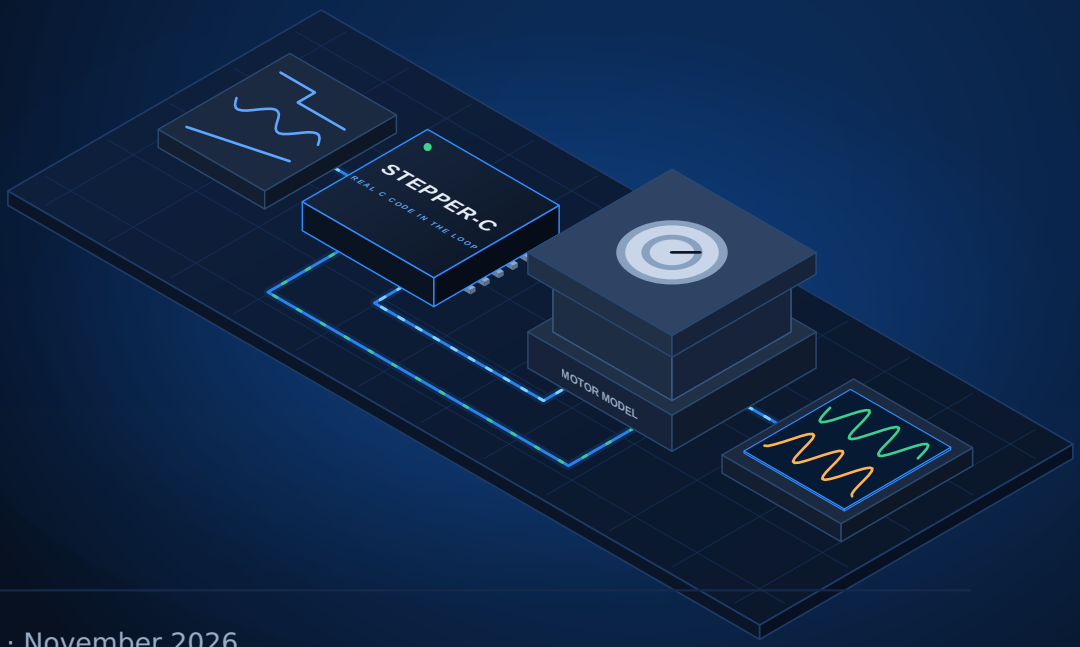
SystemFromScratch

OPEN-SOURCE SIMULATION MODEL

Stepper-Lab

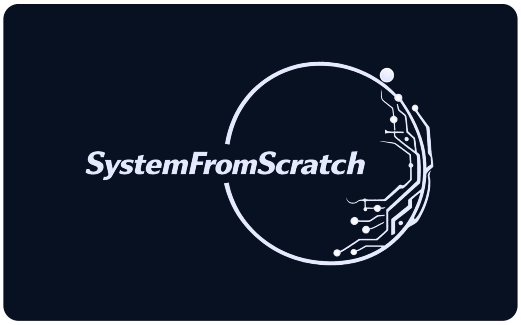
Stepper Motor Model in Scilab/Xcos

USER MANUAL



Version development · November 2026
Anton Vynohradov
systemfromscratch.com

Imprint



Stepper-Lab User Manual
Model version development
Edition: November 2026

Author
Anton Vynohradov
SystemFromScratch® — systemfromscratch.com
GitHub: github.com/AntonVynohradov
LinkedIn: linkedin.com/in/vynohradov
ORCID: orcid.org/0009-0005-6965-0911
E-mail: avynohradov@systemfromscratch.com

Online
Documentation: https://antonvynohradov.github.io/Stepper-Lab/
Source code: https://github.com/AntonVynohradov/Stepper-Lab

Copyright © 2026 Anton Vynohradov.

Stepper-Lab and this manual are licensed under the Apache License, Version 2.0 (the “License”); you may not use these files except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 and in Appendix B. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

SystemFromScratch® and the SystemFromScratch® logo are registered trademarks. The Apache License 2.0 grants no rights to use them; they may not be used to endorse or promote products derived from Stepper-Lab without prior written permission.

STM32 is a registered trademark of STMicroelectronics International N.V. Other product, motor and board names are used only to identify the hardware and tools the model describes; Stepper-Lab is not affiliated with or endorsed by their owners. All other trademarks are the property of their respective owners.

This manual is generated from the same sources as the online documentation. The information in it was checked against the model version stated above; later versions may differ.

Contents

1 Introduction 5
- 1.1 What it does and does not do 5
- 1.2 Who it is for 6
- 1.3 Related projects 6
- 1.4 Key properties 6
- 1.5 Terminology 7
- 1.6 Conventions 7

2 Getting Started 8
- 2.1 What you need 8
- 2.2 Get the sources 8
- 2.3 First run: the Lab window 8
- 2.4 First run: from a script 9
- 2.5 The original SIL diagram 9
- 2.6 The model without a controller 9
- 2.7 Tests 9
- 2.8 Documentation tooling 9

3 The Lab Window and Diagrams 11
- 3.1 The window 11
- 3.2 Scenarios 12
- 3.3 The diagram 12
- 3.4 Results 15
- 3.5 Common mistakes 15

4 MIL and SIL 17
- 4.1 What runs in each 17
- 4.2 Backends 17
- 4.3 How close they are 18
- 4.4 A configuration the library refuses 18
- 4.5 The stepper_sil diagram 18

5 Configuration 20
- 5.1 Context of a diagram 20
- 5.2 Motor: mot 20
- 5.3 Board: brd 21
- 5.4 Controller: ctl 22
- 5.5 Motion: mp 22
- 5.6 The Stepper-C folder 23
- 5.7 The stage file 23
- 5.8 Motor parameters for C 24
- 5.9 Settings file 24
- 5.10 Environment variables 24

6 Motor and Control Model 25
- 6.1 Motor 25
- 6.2 Current loop 25
- 6.3 Motion generator 26

6.4 Power stage ... 26
6.5 Stall detection ... 27
6.6 Parameters of the example ... 28

7 Measured Results ... 29
7.1 Current loop ... 29
7.2 Voltage limit ... 29
7.3 Resonance ... 30
7.4 Parameter uncertainty ... 31
7.5 Power stage ... 31
7.6 Stall detection ... 31
7.7 Where these numbers come from ... 33

8 Reference ... 34
8.1 Results: sil.values ... 34
8.2 Xcos blocks ... 35
8.3 Scilab functions ... 35
8.4 The plant in C ... 36
8.5 Folders ... 37

9 Verification ... 38
9.1 How the model is checked ... 38
9.2 Running the tests ... 38
9.3 Testing another stage file ... 39
9.4 Python twin ... 39
9.5 Test list ... 39
9.6 Known gaps ... 41

Author & Company ... 42
Author ... 42
SystemFromScratch® ... 43
Contact ... 43
Support and feedback ... 44
Trademarks and acknowledgements ... 44

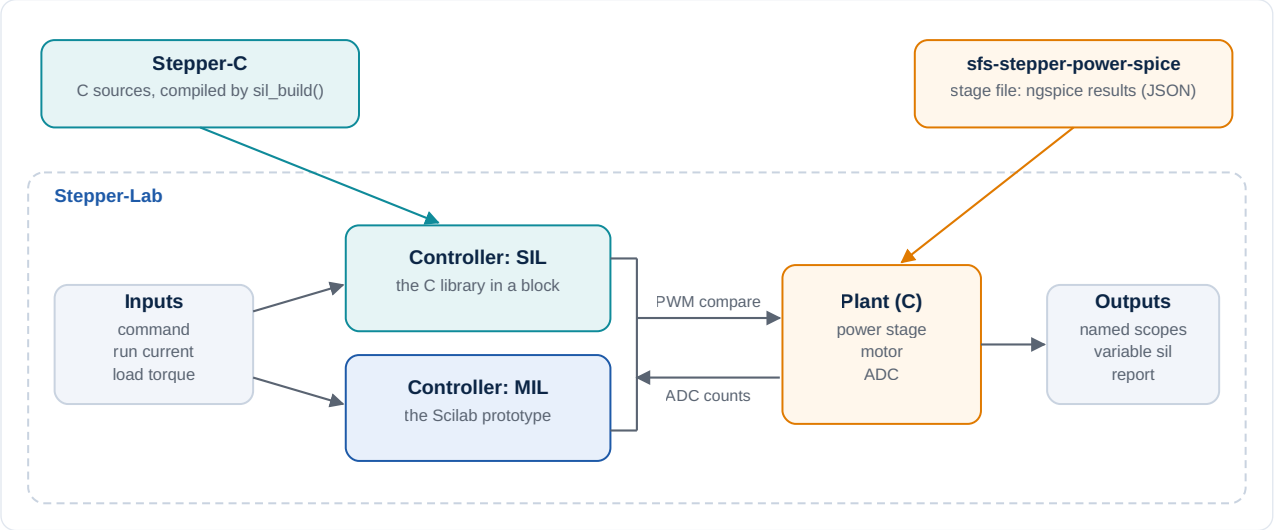
Appendix A Changelog ... 45
A.1 [Unreleased] ... 45

Appendix B License ... 49
B.1 Full license text ... 49

CHAPTER 1

Introduction

Stepper-Lab is a Scilab/Xcos model of a two-phase hybrid stepper motor with its power stage and current measurement. It lets you run the control code of Stepper-C (antonvynohradov.github.io/Stepper-C/) against that model on a PC, drive it with any signal, and read the results, before the code reaches hardware.



