AUTOMATED TEST GENERATION TOOL

INSTALLATION AND USER MANUAL

Juan José Guerrero Ruiz

Departamento de Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
Málaga, Agosto de 2022

Content

1. Connecting to Dofbot

This section describes how to connect a Dofbot machine to a computer with Cordis' MachineControlServer installed.

1.1. Connect Dofbot to the Internet

First, we need to connect Dofbot to the same net the computer is using. To achive that, we must follow these steps:

1. Generate a QR code with the corresponding net credentials.
2. Turn on Dofbot and press Button 1 to turn on the camera and make it expect a QR Code.
3. With Dofbot ready, put the QR code in front of the camera and wait for it to recognize it.
4. When Dofbot beeps it should be connected to the Internet. This can be easily checked by looking if an IP address appears on Dofbot's screen.

1.2. Compiling and Loading our Model to Dofbot

Once our Dofbot is connected to the same net our computer is, we need to compile our UModel model and load it into the robot.

1. On Altova UModel, get to CordisModeler/Configure Target menu, select **C#** on drop-down "Target" and write the address in which we want the C classes on "Generation Folder".
2. On Altova UModel, select **CordisModeler/Generate Code**.
3. unzip the compilation program (dotnet-mcp).
4. Copy the generated files on step 2 into **dotnet-mcp/Source/ProjectSpecific/GeneratedCode**.
5. Open .NET project on **dotnet-mcp/Build** in Visual Studio.
6. On the project tree, right click ProjectSpecific dependencies -¿Add Project Reference... -¿Examine... and add **DofbotArm.dll**
7. On the toolbar, Compile -¿Compile Solution. Solve any sintax errors on the code if needed.
8. Execute command **dotnet publish -c Release -r linux-arm64 -f netcoreapp3.0 –self-contained true "MCP .NET Core.sln"**
9. Using WinSCP, connect to Dofbot to copy project "linux-arm-64ïnto it, generated on **dotnet-mcp/Delivery/Bin/Release/netcoreapp3.0.** Password for connecting to Dofbot is **"yahboom"**.

10. Open Properties on file **linux-arm-64/publish/Cordis.Mcp.Executeable** and grant it all possible permissions.

11. On **linux-arm-64/publish/Settings.xml** change the OpcAddress from 127.0.0.1 to the one on Dofbot's screen.

After these steps have been completed, we should have succesfully loaded our model to our Dofbot.

1.3. Connnecting to Dofbot via MachineControlServer

The last step to be able to start executing tests on our Dofbot is connecting to it via Cordis' MachineControlServer. To do that, we will follow these steps:

1. Using any browser, connect to the IP address assigned to Dofbot. Username is **"dofbot"** with password **"yahboom"**.

2. Once inside Dofbot, execute command **"sudo ./Cordis.Mcp.Executeable"** on **Documents/linux-arm-64/publish**. Password is**"yahboom"**. Dofbot should initialize, testing its movement before putting itself on vertical position. From this point onwards the robot is ready to receive information.

3. Go to MachineControlServer's installation folder (should be **"C:/Program Files (x86)/Cordis/MachineControlServer"** by default) and change **IpAddress on "Controllers.xml"** to Dofbot's assigned IP address.

4. Run MachineControlServer as administrator. It should connect to Dofbot smoothly. From this point onwards you can send information to Dofbot.

IMPORTANT: Having MachineControlDashboard or WinSCP connected to Doffbot while trying to execute commands through MachineControlServer may cause conflicts and force a reset on Dofbot.

2. Instalation and Use of the Unitary Tests Generator

This section explains how to use a GUI to create and run unitary tests on Dofbot.

2.1. Instalation and Dependencies

In order to install the unitary test generation you only have to unzip **UnitTestGenerator** file onto a folder. Running **UnitTestGenerator.bat** estarts the GUI. The program is developed on python and requires the following packages:

- numpy
- random
- xml
- tkinter
- argparse
- os
- time
- pytest
- shutil
- sys

2.2. How to use

2.2.1. Input file

When executing the program, we are greeted with a folder navigation window asking for a XML file. This file must be an export of our Dofbot model from Altova UModel, using the option CordisModeler -¿Generate Export. We must make sure the following options are set on CordisModeler-¿Export Options... before exporting our model:

