

Monash Deep Neuron: Motorola Project:

Ursim Robot arm simulation for mind control pipeline.

Last updated 10/Jul/25

The following project allows the user to control a simulated robot arm using a python script that takes in a json file of streaming commands.

This is advantageous for prototyping uses and is suited at the end of the pipeline.

Step 1: Download + understand Docker

1.1: Get docker from here:

[Docker Desktop: The #1 Containerization Tool for Developers](#)

Docker contains **containers**

Images can exist in containers

Images are made from dockerfile + sample application

1.2: Do these in the learning centre:

You'll learn how to make the container using code and dockerfile:

```
1 FROM node
2 RUN mkdir -p
3 WORKDIR /app
4 COPY packa
```

How do I run a container?

✓ Completed



What is a container?

✓ Completed

These will teach you how to pull and run a container from a code and thus image.

It will not teach you how to access files outside the container, nor how to access the container's results persisting outside the container lifetime. There are other lessons you may watch for that but are not necessary.

However, the docker app just needs to be open for the rest of the steps.

Step 2: Prepare the Ursim container

2.1 - Get Ursim from here

Make a new directory first, Lets say ursim

Inside we pull the image from docker of the simulator.

```
docker pull universalrobots/ursim_e-series:latest
```

2.2 - Prepare the python venv

We make a virtual environment in python for clarity to store and containerise all the necessary packages in line with the docker container, making everything rather neat and tidy.

We also install ur_rtde package here, which allows us to control the simulated robot.

(This venv is called ur_venv)

```
python3 -m venv ur_venv
source ./ur_venv/bin/activate
pip install ur_rtde
```

2.3 - Placing necessary executables

Now we will also be putting two files, they should come attached with this documentation [ur_example.py](#), which is our executable python file,

And [startDocker.sh](#), which is our executable docker container maker from the image we made.

These all go under the main folder hence:

```
>>Ursim
```

```
>> ur_venv
```

```
>> startDocker.sh
```

```
>> ur_example.py
```

2.4 - You can start your docker container now.

Remember to still be in the folder; Ursim.

We use the docker executable for this, recall in some systems, you may need to provide permissions to do this:

Eg. on Linux: `chmod +x startDocker.py`

```
./startDocker.sh
```

If you are curious what is in startDocker commands: it is:

```
docker run --rm -it \
-p 5900:5900 \
-p 6080:6080 \
-p 29999:29999 \
-p 30001-30004:30001-30004 \
-e ROBOT_MODEL=UR5E \
```

```
-e URSIM_ROBOT_MODE=SIMULATION \  
--name ursim_e_series \  
universalrobots/ursim_e-series
```

Should you be successful, you should see the terminal print the following:

```
(ur_venv) erolc@lnx:~/ursim_test$ ls  
startDocker.sh ur_example.py ur_venv  
(ur_venv) erolc@lnx:~/ursim_test$ ./startDocker.sh  
Universal Robots simulator for e-Series:5.22.0  
  
IP address of the simulator  
  
172.17.0.2  
  
Access the robots user interface through this URL:  
  
http://172.17.0.2:6080/vnc.html?host=172.17.0.2&port=6080  
  
Access the robots user interface with a VNC application on this address:  
  
172.17.0.2:5900  
  
You can find documentation on how to use this container on dockerhub:  
  
https://hub.docker.com/r/universalrobots/ursim_e-series  
  
Press Ctrl-C to exit
```

