Culturally Sensitive Social Robotics
for Africa

D3.3 Software Installation Manual

Due date: **1/10/2023**
Submission Date: **18/07/2023**

Start date of project: **01/07/2023** Duration: **36 months**

Lead organisation for this deliverable: **Carnegie Mellon University Africa**

Responsible Person: **CSSR4A Team** Revision: **1.10**

Project funded by the African Engineering and Technology Network (Afretec)
Inclusive Digital Transformation Research Grant Programme

Dissemination Level		
PU	Public	**PU**
PP	Restricted to other programme participants (including Afretec Administration)	
RE	Restricted to a group specified by the consortium (including Afretec Administration)	
CO	Confidential, only for members of the consortium (including Afretec Administration)	

Contents

Executive Summary **3**

Principal Contributors **4**

Revision History **5**

1 Technical Specifications & Accessibility **6**

2 Setting up the development environment to control the physical robot **10**

3 Sensor and Actuator Unit Tests **14**

4 Setting up the development environment in a Gazebo simulator **17**

5 List of ROS topics **20**

References **26**

Executive Summary

This document serves as a comprehensive installation manual for the **Culturally Sensitive Social Robotics for Africa** (CSSR4A) project, guiding users through the step-by-step instructions for setting up the development environment necessary to control the Pepper robot physically and in a simulator, and conducting unit tests. The manual is organized into five distinct sections: technical specifications and accessibility, setting up the development environment to control the physical Pepper robot, sensor and actuator unit tests, setting up the development environment in the simulator, and a list of ROS topics.

Principal Contributors

The main authors of this deliverable are as follows (in alphabetical order).

Adedayo Akinade, CMU-Africa
Deogratias Amani, CMU-Africa
Yohannes Haile, CMU-Africa
Kleber Kabanda, CMU-Africa
Mihiretab Taye Hordofa , CMU-Africa
Natasha Mutangana, CMU-Africa
Pamely Zantou, CMU-Africa

Revision History

Version 1.0 (A.O. 31-05-2023)
First draft.

Version 1.1(A.O. 01-06-2023)
Network interface update.
Command typo fix.

Version 1.2(A.O. 02-06-2023)
Recommended remote desktop application added.

Version 1.3(A.O. 02-06-2023)
Setting up the development environment added.

Version 1.4(A.O. 02-06-2023)
Missed command and typo fix.

Version 1.5(A.O. 09-06-2023)
Add unit test.

Version 1.6(A.O. 16-06-2023)
Add missed commands.

Version 1.7(A.O. 21-06-2023)
Addition of GUI access software for Linux.

Version 1.8(A.O. 23-06-2023)
Installation manual rewrite.

Version 1.9(A.O. 01-07-2023)
Setting up the development environment in a Gazebo simulator.

Version 1.10(A.O. 12-07-2023)
Mapping subscribers between the physical robot and the simulator (sensors and actuators).

1 Technical Specifications & Accessibility

To support the software development process for CSSR4A, a ROS-based environment is necessary, requiring a Ubuntu host for writing software. While it is possible to install your own Ubuntu OS and set up the development environment independently, a dedicated server has been provisioned specifically for those affiliated with Carnegie Mellon University (CMU). Users can conveniently create profiles on a **Ubuntu 18.04** virtual machine, which functions as the designated server. Ubuntu 18.04 is the recommended version due to its compatibility with ROS Melodic, the meta-operating system utilized for the CSSR4A project. By leveraging this server, users can streamline the setup process and ensure a standardized environment for development tasks.

Specifications

The server has the following specifications:

- Storage: 1 TB
- RAM: 16 GB
- CPU: 12 Cores

Accessibility

Users can connect to the server in two different modes: non graphical user interface or the command line mode (Non-GUI) and in a graphical user interface (GUI).

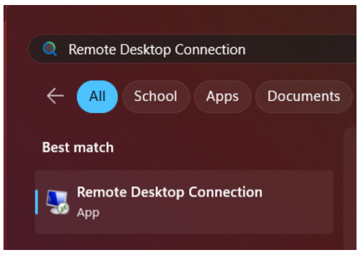
Non-Graphical User Interface -Windows, Linux, and Mac

Connecting to the server via the command line entails using the Secure Shell Protocol (SSH). You can do this using the terminals available on the three operating systems (OS) supported by this manual, namely Windows, Linux, and Mac. Additionally, you can use PuTTY, one of the most popular applications for SSH connection on Windows. Below is the bash command line template to connect to the server.

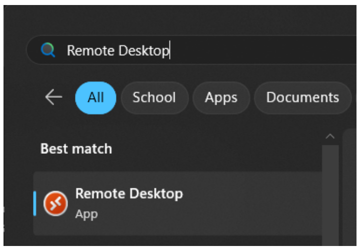
```
# step 1: enter the command line below to request a connection
# eg: ssh pamking@192.168.1.1
ssh <username>@<server_ip>
# step 2: enter your password based on the message prompted below
# eg: pamking@192.168.1.1 password: enter_the_password_here
<username>@<ip>'s password: _
```

Graphical User Interface

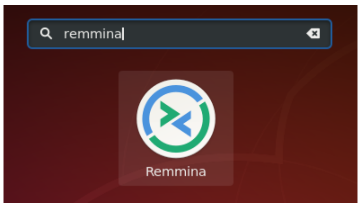
Connecting to the server's GUI requires the use of a remote desktop application. Windows users may choose two applications to connect to the server: **Remote Desktop Connection** and **Microsoft Remote Desktop**, which can be downloaded from the Windows store. Microsoft Remote Desktop is also available in the Mac OS store. On Ubuntu, you can install **Remmina** as your remote desktop application.



(a)



(b)



(c)

Figure 1: (a) Remote Desktop Connection Software. (b) Microsoft Remote Desktop application (c) Remmina.

