

Embedded testing -

- From scripts to frameworks



Introduction

- Automation of hardware is sometimes challenging
- Control of mechanical components is often needed
- Reset button is frequently used
- Use of relays is one of the solutions

Automation in automotive industry

- CAN communication is one of the standards used for communication between Electronic control units (ECUs)
- Start/Stop Engine button is a main switch for running a automotive system
- The button has 4 wires: power, GND and the two CAN signals
- Relay is one of solutions to replace the switch




- Relay imitates the push button by making a shortage on certain wires
- There are two modes of a relay – normally opened and normally closed
- Automated control of a relay can be realized using microcontrollers (e.g. Arduino)
- Arduino turns on a relay by setting a digital pin to LOW (0 volts)




- Delay function is used to wait for a certain time before changing the state of a relay
- 500 ms is enough to simulate a pressed button
- Afterwards, a digital pin is set to HIGH in order to release the relay contacts

IoT modules as designs under test

- An IoT module is a small electronic device connected to wireless networks
- IoT devices are often used in extreme conditions
- There is a variety of wireless technologies used for IoT modules, such as 3G, 4G (LTE), 5G
- LTE is divided into different categories, such as LTE Cat 1, LTE Cat M1, LTE Cat NB-IoT etc.

- 
- Most IoT modules do not need 4G speed and 2G is not supported everywhere due to its deprecation
 - Cat 1, Cat M, and Cat NB-IoT LTE modules allow highly efficient use of the current LTE spectrum
 - Each cellular LPWA standard provides different features and capabilities:
 - LTE Cat 1 IoT modules: offer data rates of 10Mbps uplink and 5Mbps downlink


- 
- LTE Cat M IoT modules provide uplink and downlink data rates of approx. 375kbps
 - LTE Cat NB-IoT IoT modules: provide uplink and downlink data rates of approx. 50kbps
 - The designs under test are Quectel's EC25 and BG96
 - The EC25-E (which has embedded Linux) is an LTE category 4 module which delivers maximum downlink rates of 150Mbps and uplink rates of 50Mbps under LTE
 - BG96 is an LTE Cat M1/Cat NB1/EGPRS module, also with embedded Linux


Automation testing of systems based on Linux


- Linux is often used in embedded systems due to its stability, support for multitasking, advanced features, scalability, low cost etc.
- Python is practical for automating the tests on embedded systems based on Linux
- Subprocess library allows us to execute the commands using Linux terminal commands directly from Python
- One example of use is execution of scripts written in C inside the module (similar to Arduino examples)



- `Subprocess.run()` and `Subprocess.Popen()` are suitable functions
- The difference between those two functions is that the first one runs the processes sequentially and the second one can run them in parallel
- `Subprocess.run()` function is ideal for modems when we need to test AT commands

- 
- Every command line for controlling a modem starts with "AT" or "at"
 - Examples of some AT commands: AT+CMGS (Send SMS message), AT+CMSS (Send SMS message from storage), AT+CMGL (List SMS messages) and AT+CMGR (Read SMS messages)
 - The starting "AT" is the prefix that informs the modem about the start of a command line
 - Every AT command has its own execution time

- 
- Subprocess.Popen() function is useful when we need to run the processes in parallel
 - One example of testing a IoT module using this function is when we need to open data connection and at the same time access to the Internet
 - This function is applied for testing Radio Input Layer (RIL) applications
 - There are 6 applications - request_operator, request_get_sim_status, request_setup_data_call, request_radio_power, request_send_sms and request_and_test_data_connection

- 
- All of the applications (functions) should display the logs with the acquired related parameters
 - Testing those functions can be automated using the subprocess library, since they are executed in a console
 - Every other function except `request_and_test_data_connection` can and should be executed with `subprocess.run()` in order to get the results sequentially
 - The function `request_and_test_data_connection` should be run using `subprocess.Popen()` which allows running ping in parallel




- This function is also specific because it is run in infinite loop
- Timer is used to stop the data connection
- After the test is finished, reports are being generated
- The script opens the generated reports, analyzes them for the key words („E_SUCCESS“, 'E_GENERIC_FAILURE', 'modem no response' and 'Segmentation fault') and appends the results at the bottom of the file



- Some of those functions demand an user input, which blocks the automation
- Due to those functions, the script would not be fully automated if it is left without an user interaction
- Input with timeout function is developed for this purpose
- After timeout, a default value is used and the test continues
- Further development of this script will include the option to repeat the tests a specified number of times, all the functions at once or individually