Step 3: Prepare the Ursim container

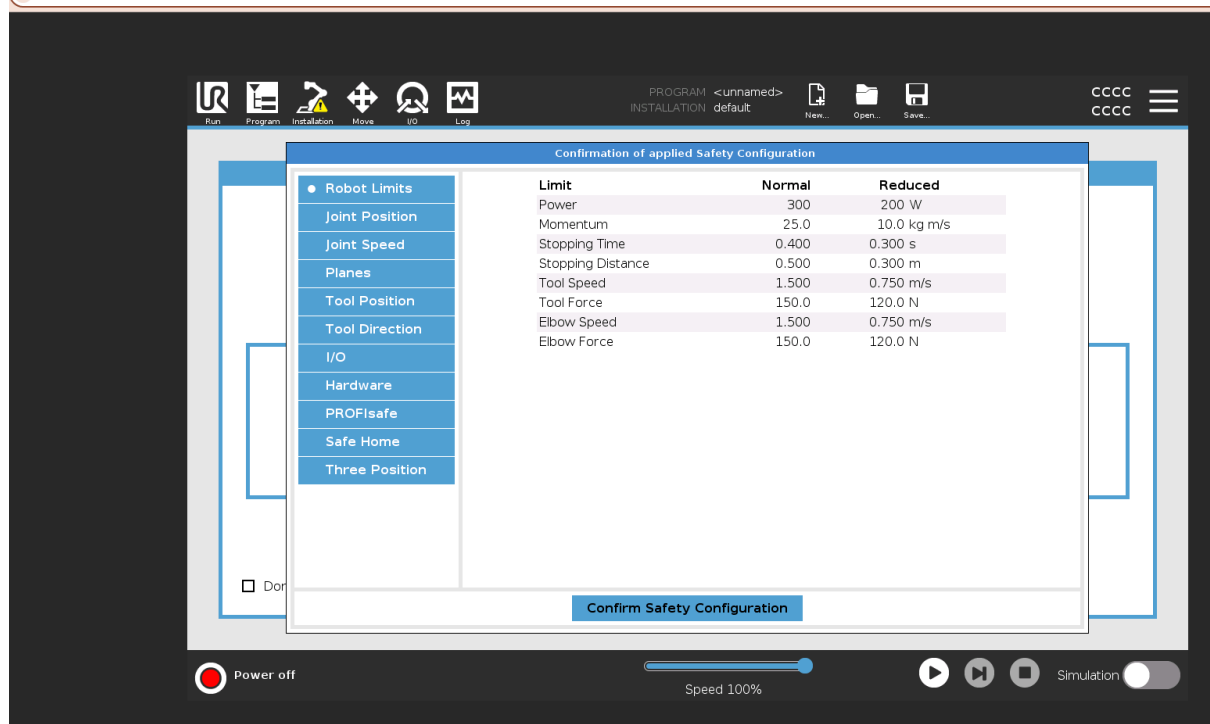
3.1 - Open the simulator:

Simply click on this link:

```
http://localhost:6080/vnc.html
```

Which should take you to here

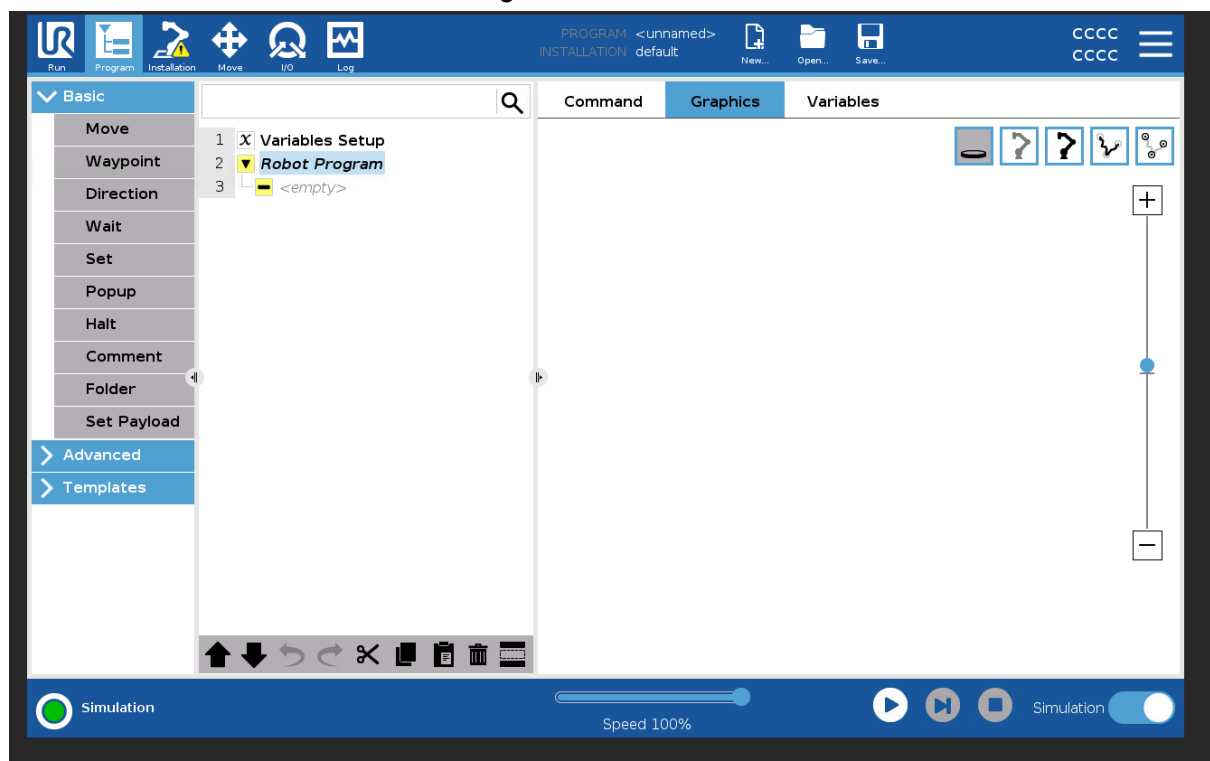
localhost:6080/vnc.html



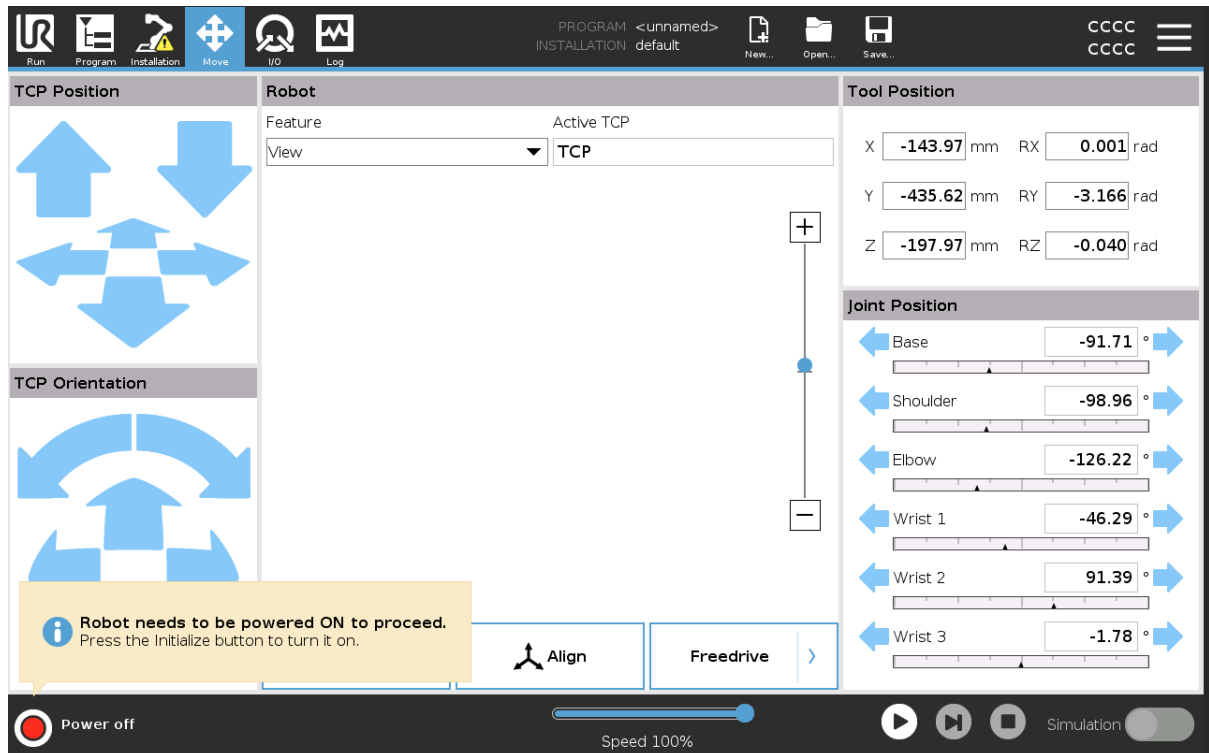
Press [Confirm Safety Configuration]