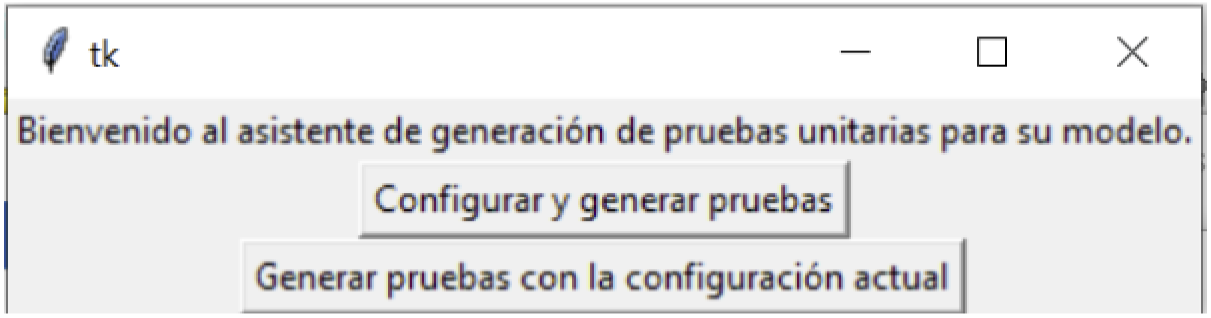
- Export Language - XML
- Scope - Full Model
- Mode - Classes
- Generate standard components

After properly importing our XML to the program, two new files are created on the root folder. APITree, which contains every route from the root to every class on the model, and Dofbot API, which contains every class, operation and input parameters, including bounds and types of said parameters.

2.2.2. Main Menu

After the import, we are presented with two options:

- **Configure and generate tests**: This option will take us to the Operation Configuration Menu, where we select which operations we want to test and assign its pre and post conditions. This step is mandatory the first time using a new model.
- **Generate tests using current configuration**: This option will take us directly to the Test Configuration Menu, without the operation configuration phase.

