Using Jupyter Notebooks with OT2

Introduction

In this tutorial, we will explore how to control the Opentrons OT2 robot in real-time using Jupyter Notebooks. Jupyter Notebooks are interactive computing environments that allow users to create documents that combine code and markdown text which can be run in real time.

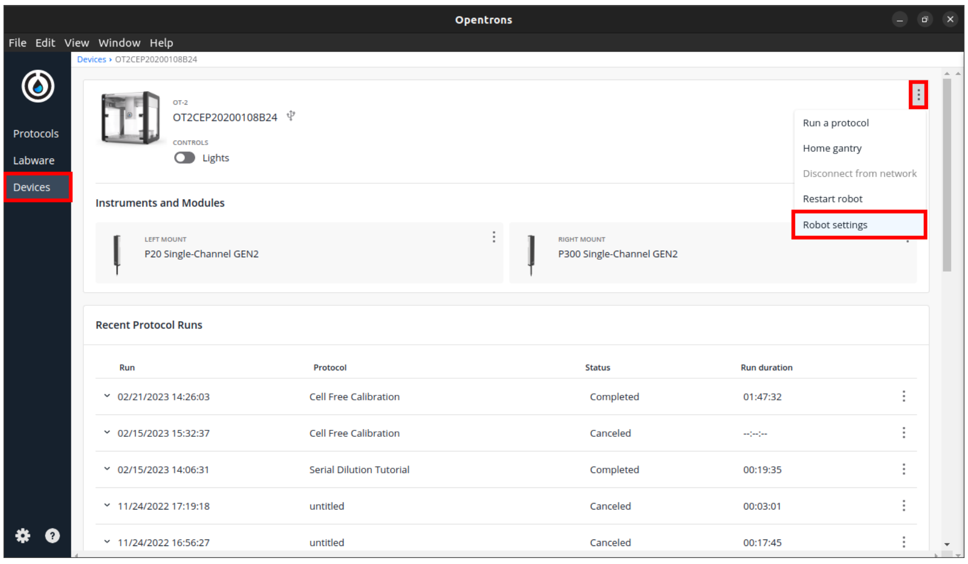
What this means for the use of the OT2 is that commands can be written and run in the robot in real time. This makes for an incredibly quicker method of prototyping experimental steps (e.g. trying to find the best way to pipette a liquid with high viscosity), since you can use the OT Python API in separate chunks. While the protocols written and run on Jupyter Notebooks are more hands-on than just using the Python API, you don't have to restart the entire protocol when a step goes wrong. Additionally, you can have labware calibration steps at any point, making it easier to have multi-step protocols where you need to change the deck configuration or manually handle labware, without having to leave the protocol.

Unfortunately, using this environment has its downsides. This is currently an experimental feature, meaning the way it functions may change in later versions of the OT software and on their next robots. You also won't be able to predict some aspects of the protocol automatically (predicting pipette tip usage, run time, etc) and interact with the Opentrons GUI.

It is recommended to familiarize yourself with the API and python commands first through reading the documentation and following along some of our other tutorials. Here, we will first cover the basics of setting up the use of Jupyter Notebooks on the OT2, for which you should also read Opentrons' official guidance for updated information. We will also cover some tips and best practices on how to write, calibrate, and run protocols in this environment.

