OPTOFIDELITY

OptoFidelity Touch and Test

Scripting Guide

Contents

1 Introduction 3
2 SW Architecture 3
3 Test Script Structure 3
4 Script context 4
5 Test case development 5
6 Gestures and non-linear movements 5
6.1 Tap 5
6.2 Swipe 6
6.3 Drag 7
6.4 Jump 7

1 Introduction

This document defines the concepts and details behind the scripting interface in the OptoFidelity Touch Panel Performance Tester (TPPT). Scripting interface is used for developing product-specific test applications utilizing robot gestures and other movements. Scripts are Python files that are executed by TnT UI. The UI is used to set script parameters and to control and visualize the script execution.

2 SW Architecture

The main test system SW components are described in the picture below. Test scripts define test cases that are configured and executed by the UI. Test cases command the robot, actuators and other HW components via TnT Client which connects to TnT Server. Scripts utilize device drivers to communicate with varying types of DUTs to receive touch events. During test case execution, test parameters and touch results are saved into database, which can then later be accessed by TPPT Analysis to compute statistics and verdicts.

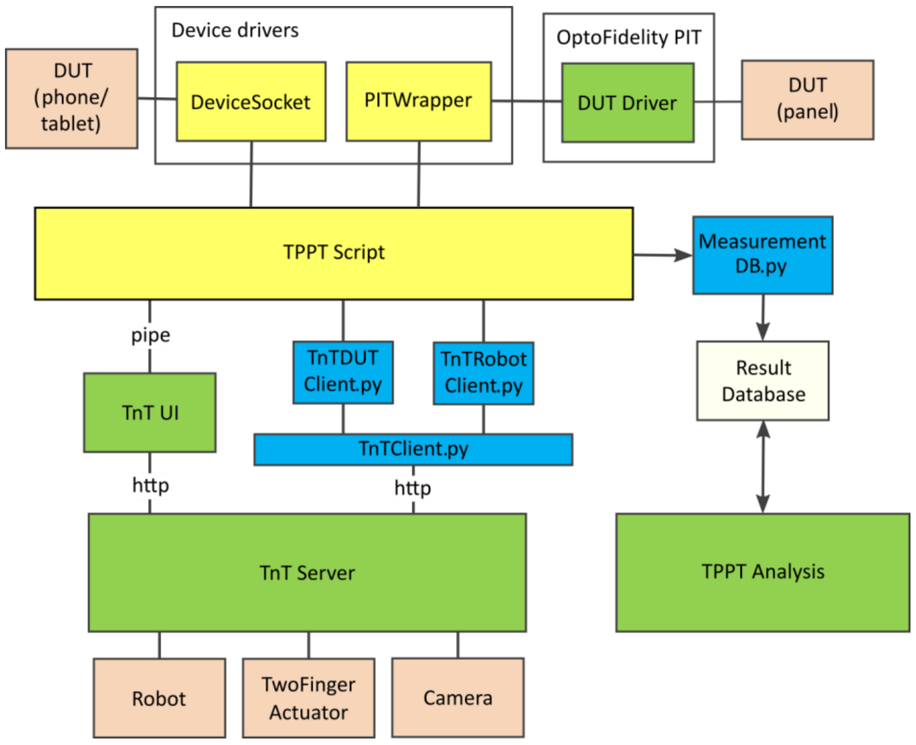


Figure 1: Software components needed for panel performance testing

3 Test script structure

TnT UI can be used to load and run test scripts. The UI provides an environment where user can give inputs to the test script and the script can easily update its progress, status, indicators etc. It offers a good control over the test scripts.

The scripts define a `Context` class that is instantiated when the UI loads the scripts. This context consists of all data required to run test cases including TnT Client object, settings, DUT information, tip information, test cases, database session and device driver objects.

Test scripts define a node hierarchy that is visible to the UI as a hierarchy of widgets such as numeric inputs and checkboxes. Each node can define a list of `Control` objects, which define initial values and valid value ranges for corresponding widgets that are displayed by the UI.

The default script implementation divides nodes under Settings, DUTs, Tips and Tests nodes. Each DUT and tip is then also a node under DUTs node and Tips node respectively. Test cases are also defined as nodes under Tests node.

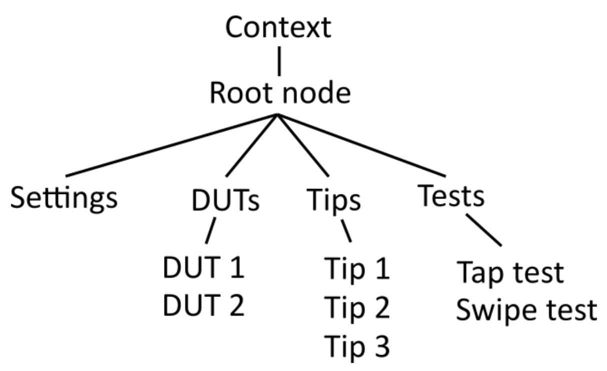


Figure 2: Basic script structure

Test cases are special `TestStep` nodes in the sense that they must implement `execute()` method that is called when the test is to be executed. The test case node class initialization is responsible for defining the controls that show up as widgets in the UI when the script is loaded.