1.1 What it does and does not do

It does	Why
Runs the real C code of Stepper-C inside an Xcos diagram (software in the loop, SIL)	The code under test is the code that ships: ADC counts in, PWM compare values out
Runs a Scilab prototype of the same controller in the same layout (model in the loop, MIL)	An algorithm can be changed and studied without compiling it
Accepts any Xcos source as motion command, run current or load torque	Step, sine, ramp and your own signals use the same three inputs
Models the power stage as averaged or switching, with results from ngspice	The fast model is enough for most runs; the slow one shows ripple and dead time
Checks itself: tests against theory, against a Python twin, and between MIL and SIL	Each model is checked by something other than itself

It does not	Why
Replace measurements on your motor	Back-EMF constant, friction and detent torque of the example motor are derived or assumed; Configuration marks which
Control a real motor	It is a simulation; the firmware that uses Stepper-C is a separate matter
Include Stepper-C	The library has its own repository; you select its folder

It does not	Why
Run ngspice	The window only selects a stage file that `sfs-stepper-power-spice` has exported
Model temperature, saturation or the motor beyond its two-phase equations	The equations are in Motor and Control Model

1.2 Who it is for

Embedded and control engineers who work with stepper motors and want to see what a controller does before the first power-up. No Scilab knowledge is needed for the first runs: the Lab window (p. 11) holds the choices, and the scenarios are ready to run.

1.3 Related projects

Project	Role here
Stepper-C (antonvynohradov.github.io/Stepper-C/)	The control library under test. Its sources are compiled into the SIL blocks
sfs-stepper-power-spice (github.com/AntonVynohradov/sfs-stepper-power-spice)	ngspice models of the power stage. Its export is the stage file: leg resistance, diode voltage, effective dead time and the leg voltage error table

Rule of this project: the Scilab model follows the algorithm of the library. A change in Stepper-C is ported to the prototype and to the Python twin in the same step, and the tests show whether they still agree.

1.4 Key properties

Property	Value
Tool	Scilab 2027 with Xcos; a C compiler for the plant and the library
Example motor	17HS19-2004S1: 1.60 Ω, 3.0 mH, 2.0 A, 200 steps per revolution
Example board	B-G431B-ESC1: 24 V, 20 kHz PWM, three legs with a shared leg
Controllers	SIL (C, FIXED or FLOAT backend), MIL (Scilab)
Stepper-C version used for the results	`main` at commit `3ef0f7e`, which reports itself as 1.1.0 and has stall detection fixes made after the tag `v1.1.0`
Tests	433 checks, 0 failures on the final full run (Verification)
Platform of all results	Windows 10 with Scilab 2027.0.0
License	Apache License 2.0

1.5 Terminology

Term	Meaning
SIL	Software in the loop: the compiled C library runs against the model
MIL	Model in the loop: the Scilab prototype of the controller runs against the same model
Plant	Power stage, motor and current measurement: everything the controller acts on
Prototype	The Scilab implementation of the controller (`model/ctrl_period.sci` and the functions it calls)
Python twin	Independent Python implementation of parts of the model, in `tools/`; it checks the Scilab results
Stage file	JSON export of `sfs-stepper-power-spice` with the measured values of a power stage
Scenario	A ready experiment of the window: context lines and source blocks
Backend	The numeric form of Stepper-C: `FIXED` (Q15) or `FLOAT`
Fast loop, slow loop	The two periods of Stepper-C: 20 kHz current control, 1 kHz motion
Load angle	Electrical angle between the commanded field and the rotor; beyond 90° the rotor is past its pull-out point

1.6 Conventions

- Units are SI unless stated. Positions are in microsteps, with 3200 microsteps per revolution at 16 microsteps per full step.
- Functions are written `sil_report()`, and files by their path from the repository root.
- The current vector is `iA = I cos(phi)`, `iB = I sin(phi)`, and the rotor settles at `p theta = phi`. Increasing `phi` turns the rotor forward. Stepper-C uses the same convention.

CHAPTER 2

Getting Started

This chapter takes you from an empty folder to the first simulation in the Lab window. Plan a few minutes, plus the one compilation of the library.

2.1 What you need

Item	Note
Scilab 2027	With Xcos. All results in this manual come from 2027.0.0 on Windows 10
C compiler	The one Scilab uses for toolboxes. On Windows: Visual Studio or the free Build Tools, with the workload *Desktop development with C++*. `sil_build()` also has a gcc path for Linux, which was not run for this manual
Stepper-C	The sources of the library; only needed once to compile, and again when they change
Python 3 with numpy and scipy	Optional: the Python twin, and test X04 with another stage file

2.2 Get the sources

Clone both repositories next to each other. The model then finds the library by itself:

```
git clone https://github.com/AntonVynohradov/Stepper-Lab.git
git clone https://github.com/AntonVynohradov/Stepper-C.git
```

The library can also be in any other folder: select it in the window, or see the search order (p. 23).

2.3 First run: the Lab window

1. Start Scilab, open **`start_lab.sce`** in its editor, and press **F5**.
2. Check the first line of the window, *Stepper-C library*. It shows the folder and `OK`. If it asks for the folder, press **Browse** and select it.
3. Leave *SIL: C code* selected. Choose the scenario *Velocity step to 1 rev/s*.
4. Press **Run**. The first run compiles the library. The status line at the bottom then says that the run finished.
5. Press **Report**. A figure shows currents, speed, position, load angle, voltages and the detector, each with a title, a unit and a legend.

To see the diagram itself, press **Open in Xcos**, then press Start in Xcos. The Lab Window and Diagrams (p. 11) explains each part.

2.4 First run: from a script

Everything the window does is available as a function. Load the functions once per Scilab session, from the root of the repository; `start_lab.sce` and `start_sil.sce` do it too. The examples in this manual assume that this has been done.

```
exec("loader.sce", -1);
res = lab_run("SIL", 3);        // scenario 3: velocity step to 1 rev/s
d = sil_report_data(res);       // the panels of the report, and the first fault
mprintf("%d samples, last speed %.0f rpm\n", size(res.values, 1), res.values($, 5));
```

Results are in `res.time` and `res.values`; Reference (p. 34) lists the columns.

2.5 The original SIL diagram

`start_sil.sce` compiles the library and opens `xcos/stepper_sil.zcos`, the diagram with the command table as its only input. It is kept unchanged next to the Lab, and its scenario is set under **Simulation → Set Context**. MIL and SIL (p. 18) describes it.

2.6 The model without a controller

The motor equations can also be used alone, with a voltage source:

```
mot = motor_17hs19_2004s1();
srcp = struct("V", 3.2, "f_el", 5, "TL", 0);
t = 0:1e-4:0.04;
x = stepper_simulate(mot, src_open_loop, srcp, t);
```

`demos/demo_open_loop.sce` and `demos/demo_switching.sce` are scripts with plots.

2.7 Tests

From the root of the repository, with the `bin` folder of Scilab in `PATH`:

```
WScilex-cli -quit -f tests/run_all.sce      # Windows
scilab-cli -quit -f tests/run_all.sce       # Linux
```

A full run takes hours, so use the groups of the window or `SSTEP_TEST_FILTER` for everyday work. Verification (p. 38) has the details.

2.8 Documentation tooling

The website and this manual come from one set of Markdown files in `docs/`.

Task	Command
Install the tools	`python -m pip install -r requirements-dev.txt`
Preview the website	`mkdocs serve`

Task	Command
Check the website	`mkdocs build --strict`
Build the PDF	`python tools/docs/build_manual.py` (needs the Pango library)
Redraw the figures	`python tools/docs/diagrams.py`
Export the figure data after a model change	`WScilex-cli -quit -f tools/docs/export_figures.sce`
Run the Scilab examples of the documentation	`WScilex-cli -quit -f tools/docs/check_examples.sce`

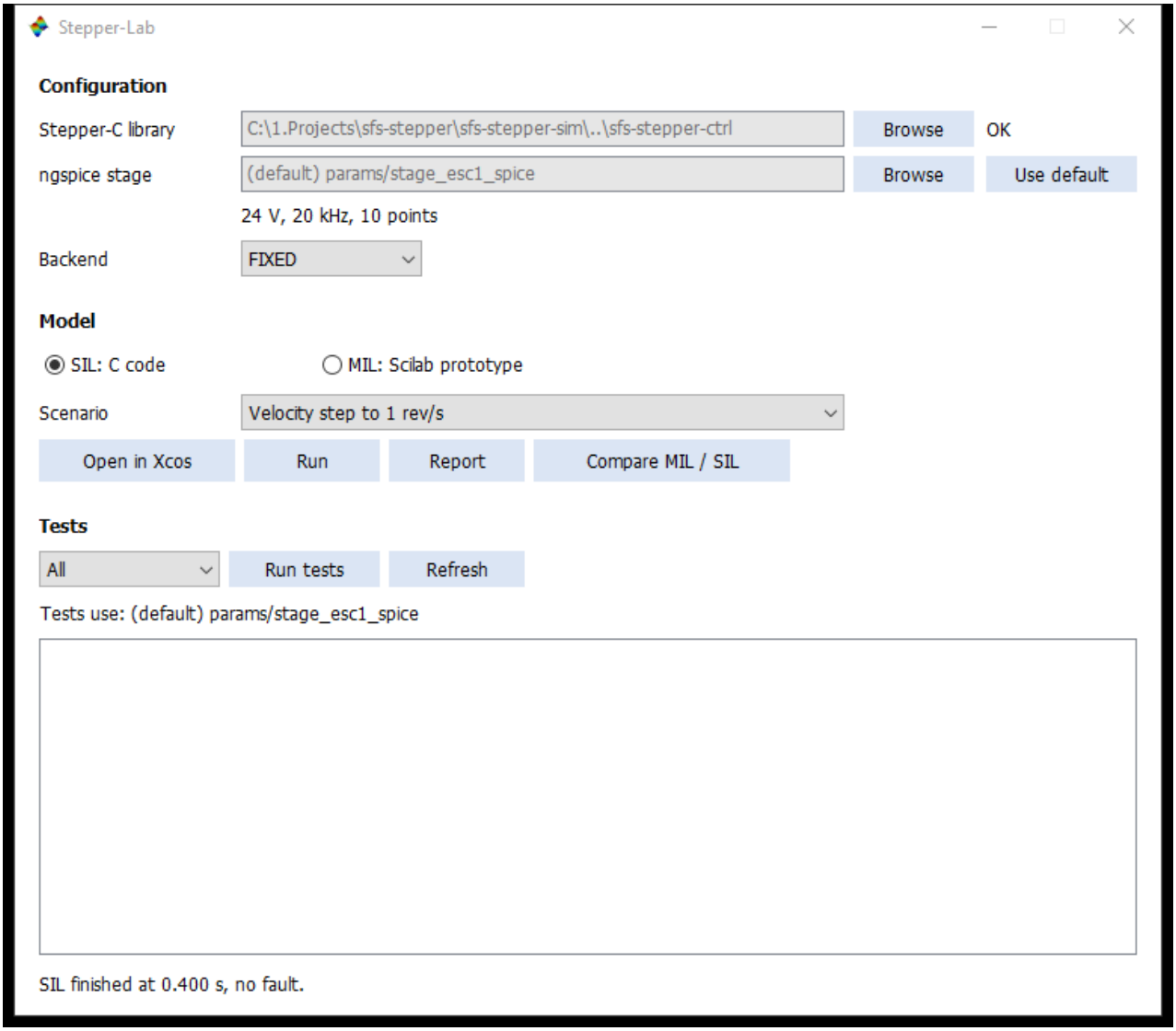
On Linux, the command is `scilab-cli`. The version printed in the manual comes from the git tag `v<version>`. Without a tag it reads `development`.