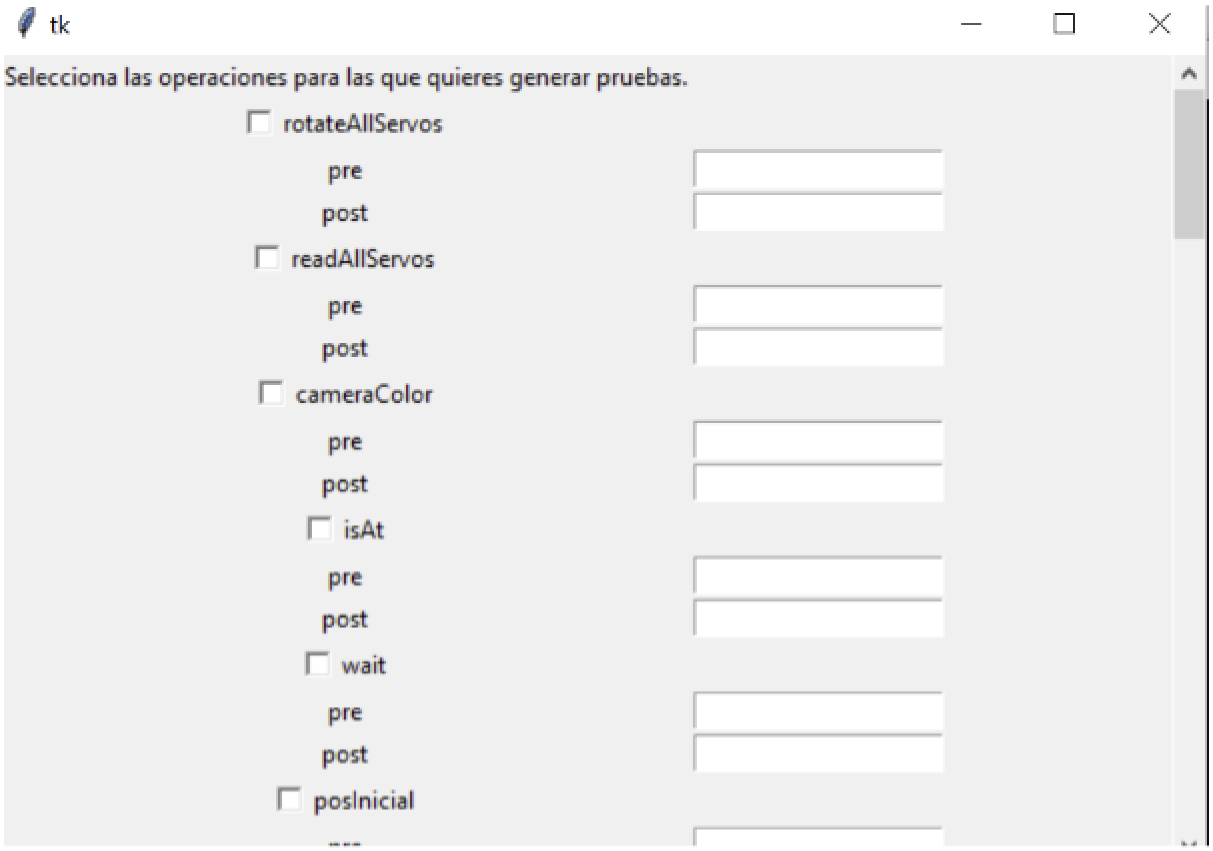

tk
Configurar y generar pruebas
Generar pruebas con la configuración actual

Bienvenido al asistente de generación de pruebas unitarias para su modelo.

Figura 1: Main Menu.

2.2.3. Operation Configuration Menu

On this menu, the user can tick the boxes of the operations he wants to test. They will be included on the test battery generated. Every operation has two blank fields below for the user to write its pre and post conditions. These can be either a boolean expression or another operation from the model. if left blank, pre and post conditions values will be true by default. After pressing the accept button, an output file will be generated. It will contain the API corresponding to the selected configuration. Then, te test generation configuration menu will appear. Pressing the cancel button will send us back to the main menu.



tk

Selecciona las operaciones para las que quieres generar pruebas.

☐ rotateAllServos

pre

post

☐ readAllServos

pre

post

☐ cameraColor

pre

post

☐ isAt

pre

post

☐ wait

pre

post

☐ posInicial

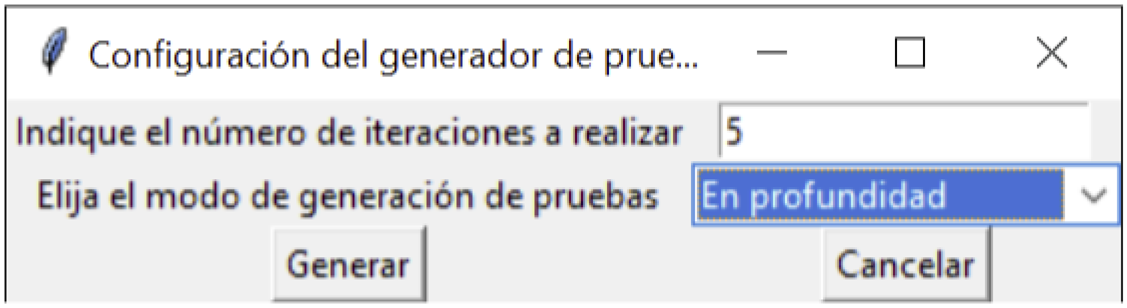
Figura 2: Operation configuration menu.

2.2.4. Test Configuration Menu

ON this menu the user will specify two parameters related to the test battery that will be generated. The first one is the number of iterations, which serves to determine the number of tests that we will be generating. If the generation is done at random, iterations equals the number of test that will be generated. In any othe case, the number of tests generated will be iterations*number of operations. The second one is the test generation mode. With this parameter we select how we want to explore the operations we selected previously. There are 3 modes:

- Random: Every unit tests selects one operation to try at random.
- Depth: Every operation gets tested a number of times in a row before testing the next.
- Width: Every operation gets tested once before repeating any of them, until we reach a certain number of tests per operation.

When pressing "Generate.^{a} test_script.py file will be created. This file will contain the py-test tests generated to be launched on the robot. After that, the GUI will close and the MachineControlServer will start sending the tests to our Dofbot.


Configuración del generador de prue...

Indique el número de iteraciones a realizar: 5

Elija el modo de generación de pruebas: En profundidad

Figura 3: Test configuration menu.