Installation can be done using either the command line or the Ubuntu Software.

1. **Installation using the command line**
   - **Install using snap**
     This is highly recommended since you are more likely to have access to the last update of the software.

```
# Step 1: get your system ready for installation
sudo apt-get update
# Step 2: install snap if not installed or skip this step
sudo apt-get install snapd
# Step 3: install remmina
sudo snap install remmina
```

   - **Install using PPA (Personal Package Archive)**
     While this works, the PPA is not actively maintained. Therefore, you are more likely to download an older version of the software.

```
# Step 1: add the ppa repository
sudo apt-add-repository ppa:remmina-ppa-team/remmina-next
# Step 2: get your system ready for installation
sudo apt update
# step 3: install remmina
sudo apt install remmina remmina-plugin-rdp remmina-plugin-secret
```

2. **Installation using Ubuntu Software (GUI)**
   This requires opening the Ubuntu Software application, searching for Remmina, and clicking on the install button at the right top of your screen to download the software.

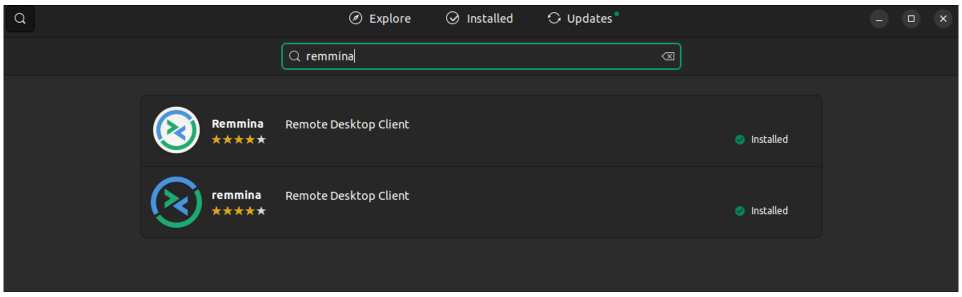
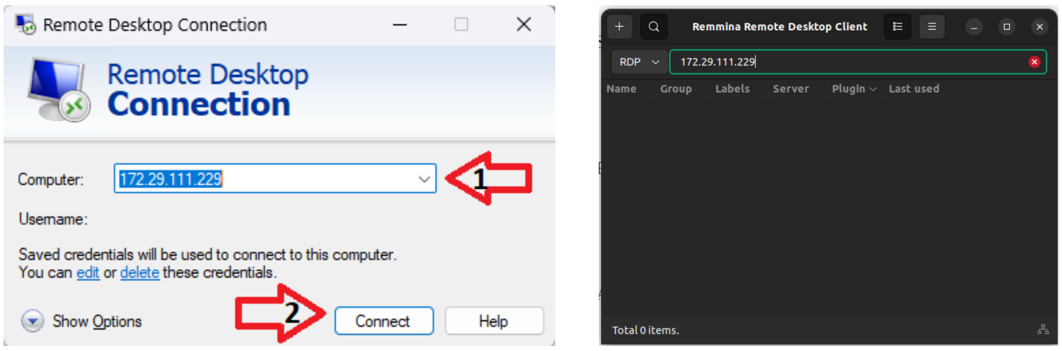

Explore
Installed
Updates
remmina
Remmina Remote Desktop Client Installed
remmina Remote Desktop Client Installed

Figure 2: Remmina on the Ubuntu Software GUI

After installing the remote desktop application of your preference, you can connect to the server. This implies that users must know the server's IP address as well as their credentials, and provide this information with the remote desktop application. It is important to note that users should be connected to the CMU-Secure network on campus or through the school's VPN. The server's IP address is: **172.29.111.229**.


Remote Desktop Connection
Remote Desktop Connection
Computer: 172.29.111.229
Username:
Saved credentials will be used to connect to this computer.
You can edit or delete these credentials.
1
2
Show Options
Connect
Help
Remmina Remote Desktop Client
RDP
172.29.111.229
Name
Group
Labels
Server
Plugin
Last used
Total 0 items.

(a) (b)

Figure 3: (a) Connection page Remote Desktop Connection. (b) Connection page Remmina.

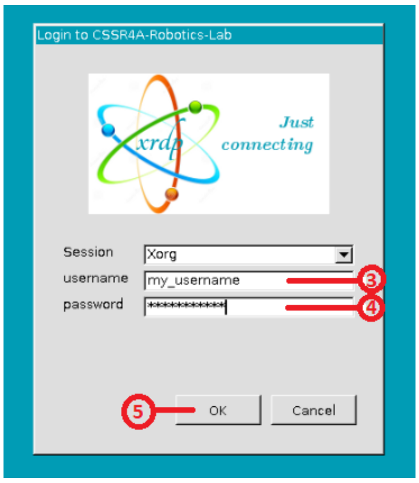
Figure 3a shows the connection page of the Remote Desktop Connection application, which features a connection button for users to establish a connection with the server. On the other hand, 3b illustrates the connection page of Remmina. In Remmina, users need to press the "Enter" key to initiate the connection to the server. It is important to note that in Remmina, the default connection type is RDP (Remote Desktop Protocol). This protocol enables users to access the server through a GUI. Refrain from changing the connection protocol to ensure uninterrupted access to the server in GUI mode. However, if you opt for the SSH protocol, you will gain access to the server through a terminal interface.

After successfully logging in using your user profile information, your screen should resemble figures 4a and 4b.

The next section addresses the different user profiles on the server.