CHAPTER 3

The Lab Window and Diagrams

The Lab is the working place of this project: one window for the configuration and the runs, and one diagram layout that takes any input signal. Open `start_lab.sce` and press F5.

3.1 The window



Part	Controls	What it does
Config-uration	*Stepper-C library*, **Browse**	Folder of the library, kept for the next starts. `OK` or the name of the missing file appears next to it
Config-uration	*ngspice stage*, **Browse**, **Use default**	The stage file. The line below shows its bus voltage, frequency and number of table points, and warns when the board runs other values
Config-uration	*Backend*	`FIXED` or `FLOAT`. One Scilab session links one library, so a change after linking asks for a restart

Part	Controls	What it does
Model	*SIL: C code, MIL: Scilab prototype*	Which controller the buttons below use
Model	*Scenario*	One of the eight ready experiments
Model	**Open in Xcos**	Opens the scenario as a diagram, from a temporary file
Model	**Run**	Runs it without windows. The result is in the variable `sil`, and the status line names the first fault
Model	**Report**	Draws the last run with `sil_report()`
Model	**Compare MIL / SIL**	Runs both and overlays iA, speed and commanded position, with the largest difference up to the first fault
Tests	Group, **Run tests**, **Refresh**	Starts `tests/run_all.sce` in a second Scilab process for All, SIL, MIL or Model. **Refresh** reads the results so far, FAIL first

The settings are saved per user; see Settings file (p. 24).

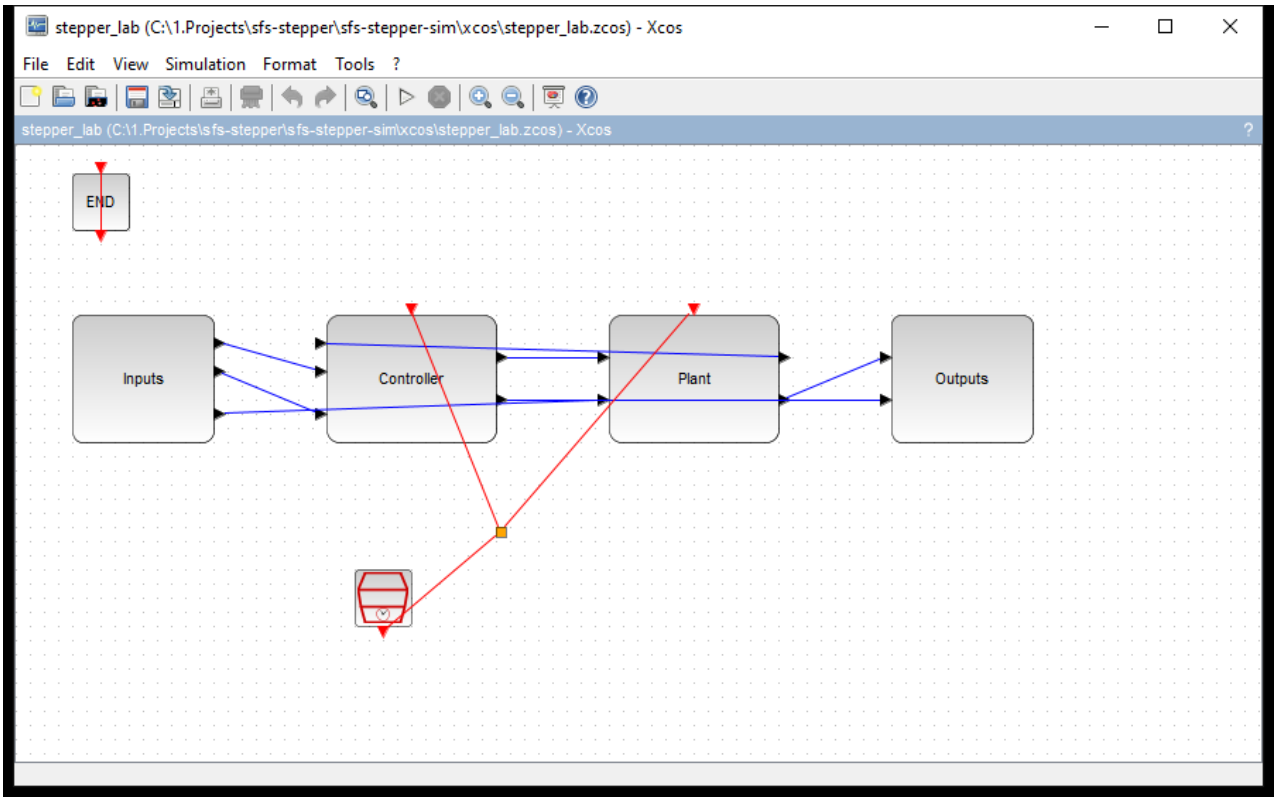
3.2 Scenarios

No.	Name	Time	Source	Expected
1	Current step 0.5 → 2 A, rotor locked	0.03 s	Step on the run current at 10 ms	No fault
2	Move 2 revolutions	0.8 s	Position command 6400 microsteps	No fault
3	Velocity step to 1 rev/s	0.4 s	Step on the command at 50 ms	No fault
4	Velocity sine, 1 rev/s, 2 Hz	0.6 s	Sine on the command	No fault
5	Velocity ramp to 5 rev/s	0.5 s	Ramp from 50 ms	No fault
6	Load step 0.25 N·m, below the pull-out torque	0.4 s	Step on the load at 0.2 s, stall detection on	No fault
7	Load step 0.5 N·m, above the pull-out torque	0.4 s	Step on the load at 0.2 s, stall detection on	Stall fault
8	Stall at the hold current, 0.12 N·m	0.62 s	Constant load, 1.5 A run, 0.5 A hold after 0.5 s, stall detection on	Stall fault

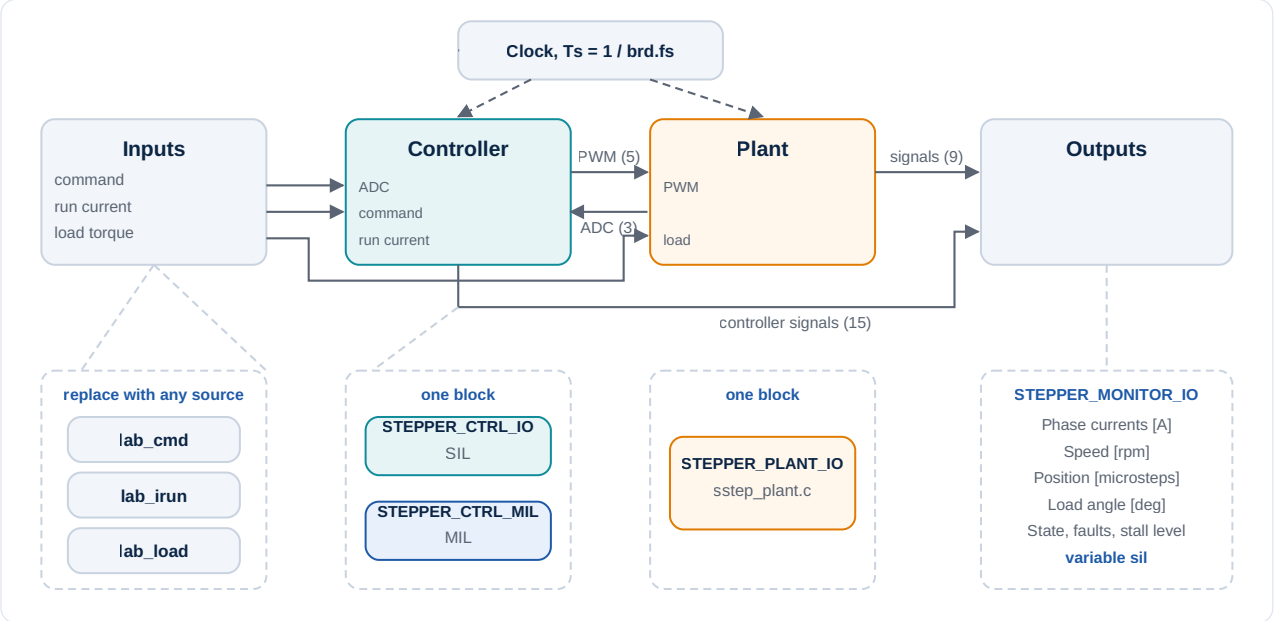
Test S10 runs all eight and checks these expectations.

3.3 The diagram

`xcos/stepper_lab.zcos` (SIL) and `xcos/stepper_lab_mil.zcos` (MIL) have the same four superblocks. Double-click one to open it.



The figure below shows what the superblocks hold and how they connect.