The default script implementation implements `execute()` method in `Context` class that loops through all tips, DUTs and test cases that are enabled by the UI. However, utilizing the existing nodes and by writing new ones, it is possible to construct more versatile test sequences that are represented by more complex tree structure.

4 Script context

Scripts define `Context` class that stores all necessary data and state for running test sequences. The UI creates an instance of this class when the script is loaded and the class exposes methods for communicating information between UI and scripts.

Script developer can easily extend and modify the widgets shown in the UI. The `parameters` member of `Context` class is a list of key-value pairs that appear in the UI as text input boxes. In the default implementation there are parameters Program, Manufacturer, Version, Operator, Serial and Notes. The values of these parameters are saved to the database.

Script developer can also easily expose button widgets in the UI by adding items to `callables` member of the `Context` class. These items are tuples (`function`, `label`) where `function` is a function object defined by script and `label` is a string used to label the button in UI.

The context object is passed down to test case initialization so that test execution can access the context resources such TnT Client object. Context also exposes `indicators` member, which can be used by test cases to update test progress in the UI. Indicators basically constitute an HTML element. For a more graphical progress indication, context exposes `add_dut_point(x, y)` method that can be used by a test case to visualize a touch event.

5 Test case development

Test cases are nodes in the tree hierarchy. They are classes that inherit the `TestStep` class. Test case class must define `execute()` method that is called when test case is executed once sequence is started from the UI and the test case has been enabled. Test case must also implement `visualize_grid(dut)` method that returns `GridVisContainer` object. This method is called when UI is commanded to show points and lines that robot will execute.

Example test case:

```
class MyTestCase(TestStep):
        def __init__(self, context):
                super().__init(“My test case”)

                # Show number input field in UI for this test case node
                self.controls.NumPoints = 5
                self.controls.info[“NumPoints”] = {“label”: “Number of points to tap”}

        def execute(self):
                # Add statements to drive robot, collect touch events and save data

        def visualize_grid(self, dut):
                return GridVisContainer(self.__class__.__name__, (dut.width,
                dut.height), self.create_grid(dut), dut.name)
```

6 Gestures and non-linear movements

This section covers gestures and non-linear movements in DUT coordinate system.

6.1 Tap

Moves finger on top of the given position (#1) and taps the panel once. User can define duration how long the finger is touching the panel in milliseconds.

To tap some distance above or below DUT surface, user can specify “clearance” value in millimeters. Negative clearance taps below surface and it is intended to be used with robots that have spring loaded effector.

When the gesture is completed, the finger is on "Base level" (#4) as it was before the execution (#1).

```
Start   [Base]  --> 1        4 <-- Ready [Base]
                    |        |
                    |        |
                    |        |
                    |        |
                    |        |
        DUT     --> +========+=====
                    |        |
        [Touch] --> 2--------3
                     duration
```

Figure 3: Tap gesture z-coordinate in relation to time.

6.2 Swipe

A swipe gesture is movement along DUT panel between two user defined points. Start point is the position where the finger touches the panel. End point is the one where the finger is lifted again. During the gesture robot performs 'U' character type of movement where the corner radius can be given as a parameter.

At the beginning of the gesture (#1), the finger is moved on top of the given start position. Because of the shape of the gesture the finger is slightly dispositional to compensate the delta caused by the corner radius of the movement. When approaching DUT surface, robot starts to make arch on Hover level (#2). When leaving DUT surface robot arches to Hover level (#5).

The movement ends to the same level (#6) from where the movement was started.

When the gesture is completed, the finger is located on top of the given end point and the finger is lifted to 'Base' level of z-axis.

```
Start   [Base]  -->   1             End -->  6 [Base]
                      |                      |
                      |                      |
                      |                      |
      [Hover] -->   2 | from              to | 5
                       \v                  v/
        DUT     -->   ==+==================+===
        [Touch] -->      `3--------------4´
                              distance
```

Figure 4: Swipe gesture z-coordinate in relation to distance

6.3 Drag

Drag almost equals to Swipe gesture but the movement is executed in three separate steps. First the finger is moved to Base level on top of the contact point (#1), after that it is moved to touch the panel (#2) and then horizontally to the end position (#3) and finally lifted again (#4) to Base level.

```
Start   [Base]  -->   1         End -->  4 [Base]
                      |                  |
                      |                  |
                      |                  |
                      |                  |
                      |                  |
        DUT     --> ==+==================+===
                      |                  |
        [Touch] -->   2------------------3
                           distance
```

Figure 5: Drag gesture z-coordinate in relation to distance.

