_ur_robot_manager robot_script_manager_ur10.launch`

You should get the following message:

```
ScriptLoader(): trying to connect to server
ScriptLoader(): Client connected on address 192.168.1.3:51687
ScriptLoader(): Client finished. Succeeded in sending the script code.
ScriptManager::waitForSocketConnection: listening on port: 50001
ScriptManager::buildSocketConnection: Got a TCP connection on port: 50001
```

If you don't see this message check the connection with the static network, see above. If everything works, then you should see in the Log tab (teach-pendant) the message: *RTMachine socket_read_binary_integer: timeout*. This is perfectly normal, it means that the script is waiting for the commands from the state machine in the robot driver.

Now, you are ready to run the real robot:

Where is your hand? and your eyes? is the robot workspace clear?

In the **Terminal 3** run again the control test:

```
$roslaunch tum_ics_ur10_controller_tutorial testSimpleEffortCtrl.launch
```

When the robot is not moving anymore, stop the controller node in the **Terminal 3**

```
$ctrl+c
```

When you are finished with the robot (you don't want to test more controllers), you need to stop the script manager. Go to **Terminal 4** and type

```
$q
```

. *q* is a command to request a *clean* exit from the script. Only use *Ctrl+C* in case that the node becomes unresponsive.

You should see a message "*Clean exit*" in both the teach-pendant log and the **Terminal 4**. If you don't see the "Clean exit" in the teach-pendant, follow the next steps:

1. Push the emergency button (red button) in the teach-pendant. This action will terminate any script running in the robot control unit (in the background).
2. Release the emergency button (pull the red button).
3. Move the robot back to *Home*.
4. Shutdown the robot.
5. Re-start robot.

5 Gripper Control

The robot has a Lacquey gripper as the end-effector. This gripper has three states: a) **Open**, the fingers are activated in an open position, b) **Closed**, the fingers are activated in a closed position, and c) **Free**, the fingers are de-activated, which means they can be manipulated.

To command this states, you need to have the robot_script_manager node running, see Section 4.2 (Terminal 4).

Then, you need to run the gripper control server.

```
$roslaunch tum_ics_ur10_controller_tutorial lacqueyGripperServerUR10.launch
```

This node will provide a service to get or change the current gripper state, for example:

```
$rosservice call /getGripperState
$rosservice call /setGripperState ‘‘newState:  ‘open’ ’’
$rosservice call /setGripperState ‘‘newState:  ‘close’ ’’
$rosservice call /setGripperState ‘‘newState:  ‘free’ ’’
```

When you are finished DO NOT FORGET TO SHUTDOWN THE WORKSTATION (PC) and THE HEATERS in the Lab!!! In the Fig. 5.1, you can see how the robot should look like before you leave the lab.

Figure 5.1: Robot in *Home* position and teach-pendant in its holder.