Superb-lock	Holds	You change
Inputs	Three `CONST_m` sources: `lab_cmd`, `lab_irun`, `lab_load`	Replace a source with any block of the Xcos palette
Controller	`STEPPER_CTRL_IO` (SIL) or `STEPPER_CTRL_MIL` (MIL)	Nothing
Plant	`STEPPER_PLANT_IO`: power stage, motor and ADC	Nothing; its parameters come from the context

Superb-lock	Holds	You change
Outputs	`STEPPER_MONITOR_IO`, five named scopes, and the work-space block writing `sil`	Scope ranges, through `scope_i`, `scope_rpm`, `scope_pos`

Use your own signal

1. In the window, choose a scenario and press **Open in Xcos**. This links the library.
2. Open **Simulation → Set Context**. A scenario adds its own lines at the end of the context, such as `T_end`, `lab_mode` or `lab_cmd`, and these win over the lines above them. Change your values there.
3. Open the superblock **Inputs** and replace the source with, for example, `STEP_FUNCTION`, `GENSIN_f` or `RAMP`.
4. Press Start. The scopes show the run, and `sil` holds the results when it ends.

To open `xcos/stepper_lab.zcos` directly, first run `exec("loader.sce", -1);` and `sil_build("FIXED");` in Scilab. Otherwise the functions of the context and the C blocks are not there, and Start fails.

The three inputs

Input	Unit	Read	Limits
Motion com-mand	Microsteps/s with `lab_mode = 1`; microsteps with `lab_mode = 2`; unused with `lab_mode = 0`	Once per slow period (1 kHz)	Velocity to ± `mp.v_max`; position to the `int32_t` range
Run cur-rent	A	Once per slow period	1 mA to the smaller of the rated current and the overcurrent trip. The hold current be-comes the smaller of hold and run
Load torque	N·m	Input of the plant	None. A value that is not finite keeps the last one, without a flag in column 23

`lab_mode`	Motion command	Library call in SIL
1	Velocity	`sstep_run()`
2	Position	`sstep_move_to()`
0	The table `cmds`, `[t kind value]`: 1 MOVE, 2 RUN, 3 STOP	As in the table

Behavior of the command and current inputs:

- A value is rounded and limited. A `NaN` keeps the last value. In both cases column 23 of `sil.values` shows the flag for that slow period.
- A new value is applied only when it differs from the last one.
- The motion generator of the library still limits acceleration and speed, so the applied com-mand (column 20) is not the speed of the rotor.
- After a fault the inputs are ignored, the outputs stay off, and the simulation goes on to `T_end`.

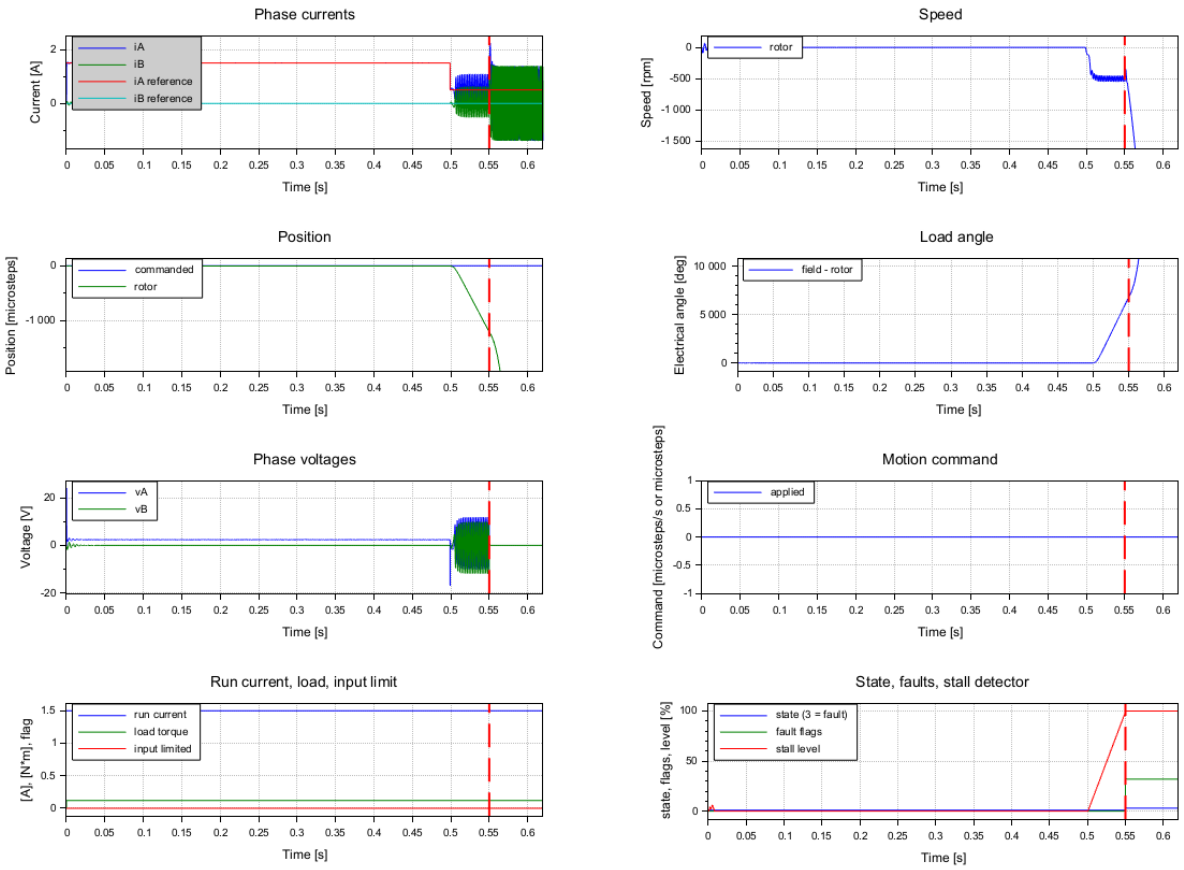
3.4 Results

The diagram writes `sil.time` and `sil.values`; the columns are in Reference (p. 34).

```
res = lab_run("SIL", 7);        // load step above the pull-out torque
d = sil_report_data(res);
mprintf("%s at %.1f ms\n", d.fault_names, d.t_fault * 1e3);
```

```
// no-run
sil_report(res);        // opens a figure
```

`sil_report(sil, visible)` draws every group with title, unit, legend and grid, and marks the first fault. The vertical scales come from the run up to shortly after the fault, so what follows it does not hide the rest. `sil_report_data(sil)` gives the same content as data, without graphics.



3.5 Common mistakes

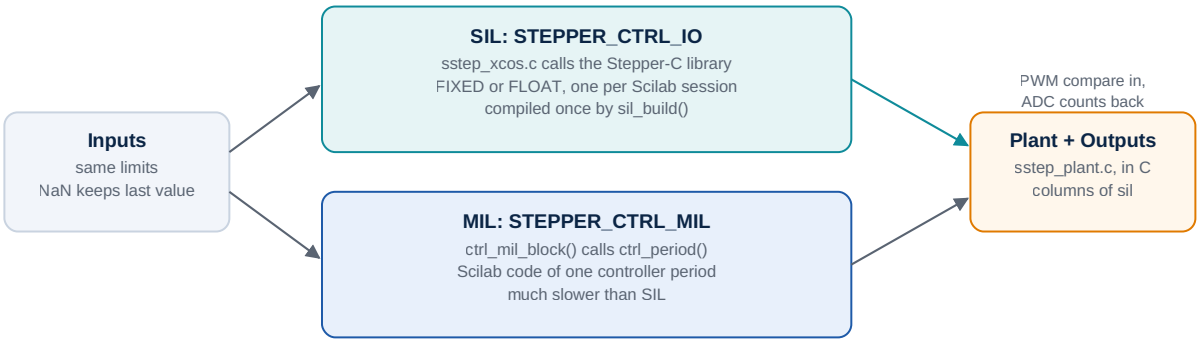
Symptom	Cause	What to do
The status asks for the folder of Stepper-C	The library was not found in the search order	Press **Browse**; the folder must hold `sstep.h`, `inc/sstep_ctrl.h` and `src/sstep_ctrl.c`
"Restart Scilab to use FLOAT"	This session already linked the other backend	Restart Scilab

Symptom	Cause	What to do
"Restart Scilab to link the library of the new folder"	The folder changed after linking	Restart Scilab
The speed does not follow a fast command	The motion generator limits it with `mp.v_max`, `mp.acc`, `mp.dec`	Raise these in the context; column 20 shows what was applied
Column 23 is 1	An input is out of range or `NaN`	See the limits above
MIL outputs stay at zero with state 3 and flag 16	The configuration is one the library refuses, or it asks for voltage drive	Read the printed message; `lab_run()` raises it as an error
A run in MIL takes long	The prototype is interpreted Scilab: 15 s per simulated second on a 0.15 s run	Use SIL for everyday runs
A selected stage file shows a message and is not used	The file is not valid; the message names the file and the field	Fix the export, or press **Use default**

CHAPTER 4

MIL and SIL

Both modes run the same plant in the same diagram layout. They differ only in the controller block, so a difference in the results points at the controller.



4.1 What runs in each

	SIL	MIL
Block	`STEPPER_CTRL_IO`	`STEPPER_CTRL_MIL`
Controller	The Stepper-C library, called by `sil/c/sstep_xcos.c`	`ctrl_mil_block()`, which calls `ctrl_period()`
Numeric form	Q15 or float, as on the MCU	Scilab doubles, with the integer motion generator of the library
Plant	`sil/c/sstep_plant.c`	The same
Input limits	`sstep_xcos.c`	`ctrl_mil_limit()`, the same limits
Speed	Used for everyday runs	15 s per simulated second on a 0.15 s run
Use it to	Test the code that ships	Study or change the algorithm without compiling it

The compiled library is needed in both modes, because the plant and the monitor are C blocks in MIL too. `lab_run()` and **Open in Xcos** link it by themselves.

4.2 Backends

Stepper-C has two numeric backends. `sil_build(backend)` compiles the chosen one into `sil/build/fixed` or `sil/build/float`, once, and again when a source file changes.

Backend	Numbers	For
`FIXED`	Q15	MCUs without an FPU
`FLOAT`	Single precision	MCUs with an FPU

A Scilab session links one backend. To change it, select the other one in the window and restart Scilab. `tests/run_all.sce` does this by starting a second Scilab process for FLOAT.