Running the scripts inside the module

- Bash is one of the languages which can be used to automate the tests
- AT commands can be executed via ttyUSB or smd(n) ports
- Background process should be started in order for AT command to be visible on terminal or in a file like in the example below:
- `cat /dev/smd10 | tee /etc/output.txt &`
- `echo -e 'AT+CGMR\r\n' > /dev/smd10`

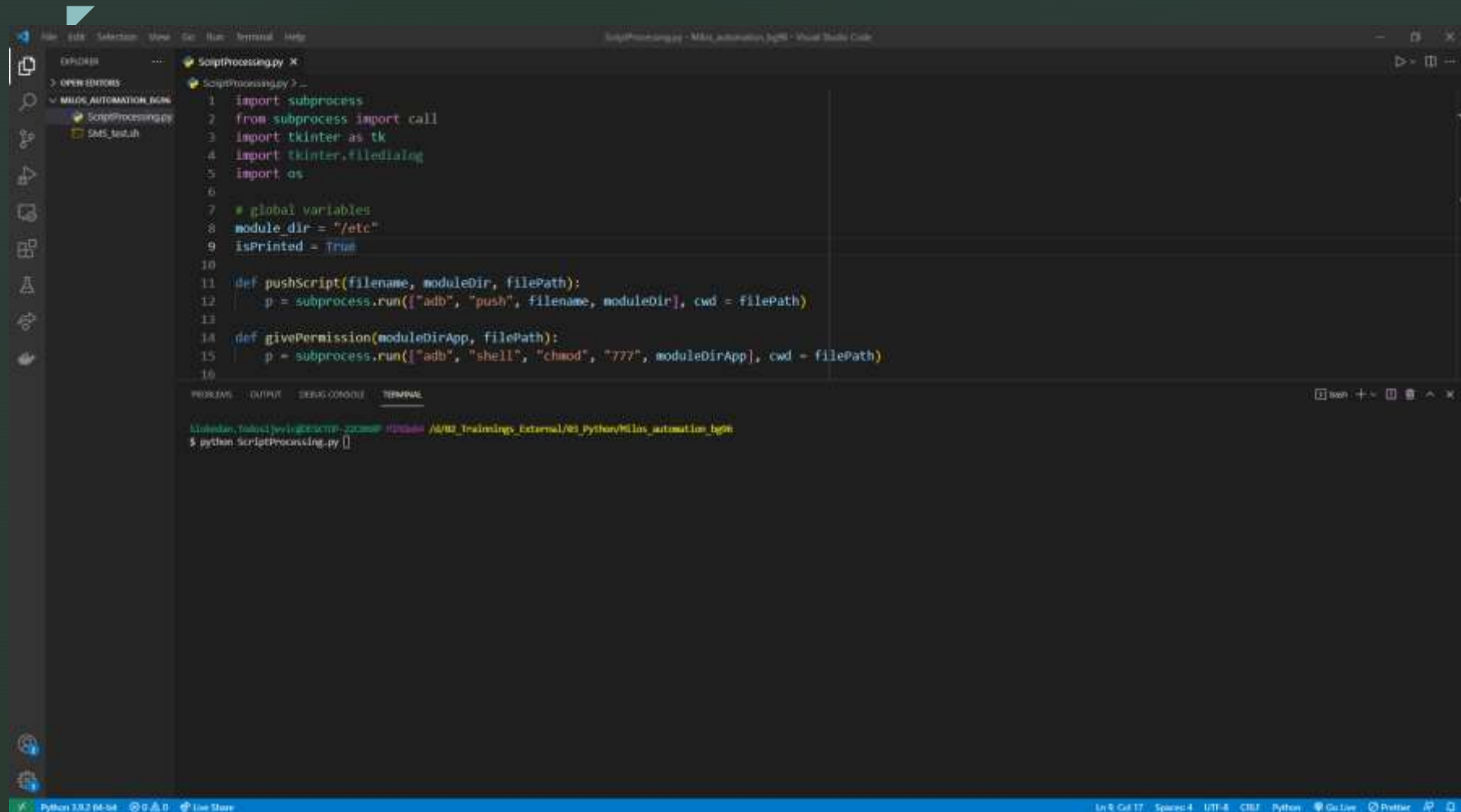
- 
- After every AT command, sleep needs to be executed in order to wait for a response from a modem
 - Response time varies from command to command – typical value is 300 ms
 - When a modem needs to connect to some server using e.g. TCP, a response time is unpredictable
 - If a response is not returned in a defined time, test fails