6.4 Jump

Move robot to the given x, y and z coordinates by first raising the robot head from starting position (#1) to a jump height (#2), which is calculated from the DUT surface (0 = DUT surface). After that robot head is moved to target x,y coordinates (#3) and moved to final z-coordinate (#4). In case jump height is not specified, the robot jump to the maximum height in workspace frame. Jump is intended to be used to e.g. jump from one DUT to another while trying to make sure that the robot does not collide with any obstacles in the workspace.

```
                       distance
Jump height  -->  2------------------3
                  |                  |
                  |                  |
                  |                  |
                  |                  |
      Start  -->  1                  |
                            End -->  4
```

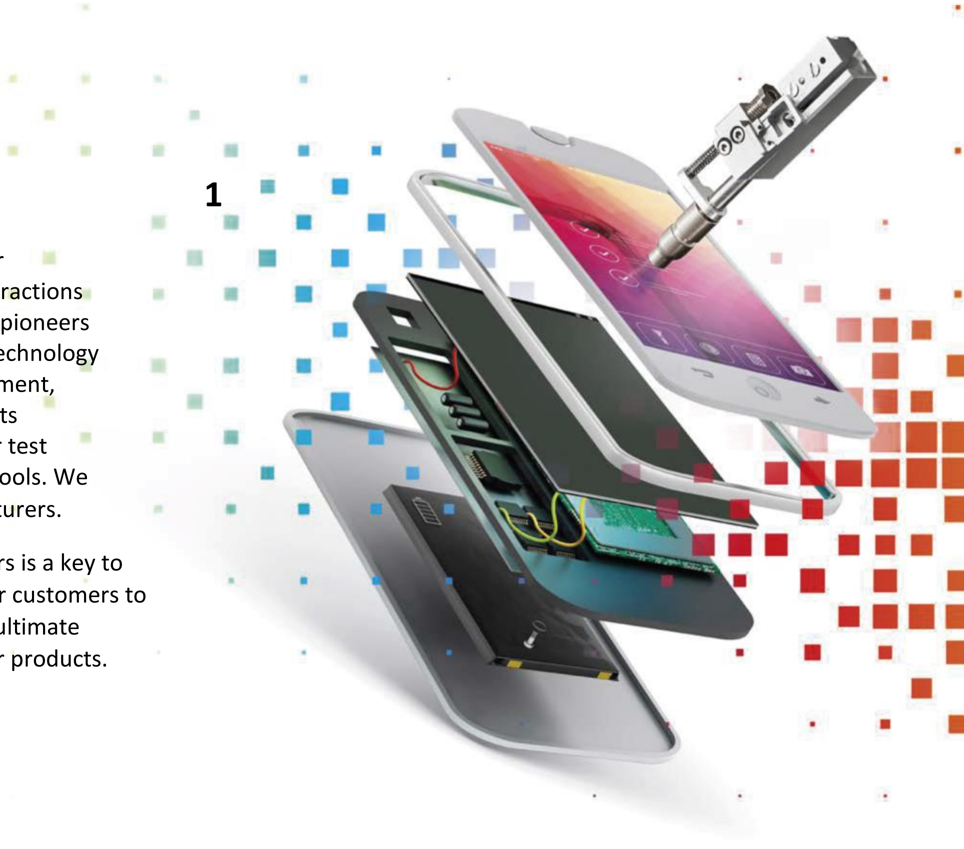
Figure 6: Jump movement z-coordinate in relation to distance.

Change history

Ver.	Status	Date	Author	Remarks
2.0	Draft	20.4.2015	TKU	Updated for the latest TnT SW version.
2.1	Final	29.9.2017	VHE	Updated to new template.
2.2	Draft	22.2.2018	MOK	Updated to tnt_mini compatible
2.3	Draft	18.6.2018	JM	Updated for Robotics platform.
2.4	Draft	2.10.2018	JM	Updated according to new UI sequencer.

OPTOFIDELITY

Who We Are



At OptoFidelity we thrive for the ultimate user experience by simulating and testing user interactions for smart devices. We are globally recognized pioneers in testing, and our humanlike robot assisted technology platforms are widely used in product development, production and quality assurance. Our products are all equipped with easy-to-use SW tools for test parametrizing, results analysis and reporting tools. We work with the world's largest device manufacturers.

Tight and loyal cooperation with our customers is a key to successful test system delivery. We enable our customers to focus on their own expertise, and ensure the ultimate performance, quality and functionality of their products.

Our People

We are a team of multitalented professionals in the fields of test automation, robotics, machine vision, signal processing and software development. 90% of our people have an engineering degree, and 100% of our people have a hands-on, problem-solving oriented mindset.

CONTACT US

WHAT IS YOUR TESTING MISSION?

WE'D LOVE TO HEAR IT!