4.3 How close they are

Tests M01 and S01 to S06 compare the modes sample by sample. These are the limits they enforce and the values measured on the final run:

Comparison	Limit	Measured
MIL against SIL, current step, phase currents	5 mA	1.02 mA
MIL against SIL, move of 640 microsteps, phase currents	5 mA	1.28 mA
MIL against SIL, commanded position	1e-6 microsteps	0
MIL against SIL, load step above pull-out	Same fault within 1 ms	`OVERCURRENT` and `STALL` at 204.9 ms in both

After the rotor slips, the ADC quantization is amplified and the samples diverge, so a comparison of samples after a pull-out means little. The stall tests compare the fault and its time there.

`Compare MIL / SIL` shows this for any scenario, and reports differences only up to the first fault. From a script:

```
rs = lab_run("SIL", 1);
rm = lab_run("MIL", 1);
c = lab_compare_data(rs, rm);
mprintf("%s\n", c.panels(1).title);
```

4.4 A configuration the library refuses

Stepper-C checks its configuration in `sstep_init()`.

- **In SIL** the simulation stops with the message of the library.
- **In MIL** the block asks the library the same question at the start. If the answer is no, or if the configuration asks for voltage drive (`ctl.drive = 1`), which the prototype does not have, the outputs stay off, the state is 3, the fault flags are 16, and the reason is printed. `lab_run()` raises the reason as an error, so the window shows it. In Xcos no dialog appears and the run reaches `T_end`, because an error raised in this kind of block at its start ends Scilab 2027.

4.5 The `stepper_sil` diagram

`xcos/stepper_sil.zcos`, opened by `start_sil.sce`, is the first SIL diagram of the project and is kept as it was. It has a flat layout, and its input is the command table.

Context variable	Meaning	Example
`cmds`	Commands `[t kind value]`: 1 MOVE to, 2 RUN, 3 STOP	`[0 2 12800; 0.15 2 -12800]`
`loads`	Load torque steps `[t TL]`	`[0 0; 0.15 0.25]`
`brd.topology`	`"shared"` or `"full"`	`"full"`
`brd.stage`	`"avg"` or `"switching"`	`"switching"`
`mp`, `ctl`	Motion and control settings	`mp.v_max = 25600`
`locked`	1: rotor locked	`1`
`T_end`	Simulated time, s	`1.0`

Its results have the first 19 columns of `sil.values` (p. 34).

CHAPTER 5

Configuration

This chapter lists what you can set: the variables of the diagram context, the parameter structures, the search for the library and the stage file, and the environment variables.

5.1 Context of a diagram

Edit it in Xcos under **Simulation → Set Context**. The Lab diagrams define:

Variable	Default	Meaning
`mot`	`motor_17hs19_2004s1()`	Motor
`brd`	`board_bg431b_esc1()`	Power stage and ADC
`ctl`	`ctrl_default()`	Current loop and stall detection
`mp`	`motion_default()`	Microsteps, speed, acceleration, currents
`lab_mode`	1	1 velocity, 2 position, 0 the table `cmds`
`cmds`	Empty	Command table `[t kind value]`, for `lab_mode = 0`
`lab_cmd`	3200	Value of the command source: 1 rev/s at 16 microsteps
`lab_irun`	`mp.i_run`	Value of the run current source, A
`lab_load`	0	Value of the load torque source, N·m
`locked`	0	1: rotor locked, for current loop tests
`T_end`	0.6	Simulated time, s
`scope_i`, `scope_rpm`, `scope_pos`	3, 400, 7000	Scope ranges

5.2 Motor: `mot`

`motor_17hs19_2004s1()` holds the example motor. Copy the file for another motor.

Field	Value	Unit	Source
`R`	1.60	Ω	Datasheet, ±10 %; older listings give 1.4 Ω. Measure it
`L`	3.0e-3	H	Datasheet at 1 kHz, ±20 %
`I_rated`	2.0	A	Datasheet
`Kt`	0.2086	N·m/A	Derived: 0.59 N·m holding torque, both phases at 2 A. If the datasheet value is for one phase, it is 0.295. Measure the back-EMF

Field	Value	Unit	Source
`Ns`	200	Steps/rev	1.8° motor
`Td`	0	N·m	Detent torque, not in the datasheet. T11 uses 0.02 as an assumption
`J`	82e-7	kg·m²	Datasheet, rotor only
`B`	1e-5	N·m·s/rad	Viscous friction, assumption
`Tc`	0	N·m	Dry friction, unknown. T12 uses 0.01
`w_c`	0.05	rad/s	Dry friction is smoothed as `Tc tanh(w / w_c)`, so there is no stiction

To measure `Kt`, spin the shaft with a drill and measure the open-circuit phase voltage amplitude `E` and the speed `w`. Then `Kt = E / w`, in V·s/rad, which equals N·m/A.

5.3 Board: `brd`

`board_bg431b_esc1()` holds the example board.

Field	Value	Unit	Meaning
`vbus`	24	V	Bus voltage
`topology`	`"shared"`		`"shared"`: three legs; `"full"`: two H-bridges
`fs`	20000	Hz	PWM frequency, equal to the fast loop rate
`vbus_min`, `vbus_max`	10, 30	V	Undervoltage and overvoltage trips
`i_full`	4.0	A	Phase current at ADC full scale
`i_trip`	3.5	A	Overcurrent trip
`period`	4250	Counts	PWM compare value for 100 %
`adc_zero`, `adc_span`	2048, 2047	Counts	ADC reading at 0 A; counts from 0 A to `i_full`
`adc_vbus_full`	34.32	V	Bus voltage at the largest ADC reading
`adc_full`	4095	Counts	Largest ADC reading
`stage`	`"avg"`		`"avg"` averaged, or `"switching"` with dead time and diodes
`sw_nstep`	16		RK4 steps per PWM period in the switching stage, plus one at each edge
`dead_ns`, `dead_timer_ns`	From the stage file	ns	Effective dead time; dead time set in the timer
`rds_on`, `vd`	From the stage file	Ω, V	Leg resistance; body diode voltage
`leg_error`	`[]`		Table of leg voltage error against current; `[]` means ideal legs. Set `brd.leg_error = stage_active().leg_error` to use it

The ADC scaling is a model of the measurement chain. Replace it with the measured values of your board.

5.4 Controller: `ctl`

`ctrl_default()` gives the table below. Not every field reaches every controller: `delay` and `nsub` apply only to `sim_closed_loop()`, the prototype outside the diagrams. `ff` applies to the prototype and MIL; the library in SIL computes its own feed-forward from the back-EMF constant of the motor. `protect` is always on in MIL and has no effect in SIL, where the library has its overcurrent trip in current drive (`ctl.drive = 0`) and none in voltage drive.

Field	Value	Unit	Meaning
`bw_hz`	1000	Hz	Current loop bandwidth: `Kp = L wc`, `Ki = R wc`
`delay`	1	Periods	Computation delay, 0 or 1; on the MCU it is 1
`ff`	1		Back-EMF feed-forward, 0 or 1
`nsub`	4		RK4 sub-steps of the plant per fast period, in the prototype
`protect`	0		1: the overcurrent trip of the library in `sim_closed_loop()`
`stall_time_ms`	0	ms	Condition time before the stall fault; 0 is off, as in the library. The tests use 50
`stall_filter_us`	1000	µs	Time constant of the back-EMF filter
`stall_speed`	1600	Microsteps/s	0.5 rev/s: slip threshold, and the speed from which the synchronism check runs
`stall_sync_pct`	30	%	Smallest back-EMF along the motion, as a percentage of `Ke w`. The library example is 50

Optional fields: `ctl.mot` tunes the controller for a motor other than the plant (test T08), and `ctl.drive = 1` selects voltage drive, in SIL only.

5.5 Motion: `mp`

`motion_default()` gives:

Field	Value	Unit	Meaning
`M`	16		Microsteps per full step: 3200 per revolution
`f_slow`	1000	Hz	Slow loop rate
`v_max`	16000	Microsteps/s	5 rev/s
`acc`, `dec`	160000	Microsteps/s²	50 rev/s²
`i_run`	2.0	A	Current while moving
`i_hold`	1.0	A	Current at standstill
`hold_delay`	0.1	s	Standstill time before the hold current

5.6 The Stepper-C folder

`sil_ctrl_dir()` takes the first of these:

1. The environment variable `SSTEP_CTRL_DIR`.
2. The folder saved by the window. If it is no longer valid, a warning is printed and the search goes on.
3. `ext/Stepper-C`, then `../Stepper-C`.
4. `ext/sfs-stepper-ctrl`, then `../sfs-stepper-ctrl`.

A valid folder holds `sstep.h`, `inc/sstep_ctrl.h` and `src/sstep_ctrl.c`. Without one, the error message points to the window. `sil_build()` prints the folder it uses.

5.7 The stage file

The stage file carries the results of the ngspice model of a power stage. This project does not start ngspice. To create a file with sfs-stepper-power-spice (github.com/AntonVynohradov/sfs-stepper-power-spice):

1. Create an empty folder with a subfolder `params`, for example `my_stage/params`.
2. In `sfs-stepper-power-spice`, run `python tools/run_tests.py --export --sim <path to my_stage>`.
3. In the window, select `my_stage/params/stage_esc1_spice.json`.

Select that file, and not `export/stage_esc1.json`, which has no `format`, `vbus` or `fs` and which the window refuses. Do not point `--sim` at this repository unless you mean to replace its default stage, `params/stage_esc1_spice.*`.

The active file is the first of:

1. The environment variable `SSTEP_STAGE_FILE`.
2. The file saved by the window.
3. The default, `params/stage_esc1_spice`.

Simulations and tests use the active file, and `tests/run_all.sce` prints it at its start and end. A selected file that is not valid counts as a test failure, so a result never comes from another source without saying so.