2.2.5. Executing previously generated tests without using the GUI

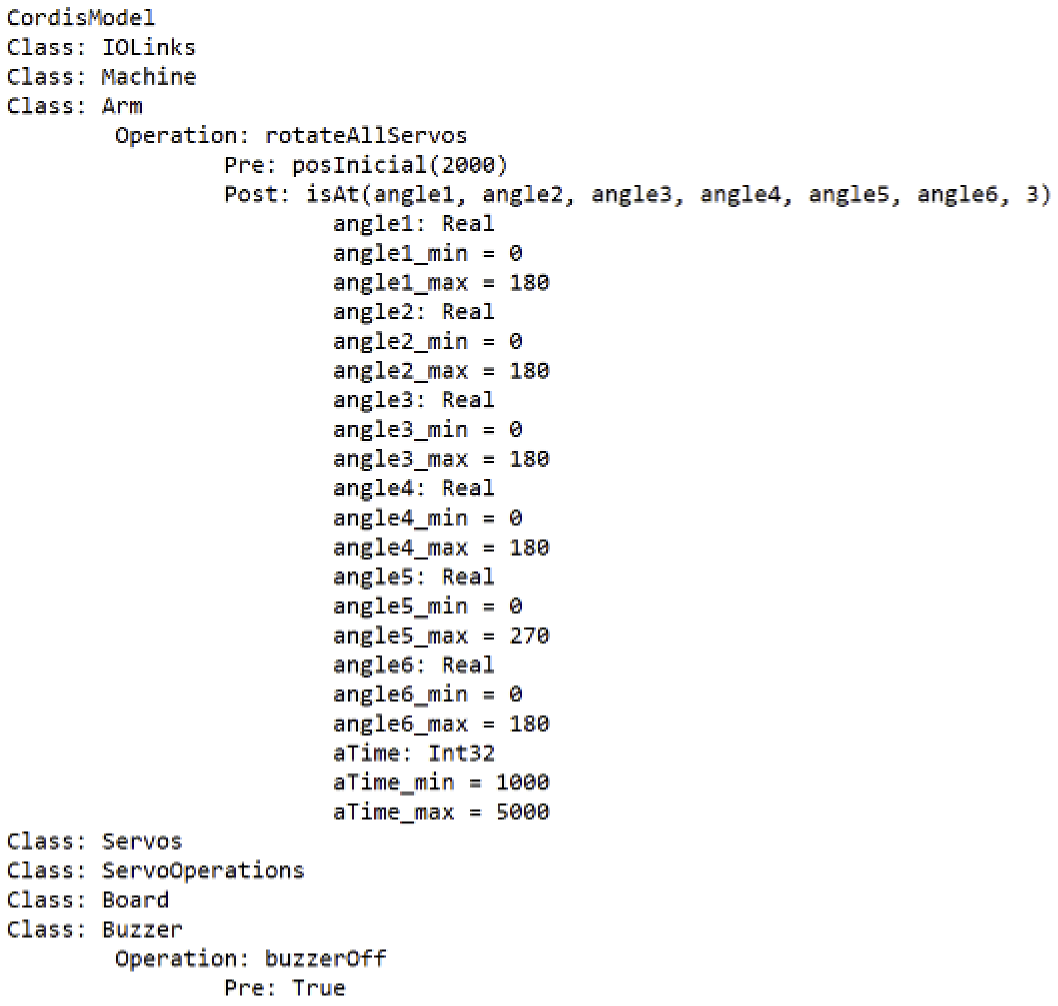
If we want to execute a previously generated test battery without using the GUI, we can simply use the command "python RunCordisTests.py" on the program's root folder.

IMPORTANT: If thre is more than one pytest file on the root folder the execution will fail. If you want to preserve more than one test battery you must copy it to another folder. Creating a new test battery will override the previous one on the root folder

2.2.6. Manually editing input and configuration files

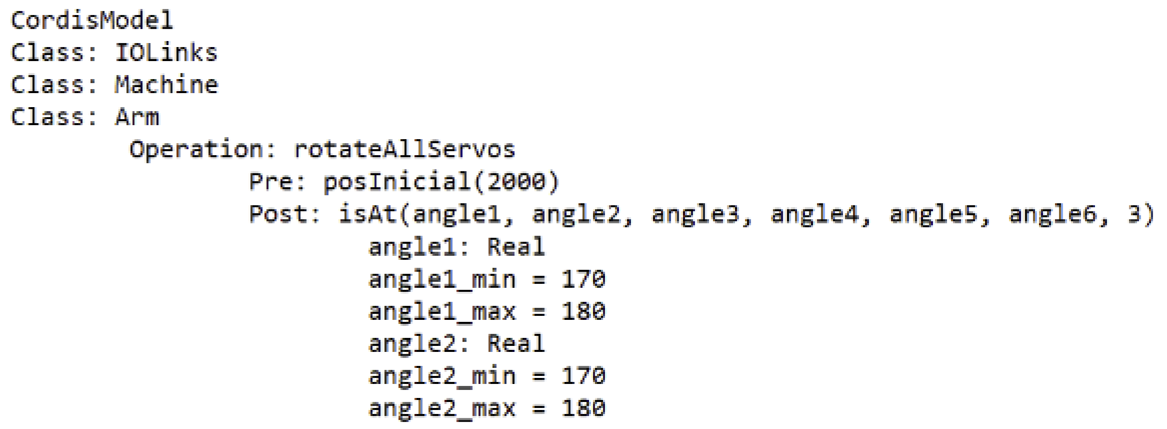
Both APIs are written in a simple, easy to understand pivot language and can be edited by hand if prefered. If we want to change a pre or post condition, or w have detected an error on an operation name, we can avoid having to use the GUI again by just editing it on the configuration files.