Setup

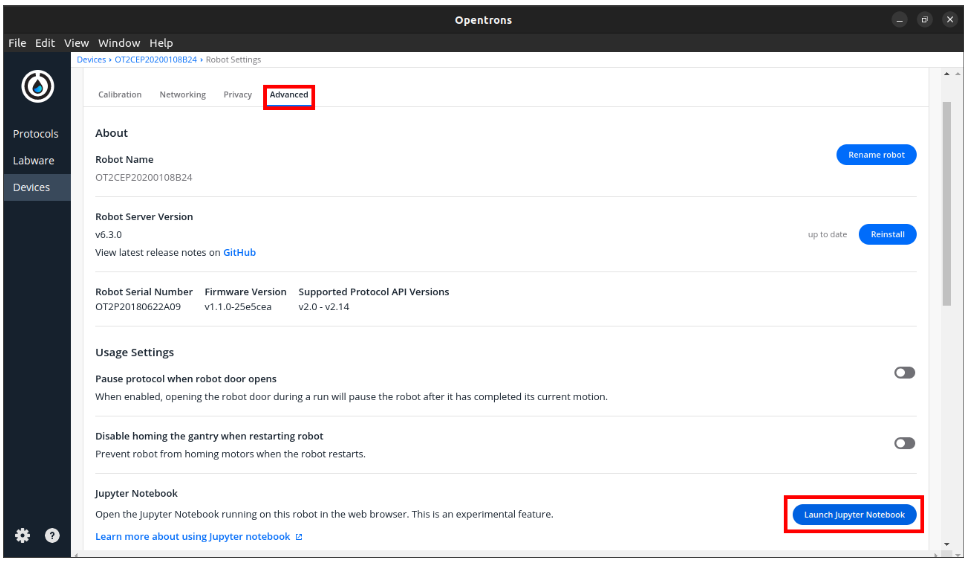
Since the Notebook runs in the Raspberry Pi inside the OT2 robot, Jupyter and basic python modules come pre-installed. To launch Jupyter with the robot, access your robot through the Opentrons app, open the three-dot menu (⁝) in the top right and select "Robot Settings".



Recent Protocol Runs

Run	Protocol	Status	Run duration
02/21/2023 14:26:03	Cell Free Calibration	Completed	01:47:32
02/15/2023 15:32:37	Cell Free Calibration	Canceled	--:--:--
02/15/2023 14:06:31	Serial Dilution Tutorial	Completed	00:19:35
11/24/2022 17:19:18	untitled	Canceled	00:03:01
11/24/2022 16:56:27	untitled	Canceled	00:17:45

Select the “Advanced” tab and scroll down to “Jupyter Notebook”. When you click “Launch Jupyter Notebook”, your standard web browser should open a tab with the Jupyter Notebooks environment.



About

Robot Name
OT2CEP20200108B24

Robot Server Version
v6.3.0
View latest release notes on GitHub

Robot Serial Number	Firmware Version	Supported Protocol API Versions
OT2P20180622A09	v1.1.0-25e5cea	v2.0 - v2.14

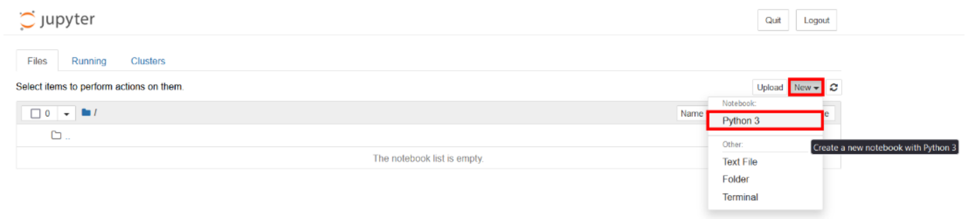
Usage Settings

Pause protocol when robot door opens
When enabled, opening the robot door during a run will pause the robot after it has completed its current motion.

Disable homing the gantry when restarting robot
Prevent robot from homing motors when the robot restarts.

Jupyter Notebook
Open the Jupyter Notebook running on this robot in the web browser. This is an experimental feature.
Learn more about using Jupyter notebook

From here, you can either open pre-written protocols or start writing your own with clicking “New > Python3” on the top right.



Select items to perform actions on them.

The notebook list is empty.

Create a new notebook with Python 3

Before you can send commands to the robot, you first need to get the OT module, the protocol API and declare your labware, like you would in a normal Python-based protocol. The difference here is that you don’t need a “run” function, since you will be running all code in real time. Start the protocol running the cell:

```
import opentrons.execute
protocol = opentrons.execute.get_protocol_api('2.12')
protocol.home()
```

Then declare your labware (modules, plates, tipracks, pipettes) in a new cell:

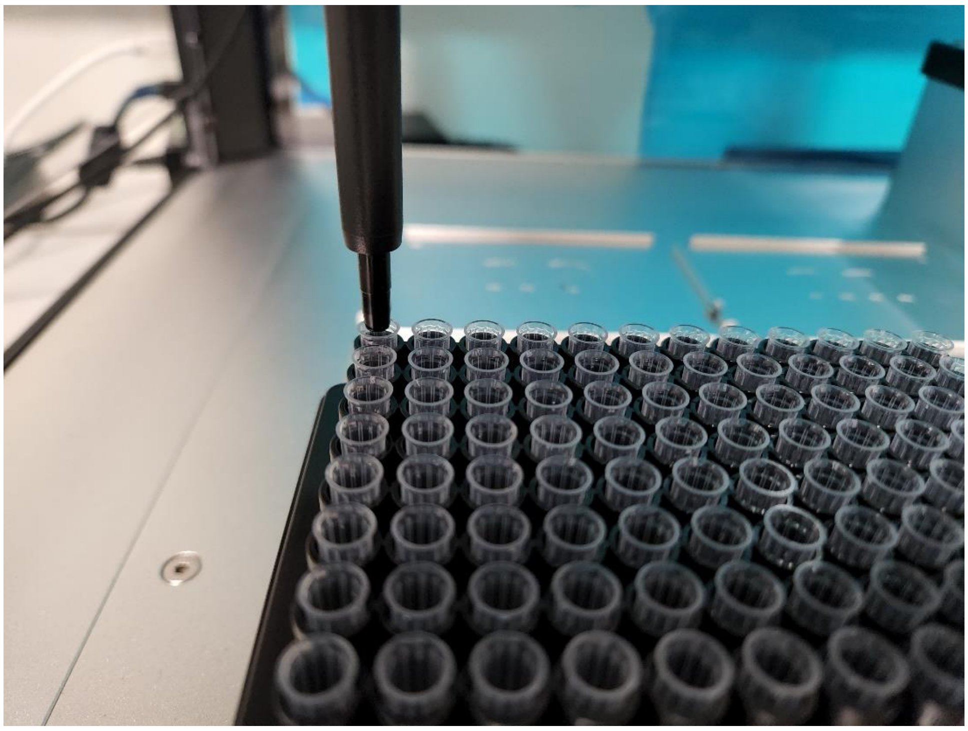
```
samples = protocol.load_labware('opentrons_96_aluminumblock_generic_pcr_strip_200ul', '1')
tiprack_300 = protocol.load_labware('opentrons_96_tiprack_300ul', '2')
p300 = protocol.load_instrument('p300_single_gen2', 'left', tip_racks=[tiprack_300])
```

Calibration

While calibration of labware on Protocol Designer and Python-based protocols is done before each run with the Opentrons app GUI, this is not an option for Jupyter Notebook protocols. For this reason, we have calibration cells for each piece of labware that set an offset and move into that position to test it. This cell's numbers can then be tweaked, and you can re-run the same cell until you're happy with the calibration. We recommend you always calibrate you labware before running a protocol and after any labware is moved while performing a protocol.

At the start of a protocol, you first need to test the offsets of your tipracks. If you're unsure of which values to use (or if you think starting with all zeroes is going to be significantly off), you can run an empty protocol with the Opentrons App and get your initial calibration offsets from that. You can use those values on the cell below or just start with all zeroes:

```
tiprack_300.set_offset(x=0.0, y=0.0, z=0.0)
p300.move_to(tiprack_300['A1'].top())
```