And then at the bottom right corner, turn ON [Simulation]

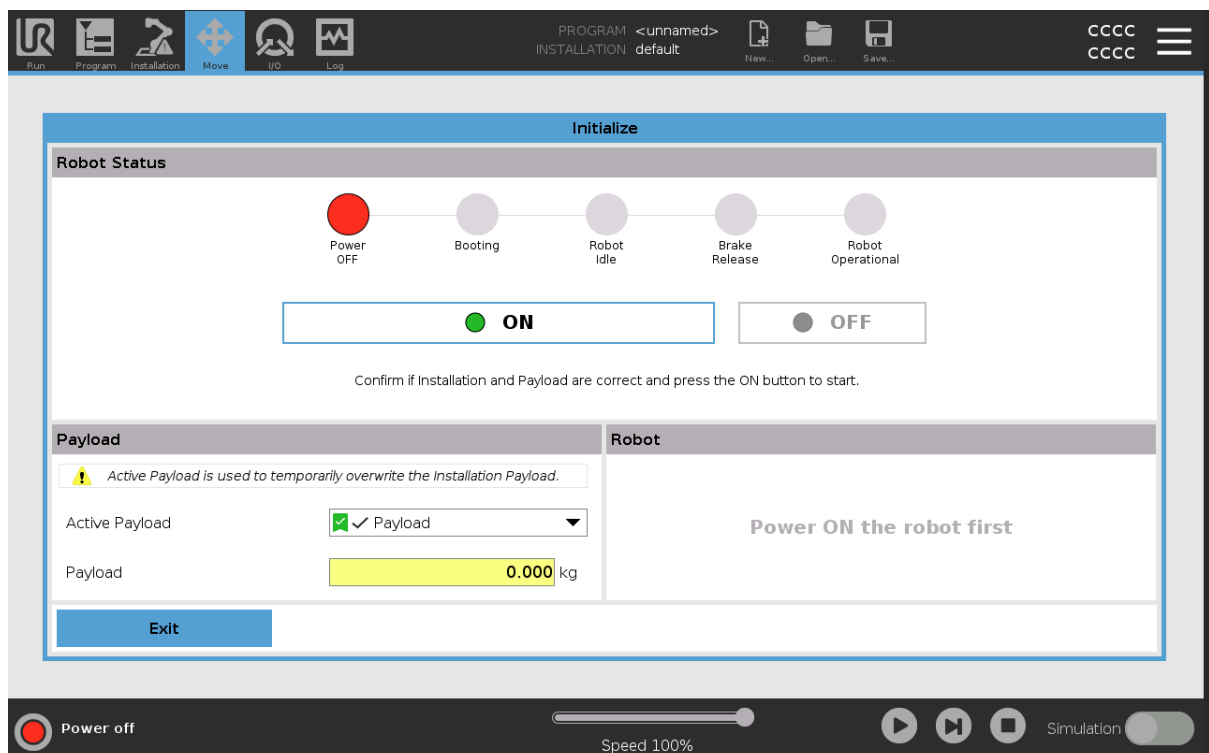
Then go to Program → Graphics: This is where the robot simulation will take place. Notice, however, that there is no robot showing:



