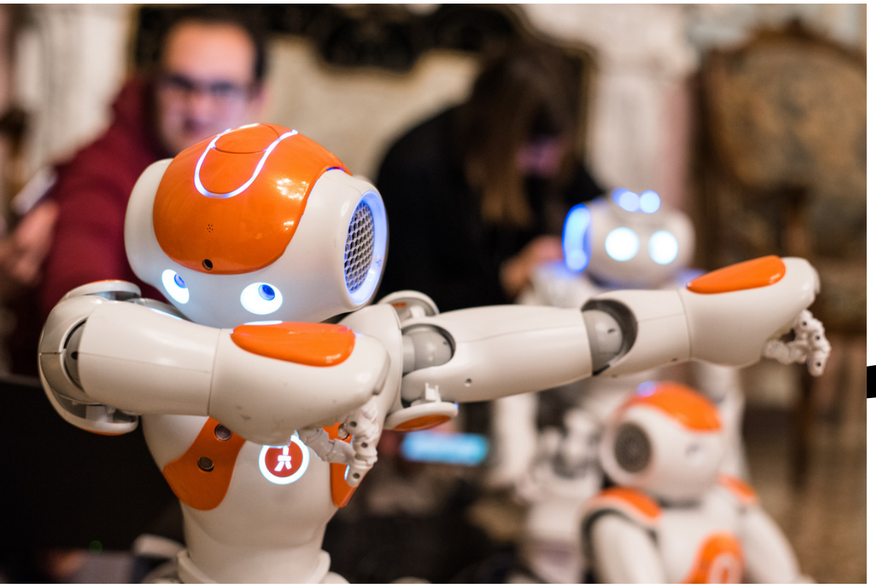
Università di Bologna
Alma Mater Studiorum