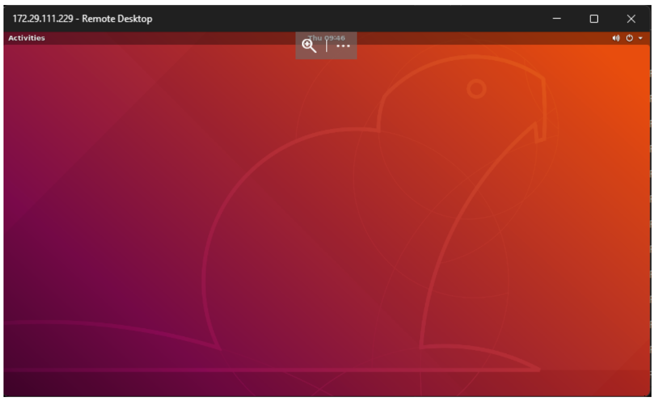
User profiles

Two types of users can connect to the server: an administrator and simple users. The administrator or admin, has all the privileges. The admin can create and remove users. Simple users have access to



Session: Xorg
username: my_username
password: ************

(a) Login with user credentials



(b) Server's Ubuntu home page.

Figure 4

the virtual machine and can write software to control the robot. First, we discussed how to connect to the server as a simple user. Now, we address how an admin connects, adds a user profile, and grants admin privileges to other users.

Login to the server as an admin

```
# Step 1: request for connection as admin
ssh admin@172.29.111.229
# Step 2: enter your password based on the message prompted below
admin@172.29.111.229's password: -
```

Add user profile

1. Create a user profile with a username as shown below.

```
adduser <username>
```

2. Enter the password for the username above.

```
password: _
```

3. Retype the password entered above

```
retype password: _
```

4. Fill in other optional prompts or skip (hit enter)

Give the user an admin privilege and allow to login.

1. Add user in the sshd_config file

```
retype password: echo "<username>" >> /etc/ssh/sshd_config
```

2. Change the user mode to be added in the admin group with super-user access.

```
usermod -aG sudo <username>
```

3. Restart the ssh service.

```
systemctl restart ssh
```

2 Setting up the development environment to control the physical robot

This section provides step-by-step instructions for installing a critical set of software packages required to control the Pepper humanoid. If you are accessing the server in the GUI mode, open a new terminal ***(ctrl + shift + t)*** and type the following commands carefully.

Installing ROS Melodic

1. Setup the computer to accept software from packages.ros.org

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc)
main" > /etc/apt/sources.list.d/ros-latest.list'
```

2. Install curl

```
sudo apt install curl
```

3. Setup your keys

```
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc |
sudo apt-key add -
```

4. Update Debian package index

```
sudo apt-get update
```

5. Install ROS Melodic with the default configurations

```
sudo apt install ros-melodic-desktop-full
```

6. Make ROS environment variables automatically added every time a new shell is launched

```
echo "source /opt/ros/melodic/setup.bash" >> $HOME/.bashrc
source $HOME/.bashrc
```

Installing and configuring the C++ NAOqi SDK

NAOqi is the main software that runs on the Pepper robot and controls it. In order to control Pepper with ROS, we need to install NAOqi and then find a driver with ROS to enable communication between the two software. The NAOqi Framework is a cross-language programming framework used to program Pepper. We can program the robot using C++ and Python. We have chosen to develop the bridge in C++. We, therefore, need to install the NAOqi C++ SDK following the steps detailed below:

1. Check compiler version
   A GCC compiler version 4.8.2 or higher is required. On most Ubuntu distributions, it is installed by default. We can check the GCC version by typing the following command:

   ```
   gcc -v
   ```

2. Install pip
   If you don't have pip installed on your machine, please run the code below.

   ```
   sudo apt install python-pip
   ```

3. Install qbuild
   qbuild is used to create a cross-platform executable.

   ```
   pip install qibuild --user
   ```

4. Update the path environment variable

   ```
   PATH=$PATH:$HOME/.local/bin # to load qBuild when your shell starts
   ```

5. Configure qBuild.

   ```
   qibuild config --wizard
   # Input 1 and hit enter (1 Unix Makefiles (default))
   # Input 1 and enter (1 None (default))
   ```

6. Make a directory to work in.

```
mkdir myworktree && cd myworktree
```

7. Download the Naoqi-SDK

```
wget -P $HOME/Downloads/ https://community-static.aldebaran.com/resources/2.5.10/
NAOqi%20SDK/naoqi-sdk-2.5.7.1-linux64.tar.gz
```

8. Go to the download folder and extract the downloaded file.

```
cd $HOME/Downloads && tar -xvzf naoqi-sdk-2.5.7.1-linux64.tar.gz
```

9. Create a toolchain

```
qitoolchain create mytoolchain $HOME/Dowloads/naoqi-sdk-2.5.7.1-linux64/toolchain.xml
```

10. Go to the myworktree directory and configure the SDK

```
cd $HOME/myworktree && qibuild add-config myconfig -t mytoolchain --default
```

Installing NAOqi driver and packages

The NAOqi driver is a module that provides some bridge capabilities. It publishes sensory data, the robot position (Pepper in our case) and enables ROS to call part of the NAOqi API. The easiest way is to install it with the apt package tool used to install software on Ubuntu. Installing the driver through the apt will not provide all the packages we need to communicate with the robot through ROS. For example, the **naoqi_dcm_driver**, is the package controlling the robot's actuators through Thus, we compile the remaining packages from their sources through catkin make. This requires cloning official sources code from GitHub. Below, are the steps to follow

```
# install the naoqi driver and
sudo apt-get install ros-.*-naoqi-driver

# create ROS workspace
mkdir -p $HOME/workspace/ros/src

# move to workspace directory
cd $HOME/workspace/ros/src

# install git
sudo apt install git

# clone naoqi dcm driver repository
git clone https://github.com/ros-naoqi/naoqi_dcm_driver.git

```

```
# clone pepper dcm driver repository
git clone https://github.com/ros-naoqi/pepper_dcm_robot.git