Things like upper and lower bounds, normally extracted from the model directly, can also be edited in these file should we want the test data to be generated on certain ranges.



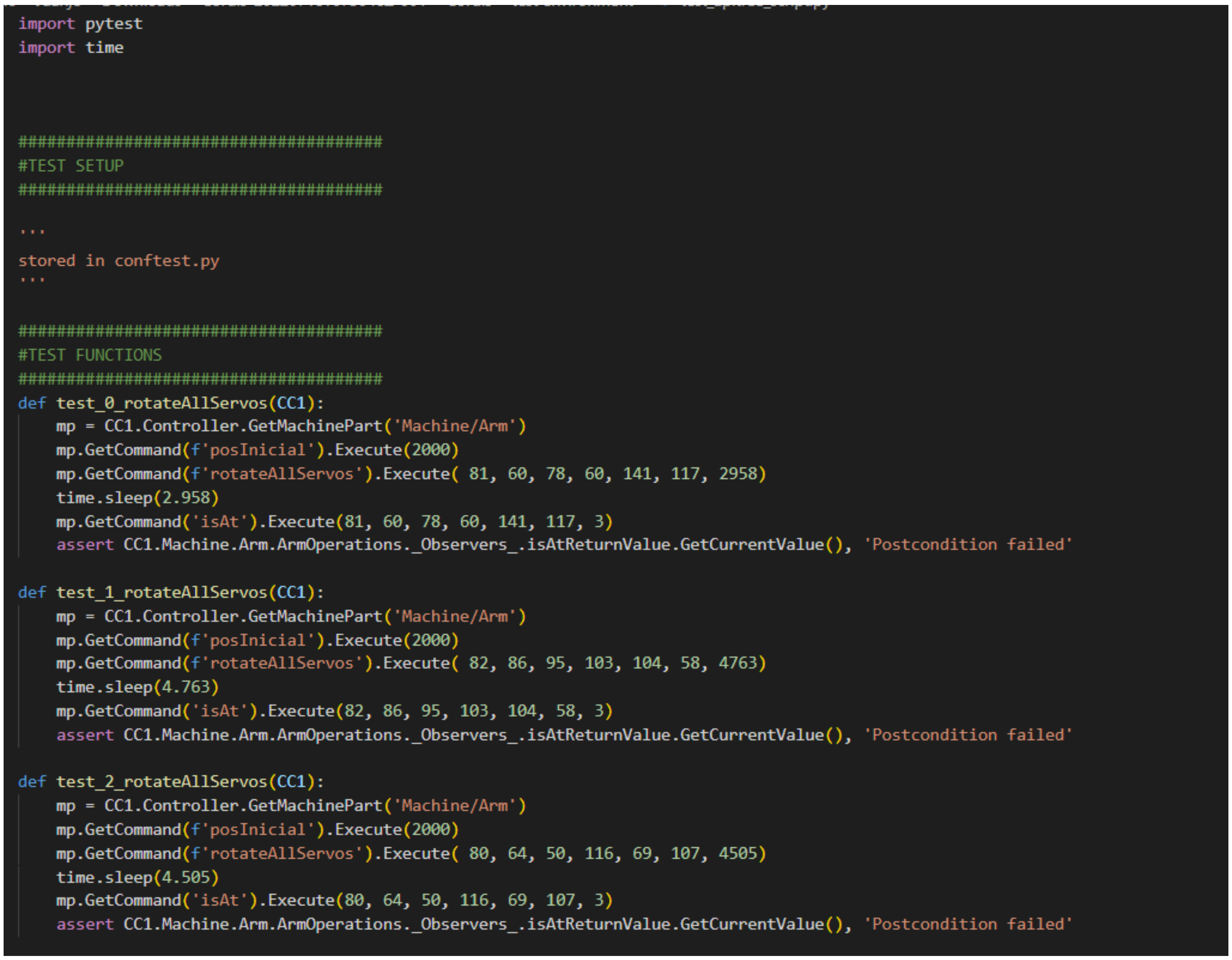
```
CordisModel
Class: IOLinks
Class: Machine
Class: Arm
        Operation: rotateAllServos
                Pre: posInicial(2000)
                Post: isAt(angle1, angle2, angle3, angle4, angle5, angle6, 3)
                        angle1: Real
                        angle1_min = 0
                        angle1_max = 180
                        angle2: Real
                        angle2_min = 0
                        angle2_max = 180
                        angle3: Real
                        angle3_min = 0
                        angle3_max = 180
                        angle4: Real
                        angle4_min = 0
                        angle4_max = 180
                        angle5: Real
                        angle5_min = 0
                        angle5_max = 270
                        angle6: Real
                        angle6_min = 0
                        angle6_max = 180
                        aTime: Int32
                        aTime_min = 1000
                        aTime_max = 5000
Class: Servos
Class: ServoOperations
Class: Board
Class: Buzzer
        Operation: buzzerOff
                Pre: True
```