NAO CHALLENGE

2020



FAST FREQUENCY

SANDEEP KUMAR KUSHWAHA,
SAMRAL TAHIRLI,
ZARMINA URSINO

SUMMARY OF CONTENTS

TODAY'S AGENDA

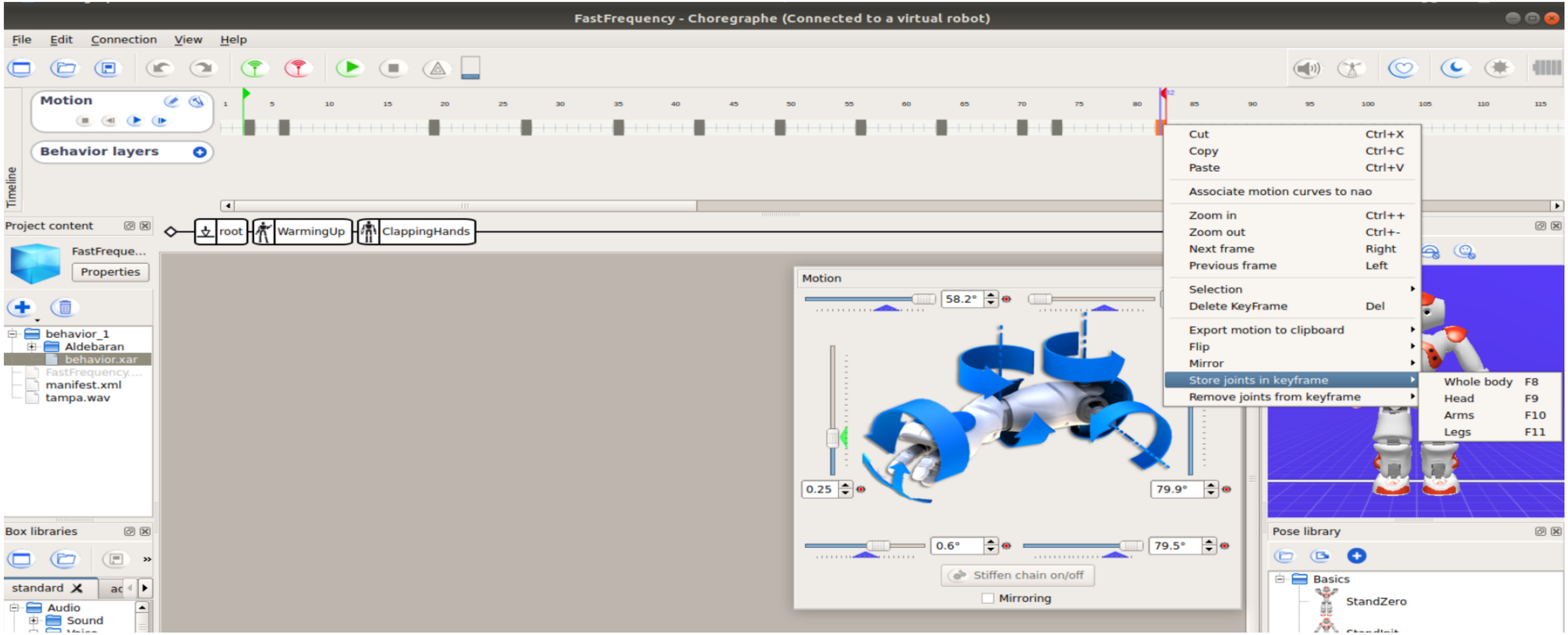
History of our project

WATCH THIS VIDEO

This is the link that you can watch here:

lhttps://www.instagram.com/p/CEudpYKDUv0/?utm_source=ig_web_copy_link

Movement creation

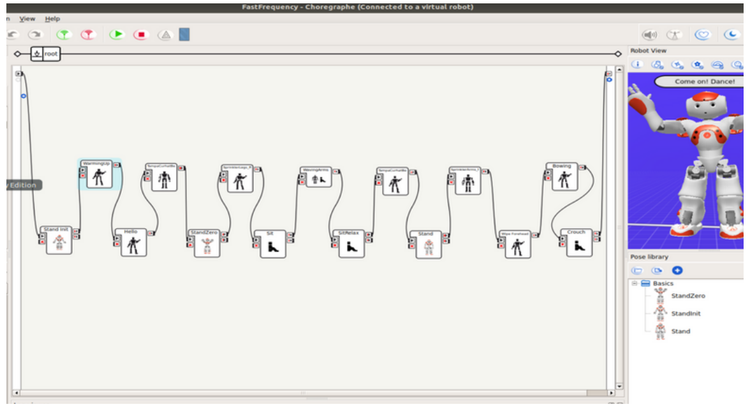


For the creation of the movements, we have used the Timeline panel, displayed double-clicking on a Timeline box.

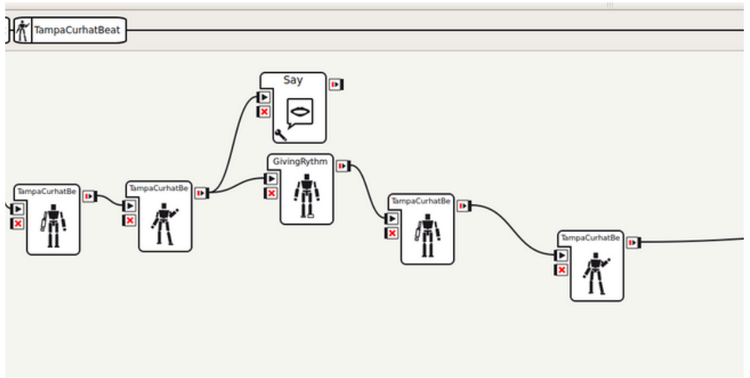
Thanks to the Motion widget we modified the joint values of each limb and we stored them in the keyframe.