Field	Check	Meaning
`format`	Must be 1	Version of the file format
`vbus`, `fs`	One finite number, positive	Operating point of the table, V and Hz
`r_leg`	One finite number, not negative	Leg resistance: switch and shunt, Ω
`vd`	The same	Body diode voltage, V
`dead_timer_ns`, `dead_eff_ns`	The same	Dead time in the timer; effective dead time
`leg_error_table`	1 to 16 rows `[i v]`, finite, current rising	Leg voltage error against leg current

The table is valid at its operating point. The window warns when `vbus` or `fs` differs from the board.

The default file gives:

Value	Result	Note
Effective dead time	133 ns	Timer 500 ns (assumption), minus the turn-off delay of the MOSFET; fitted for 1 to 3 A
Leg resistance	3.4 mΩ	Switch 1.9 mΩ, with the low-side 3 mΩ shunt (assumption), averaged
Body diode	0.73 V	At 2 A

Below 1 A the real dead time error is smaller than the model, because the phase current charges the switch node too slowly: 50 % of the model at 0.5 A, and 25 % at 0.25 A.

```
stg = stage_active();
mprintf("%g V, %g kHz, %d table points\n", stg.vbus, stg.fs / 1000, size(stg.leg_error, 1)
);
```

5.8 Motor parameters for C

`params/motor_*.json` holds the motor parameters for C users of the plant. The `.sci` file is the source: `python tools/export_params.py` writes the JSON, and `--check` fails when it is older than the `.sci` file. Both JSON kinds carry `"format": 1`, and readers check it.

5.9 Settings file

The window keeps its choices in `SCIHOME/sfs_stepper_lab.json`: `ctrl_dir`, `stage_file`, `backend`, `model` and `scenario`. A missing or damaged file, or a value of the wrong type, gives the default. `SSTEP_LAB_SETTINGS` names another file.

5.10 Environment variables

Variable	Meaning
`SSTEP_CTRL_DIR`	Stepper-C folder; wins over the window
`SSTEP_STAGE_FILE`	Stage file; wins over the window
`SSTEP_LAB_SETTINGS`	Settings file
`SSTEP_TEST_FILTER`	Regular expression over the test file names
`SSTEP_PYTHON`	Python with numpy and scipy for test X04, when another stage file is active
`SSTEP_SIL_BACKEND`, `SSTEP_STALL_PROTO`	Used between `run_all.sce` and its second Scilab process

CHAPTER 6

Motor and Control Model

This chapter gives the equations of the plant and of the prototype controller, in the order the code computes them. The plant is implemented twice, in Scilab (`model/`) and in portable C (`sil/c/sstep_plant.c`), and the controller prototype follows Stepper-C. Tests compare them.

Rule: a change of the model is made in `sstep_plant.c` and in the Scilab prototype together, and a change in Stepper-C is ported to the prototype and to the Python twin in the same step.

6.1 Motor

The state is $x = [i_A,\ i_B,\ \omega,\ \theta]$ in A, A, rad/s and rad, with $p = N_s/4$ electrical turns per mechanical turn.

$$
\begin{aligned}
L\frac{di_A}{dt} &= v_A - R\,i_A - e_A, & e_A &= -K_t\,\omega\sin(p\theta)\\
L\frac{di_B}{dt} &= v_B - R\,i_B - e_B, & e_B &= K_t\,\omega\cos(p\theta)\\
J\frac{d\omega}{dt} &= T_e - T_L - T_f\\
T_e &= K_t\big(-i_A\sin(p\theta) + i_B\cos(p\theta)\big) - T_d\sin(4p\theta)\\
T_f &= B\,\omega + T_c\tanh(\omega/\omega_c)\\
\frac{d\theta}{dt} &= \omega
\end{aligned}
$$

T_d is the detent torque, with one period per full step, and T_c is the dry friction, smoothed below ω_c, so the model has no stiction. Both are 0 by default, because the datasheet does not give them. The power balance $e_A i_A + e_B i_B = T_e\,\omega$ holds without detent torque (test T00).

Convention. With $i_A = I\cos\varphi$ and $i_B = I\sin\varphi$ the rotor settles at $p\theta = \varphi$. Increasing φ turns the rotor forward, and phase A leads phase B by 90°. Stepper-C uses the same convention.

6.2 Current loop

Every fast period $T_s = 1/f_s$, `ctrl_period()` does the following, as the fast loop of Stepper-C:

1. Sample i_A, i_B.
2. Compute the references and the back-EMF feed-forward at the commanded speed ω^*:

$$
i_A^* = I\cos\varphi, \qquad i_B^* = I\sin\varphi, \qquad v_{ff} = K_t\,\omega^*\begin{bmatrix}-\sin\varphi\\ \cos\varphi\end{bmatrix}
$$

3. Run one PI controller per phase, with pole-zero cancellation, so the closed loop is of first order with bandwidth $\omega_c = 2\pi\cdot$ `bw_hz`:

$$
v = K_p\,e + integ + v_{ff}, \qquad K_p = L\,\omega_c, \qquad K_i = R\,\omega_c
$$

4. Limit the voltage to what the power stage can give, scaling the vector and keeping its direction (`bridge_avg()`). With two H-bridges, $|v_A|,\ |v_B| \le V_{bus}$. With the shared leg, $\max(a,\ b,\ 0) - \min(a,\ b,\ 0) \le 1$ for the duty differences a and b, which gives a circular amplitude of $V_{bus}/\sqrt{2}$.

5. Update the integrator with back-calculation, $T_t = L/R$:

$$integ \leftarrow integ + K_i T_s\, e + \frac{T_s}{T_t}(v_{sat} - v)$$

6. The voltage is applied in the **next** period, the one-period computation delay of the MCU.

7. The prototype integrates the motor over T_s with RK4, with `ctl.nsub` sub-steps.

6.3 Motion generator

`motion_init()` and `ref_motion()` reproduce the integer algorithm of the slow loop of Stepper-C. Positions are in 1/65536 microstep, and velocities in 1/65536 microstep per slow period. The commands are `[t kind value]`: 1 MOVE target, 2 RUN velocity, 3 STOP, and in the prototype also 4 LOAD torque.

Every slow period:

- The velocity changes by at most `acc` when speeding up, or `dec` when slowing down.
- A move aims at the largest v_{new} for which this period's distance $(v + v_{new})/2$ plus the braking distance $v_{new}^2/(2\,dec)$ fits into the distance left:

$$v_{new} = \frac{-dec + \sqrt{dec^2 + 8\,dec\,|rem| - 4\,dec\,|v|}}{2}$$

  This is exact for the discrete profile: no overshoot, and an exact stop on the target. The last period stops from a velocity at or below 2 `dec`. - The fast loop advances the electrical angle by a constant increment per period, and it is set again every slow period. One electrical turn is 2^{32}, and one microstep is $2^{30}/M$. - The run current applies while moving, and the hold current after `hold_delay` without motion.

Test X03 compares this generator with the Python twin.

6.4 Power stage

`brd.stage` selects the model, in the prototype and in the plant block.

Averaged

`"avg"` is the default: each leg produces its mean voltage, duty × V_{bus}. It is fast and is used by most tests.

With `brd.leg_error = stage_active().leg_error` every leg also gets the voltage error measured with ngspice against the leg current, linear between the table points. This covers dead time, body diodes and leg resistance, at the speed of the averaged model. The table is valid at its operating point (`stg.vbus`, `stg.fs`). The default `[]` is ideal legs.

Switching

`"switching"` splits every PWM period at the switching instants of the centre-aligned legs, from counter peak to counter peak, which is between two current samples.

- Every turn-on is delayed by the dead time `dead_ns`.
- During the dead time the body diode carries the current, so the leg is at $-V_d$ or $V_{bus} + V_d$, depending on the current direction.
- Conducting switches drop $R_{ds(on)}\, i$.

It shows the current ripple, the peak current per period and the voltage error of the dead time:

$$\Delta v = -2\,\mathrm{sign}(i)\,(V_{bus} + 2V_d)\,\frac{t_d}{T_s}$$

per phase. The current loop removes this in steady state, but it distorts the current near zero crossings. The ripple on a locked phase is

$$\Delta i_{pp} = (V_{bus} - RI)\,\frac{|v_A|}{V_{bus}}\,\frac{T_s}{2L}$$

The results are in Measured Results (p. 31).

6.5 Stall detection

`stall_init()`, `stall_observe()`, `stall_check()` and `stall_status()` are the detector of Stepper-C. `ctl.stall_time_ms > 0` switches it on, and after the fault the outputs are off and the reference is frozen, as in the library.

Fast loop. The back-EMF comes from the limited voltage. The voltage computed two calls ago is the one applied from sample $k-1$ to k:

$$e_k = v_{k-2} - \left(\frac{L}{T_s} + \frac{R}{2}\right) i_k + \left(\frac{L}{T_s} - \frac{R}{2}\right) i_{k-1}$$

It is saturated at $\pm V_{bus}$, rotated into the commanded frame (d along the current, q 90° ahead), and filtered by a first-order low-pass with the time constant `stall_filter_us`.

Slow loop. In integer millivolts, against the command of the period that just ended, with $E = |K_e\,\omega|$, `sync` = `stall_sync_pct`, and e_s the back-EMF at `stall_speed`:

Case	The condition is true when
In motion, $\lvert v\rvert \geq$ `stall_speed`	$\mathrm{sgn}(v)\,e_q < \frac{sync}{100}E$, with e_d between $-(E/3 + RI/10)$ and $E/3 + 2RI/5$: a rotor that does not turn
Reversed back-EMF	$\mathrm{sgn}(v)\,e_q < -(E/2 + e_s)$ keeps the condition true for $N/2$ periods
At rest or slow	The deviation of $(e_d,\ e_q - K_e\omega)$ from its running mean, with a time constant of N periods, is larger than e_s

An up/down counter rises while the condition is true and falls otherwise. The fault trips at $N =$ `stall_time_ms` in slow periods. With `ctl.protect = 1`, an overcurrent trip while the counter is above zero reports both `SSTEP_FAULT_OVERCURRENT` and `SSTEP_FAULT_STALL`.

The results and the limits of the detector are in Measured Results (p. 31).

The Python twin has no stall detector; the X tests run with detection off.