6 Robot Skin

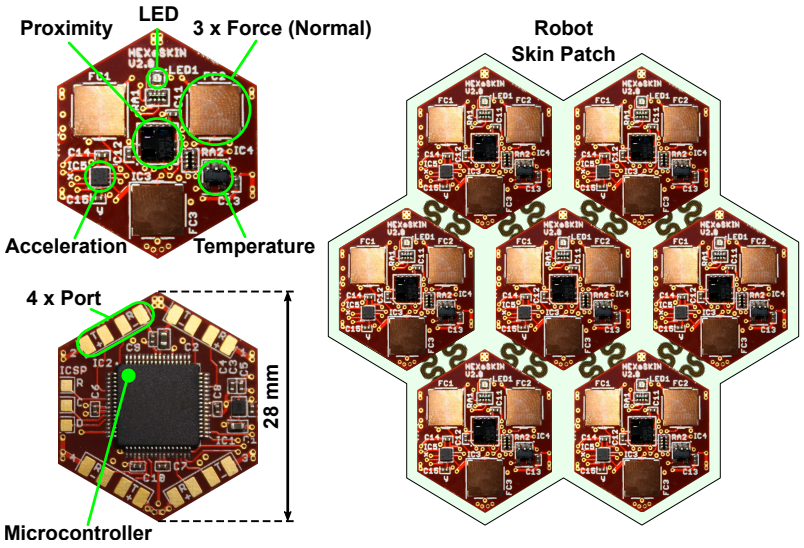


Figure 6.1: Robot Skin Cells

6.1 Robot Skin Cells

The smallest elements of our robot skin system are the hexagonally shaped skin cells (see Fig. 6.1). These skin cells integrate the sensors our robot skin system uses to perceive tactile stimuli from the environment. Each skin cell employs the same set of sensors, namely a *3D acceleration sensor*, to sense vibrations and net-linear accelerations, *three capacitive force sensors*, to measure normal contact forces, an optical *proximity sensor* to measure the distance to close objects, and a *temperature sensor*, to measure thermal properties of contacts. Thus, each skin cell provides *multi-modal contact information*, and, in the case of the proximity sensor, pre-contact information. All skin cells employ microcontrollers, mounted on their back-side, which provide local processing capabilities, allowing the skin cells to acquire and filter sensory information locally before sending it to higher processing layers. A group of directly connected skin cells forms a *Skin Patch*, see Fig. 6.1. Our robot skin system can combine net-linear acceleration information with neighbor and shape information to automatically reconstruct the 3D surface a skin patch covers. This greatly facilitates the process of acquiring the skin cells' poses with respect to a robot coordinate frame.

To utilize the robot skin, we provide a ROS interface and API. This interface offers intuitive tools to calibrate the skin patches.

7 ROS Robot Skin Interface

7.1 ROS robot skin driver

First you need to connect the skin patch to the PC. To this end, plug the USB-interface (FTDI) to the computer. Then, you need to run the robot skin ROS driver in **Terminal 1**

```
$roslaunch tum_ics_skin_driver_events skin_driver_ftdi.launch FTDI_SERIAL:=FTYZ2YG4
```

NOTE: If you don't have the serial number, then run the file without the serial number, the node will will fail, but in the screen you will find the correct information of the FTDI, e.g.

```
FTDI device info:
node[0]:
 opened = 0
 high speed device = 1
 type = 232H
 locId = 264
 serialNumber = FTYZ2YG4 « This is the correct serial number!!!
 description = SkinCellAdapter
 ftHandle = 0x0
```

7.2 Robot visualization

If not already running, launch the bringup launch file of the robot and visualize it in rviz:

```
$roslaunch tum_ics_ur10_description bringUR10.launch
```

7.3 Robot skin calibration

To calibrate the robot skin, you need to create a skin config file, which basically contains the structural information of the skin patch. In order to create this config file run:

```
$roslaunch tum_ics_skin_tutorials full_config.launch
```

This will open a GUI divided in 3 different steps (GUI Blocks)