Figura 4: Pivot language



```
CordisModel
Class: IOLinks
Class: Machine
Class: Arm
        Operation: rotateAllServos
                Pre: posInicial(2000)
                Post: isAt(angle1, angle2, angle3, angle4, angle5, angle6, 3)
                        angle1: Real
                        angle1_min = 170
                        angle1_max = 180
                        angle2: Real
                        angle2_min = 170
                        angle2_max = 180
```

Figura 5: Pivot language operation with modified lower bounds.



```
import pytest
import time



########################################
#TEST SETUP
########################################

'''
stored in conftest.py
'''

########################################
#TEST FUNCTIONS
########################################
def test_0_rotateAllServos(CC1):
    mp = CC1.Controller.GetMachinePart('Machine/Arm')
    mp.GetCommand(f'posInicial').Execute(2000)
    mp.GetCommand(f'rotateAllServos').Execute( 81, 60, 78, 60, 141, 117, 2958)
    time.sleep(2.958)
    mp.GetCommand('isAt').Execute(81, 60, 78, 60, 141, 117, 3)
    assert CC1.Machine.Arm.ArmOperations._Observers_.isAtReturnValue.GetCurrentValue(), 'Postcondition failed'

def test_1_rotateAllServos(CC1):
    mp = CC1.Controller.GetMachinePart('Machine/Arm')
    mp.GetCommand(f'posInicial').Execute(2000)
    mp.GetCommand(f'rotateAllServos').Execute( 82, 86, 95, 103, 104, 58, 4763)
    time.sleep(4.763)
    mp.GetCommand('isAt').Execute(82, 86, 95, 103, 104, 58, 3)
    assert CC1.Machine.Arm.ArmOperations._Observers_.isAtReturnValue.GetCurrentValue(), 'Postcondition failed'

def test_2_rotateAllServos(CC1):
    mp = CC1.Controller.GetMachinePart('Machine/Arm')
    mp.GetCommand(f'posInicial').Execute(2000)
    mp.GetCommand(f'rotateAllServos').Execute( 80, 64, 50, 116, 69, 107, 4505)
    time.sleep(4.505)
    mp.GetCommand('isAt').Execute(80, 64, 50, 116, 69, 107, 3)
    assert CC1.Machine.Arm.ArmOperations._Observers_.isAtReturnValue.GetCurrentValue(), 'Postcondition failed'
```

