# Stress Testing Control Loops in Cyber-Physical Systems - RCR Report

CLAUDIO MANDRIOLI, University of Luxembourg, Luxembourg
SEUNG YEOB SHIN, University of Luxembourg, Luxembourg
MARTINA MAGGIO, Saarland University, Germany and Lund University, Sweden
DOMENICO BIANCULLI, University of Luxembourg, Luxembourg
LIONEL BRIAND, Lero Centre, University of Limerick, Ireland and University of Ottawa, Canada

This is the Replicated Computational Results (RCR) Report for the article "*Stress Testing Control Loops in Cyber-Physical Systems*." The article proposes a novel approach for testing Cyber-Physical Systems (CPS) based on the integration of the guarantees that can be provided with the control theoretical models into the software testing practices. This RCR report describes how to reproduce the empirical results of the article. We make available the different scripts needed to fully replicate the results obtained in our article.

CCS Concepts: • **Software and its engineering → Software verification and validation**; • **Computer systems organization → Embedded and cyber-physical systems**.

Additional Key Words and Phrases: Cyber-physical Systems, Software Testing, Control Theory

**ACM Reference Format:**
Claudio Mandrioli, Seung Yeob Shin, Martina Maggio, Domenico Bianculli, and Lionel Briand. 2025. Stress Testing Control Loops in Cyber-Physical Systems - RCR Report. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (January 2025), 6 pages. https://doi.org/10.1145/3733715

## 1 Overview

### 1.1 Summary of Motivation and Proposed Approach

Cyber-Physical Systems (CPS) are systems characterised by the tight coupling of software and physical components [1]. This interaction with the physical world exposes the software to its uncertainty, making its verification challenging. At the same time, the ability to affect the physical world often makes CPS safety-critical. Thus, their verification is both challenging and important. This work addresses the problem of identifying critical scenarios where the CPS shows unpredictable behaviour.

The development of CPS is multidisciplinary, involving control engineers, software engineers and engineers specific to the given application domain (e.g., automotive). We propose an approach [3] that integrates the a priori guarantees from control theory on CPS development with the a posteriori guarantees obtained from the software testing process. To do so, the approach complements the traditional requirement-based testing with the testing of design assumptions that underlie the control theoretical models. In fact, when the design assumptions hold, then the control theoretical models are valid and we can rely on the a priori guarantees. Conversely, when the design assumptions

Authors' Contact Information: Claudio Mandrioli, claudio.mandrioli@uni.lu, University of Luxembourg, Luxembourg; Seung Yeob Shin, seungyeob.shin@uni.lu, University of Luxembourg, Luxembourg; Martina Maggio, maggio@cs.uni-saarland.de, Saarland University, Germany;, Lund University, Sweden; Domenico Bianculli, domenico.bianculli@uni.lu, University of Luxembourg, Luxembourg; Lionel Briand, lbriand@uottawa.ca, Lero Centre, University of Limerick, Ireland;, University of Ottawa, Canada.

ACM 1557-7392/2025/1-ART1
https://doi.org/10.1145/3733715

Table 1. Summary of test subjects, corresponding artefacts, and datasets.

| **Name** | **Implementation** | **Artefact** | | **Dataset** | |
|---|---|---|---|---|---|
| | | **DOI** | **License** | **DOI** | **License** |
| Drone | Python | 10.5281/zenodo.13123116 | 🅭🅯 | 10.5281/zenodo.13123582 | 🅭🅯 |
| DC servo | Matlab | 10.5281/zenodo.7274106 | 🅭🅯 | 10.5281/zenodo.8043260 | 🅭🅯 |
| Aircraft | | | | 10.5281/zenodo.8043351 | 🅭🅯 |

do not hold, the control theoretical models are not valid and cannot be used to predict the CPS behaviour. Thus, in such scenarios, there is need for *a posteriori* empirical verification, i.e., testing. The proposed testing approach leverages this intuition and stresses the CPS by generating scenarios that falsify the design assumptions. Among the classes of assumptions identified in the article, we target the falsification of the assumption underlying the use of linear models. To identify when the design assumptions falsification affects the CPS behaviour, we have developed metrics (degree of non-linearity and degree of filtering) and metamorphic relations. We generate tests with varying rates of change (i.e., frequency content) and amplitudes, and then use the degree of non-linearity and degree of filtering metrics to identify the scenarios where the linearity assumption is falsified.

## 1.2 Summary of Results

In our empirical evaluation, we address two Research Questions (RQ):

**RQ1**: How effective is our test case generation approach in pushing a CPS control loop away from the linearisation point and triggering non-linear phenomena around the boundary where they appear?

**RQ2**: How the proposed Metamorphic Relations (MRs) identify stress test cases at the bounds of the System-Under-Test (SUT) design scope?