Timeline reference

LOOK AT THESE EXAMP-LES

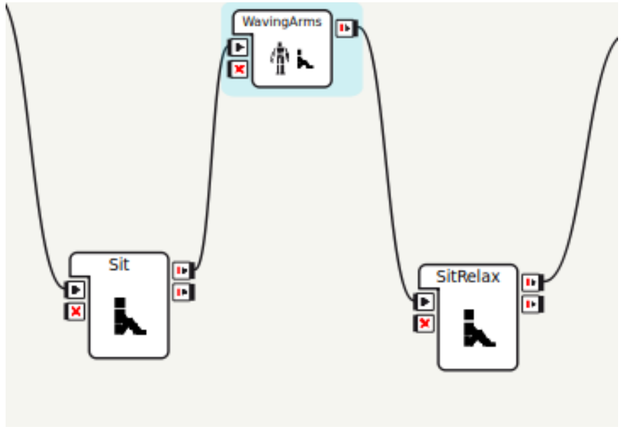


Example 1



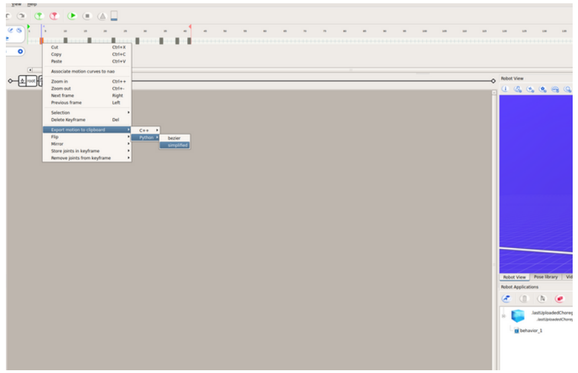
Example 3

Example 2



Exporting into python and linking to Pycharm

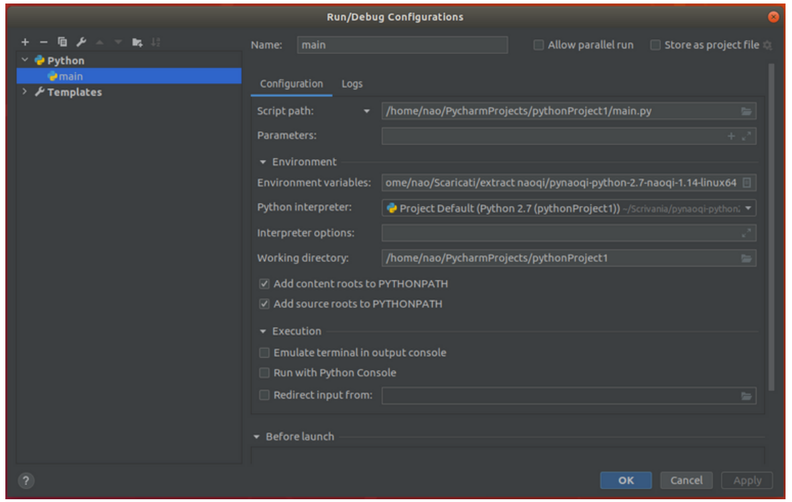
LOOK AT THESE STEPS



STEP1

After creating movements, we should export them to python.

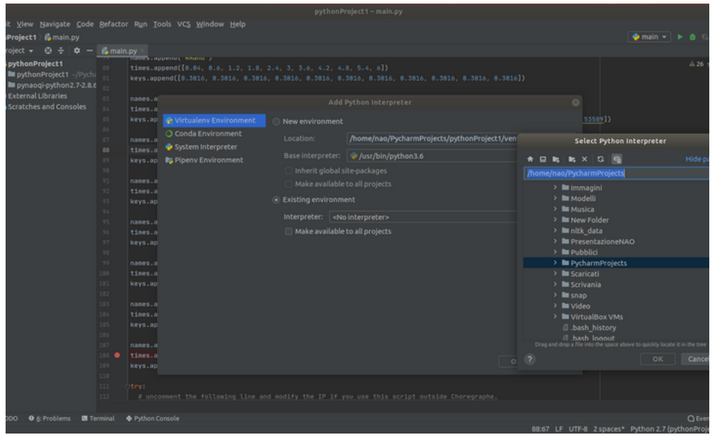
1.click right click on the motion
2.click 'extract motion to clipboard'
3.choose 'python'
4.choose 'simplified'



STEP2

Time for adding new interpreter to pycharm:

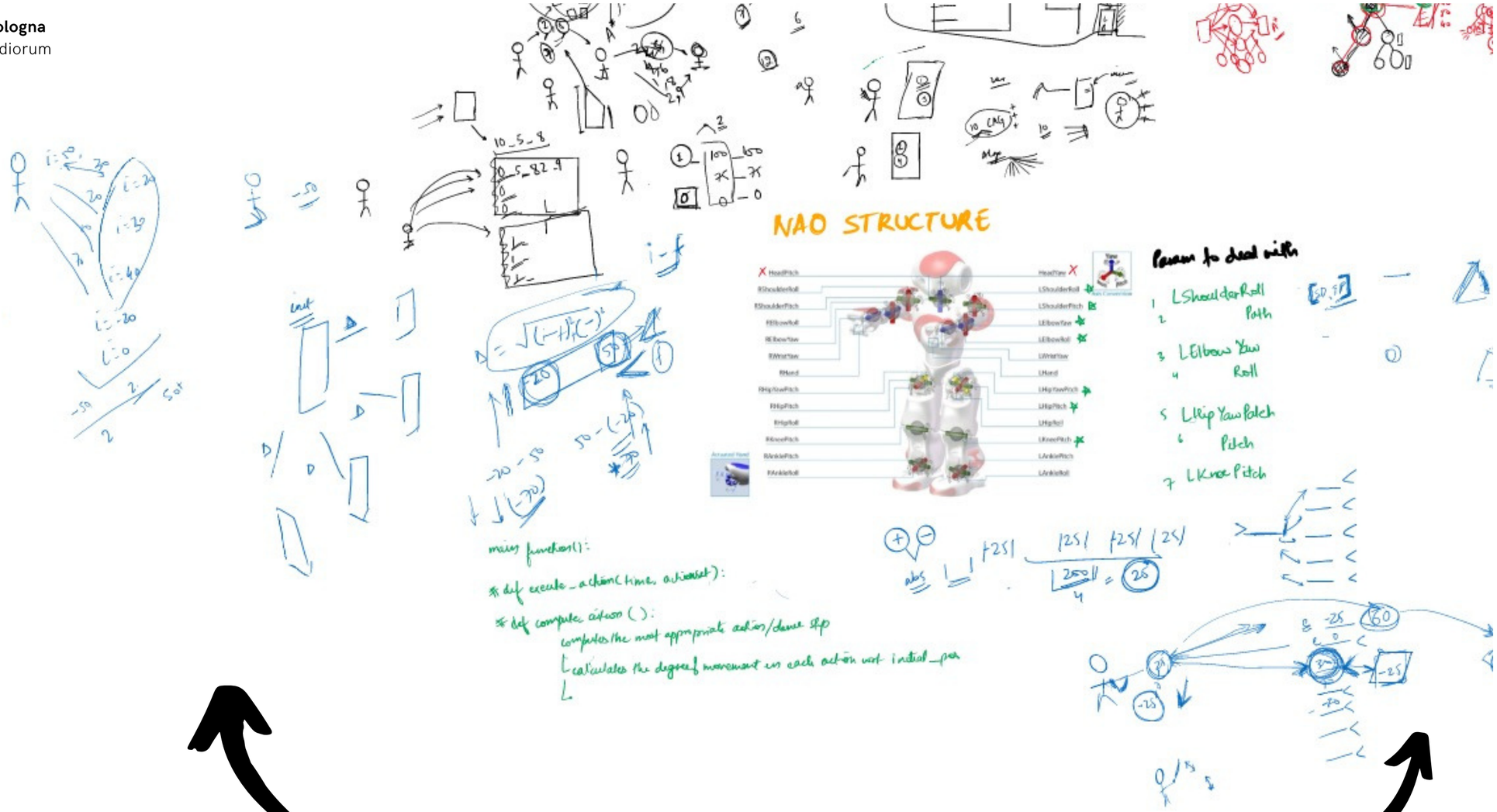
1.click in the right below part of main window
2.choose 'Add interpreter'
3.choose 'Existing environment'
4.choose 3 dots and select python interpreter



STEP3

Adding new path:

1.click 'Edit configuration' in the 'RUN' bar
2.click 'Environment behavior'
3.click '+'
4.add 'PYTHONPATH' to the 'Name'
5.add path of the folder to the 'Value'



Calculating the difference/weight points

In this notepad we decided that which parts of the robots play main role and how we can choose them. Moreover, the calculation of the variables was discussed in the file.

A* star/algo based upon the weight points

```python
import numpy as np
# randomness for the movements
data = list()
data.append(np.std(keys[5]))
data.append(np.std(keys[6]))
data.append(np.std(keys[8]))
data.append(np.std(keys[10]))
data.append(np.std(keys[11]))
data.append(np.std(keys[12]))
data.append(np.std(keys[13]))
print("KeyValue ", data)
keysValue = list()

# ending positions for the moves
tempPositionValue = list()
tempPositionValue.append(keys[5][-1])
tempPositionValue.append(keys[6][-1])
tempPositionValue.append(keys[8][-1])
tempPositionValue.append(keys[10][-1])
tempPositionValue.append(keys[11][-1])
tempPositionValue.append(keys[12][-1])
tempPositionValue.append(keys[13][-1])
print("tempPosition ", tempPositionValue)
finalPositionValue = list()

x = np.hstack(times)
print("time taken", np.sum(times[0]))
```

```python
import o_Waving_Arms, o_unleashing, o_two, o_three, o_tampa02,o_tampa01, o_take_energy, o_snap, o_raising_arms, o_pre_dance

# lists that contains the optionalActions/ MandatoryAction
mandatoryPos = [m_StandInit, m_Hello, m_StandZero, m_Sit, m_SitRelax, m_Stand, m_WIpeForehead, m_Crouch]
availablePos = [o_Waving_Arms, o_unleashing, o_two, o_three, o_tampa02, o_tampa01, o_take_energy, o_snap, o_raising_arms, o_pre_dance]

# this functionn exeutes the action according to the given action file
```

```python
def mainFunctionToRun():
    # get the initial positions
    while len(mandatoryPos) != 1:
        execute_performance(mandatoryPos[0])
        findTheNextNode()
        mandatoryPos.pop(0)
```