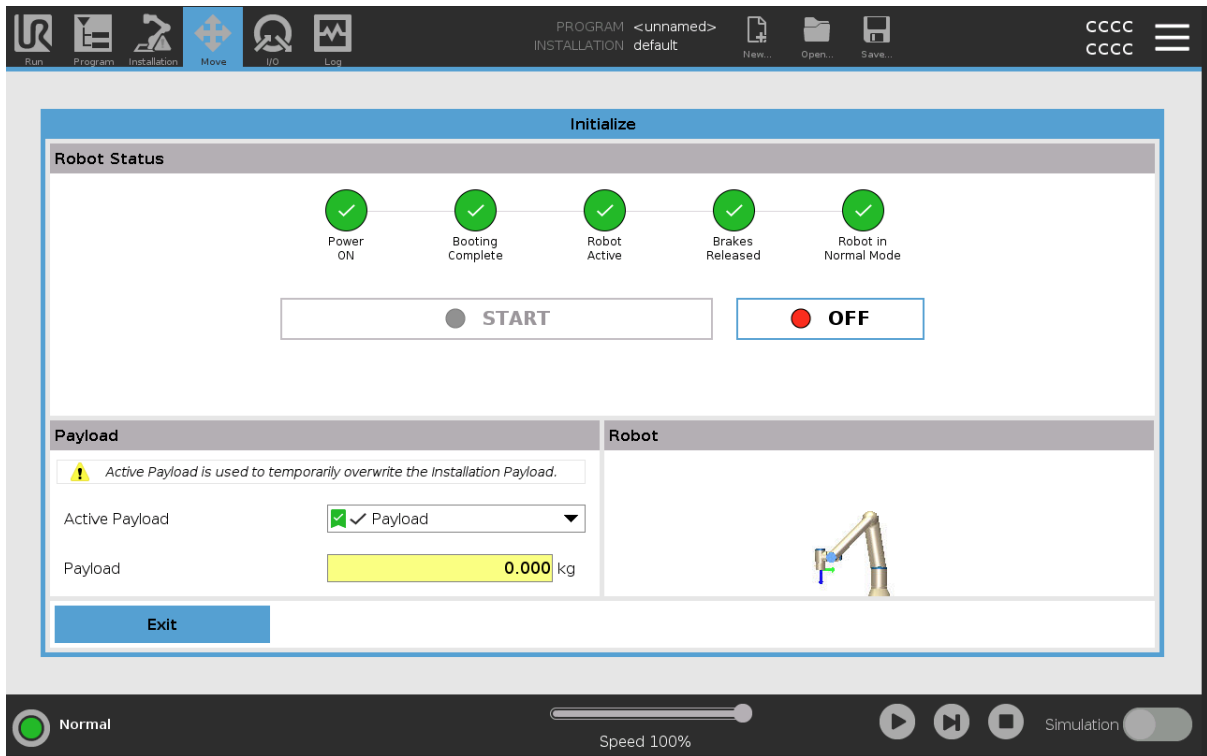
So to activate the robot arm we need a second step: Go to Move on the top left corner.



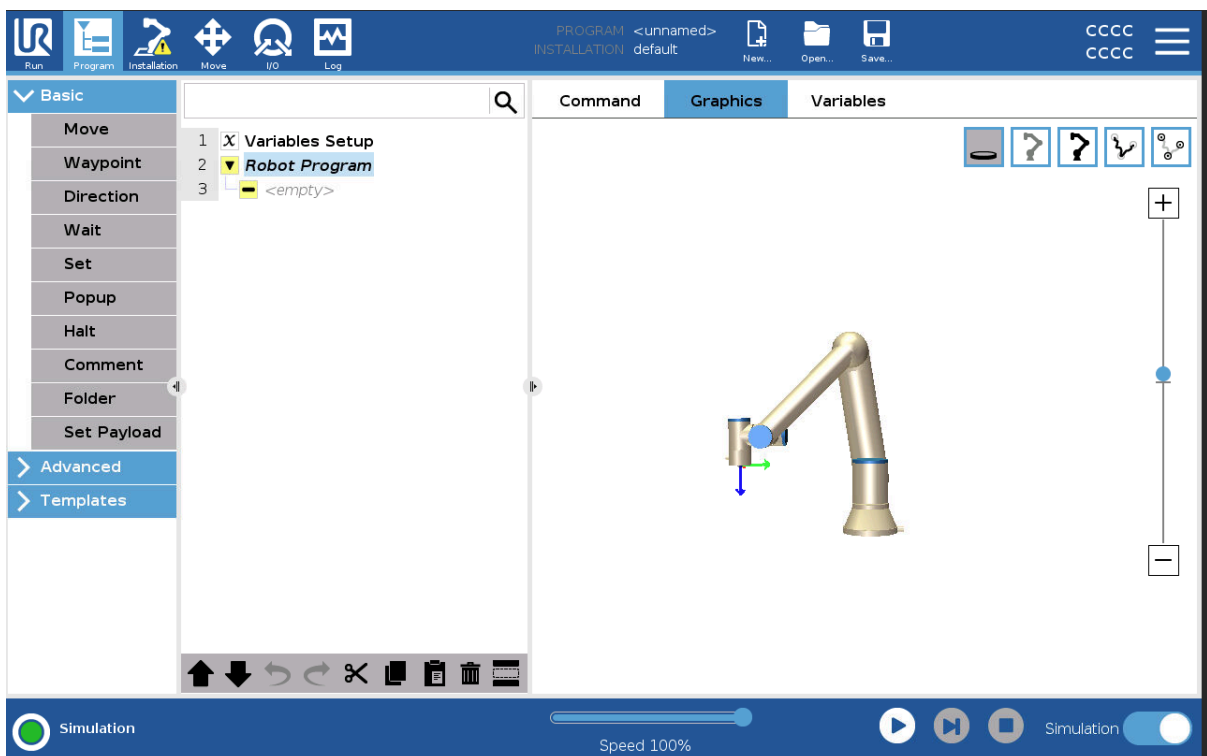
When you go in this menu, you'll notice that there is still no robot in the middle. But you get a suggestion to power ON the robot via the bottom left button: Press that.