6.6 Parameters of the example

The values and their sources are in Configuration (p. 20). Those marked as derived or assumed, in particular `Kt`, `B` and `Td`, should be measured on your motor before you trust results near a limit.

CHAPTER 7

Measured Results

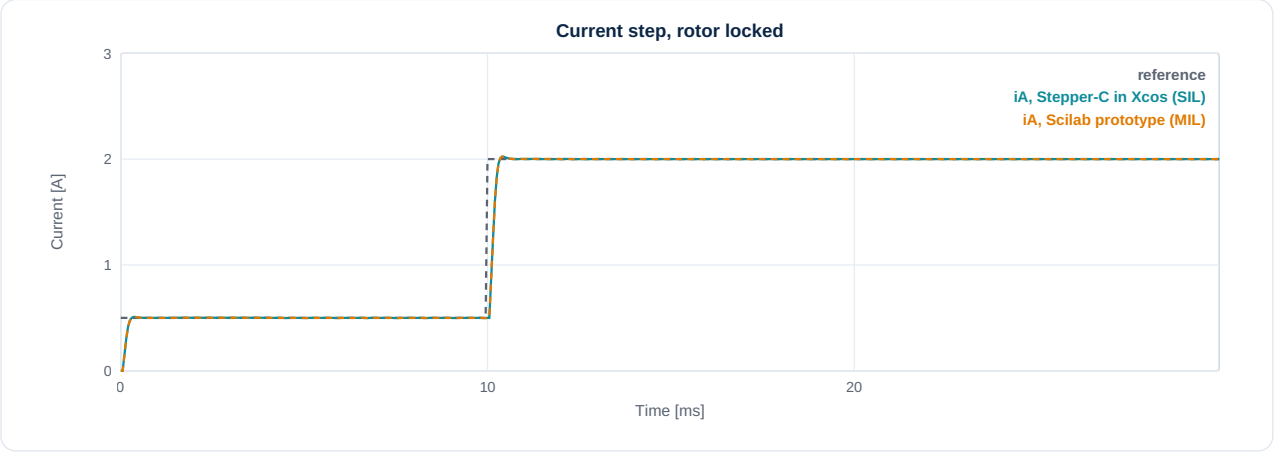
This chapter shows what the model says about the example: 17HS19-2004S1, 24 V, 2 A, 20 kHz, 1 kHz current loop bandwidth. "Measured" means measured on the model. These are simulation results, valid as far as the parameters (p. 20) are, and they have not been compared here with a real motor.

7.1 Current loop

A 0 → 2 A step on a locked rotor, in test T04, gives the same result on both topologies:

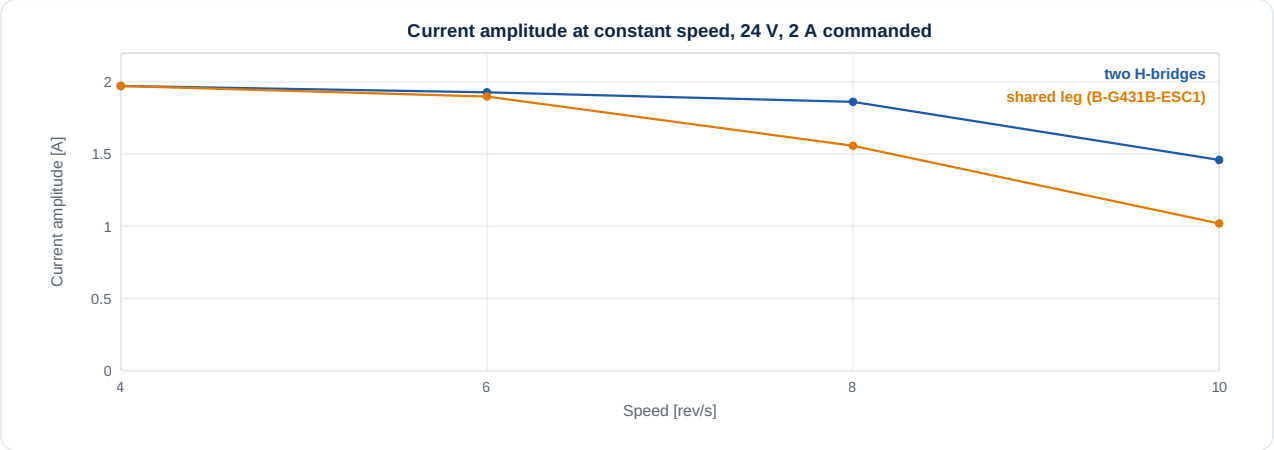
Overshoot	Rise (90 %)	Settling (2 %)	Steady error
1.12 %	0.350 ms	0.400 ms	0.009 %

The figure shows the first Lab scenario, a step from 0.5 to 2 A, with both controllers. The two curves lie on top of each other: they differ by at most 1.2 mA in this run, and the ADC step is 1.95 mA.



7.2 Voltage limit

The current amplitude falls with speed once the bridge reaches its voltage limit. The shared leg reaches it first, because its largest amplitude is $V_{bus}/\sqrt{2}$.



Speed	Two H-bridges	Shared leg (B-G431B-ESC1)
4 rev/s	1.97 A	1.97 A
6 rev/s	1.93 A	1.90 A
8 rev/s	1.86 A	1.56 A
10 rev/s	1.46 A	1.02 A

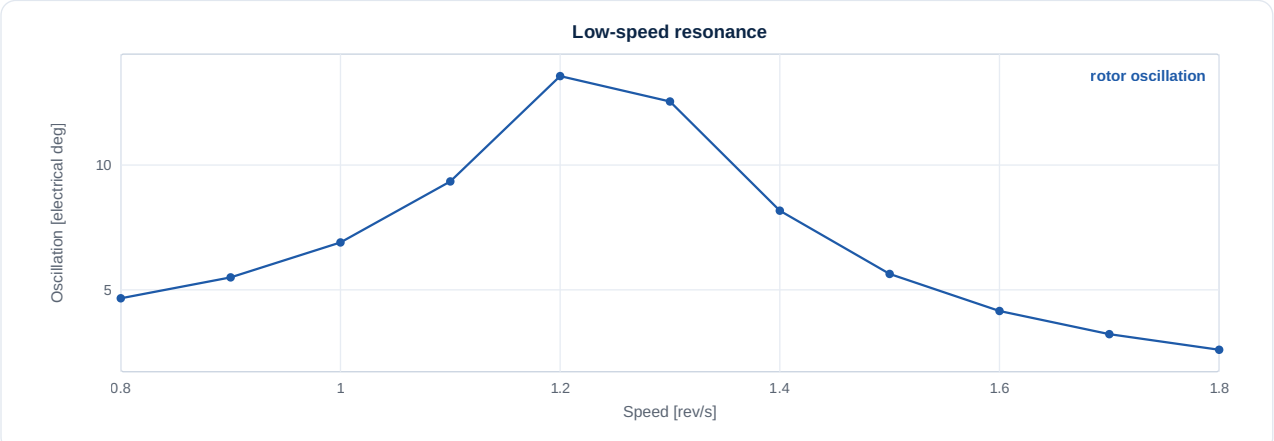
Below the limit the amplitude drops slightly with speed, because the current loop lags the rotating reference.

Higher speeds are left out: from about 11 rev/s on the shared leg the operating point is a limit cycle (see below), so a mean amplitude depends on the time window it is taken over.

7.3 Resonance

Natural frequency. The rotor on the current vector oscillates at $\sqrt{pK_tI/J}/2\pi$, which is 253.8 Hz at 2 A without load. The loop gives 248.1 Hz (test T11).

Low-speed resonance. A detent torque excites it when $4pn$ equals the natural frequency, about 1.27 rev/s. With 0.02 N·m, which is an assumption and not a datasheet value, the rotor oscillation peaks at 1.2 rev/s, at 13.6 electrical degrees. Avoid running continuously around this speed, or add load inertia or damping.



Mid-band instability. Below the voltage limit the current loop damps the rotor. Once the bridge limits the voltage the oscillation grows. A higher bus voltage moves this up.

Speed, shared leg, 24 V	Oscillation (electrical)	Voltage scale
5 rev/s	0.0°	1.00
8 rev/s	1.3°	0.51
11 rev/s	47.6°, a limit cycle	0.30

7.4 Parameter uncertainty

With the gains for the nominal motor and a different plant (test T08):

Plant	Overshoot	Settling (2 %)	Steady error
R ×0.9, L ×0.8	5.16 %	0.450 ms	0.060 %
R ×0.9, L ×1.2	2.15 %	1.000 ms	0.216 %
R ×1.3, L ×0.8 (hot winding)	1.99 %	1.600 ms	0.344 %
R ×1.3, L ×1.2	−0.06 %	0.550 ms	0.060 %

A ramp to 4 rev/s stays synchronous with R ×1.3, L ×0.8, J ×5 and 0.15 N·m of load: the load angle reaches 46.3°.