# clone pepper virtual repository.
git clone https://github.com/ros-naoqi/pepper_virtual.git

# clone naoqi driver repository
git clone https://github.com/ros-naoqi/naoqi_driver.git

# clone pepper robot repository
git clone https://github.com/ros-naoqi/pepper_robot.git

# clone pepper moveit config repository
git clone https://github.com/ros-naoqi/pepper_moveit_config.git

# clone pepper diagnostic routines repository
git clone https://github.com/tmihiret/pepper_diagnostic_routines.git

# Build the repository
cd .. && catkin_make

# Add the workspace to your ROS environment by sourcing setup file in devel folder
source devel/setup.bash

# Add the setup to your .bashrc file so that you don't have to do this every time
# you open a new terminal
echo "source ~/workspace/ros/devel/setup.bash" >> $HOME/.bashrc

# install additional packages
sudo apt-get install ros-melodic-joint-trajectory-controller

sudo apt-get install ros-melodic-ros-controllers

sudo apt-get install ros-melodic-pepper-meshes
# when configuring window opens up, you may agree to the license terms using
# right/left arrow key and select <ok> and hit enter and then select <Yes>
# to accept the terms and hit enter.

# install rosdep
sudo pip install -U rosdep
sudo rosdep init
rosdep update

rosdep install --from-paths src --ignore-src -r -y
```

Important

Deactivate Audio Service - Open boot config file and set the audio key to false

```
sudo nano ~/workspace/ros/src/naoqi_driver/share/boot_config.json
```

Network configuration

1. Install network tool (if not installed)

```
sudo apt install net-tools
```

2. Identify network ip and interface.

```
ifconfig
```

Bring up Pepper

The Pepper Robot is now set up to be controlled over ROS. To wake Pepper up over ROS, follow the instructions below:

1. Bring up the Pepper robot, assuming the robot is turned on

```
roslaunch pepper_dcm_bringup pepper_bringup.launch robot_ip:=172.29.111.230
roscore_ip:=172.29.111.229 network_interface:=wlan0
```

2. Launch the Naoqi driver

```
roslaunch naoqi_driver naoqi_driver.launch nao_ip:=172.29.111.230
roscore_ip:=172.29.111.249 network_interface:=ens160
```

3 Sensor and Actuator Unit Tests

The following section presents a set of unit tests designed to evaluate the functionality and performance of the sensors and actuators on the Pepper humanoid robot. These tests aim to ensure that the sensors accurately perceive the environment, and provide reliable data for further processing and decision-making in the CSSR4A project. Before running the unit tests, we will first describe how the software is organized. The software developed as part of this work is ROS-based. Thus, we follow ROS best practices to organize the code. Below is what the project directory looks like:

```
workspace
└─ros
  ├─build
  ├─devel
  └─src
    ├─naoqi_dcm_driver
    ├─naoqi_driver
    ├─pepper_dcm_robot
    ├─pepper_diagnostic_routines
    │ ├─config
    │ ├─data
    │ ├─include
    │ ├─launch
    │ │ └─diagnostic.launch
    │ ├─msg
    │ ├─src
    │ │ ├─back_sonar.cpp
    │ │ ├─bottom_camera.cpp
    │ │ ├─depth_camera.cpp
    │ │ ├─front_camera.cpp
    │ │ ├─front_sonar.cpp
    │ │ ├─goToPositionMIMO.cpp
    │ │ ├─goToPositionMimo.cpp
    │ │ ├─handTouch.cpp
    │ │ ├─headTouch.cpp
    │ │ ├─laser.cpp
    │ │ ├─left_arm_control.cpp
    │ │ ├─left_hand_control.cpp
    │ │ ├─pelvis_control.cpp
    │ │ ├─right_arm_control.cpp
    │ │ └─right_hand_control.cpp
    │ ├─srv
    │ ├─CMakeLists.txt
    │ └─package.xml
    ├─pepper_moveit_config
    ├─pepper_robot
    ├─pepper_sensors_py
    └─pepper_virtual
```

The **src** folder in the project tree contains all the packages we used to enable the communication between ROS and NAOqi. However, only the **pepper_diagnostic_routines** package contains the source files that have been written to perform the unit tests. In order to run the software, run the following commands.

```
# move to the workspace directory
cd $HOME/workspace/ros/

```

```
# build the source files
catkin_make

# launch the diagnostic routines
roslaunch pepper_diagnostic_routines diagnostic.launch robot_ip:=<robot_ip>
roscore_ip:=<roscore_ip> network_interface:=<network_interface>

# launch the naoqi driver
roslaunch naoqi_driver naoqi_driver.launch nao_ip:=<robot_ip> roscore_ip:
<roscore_ip> network_interface:=<network_interface>

```

The commands above will wake the Pepper robot up. The **roscore_ip** is the IP address of the host from which you are running the software. You might encounter the following error while installing the driver.

Error

terminate called after throwing an instance of 'qi::FutureUserException' what(): Can't find service: ROS-Driver-Audio

There are many possible causes of this error. Based on the comments provided by the maintainers of the naoqi driver on their github issues page and some developers, this error can be due to network configuration issues. However, in our case, it was not a network issue. In fact, we were able to work around the error by deactivating the audio service in order to avoid this crash. Below is what we did:

```
sudo emacs /opt/ros/melodic/naoqi_driver/share/boot_config.json
```

The command above opens the **boot_config.json** file in Emacs. Set the audio key to false. This should be able to stop the program from crashing. After that, the robot can wake up!

However, setting the audio key to false will lead to the user not accessing the microphone in the robot. Hence, no tests are included below to check the functionality of the microphone.