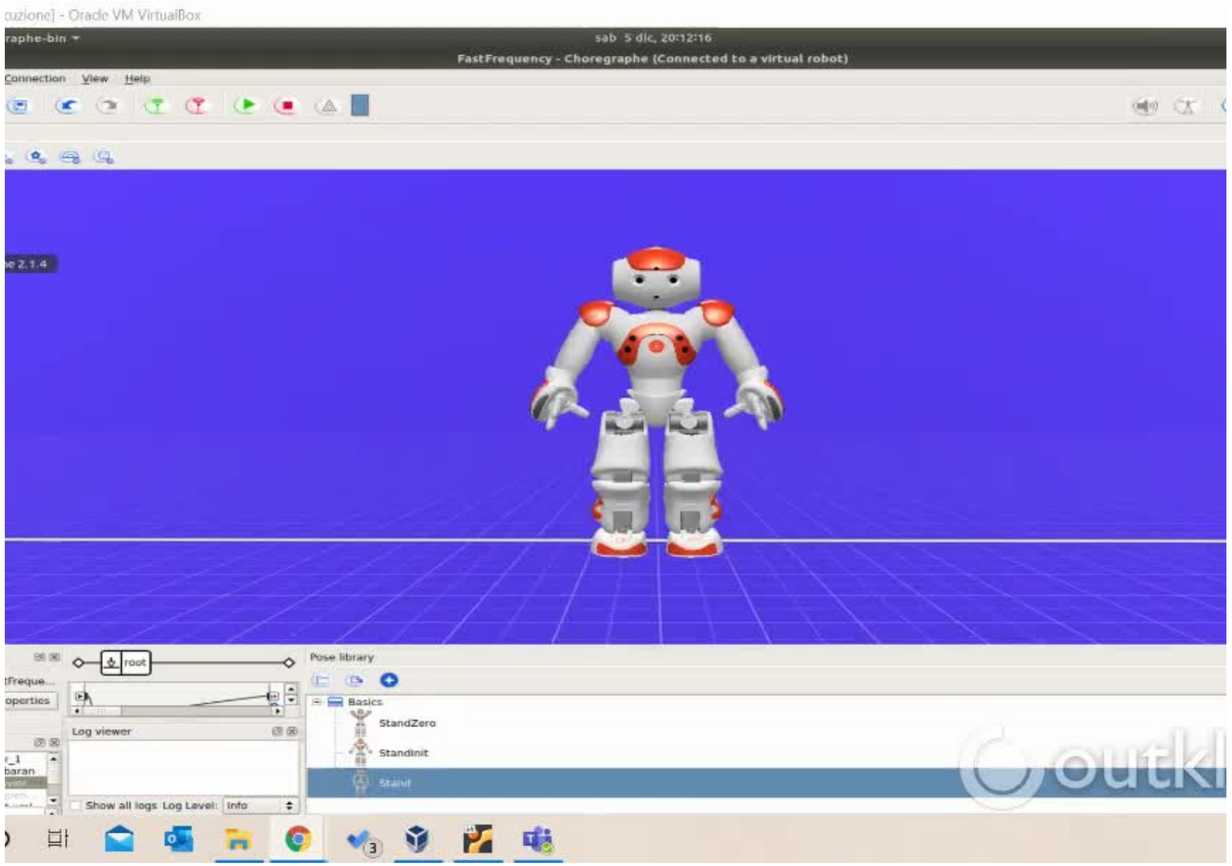
```python
# this functionn exeutes the action according to the given action file
def execute_performance(x):
    try:
        motion = ALProxy("ALMotion", "127.0.0.1", 38165)
        motion.angleInterpolation(x.names, x.keys, x.times, True)
    except BaseException as err:
        print(err)


# calculates the weight differences
def costDifference(coreValues1, coreValues2, coreValues3):
    return np.sum(np.abs(coreValues3-coreValues2) + np.abs(coreValues1-coreValues2))


def findTheNextNode():
    optimum = 100
    index = 0
    # picks up the most efficient position  & index
    for i in range(len(availablePos)):
        cost = costDifference(mandatoryPos[0], availablePos[i], mandatoryPos[1])
        if optimum > cost:
            optimum, index = cost, i
    print("Action chosen at index - ", index)
    execute_performance(availablePos[index])
    availablePos.pop(index)
```

Demo

WATCH THIS VIDEO



THANK YOU FOR LISTENING TO US