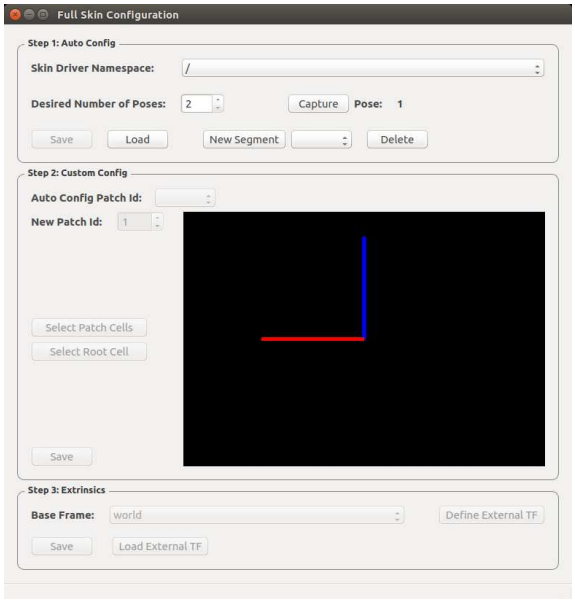


Figure 7.1: Step 1.1

7.3.1 GUI Block Step 1:

1. If you don't need a namespace use "/", see Fig. 7.1. For simple patches 2 poses is ok as default.

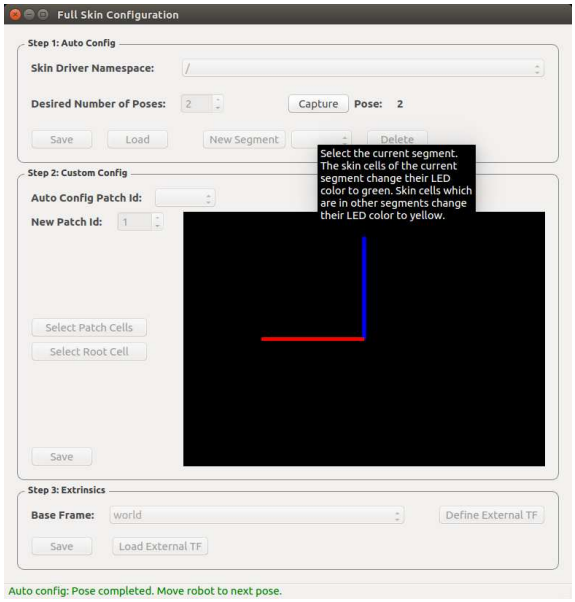


Figure 7.2: Step 1.2

2. Put the skin in the first pose, and press “Capture”, see Fig. 7.2.

3. Do this process as many times as indicated in “Desired Number of Poses”, until you see *Pose: finished*, see Fig. 7.3

7.3.2 GUI Block Step 2:

1. Please verify that the skin patch shown in the display is similar to the real skin patch, see Fig. 7.4. The GUI offers different configuration options, but for this simple case you can directly go to **Step 3**.

7.3.3 GUI Block Step 3:

1. Select the base frame of the patch, i.e. the robot link where the patch is attached to (e.g. ee_link), see Fig. 7.5, and press the button “Define External TF”, see Fig. 7.6.

2. In rviz add the “InteractiveMarkers” plugin and select the “Update Topic”. The interactive marker gimbal is shown in rviz (Fig. 7.7). You need to move the visualized skin patch to the correct pose, using the arrows and rings of the interactive marker. Also, activate the Marker Array plug-in to visualize the skin patch in rviz, see Fig. 7.8.

3. When the patch has its correct pose, e.g. the patch in rviz is located and orientated as is in the real robot (use the acceleration signals shown in rviz to orientate the patch correctly), then press the button “Confirm External TF” “Step3: Extrinsics” in the calibration GUI, see Fig. 7.9.

4. Press the “Save” button in “Step3: Extrinsics” group box of the GUI. Select a proper name and path for the skin config file. (fileNamePath), see Fig. 7.10. This button will create a skin configuration file where the intrinsic and extrinsic parameters of the patch will be

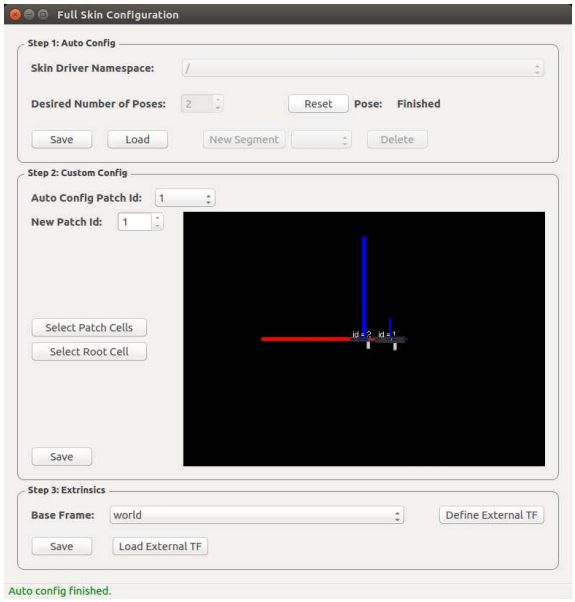


Figure 7.3: Step 1.3

stored. This skin config file will be used to obtain and visualize data from the skin patch, see Sec. 7.4.