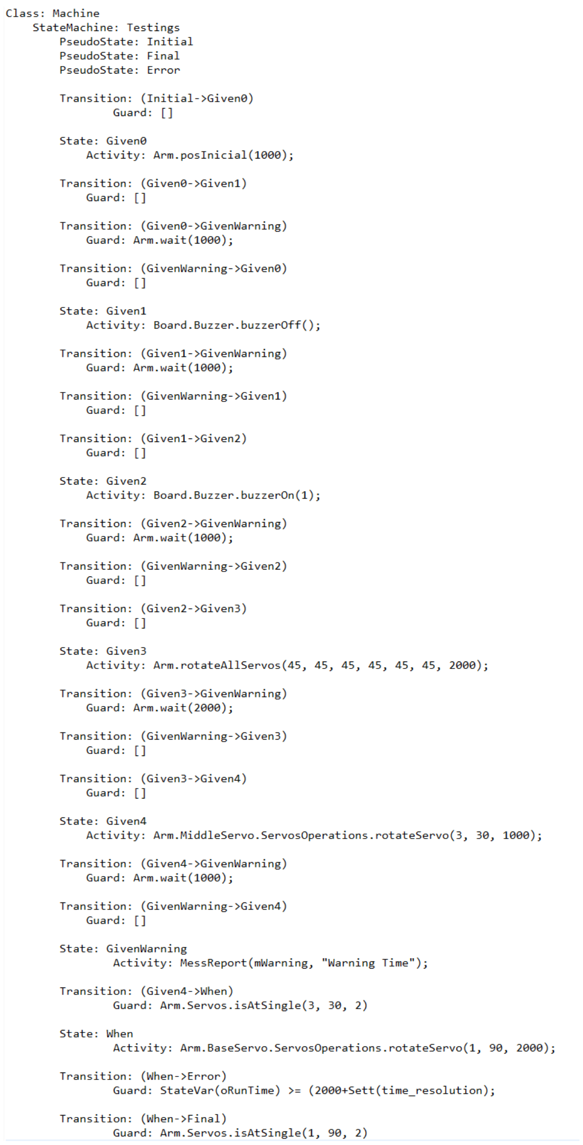
Figura 6: Unit tests generated.

3. Generating Tests from Use Case State Machines

This section describes how to generate use case tests from use case states machines generated with Gherkin.

For installation, download 'UseCaseTesting' from the repository and unzip it on its own folder.