- Writing a bash script is the first part of automation
- The second part is uploading the script to a module
- Android Debug Bridge (ADB) is used for communication with the devices such as Quectel's EC25 and BG96
- ADB (Android Debug Bridge) is a command-line tool that allows a communication with a device
- Python subprocess library is used to execute the ADB commands



- All the commands for manipulation with the bash script can be executed using `subprocess.run()`
- Example of the syntax:
- `subprocess.run(["adb", "push", filename, moduleDir], cwd = filePath)`
- The automation process looks like this:
- ADB shell is opened using subprocess, a script is pushed via `adb push`, permission is given with `chmod`, script is executed like `./script.sh` and an output file is pulled to the PC with `adb pull`



```
ScriptProcessing.py
1 import subprocess
2 from subprocess import call
3 import tkinter as tk
4 import tkinter.filedialog
5 import os
6
7 # global variables
8 module_dir = "/etc/"
9 isPrinted = True
10
11 def pushScript(filename, moduleDir, filePath):
12     p = subprocess.run(["adb", "push", filename, moduleDir], cwd = filePath)
13
14 def givePermission(moduleDirApp, filePath):
15     p = subprocess.run(["adb", "shell", "chmod", "777", moduleDirApp], cwd = filePath)
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Linux: /home/.../python3.8.2-64-bit
$ python ScriptProcessing.py
```

Figure 1. Uploading the bash script to the module and pulling an output file

- Example of the stress test:

```
#!/bin/bash
for ((i=0; i< 1000; i++))
do
echo "\n\n"
echo "Run number: $i"
echo "\n\n"
sudo QFirehose -f .
dmesg
adb wait-for-device devices
echo "\n\n"
echo "ADB device booted successfully"
echo "\n\n"
Done
```

Figure 2. Stress test example


- The idea behind this test is to detect faulty memory blocks by repeating the flashing process many times

Test automation framework for non-Linux systems

- For testing the modules without embedded Linux, more general approach is needed
- The base of this framework is the Pytest framework
- Configuration file defines which functionality will be tested, a command which runs an image burning script and a number of test repetitions





- Decorators, which are part of a pytest framework are used to run, repeat, skip and enable certain scripts and functionalities
- Input file is mandatory for the framework
- The older version of the framework used a .csv input file with the next columns: input AT command, expected result and timeout
- Functioning principle is a comparison of the output result with the expected result

- 
- The easiest part of automating the tests for the modems is the execution of simple AT commands which do not demand a user input
 - Examples are AT+COPS?, AT+QCFG=? And AT+CMGF=1
 - Exception is a command AT+COPS=?
 - Such commands are sent via serial communication in a straightforward manner



- There is a set of commands which demands user interaction
- A special tag is put in a csv file before such AT commands in order for the framework to parse them differently
- This tag is in a form of [DATA], which means that x1A character has to be put at the end of the input in order to send a message
- x1A represents Ctrl+Z in ASCII and serves as a termination character

- 
- The second important tag in a csv file is [FILE] which is put before a command for uploading a file to the module storage
 - There is a special algorithm for a file upload, hence, a different tag is needed
 - In order to start the test, starter.py script is run along the arguments which represent path to the test script and a serial port
 - Some of the flaws of csv is its readability and a lack of possibility to add special attributes to certain test cases, which is solved using yaml configuration files


- 
- The new version uses yaml file for an input
 - Some of the properties of an input file are the AT commands, chained property, data, termination exit etc.
 - Regular expressions are used for the expected result of certain commands which have URC's with variable size
 - AT commands with such URC's have unpredictable responses and regex is a good way to solve the matching of the results



- The first property of a yaml file are the AT commands
- Framework uses a while loop which reads the response from the module after the AT command is executed
- While loop is terminated once the timeout expires or a response is received
- Execution time is saved this way



- The next property used in yaml file is called **chained**
- This property is used in the functional tests where the execution of the latter commands depend on the former
- In case of failure of opening a connection to a server, no following commands in that chain will be executed and time of test execution will be preserved

- 
- Many test scripts have their own configuration file
 - The example command for a start of an automated test:
 - ```
python starter.py -p testnew/test_mqtts.py
/dev/ttyUSB0
```
  - The examples of some of the test runs:



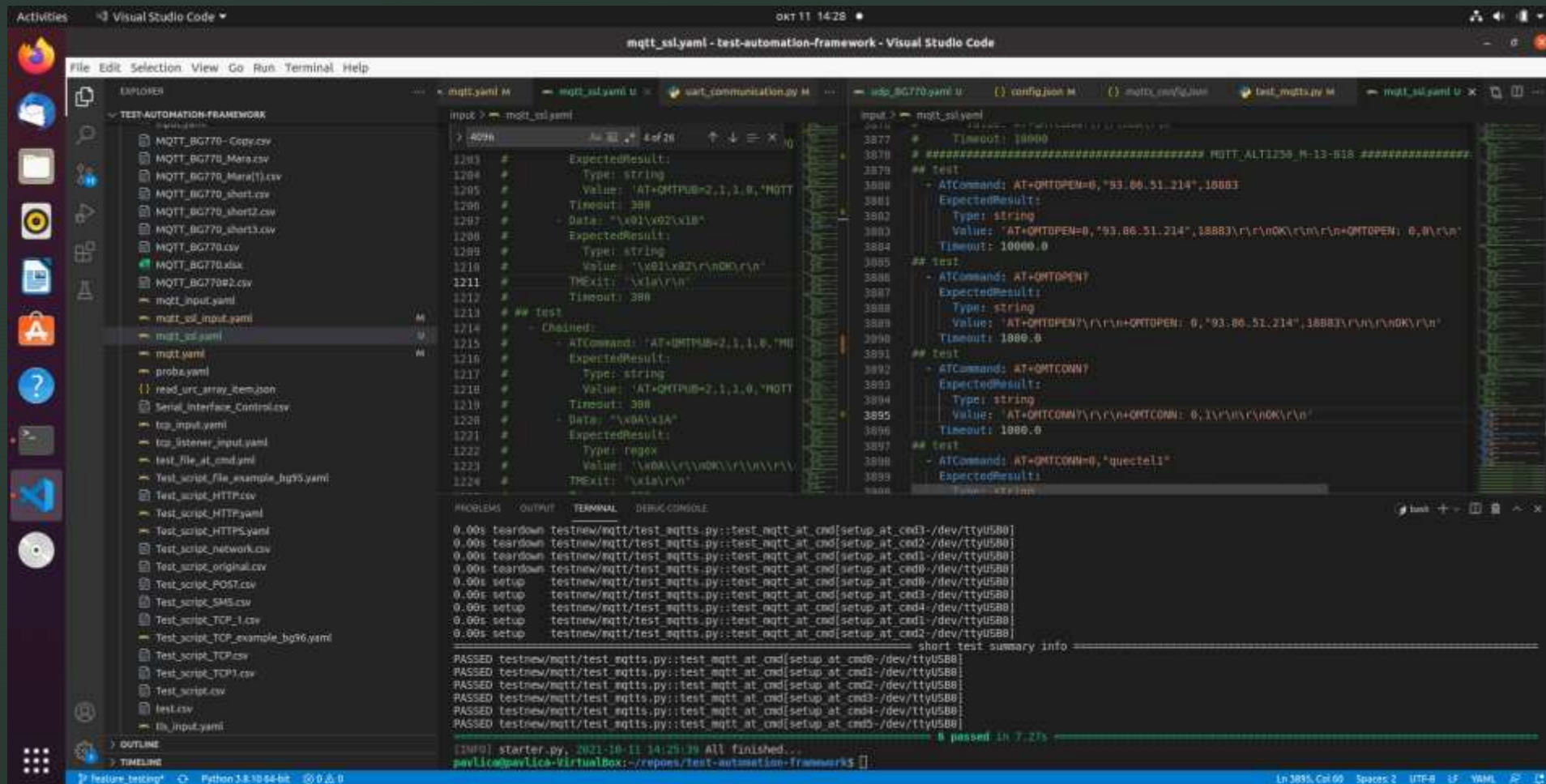



Figure 3. Example 1 – Running the framework





- 
- AT commands which demand a user input are a bit more complicated
  - TCP is one of the features which demands a user input
  - Termination character must be put at the end of every message
  - TCP has three different modes – buffer mode, direct push and a transparent mode – different termination characters are used for them



# Future improvement

- Development of GUI
- Report
- Integration with Jenkins



- Thanks for your  
ATtention