Press ON, then press START to take the robot out of the safety brake and allow all the 5 circles to be green; this means the robot is operational and ready



Here, now just press EXIT at the bottom left corner, and then go back to PROGRAM -> Graphics, you should see the robot like the following:



(Just double check that Simulation at the bottom right corner is still turned ON)

Step 4: Run the test python script:

4.1 - Open a new terminal tab in the same directory

The python script is another node that runs concurrently with this docker container, it should ideally run continuously and thus can take streaming data such as from a json file.

We do this by opening a new terminal in the same directory.

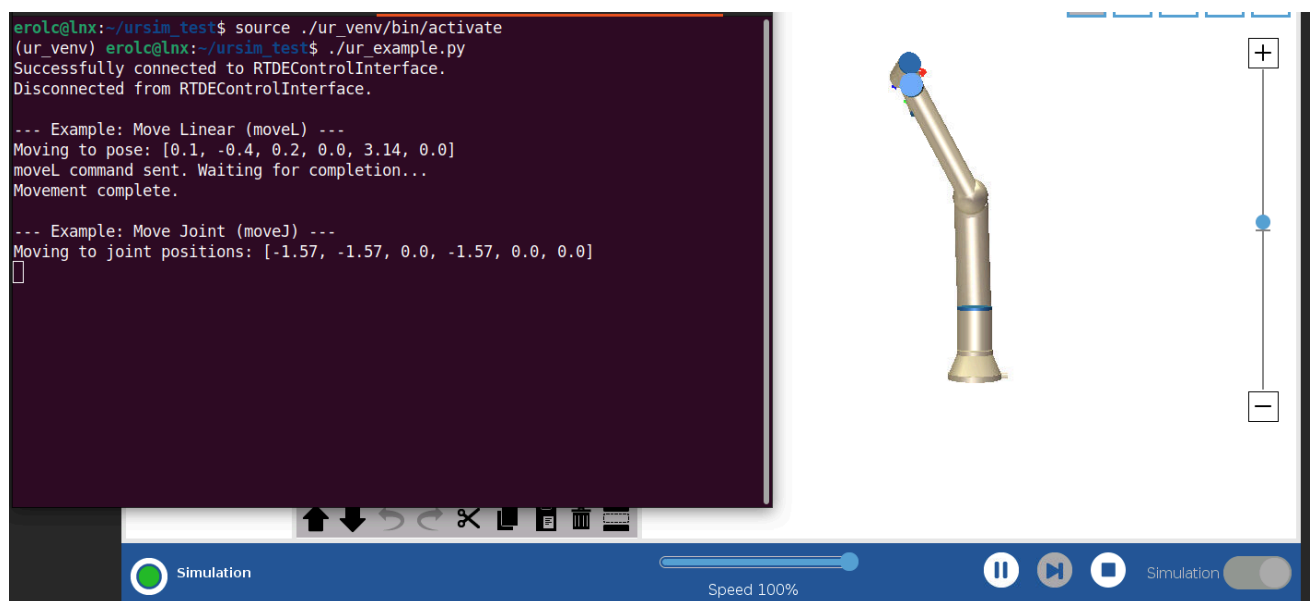
Remember to activate the venv!