We answer the RQs by applying our testing approach to three test subjects. The "Drone" subject, which is taken from previous literature [2], the "DC Servo" (a continuous current electric motor, that we study in five different configurations), which is based on laboratory hardware used for educational purposes at Lund University (Sweden), and the "Lightweight Aircraft" which is developed by Mathworks.[1]

*Answer to RQ1.* We use in-depth knowledge of our test subjects to quantify non-linear phenomena that occur in their execution. All of the test subjects include the actuator saturation, the DC Servo also includes sensor saturation and other non-linear phenomena depending on the configuration. We observe that our tests effectively trigger the non-linear phenomena that characterise the test subjects. Specifically, actuator and sensor saturation, and quadratic friction are triggered to various degrees. Other phenomena (coulomb friction, dead-zone, and backlash) are more easily triggered and do not require a dedicated testing approach.

*Answer to RQ2.* The three proposed MRs describe expected patterns of the degree of non-linearity and degree of filtering, for varying input rate of change (frequency) and amplitude. We observe that the three MRs hold in the majority of the tests for our test subjects. Notably, the invalidation of the MRs that concern the degree of filtering (MR2 and MR3), can be used to complement the degree of non-linearity and MR1 to detect the impact of non-linear phenomena on the CPS behaviour.

[1]https://nl.mathworks.com/help/aeroblks/lightweight-airplane-design.html

## 2 Artefacts

We provide two separate repositories containing the scripts that implement our testing approach both in Python and Matlab. Since the artefacts require only Python and Matlab with common libraries and add-ons, we do not deem a VM image or Docker container necessary.

We use the Python implementation for the drone (included as a submodule of the repository), and the Matlab one for the DC Servo and Lightweight Aircraft (included as part of the repository). Table 1 summarises the organisation of the repositories and test subjects and provides the DOIs for the code and pre-computed datasets. All of the artefacts are released with the permissive MIT license under the CC-BY copyright. Besides the long-term archives on Zenodo, the code is also available on Github.[2]

Since the testing campaigns take some time to execute, we provide pre-computed test output traces. Those traces can be used to obtain the figures of the article without running the full testing campaigns. The datasets can be found using the DOIs in the Datasets column of Table 1. They are simple directories containing pickled Python objects in the case of the Drone test subject and csv files in the case of the Lightweight Aircraft and DC Servo. Instructions of how to use the datasets are provided in Section 5.

### Python Implementation

The repository contains:

- the main script that executes the testing campaign (`main-crazyflie.py`),
- several plotting scripts (named `plot-*.py`), and
- the *ControlBasedTesting* directory that contains the python module implementing the functionalities needed by the testing approach.

### Matlab Implementation

The repository contains:

- the main scripts to run the testing campaign (named `main_DCservo.m` and `main_LWaltitude.m`)
- the scripts implementing the approach steps (named `step*.m`),
- the initialisation scripts for the two test subjects (`DCservo_init.m` and `LWaltitude_init.m`),
- the *model* directory containing the Simulink models of the test subjects, and
- the *testing* directory containing the Matlab functions used for the testing approach implementation.

## 3 Prerequisites and Requirements

The artefacts do not require any specific hardware and can be executed on any machine that can run Python3 and Matlab. The artefacts have been developed with Python3.11 and Matlab R2022b.

The Matlab installation requires the following add-ons:

- Simulink,
- Aerospace Blockset,
- Aerospace Toolbox,
- Control System Toolbox,
- Signal Processing Toolbox,
- Simulink Control Design,
- Statistics and Machine Learning Toolbox.

[2]The Python implementation is available at https://github.com/ManCla/control-based-cps-testing-crazyflie, and the Matlab implementation is available at https://github.com/ManCla/control-based-cps-testing-DCservo-aircraft.

The DCservo test subject requires a C compiler, which should be installed on the computer.[3]

## 4 Setup

### 4.1 Python Implementation

To set up the drone case study, first clone the repository with:

```
git clone https://github.com/ManCla/control-based-cps-testing-crazyflie.git
```

Then, you need to initialise the git submodule containing the model of the drone. After cloning, use the following commands from the command line to retrieve the submodule:

```
cd control-based-cps-testing-crazyflie
git submodule init
git submodule update
```

You also need to install the required Python libraries in the given version. These can be installed with the Python package manager, using the following commands in the terminal:

```
pip3 install numpy==1.24.1
pip3 install scipy==1.10.0
pip3 install matplotlib==3.8.2
```

### 4.2 Matlab Implementation

Once the Matlab add-ons are installed (see Section 3), you only need to clone the repository with:

```
git clone \
https://github.com/ManCla/control-based-cps-testing-DCservo-aircraft.git
```

The artefact already contains the test subject models and is ready for execution.