```
# Test the actuator's control
rosrun pepper_diagnostic_routines actuators_control

# Test the arm control
rosrun pepper_diagnostic_routines arm_control

# Test the back sonar
rosrun pepper_diagnostic_routines back_sonar

# Test the bottom camera
rosrun pepper_diagnostic_routines bottom_camera

# Test the depth camera
```

```
rosrun pepper_diagnostic_routines depth_camera

# Test the front camera
rosrun pepper_diagnostic_routines front_camera

# Test the front sonar:
rosrun pepper_diagnostic_routines front_sonar
```

4 Setting up the development environment in a Gazebo simulator

Sections 2 and 3 assume that you have access to a physical Pepper robot, however, there are times you may need to test your software without access to the physical robot. In this section, we explain how you can control the Pepper robot using a Gazebo simulator. The simulation is based on an open-source Pepper ROS environment by Sam Pfeiffer and Finn Rietz [1]. For setup on Ubuntu 18.04, ensure ROS Melodic is installed. For ROS installation, refer to section 2.

Install Docker using the apt repository

Below are a set of instructions to install the Docker engine on Ubuntu [2].

1. Make sure to uninstall any conflicting packages.

```
for pkg in docker.io docker-doc docker-compose podman-docker containerd runc;
do sudo apt-get remove $pkg;
done
```

2. Update the apt package index and install packages to allow apt to use a repository over HTTP

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
```

3. Add Docker's official GPG key

```
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

4. Use the following command to set up the repository

```
echo \"deb
[arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
```

[1] The Pepper ROS simulator is available on the following link: https://github.com/frietz58

[2] This is a subset of the official documentation available at https://docs.docker.com/engine/install/ubuntu/.

```
"$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

5. Update the apt package index

```
sudo apt-get update
```

6. Install Docker Engine, container, and Docker Compose

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
docker-buildx-plugin docker-compose-plugin
```

7. Verify that the Docker Engine installation is successful by running the hello-world image.

```
sudo docker run hello-world
```

   This command downloads a test image and runs it in a container. When the container runs, it prints a confirmation message and exits.

Install the Pepper Gazebo Simulator

1. Clone the GitHub repository

```
git clone https://github.com/frietz58/pepper_virtual
```

2. Move to the pepper_virtual directory

```
cd pepper_virtual
```

3. Build image from provided Dockerfile

```
sudo docker build -t awesome-pepper-sim .
```

4. Allow Docker to open GUIs on the surrounding OS

```
xhost +local:root
```

5. Run the Docker container

```
sudo docker run -it -v /tmp/.X11-unix:/tmp/.X11-unix
-e DISPLAY=unix$DISPLAY awesome-pepper-sim
```

6. Start the Gazebo simulation

```
roslaunch pepper_gazebo_plugin pepper_gazebo_plugin_in_office_CPU.launch
```

If everything goes well, your simulation environment should be similar to the image below.

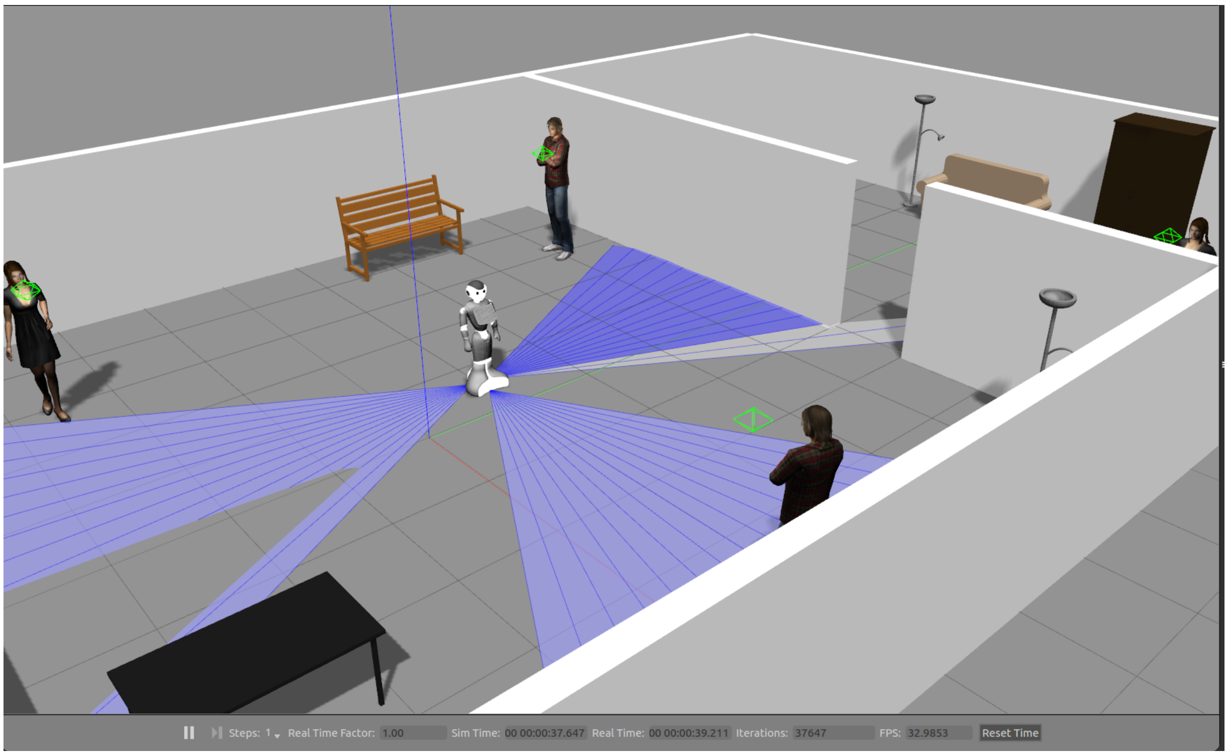


Figure 5: Pepper Gazebo simulation environment

Sanity check: Driving the Pepper robot in the simulator

In this section, we drive the Pepper robot in the simulation environment using the **rqt** GUI joint trajectory controller application.

1. Find the ID of the container:

```
sudo docker ps
```

2. Attach new shell to the container with the given ID

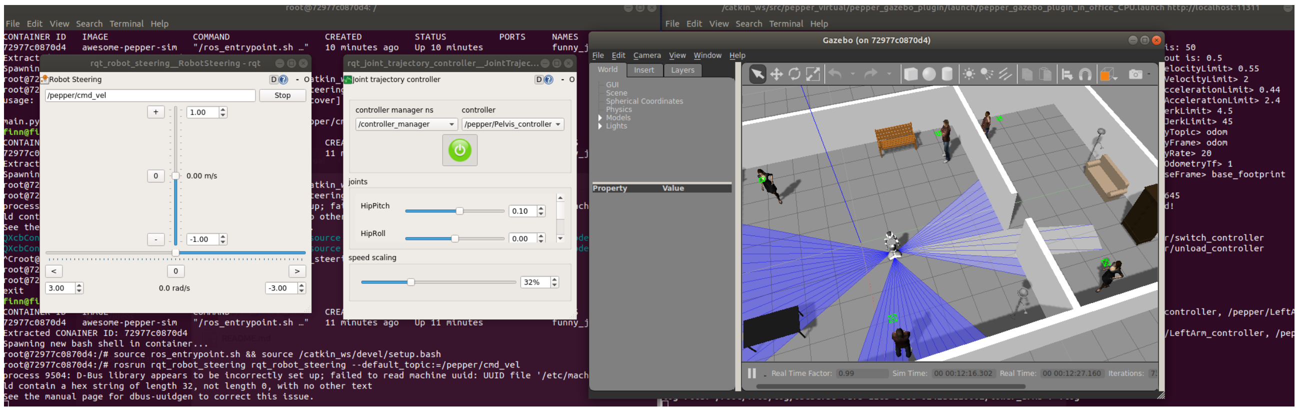
```
sudo docker exec -it <CONTAINER-ID> bash
```

3. Start rqt steering with Pepper's base topic

```
rosrun rqt_robot_steering rqt_robot_steering --default_topic:=/pepper/cmd_vel
```

4. Start joint trajectory controller

```
rosrun rqt_joint_trajectory_controller rqt_joint_trajectory_controller
```



```
CONTAINER ID   IMAGE               COMMAND                  CREATED          STATUS          PORTS     NAMES
72977c0870d4   awesome-pepper-sim  "/ros_entrypoint.sh …"   10 minutes ago   Up 10 minutes             funny_j
```

```
CONTAINER ID   IMAGE               COMMAND                  CREATED          STATUS          PORTS     NAMES
72977c0870d4   awesome-pepper-sim  "/ros_entrypoint.sh …"   11 minutes ago   Up 11 minutes             funny_j
Extracted CONAINER ID: 72977c0870d4
Spawning new bash shell in container...
root@72977c0870d4:/# source ros_entrypoint.sh && source /catkin_ws/devel/setup.bash
root@72977c0870d4:/# rosrun rqt_robot_steering rqt_robot_steering --default_topic:=/pepper/cmd_vel
process 9504: D-Bus library appears to be incorrectly set up; failed to read machine uuid: UUID file '/etc/mach
ld contain a hex string of length 32, not length 0, with no other text
See the manual page for dbus-uuidgen to correct this issue.
```

```
is: 50
out is: 0.5
elocityLimit> 0.55
VelocityLimit> 2
ccelerationLimit> 0.44
AccelerationLimit> 2.4
erkLimit> 4.5
JerkLimit> 45
yTopic> odom
yFrame> odom
yRate> 20
OdometryTf> 1
seFrame> base_footprint

645
d!