5. Close the main window.

7.4 Robot skin visualization and skin cell data access

To visualize the skin patch (rviz) and acquire data of each of its skin cells, we need to run a special ROS node.

```
$roslaunch tum_ics_skin_tutorials
 load_and_view_patch_robot.launch CONFIG_PATH:=<fileNamePath>
```

<fileNamePath> defines the path and filename of the skin config file generated during the calibration, see Sec. 7.3.3, Step 3.4. The visualization of the skin patch in rviz should show the skin patch located at the right pose. The proximity, acceleration, temperature and force signals of each skin cell are also displayed in rviz.

You can immediately check whether the node is working, because it will create a ros publisher with a topic named as the patch name. This topic contains all the information of all the cells in the skin patch. To find out the proper topic name simply run: `$rostopic list` Check the topic name, it should be something like: */patchX*. Then, you can printout the data with `$rostopic echo /patchX`

Finally, to integrate the patch into your application, take a look on the file:
tum_ics_skin_tutorials/src/Applications/main_load_and_view_patch_robot_tut.cpp
There, you will find an example on how to use the skin patch API.

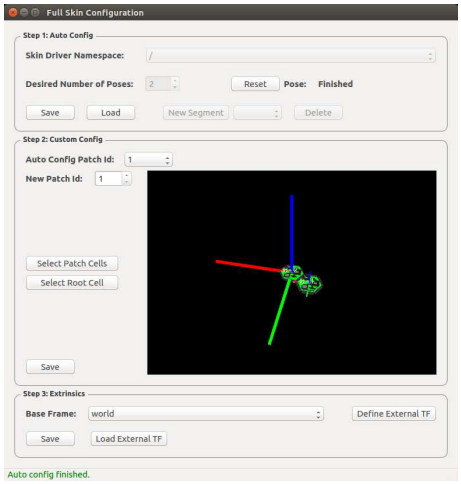


Figure 7.4: Step 2.1

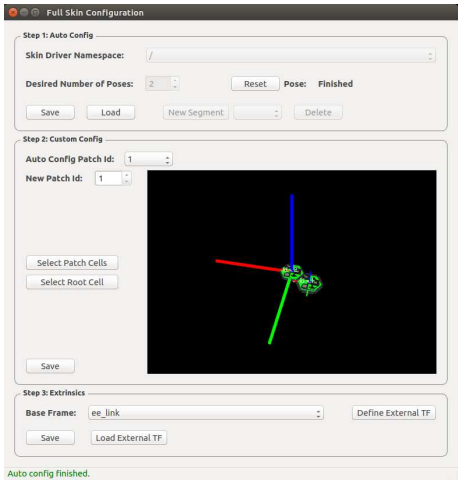


Figure 7.5: Step 3.1

References

[1] Universal Robots: User Manual Robot UR-10, Agust 1, 2013. https://www.universal-robots.com/download/