>> ursim

```
source ./ur_venv/bin/activate
```

4.2 - Run the python file executable

```
./ur_example.py
```



Opening the website again while this is running, you should see the the robot do a series of movement tests.

Ur_example.py is a testing document to validate that the simulation is working as expected. And simply diagnoses the operational state of this simulation system.

If the system has worked correctly, we can move on with the more interesting python file, **ur_synchronous.py** which will actuate commands based on the stream of json files entering.

Step 5: Operate the arm using json files:

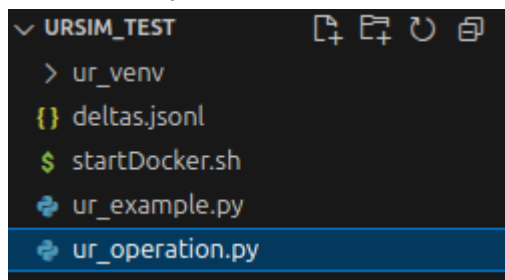
5.1 - Place the operation files

Attached you will also find a python file called

ur_synchronous.py, using that, place under the ur_env directory, same as all the other files we've put so far.

Additionally, there is an example commands input jsonl file for use to test this code called **Deltas.jsonl**

Your directory should look like this now:

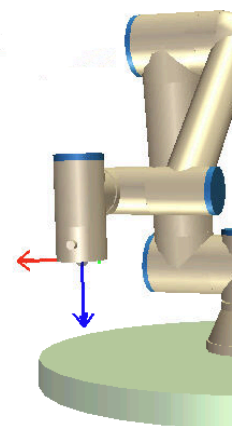


If you investigate the json file, you'll find it simply moves the arm one direction and then back, feel free to modify it to your preference.

5.2 - Operation

```
./ur_operation.py
```

```
(ur_env) erolc@lnx:~/ursim_test$ ./ur_operation.py
Connected to UR at 127.0.0.1
[19:59:01] Streaming velocity Δ/sec = [-0.05, 0.0, 0.0, 0.0, 0.0, 0.0]
[19:59:02] Streaming velocity Δ/sec = [-0.05, 0.0, 0.0, 0.0, 0.0, 0.0]
[19:59:03] Streaming velocity Δ/sec = [-0.05, 0.0, 0.0, 0.0, 0.0, 0.0]
[19:59:04] Streaming velocity Δ/sec = [0.05, 0.0, 0.0, 0.0, 0.0, 0.0]
[19:59:05] Streaming velocity Δ/sec = [0.05, 0.0, 0.0, 0.0, 0.0, 0.0]
[19:59:06] Streaming velocity Δ/sec = [0.05, 0.0, 0.0, 0.0, 0.0, 0.0]
[19:59:07] Streaming velocity Δ/sec = [0.05, 0.0, 0.0, 0.0, 0.0, 0.0]
[19:59:08] Streaming velocity Δ/sec = [-0.05, 0.0, 0.0, 0.0, 0.0, 0.0]
[19:59:09] Streaming velocity Δ/sec = [-0.05, 0.0, 0.0, 0.0, 0.0, 0.0]
[19:59:10] Streaming velocity Δ/sec = [-0.05, 0.0, 0.0, 0.0, 0.0, 0.0]
[19:59:11] Streaming velocity Δ/sec = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Skipping invalid JSON message: Expecting value: line 2 column 1 (char 1)
Skipping invalid JSON message: Expecting value: line 2 column 1 (char 1)
End of JSON source.
Disconnected from UR.
(ur_env) erolc@lnx:~/ursim_test$
```



You should find the arm moving left and then right and then left again.

5.3 - How to modify the json commands file?

How does the jsonl work?

Jsonl is a json file where each line represents an appendable, separate instance, in this case it is an instruction.