r/switch_controller
r/unload_controller

controller, /pepper/LeftA

/LeftArm_controller, /pep
```

Figure 6: Joint trajectory controller

5 List of ROS topics

This section provides a comprehensive set of tables that summarize the available topics we can subscribe to through ROS. The topics are categorized based on the sensors and the end effectors utilized, as well as the robot platforms employed (physical robot or simulator).

Sensors

Sensors	Subscriber physical robot	Subscriber simulator
3D depth camera	/naoqi_driver/camera/depth/camera_info	/pepper /camera /depth /camera_info
	/naoqi_driver/camera/depth/image_raw	/pepper /camera /depth /image_raw
	/naoqi_driver/camera/depth/image_raw/compressed/ parameter_descriptions	-
	/naoqi_driver/camera/depth/image_raw/ compressedDepth	-
	/naoqi_driver/camera/depth/image_raw/ compressedDepth/parameter_descriptions	-
	/naoqi_driver/camera/depth/image_raw/ compressedDepth/parameter_updates	-
	/naoqi_driver/camera/depth/image_raw/theora	-
	/naoqi_driver/camera/depth/image_raw/theora/ parameter_descriptions	-
	/naoqi_driver/camera/depth/image_raw/theora/ parameter_descriptions	-
	/naoqi_driver/camera/depth/image_raw/theora/ parameter_updates	-
	-	/pepper /camera /depth /points
2D front camera	/naoqi_driver/camera/front/camera_info	/pepper/camera/front/camera_info
	/naoqi_driver/camera/front/image_raw	/pepper/camera/front/image_raw
	/naoqi_driver/camera/front/image_raw/compressed/ parameter_descriptions	/pepper/camera/front/image_raw/compressed/ parameter_descriptions
	/naoqi_driver/camera/front/image_raw/compressed/ parameter_updates	/pepper/camera/front/image_raw/compressed/ parameter_updates
	/naoqi_driver/camera/front/image_raw/theora	/pepper/camera/front/image_raw/theora
	/naoqi_driver/camera/front/image_raw/theora / parameter_descriptions	/pepper/camera/front/image_raw/theora/ parameter_descriptions
	/naoqi_driver/camera/front/image_raw/theora / parameter_updates	/pepper/camera/front/image_raw/theora/ parameter_updates
	-	/pepper/camera/front/parameter_descriptions
	-	/pepper/camera/front/parameter_updates
	-	/pepper/camera/front/image_raw/compressedDepth
	-	/pepper/camera/front/image_raw/compressedDepth/ parameter_descriptions
	-	/pepper/camera/front/image_raw/compressedDepth/ parameter_updates

Sensors	Subscriber physical robot	Subscriber simulator
2D bottom camera	/naoqi_driver/camera/bottom/camera_info	/pepper/camera/bottom/camera_info
	/naoqi_driver/camera/bottom/image_raw	/pepper/camera/bottom/image_raw
	/naoqi_driver/camera/bottom/image_raw/compressed	/pepper/camera/bottom/image_raw/compressed
	/naoqi_driver/camera/bottom/image_raw/compressed/ parameter_descriptions	/pepper/camera/bottom/image_raw/compressed/ parameter_descriptions
	/naoqi_driver/camera/bottom/image_raw/compressed/ parameter_updates	/pepper/camera/bottom/image_raw/compressed/ parameter_updates
	/naoqi_driver/camera/bottom/image_raw/theora	/pepper/camera/bottom/image_raw/theora
	/naoqi_driver/camera/bottom/image_raw/theora / parameter_descriptions	/pepper/camera/bottom/image_raw/theora/ parameter_descriptions
	/naoqi_driver/camera/bottom/image_raw/theora / parameter_updates	/pepper/camera/bottom/image_raw/theora/ parameter_updates
	-	/pepper/camera/bottom/parameter_descriptions
	-	/pepper/camera/bottom/parameter_updates
	-	/pepper/camera/bottom/image_raw/compressedDepth
	-	/pepper/camera/bottom/image_raw/ compressedDepth/parameter_descriptions
	-	/pepper/camera/bottom/image_raw/ compressedDepth/parameter_updates
IR camera	/naoqi_driver/camera/ir/camera_info	-
	/naoqi_driver/camera/ir/image_raw	/pepper/camera/ir/image_raw
	/naoqi_driver/camera/ir/image_raw/compressed	/pepper/camera/ir/image_raw/compressed
	/naoqi_driver/camera/ir/image_raw/compressed/ parameter_descriptions	/pepper/camera/ir/image_raw/compressed/ parameter_descriptions
	/naoqi_driver/camera/ir/image_raw/compressed/ parameter_updates	/pepper/camera/ir/image_raw/compressed/ parameter_updates
	/naoqi_driver/camera/ir/image_raw/theora	/pepper/camera/ir/image_raw/theora
	/naoqi_driver/camera/ir/image_raw/theora/ parameter_descriptions	/pepper/camera/ir/image_raw/theora/ parameter_descriptions
	/naoqi_driver/camera/ir/image_raw/theora/ parameter_updates	/pepper/camera/ir/image_raw/theora/ parameter_updates
	-	/pepper/camera/parameter_descriptions
	-	/pepper/camera/parameter_updates
	-	/pepper/camera/ir/image_raw/compressedDepth
	-	/pepper/camera/ir/image_raw/compressedDepth/ parameter_descriptions
	-	/pepper/camera/ir/image_raw/compressedDepth/ parameter_updates

Sensors	Subscriber physical robot	Subscriber simulator
Sonar	/naoqi_driver/sonar/back	/pepper/sonar_back
	/naoqi_driver/sonar/front	/pepper/sonar_front
IMU	/naoqi_driver/imu/base	-
	/naoqi_driver/imu/torso	-
Laser	/naoqi_driver/laser	-

End effectors

End effectors	Subscriber physical robot	Subscriber simulator
Head	/pepper_dcm/Head_controller/command	/pepper/Head_controller/command
	/pepper_dcm/Head_controller/follow_joint_trajectory	-
	/pepper_dcm/Head_controller/follow_joint_trajectory/cancel	/pepper/Head_controller/follow_joint_trajectory/cancel
	/pepper_dcm/Head_controller/follow_joint_trajectory/feedback	/pepper/Head_controller/follow_joint_trajectory/feedback
	/pepper_dcm/Head_controller/follow_joint_trajectory/goal	/pepper/Head_controller/follow_joint_trajectory/goal
	/pepper_dcm/Head_controller/follow_joint_trajectory/result	/pepper/Head_controller/follow_joint_trajectory/result