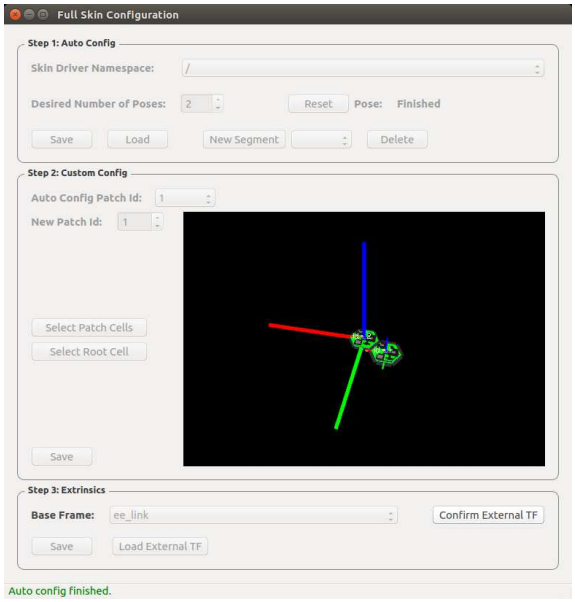


Figure 7.6: Step 3.1

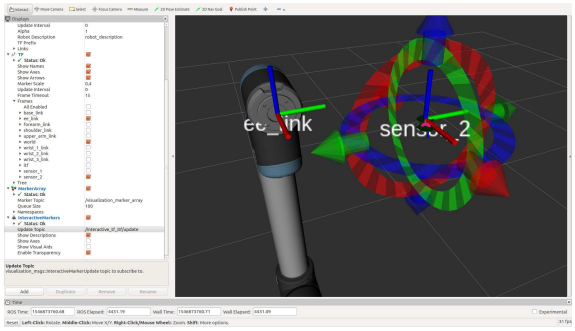


Figure 7.7: Step 3.2

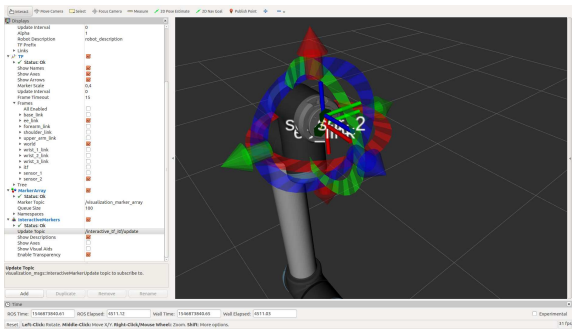


Figure 7.8: Step 3.2

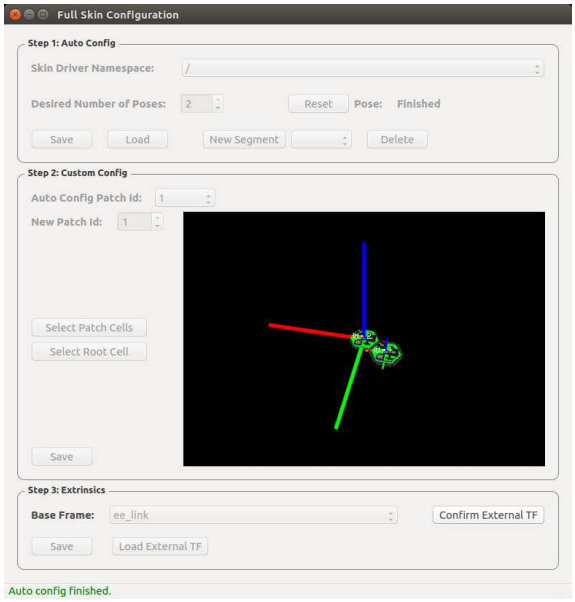


Figure 7.9: Step 3.3

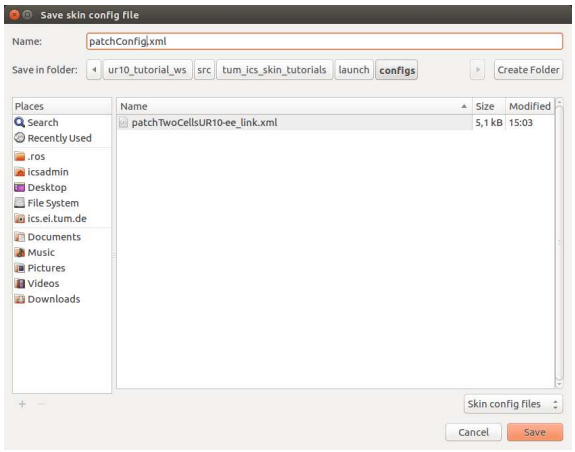


Figure 7.10: Step 3.4