Project dependencies are the same that the ones described on section 2.1

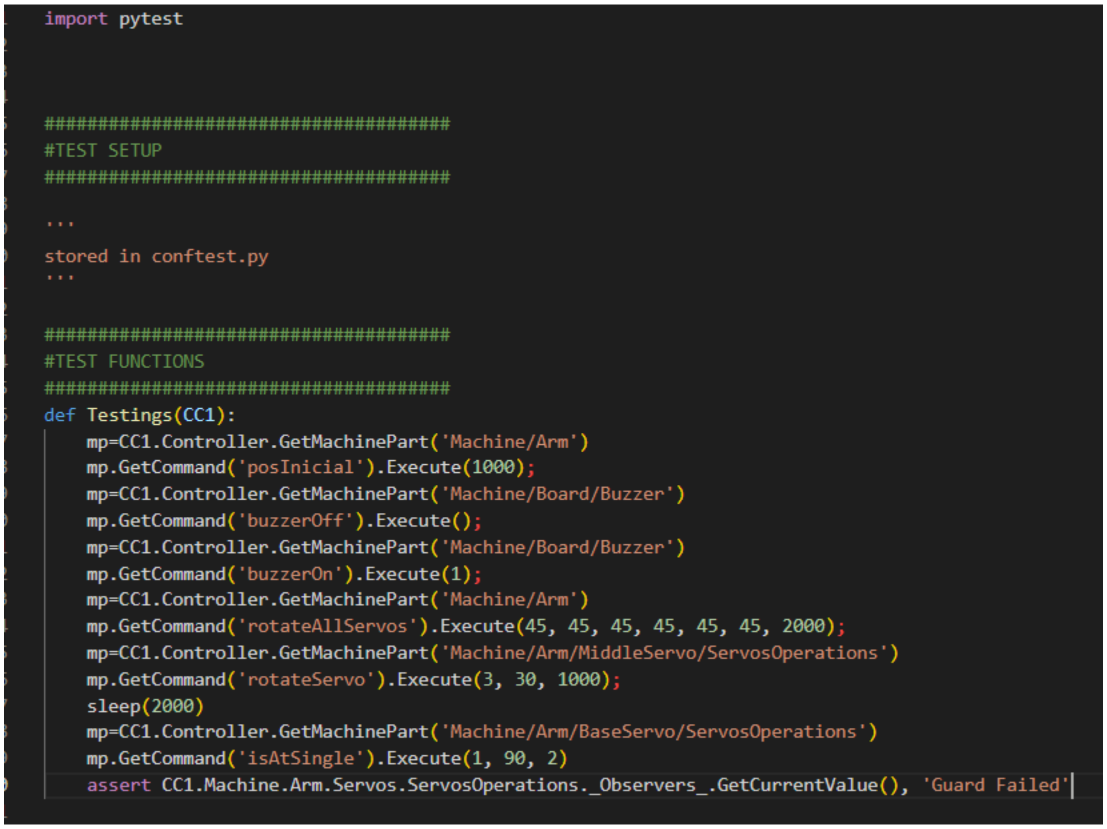


```
Class: Machine
    StateMachine: Testings
        PseudoState: Initial
        PseudoState: Final
        PseudoState: Error

        Transition: (Initial->Given0)
                Guard: []

        State: Given0
            Activity: Arm.posInicial(1000);

        Transition: (Given0->Given1)
            Guard: []

        Transition: (Given0->GivenWarning)
            Guard: Arm.wait(1000);

        Transition: (GivenWarning->Given0)
            Guard: []

        State: Given1
            Activity: Board.Buzzer.buzzerOff();

        Transition: (Given1->GivenWarning)
            Guard: Arm.wait(1000);

        Transition: (GivenWarning->Given1)
            Guard: []

        Transition: (Given1->Given2)
            Guard: []

        State: Given2
            Activity: Board.Buzzer.buzzerOn(1);

        Transition: (Given2->GivenWarning)
            Guard: Arm.wait(1000);

        Transition: (GivenWarning->Given2)
            Guard: []

        Transition: (Given2->Given3)
            Guard: []

        State: Given3
            Activity: Arm.rotateAllServos(45, 45, 45, 45, 45, 45, 2000);

        Transition: (Given3->GivenWarning)
            Guard: Arm.wait(2000);

        Transition: (GivenWarning->Given3)
            Guard: []

        Transition: (Given3->Given4)
            Guard: []

        State: Given4
            Activity: Arm.MiddleServo.ServosOperations.rotateServo(3, 30, 1000);

        Transition: (Given4->GivenWarning)
            Guard: Arm.wait(1000);

        Transition: (GivenWarning->Given4)
            Guard: []

        State: GivenWarning
                Activity: MessReport(mWarning, "Warning Time");

        Transition: (Given4->When)
                Guard: Arm.Servos.isAtSingle(3, 30, 2)

        State: When
                Activity: Arm.BaseServo.ServosOperations.rotateServo(1, 90, 2000);

        Transition: (When->Error)
                Guard: StateVar(oRunTime) >= (2000+Sett(time_resolution);

        Transition: (When->Final)
                Guard: Arm.Servos.isAtSingle(1, 90, 2)
```

Figura 7: Gherkin generated use case state machine used as an example.

3.1. Example

First, you must copy your newly generated use case state machine file to the root folder of the project. It should have the name ”test.txt”. Once that is done, executing ’UseCaseTesting.bat’ will automatically generate the pytest test script. After that, running the command ”python RunCordisTests.py”will launch the test on the robot.



```
import pytest



#######################################
#TEST SETUP
#######################################

'''
stored in conftest.py
'''

#######################################
#TEST FUNCTIONS
#######################################
def Testings(CC1):
    mp=CC1.Controller.GetMachinePart('Machine/Arm')
    mp.GetCommand('posInicial').Execute(1000);
    mp=CC1.Controller.GetMachinePart('Machine/Board/Buzzer')
    mp.GetCommand('buzzerOff').Execute();
    mp=CC1.Controller.GetMachinePart('Machine/Board/Buzzer')
    mp.GetCommand('buzzerOn').Execute(1);
    mp=CC1.Controller.GetMachinePart('Machine/Arm')
    mp.GetCommand('rotateAllServos').Execute(45, 45, 45, 45, 45, 45, 2000);
    mp=CC1.Controller.GetMachinePart('Machine/Arm/MiddleServo/ServosOperations')
    mp.GetCommand('rotateServo').Execute(3, 30, 1000);
    sleep(2000)
    mp=CC1.Controller.GetMachinePart('Machine/Arm/BaseServo/ServosOperations')
    mp.GetCommand('isAtSingle').Execute(1, 90, 2)
    assert CC1.Machine.Arm.Servos.ServosOperations._Observers_.GetCurrentValue(), 'Guard Failed'
```

Figura 8: Use case test generated from the previously showed use case state machine.