	/pepper_dcm/Head_controller/follow_joint_trajectory/status	/pepper/Head_controller/follow_joint_trajectory/status
	/pepper_dcm/Head_controller/state	/pepper/Head_controller/state
	-	/pepper/Head_controller/gains/HeadPitch/parameter_descriptions
	-	/pepper/Head_controller/gains/HeadPitch/parameter_updates
	-	/pepper/Head_controller/gains/HeadYaw/parameter_descriptions
	-	/pepper/Head_controller/gains/HeadYaw/parameter_updates
Left arm	/pepper_dcm/LeftArm_controller/command	/pepper/LeftArm_controller/command
	/pepper_dcm/LeftArm_controller/follow_joint_trajectory/cancel	/pepper/LeftArm_controller/follow_joint_trajectory/cancel
	/pepper_dcm/LeftArm_controller/follow_joint_trajectory/feedback	/pepper/LeftArm_controller/follow_joint_trajectory/feedback
	/pepper_dcm/LeftArm_controller/follow_joint_trajectory/goal	/pepper/LeftArm_controller/follow_joint_trajectory/goal
	/pepper_dcm/LeftArm_controller/follow_joint_trajectory/result	/pepper/LeftArm_controller/follow_joint_trajectory/result
	/pepper_dcm/LeftArm_controller/follow_joint_trajectory/status	/pepper/LeftArm_controller/follow_joint_trajectory/status
	/pepper_dcm/LeftArm_controller/state	/pepper/LeftArm_controller/state
	-	/pepper/LeftArm_controller/gains/LElbowRoll/parameter_descriptions
	-	/pepper/LeftArm_controller/gains/LElbowRoll/parameter_updates
	-	/pepper/LeftArm_controller/gains/LElbowYaw/parameter_descriptions
	-	/pepper/LeftArm_controller/gains/LElbowYaw/parameter_updates
	-	/pepper/LeftArm_controller/gains/LShoulderPitch/parameter_descriptions

Left arm	-	/pepper/LeftArm_controller/gains/LShoulderPitch/parameter_updates
	-	/pepper/LeftArm_controller/gains/LShoulderRoll/parameter_descriptions
	-	/pepper/LeftArm_controller/gains/LShoulderRoll/parameter_updates
	-	/pepper/LeftArm_controller/gains/LWristYaw/parameter_descriptions
	-	/pepper/LeftArm_controller/gains/LWristYaw/parameter_updates
Right arm	/pepper_dcm/RightArm_controller/command	/pepper/RightArm_controller/command
	/pepper_dcm/RightArm_controller/follow_joint_trajectory/cancel	/pepper/RightArm_controller/follow_joint_trajectory/cancel
	/pepper_dcm/RightArm_controller/follow_joint_trajectory/feedback	/pepper/RightArm_controller/follow_joint_trajectory/feedback
	/pepper_dcm/RightArm_controller/follow_joint_trajectory/goal	/pepper/RightArm_controller/follow_joint_trajectory/goal
	/pepper_dcm/RightArm_controller/follow_joint_trajectory/result	/pepper/RightArm_controller/follow_joint_trajectory/result
	/pepper_dcm/RightArm_controller/follow_joint_trajectory/status	/pepper/RightArm_controller/follow_joint_trajectory/status
	/pepper_dcm/RightArm_controller/state	/pepper/RightArm_controller/state
	-	/pepper/RightArm_controller/gains/RElbowRoll/parameter_descriptions
	-	/pepper/RightArm_controller/gains/RElbowRoll/parameter_updates
	-	/pepper/RightArm_controller/gains/RElbowYaw/parameter_descriptions
	-	/pepper/RightArm_controller/gains/RElbowYaw/parameter_updates
	-	/pepper/RightArm_controller/gains/RShoulderPitch/parameter_descriptions
	-	/pepper/RightArm_controller/gains/RShoulderRoll/parameter_descriptions
	-	/pepper/RightArm_controller/gains/RShoulderRoll/parameter_descriptions
	-	/pepper/RightArm_controller/gains/RShoulderRoll/parameter_updates
	-	/pepper/RightArm_controller/gains/RWristYaw/parameter_descriptions
	-	/pepper/RightArm_controller/gains/RWristYaw/parameter_updates

Pelvis	/pepper_dcm/Pelvis_controller/command	/pepper/Pelvis_controller/command
	/pepper_dcm/Pelvis_controller/follow_joint_trajectory/cancel	/pepper/Pelvis_controller/follow_joint_trajectory/cancel
	/pepper_dcm/Pelvis_controller/follow_joint_trajectory/feedback	/pepper/Pelvis_controller/follow_joint_trajectory/feedback
	/pepper_dcm/Pelvis_controller/follow_joint_trajectory/goal	/pepper/Pelvis_controller/follow_joint_trajectory/goal
	/pepper_dcm/Pelvis_controller/follow_joint_trajectory/result	/pepper/Pelvis_controller/follow_joint_trajectory/result
	/pepper_dcm/Pelvis_controller/follow_joint_trajectory/status	/pepper/Pelvis_controller/follow_joint_trajectory/status
	/pepper_dcm/Pelvis_controller/state	/pepper/Pelvis_controller/state
	-	/pepper/Pelvis_controller/gains/HipPitch/parameter_descriptions
	-	/pepper/Pelvis_controller/gains/HipPitch/parameter_updates
	-	/pepper/Pelvis_controller/gains/HipRoll/parameter_descriptions
	-	/pepper/Pelvis_controller/gains/HipRoll/parameter_updates
	-	/pepper/Pelvis_controller/gains/KneePitch/parameter_descriptions
	-	/pepper/Pelvis_controller/gains/KneePitch/parameter_updates
Left hand	/pepper_dcm/LeftHand_controller/command	-
	/pepper_dcm/LeftHand_controller/follow_joint_trajectory/cancel	-
	/pepper_dcm/LeftHand_controller/follow_joint_trajectory/feedback	-
	/pepper_dcm/LeftHand_controller/follow_joint_trajectory/goal	-
	/pepper_dcm/LeftHand_controller/follow_joint_trajectory/result	-
	/pepper_dcm/LeftHand_controller/follow_joint_trajectory/status	-
	/pepper_dcm/LeftHand_controller/state	-
Right hand	/pepper_dcm/RightHand_controller/command	-
	/pepper_dcm/RightHand_controller/follow_joint_trajectory/cancel	-
	/pepper_dcm/RightHand_controller/follow_joint_trajectory/feedback	-
	/pepper_dcm/RightHand_controller/follow_joint_trajectory/goal	-
Right hand	/pepper_dcm/RightHand_controller/follow_joint_trajectory/result	-
	/pepper_dcm/RightHand_controller/follow_joint_trajectory/status	-
	/pepper_dcm/RightHand_controller/state	-
Wheels	/cmd_vel	/cmd_vel
	/pepper_dcm/cmd_moveto	-

References