## 5 Steps to Reproduce

### 5.1 Python - Drone

The whole testing campaign is executed by running one main script:

```
python main-crazyflie.py
```

All the tests results are stored in Python pickled objects in the directory *cfdata_nlmax015*. The main script generates two files named *characterization-only-main* and *characterization-all*. Using these results, the exact same figures as in the article can be generated running this script (the titles of the figures correspond to the article's figure numbering for convenience):

```
python plot-paper-figures.py
```

*Reproducing the Results Using the Pre-Computed Datasets.* To use the provided dataset, download it from the Zenodo archive[4] and unzip the file. Then, move the directory named *cfdata_nlmax015* from the archive to the artefact directory using the following command (assuming the unzipped directory and artefact repository are in the same directory):

```
mv cfdata_nlmax015 control-based-cps-testing-crazyflie/cfdata_nlmax015
```

Now you can run the main script in the same way as above:

```
cd control-based-cps-testing-crazyflie
python main-crazyflie.py
```

[3]Information on the compatible compilers can be found at https://mathworks.com/support/requirements/supported-compilers.html.
[4]https://zenodo.org/records/13123582/files/cfdata_nlmax015.zip?download=1

The script will automatically look for the pre-computed test flights and use them instead of re-running the tests.

### 5.2 Matlab - DC Servo and Lightweight Aircraft

We provide scripts for both test subjects that execute the full testing campaign and generate the article figures. In the Matlab terminal, navigate to the repository directory, and then run the main script for the test subject that you want to execute:

```
cd PATH/TO/REPOSITORY/control-based-cps-testing-DCservo-aircraft
main_DCservo or main_LWaltitude
```

The main scripts first call an initialisation script specific to each test subject, and then call the scripts that implement each approach step. We recall that we assessed our testing approach on five different configurations of the DC Servo, each effectively constituting a different test subject. Besides the basic configuration without extra non-linearity, in the other configurations we inject either an input non-linearity or a friction non-linearity. To inject input or friction non-linearity, change lines 57 or 58 in the `DCservo_init.m` script with the non-linearity that you want to inject into the system:

```
57 sut_nl.input_non_linearity = inl_none;       % or inl_dead_zone or inl_backlash
58 sut_nl.friction_non_linearity = fnl_linear;  % or fnl_quadratic or fnl_coulomb
```

The output of each step is stored (both test logs and analysis results) as csv files in the *dcServo_test _data_7periods* directory for the DC Servo and in the *lwAltitude_test_data_10periods* for the Lightweight Aircraft.

*Reproducing the Results Using the Pre-Computed Datasets.* As shown in Table 1, we provide two separate archives with different DOIs for the DC Servo and the Lightweight Aircraft. To use the pre-computed tests results, download the desired dataset from Zenodo and unzip it.[5] Then, move the dataset folder to the artefact directory with the following command from the OS shell, for the DC Servo:

```
mv dcServo_test_data_7periods \
control-based-cps-testing-DCservo-aircraft/dcServo_test_data_7periods
```

or, for the Lightweight Aircraft:

```
mv lwAltitude_test_data_10periods \
control-based-cps-testing-DCservo-aircraft/lwAltitude_test_data_10periods
```

Afterwards, from the Matlab shell, run the main script like above, for the DC Servo:

```
main_DCservo
```

or, for the Lightweight Aircraft:

```
main_LWaltitude
```

The main scripts will automatically recognise the dataset and use the pre-computed tests results instead of executing the testing campaign from scratch.

## Acknowledgments

We thank Drishti Yadav for her help in testing the installation instructions. This project has received funding from SES and the Luxembourg National Research Fund under the Industrial Partnership Block Grant (IPBG), ref. IPBG19/14016225/INSTRUCT; the European Union's Horizon 2020 research and innovation programme under Grants No. 871259 (ADMORPH project) and

[5]DC servo: https://zenodo.org/records/8043260/files/dcServo_test_data_7periods.zip?download=1, Lightweight Aircraft:https://zenodo.org/records/8043351/files/lwAltitude_test_data_10periods.zip?download=1.

No. 101148870 (ContTestCPS); and from the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Martina Maggio is member of the ELLIIT Strategic Research Area at Lund University. Lionel Briand was partly funded by the Science Foundation Ireland grant 13/RC/2094-2 and NSERC of Canada under the Discovery and CRC programs. This publication reflects only the authors' view and the European Commission is not responsible for any use that may be made of the information it contains.

## References

[1] Edward A Lee. 2015. The past, present and future of cyber-physical systems: A focus on models. *Sensors* 15, 3 (2015), 4837–4869.

[2] Claudio Mandrioli, Max Nyberg Carlsson, and Martina Maggio. 2023. Testing Abstractions for Cyber-Physical Control Systems. *ACM Trans. Softw. Eng. Methodol.* 33, 1, Article 18 (nov 2023), 32 pages. https://doi.org/10.1145/3617170

[3] Claudio Mandrioli, Seung Yeob Shin, Martina Maggio, Domenico Bianculli, and Lionel Briand. 2023. Stress Testing Control Loops in Cyber-physical Systems. *ACM Trans. Softw. Eng. Methodol.* 33, 2, Article 35 (dec 2023), 58 pages. https://doi.org/10.1145/3624742