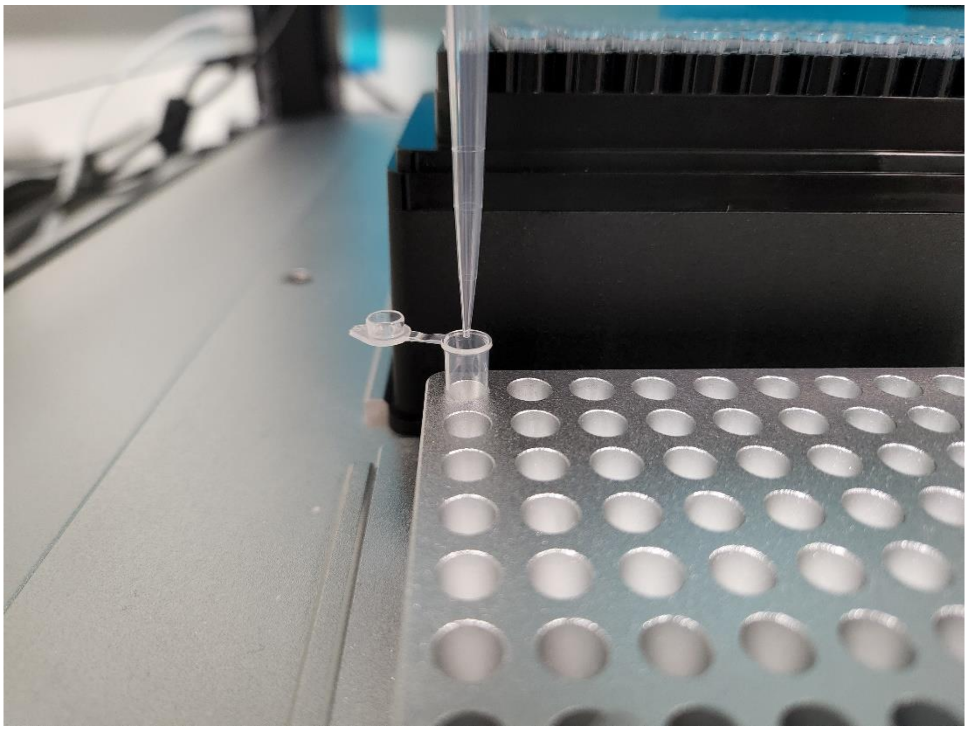


When you're happy with the positioning, test the calibration by picking up a tip:

```
p300.pick_up_tip()
```

If a tip is picked up successfully, you can proceed with the calibration of the remaining labware. For our example, this is will only require a single cell, but you will need more if your protocol requires more plates and tubes. Here the objective is to get the tip aligned to the centre of your well and level with the well's top:

```
samples.set_offset(x=0.0, y=0.0, z=0.0)
p300.move_to(samples['A1'].top())
```



Sometimes a labware definition doesn't accurately represent the dimensions of real labware. This is especially impactful in plates with many rows and columns of wells, and a slight difference in expected-to-real well size adds up as you go along the plate. To test for this, you can optionally add a cell to move to the last well and adjust the previous cell accordingly:

```
p300.move_to(tiprack_300['H12'].top())
```

Once your labware calibration is done, you're ready to proceed with the rest of the protocol. You can keep using the current tip or choose to return it and start from the next one:

```
p300.return_tip()
```

Tips and best practices

Running protocols

Do not rely on the "Run All" commands at the top menu bar. Calibration steps and others may require human interaction and the code will not stop and wait in between cells.

When calibrating labware, always make sure there is a tip attached to the pipette. Without a tip, the robot will interpret the end of the nozzle as it's extremity for the move_to() command and will try to align it with your wells. This can cause bumping with other labware and thus damage the equipment.

If something unexpected happens, it can be useful to use the protocol.home() command to reset the robot to a neutral state (starting position and no tips/liquid).

If you close the Notebook or restart it's Kernel in any way, the robot will forget it's position, any labware and tip tracking. We recommend keeping the notebook open for the duration of the protocol.

Communicating with users

While protocols in this environment can't get GUI messages, you can still communicate important notes and steps to users with Markdown cells or commented lines. To make comments in the code use # prefacing a single line or use triple quotes flanking multiple lines. To change a cell from code to markdown, select that cell and choose "Markdown" at the top menu bar. Refer to a Jupyter Notebooks Markdown cheatsheet for a quick reference on how to write on these cells.

Using custom labware

The Jupyter Notebook can't just pull names from your custom labware list in your OT app like a Python protocol would. Here, you must import the labware files into the OT2 Raspberry Pi and load them into the script with python. We recommend having a folder specifically for custom labware, then you can use the "Upload" button to load any custom labware file into the robot's system:

Then, in your script, you must open that file and load the labware definition:

```
import json
```

```
with open(' custom_labware/custom_96_wellplate_flat.json ') as labware_file:
   labware_def = json.load(labware_file)
   plate_96 = protocol.load_labware_from_definition(labware_def, 1)
```

Note you will have to import the json library at the start of your script and fill in your file's path appropriately (the quotation marks are necessary).