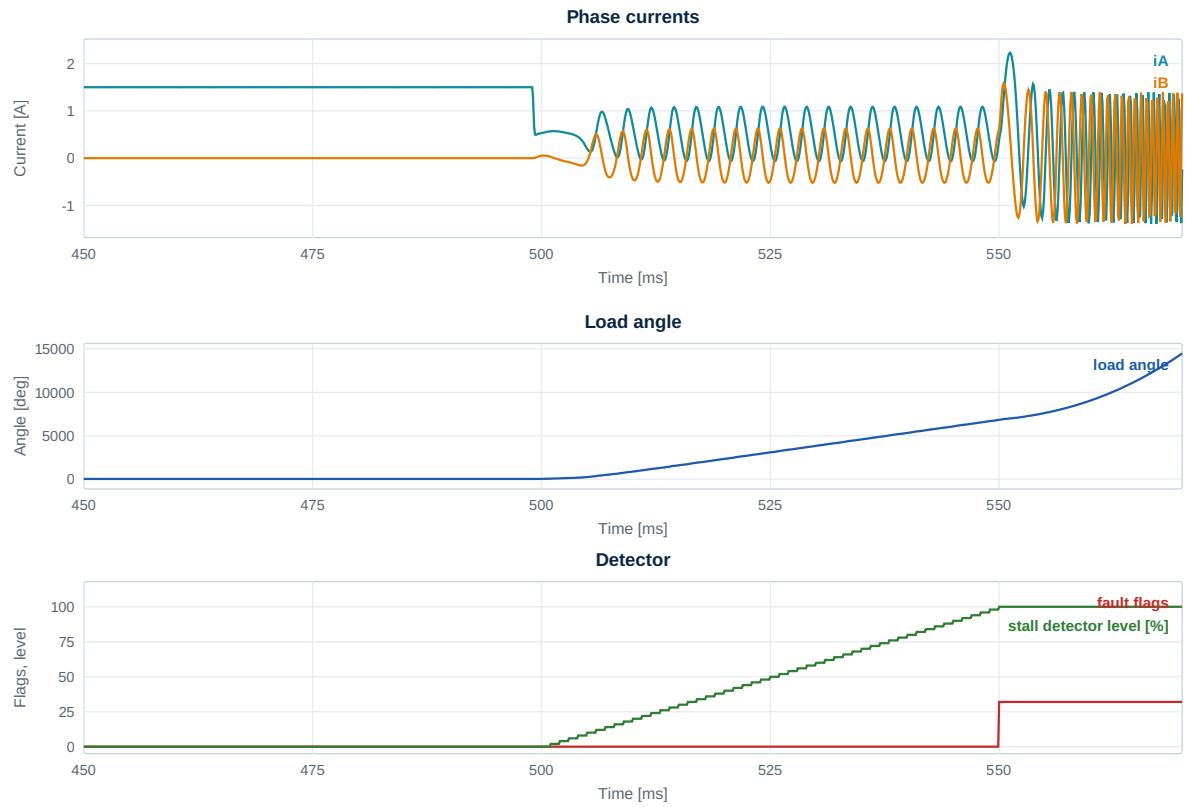
7.5 Power stage

Result	Value	Test
Ideal switching stage against the averaged model	At most 0.0113 mA	T10
Ripple at 2 A, locked phase	23.11 mA p-p, equal to the formula	T10
Dead time voltage error at 133 ns	0.135 V per phase, 0.07 mV from the formula	T10
Peak current at 4 rev/s with dead time	2.027 A	T10
Commanded voltage against the switching stage, 4 rev/s, ideal legs	118 mV	S05
The same with the ngspice leg error table	19 mV, amplitude 1.9702 A against 1.9701 A	S05

7.6 Stall detection

Configuration: `ctrl_default()` with `stall_time_ms = 50`. Pull-out means a load angle beyond 90°. Steps lost is the difference between commanded and rotor position at the fault. The values are from the prototype and the C library, on the averaged stage with the ngspice leg error and on the switching stage. Where they differ, the table gives a range.

The figure shows Lab scenario 8: the hold current starts at 500 ms, the rotor slips under 0.12 N·m, the detector level rises to 100 %, and the fault switches the outputs off at 550.0 ms.



Must trip	Fault	Time	After the pull-out	Full steps lost
0.12 N·m, run 1.5 A, hold 0.5 A after 500 ms	Stall	550.0 ms	48.1 ms	76.4 to 76.7
Blocked rotor, run 1 rev/s from rest	Stall	60.0 ms	45.9 ms	10.0
0.5 N·m load step at 1 rev/s	Overcurrent and stall	154.9 ms	3.7 ms	13.4
Overload at start: 0.5 N·m, run 1 rev/s	Overcurrent and stall	39.5 to 40.3 ms	38 to 39 ms	1298 to 1354
Heavy axis, J ×100, 0.5 N·m at 1 rev/s	Stall	212.0 ms	57.6 ms	36.1
Heavy axis, J ×20, 0.5 N·m at 3 rev/s	Stall	316.0 ms	61.1 ms	162.9

Must not trip	Largest detector level
Enable, run → hold 2 → 1 A, plant R +30 %, L −20 %	52 to 54 %
Enable, run → hold 1.5 → 0.5 A with 0.07 N·m	6 %
0.30 N·m at ±1 and ±3 rev/s, plant R +30 %, L −20 %	18 %
4 rev/s, plant R +30 %, L −20 %, J ×5, 0.15 N·m	26 %

Must not trip	Largest detector level
Move of 6400 microsteps; reversal ±4 rev/s; 8 rev/s on both topologies	0 %
Detent 0.02 N·m, 0.8 to 1.8 rev/s	0 %
Mid-band 5, 8 and 11 rev/s, load angle up to 107°, no step lost	0 %
8 rev/s at 18 V, 9 rev/s at 20 V	0 %
11 rev/s at 24 V, plant L +20 %, load angle up to 149°, no step lost	2 %

The verdict asks for the stall flag in the **first** fault, also together with the overcurrent flag: a bare rotor torn away by a large load reaches the overcurrent trip before `stall_time_ms`.

Agreement. The run in Xcos equals the prototype: the same fault within one slow period, and currents within 5 mA while the rotor is synchronous. After a pull-out the free slip amplifies the ADC quantization, and the 11 rev/s limit cycle grows from it, so those cases are compared by their verdict only.

After a fault. The outputs are off, which in the Xcos plant means 0 V on the phases, and the load torque stays as an input. A constant load then keeps accelerating the free rotor without limit, and the simulated currents and speed reach values that mean nothing. Read a result up to the fault.

Limits of the detector:

- A rotor that stops while the command is below `stall_speed` is not seen. A rotor blocked at 0.4 rev/s gives no fault and a detector level of 0 % in 0.5 s, while blocked at 1 rev/s it trips at 60.0 ms.
- A load angle swinging repeatedly beyond about 120° near the voltage limit is reported before a full step is lost. At 11 rev/s and 20 V the fault comes at 250.0 ms, with the rotor 0.83 full steps from the command.

7.7 Where these numbers come from

Each comes from a line of the test log of the final full run, or from the data of a figure, which `tools/docs/export_figures.sce` writes with the setups of the tests. Run the tests or the export to measure them again; Verification (p. 38) shows how.

CHAPTER 8

Reference

This chapter lists the columns of the results, the Xcos blocks, the main Scilab functions, the C module of the plant and the folders of the repository.

8.1 Results: `sil.values`

A run writes `sil.time` (s) and `sil.values`, one row per fast period. The Lab diagrams have 23 columns; `stepper_sil` has the first 19.

Col.	Signal	Unit	Note
1, 2	iA, iB	A	Phase currents of the plant
3, 4	iA reference, iB reference	A	From the controller
5	Speed	rpm	Rotor
6	Load angle	Electrical deg	Commanded field minus rotor
7	Commanded position	Microsteps	
8	Rotor position	Microsteps	
9, 10	vA, vB	V	Commanded phase voltages: compare values × Vbus, 0 when the outputs are off. Dead time, diodes and the leg error table are not in them
11, 12	Largest |iA|, |iB| in the period	A	With the PWM ripple when `brd.stage = "switching"`
13	State		3 = fault
14	Fault flags		1 `SSTEP_FAULT_OVERCURRENT`, 16 `SSTEP_FAULT_EXTERNAL` (MIL with a refused configuration), 32 `SSTEP_FAULT_STALL`
15, 16	Back-EMF estimate d, q	mV	From `sstep_get_status()`, or `stall_status()` in MIL
17	Rotor velocity estimate	Microsteps/s	
18	Load angle estimate	deg	
19	Detector level	%	100 % is the trip
20	Applied motion command	Microsteps/s or microsteps	After the limits
21	Applied run current	A	After the limits

Col.	Signal	Unit	Note
22	Load torque	N·m	Input of the plant
23	Input limit flag		1: an input was limited, or `NaN`, in this slow period

8.2 Xcos blocks

All are in `xcos/blocks/`. Each runs at the fast loop rate.

Block	Inputs	Outputs	Parameters
`STEPPER_CTRL_IO`	ADC counts (3), command, run current	Compare values with enable (5), controller signals (15)	`sil_ctrl_rpar(mot, brd, ctl, mp, cmds)`, `lab_mode`
`STEPPER_CTRL_MIL`	The same	The same	`ctrl_mil_config(mot, brd, ctl, mp, cmds)`, `lab_mode`
`STEPPER_PLANT_IO`	Compare values with enable (5), load torque	ADC counts (3): iA, iB, Vbus; plant signals (9)	`sil_plant_rpar(mot, brd, [0, 0], locked, 4)`
`STEPPER_MONITOR_IO`	Plant signals (9), controller signals (15)	Scope groups (4, 1, 2, 1, 3), all columns (23)	`[mot.Ns, mp.M]`
`STEPPER_CTRL`, `STEPPER_PLANT`, `STEPPER_MONITOR`	Used by `stepper_sil`		Command table and load table in their parameters

8.3 Scilab functions

Lab

Function	Purpose
`lab_window(visible)`	Opens the window. `start_lab.sce` calls it
`lab_run(kind, k)`	Runs scenario `k` with `"SIL"` or `"MIL"` without windows. Returns the results, and the diagram as a second output
`lab_scenarios()`	The list of scenarios
`lab_scenario_diagram(kind, k)`	The diagram of a scenario
`lab_diagram(kind, path)`	Builds a Lab diagram, and writes it when `path` is given
`lab_set_source(scs_m, label, b)`	Replaces a source block of Inputs
`lab_compare_data(rs, rm)`	The comparison of two results, as data
`sil_report(sil, visible)`	Draws a result
`sil_report_data(sil)`	The panels of the report and the first fault, as data
`lab_settings_load()`, `lab_settings_save(s)`	The saved settings

Build and configuration

Function	Purpose
`sil_build(backend, force)`	Compiles if a source changed, and links
`sil_ctrl_dir()`	The Stepper-C folder, or an error that names the window
`sil_check_ctrl_dir(d)`	`""`, or the first missing file
`sil_check_rpar(rpar)`	`""`, or what `sstep_init()` says about a configuration
`stage_load(path)`	Reads and checks a stage file
`stage_source()`	The active file, the reason a selected one is not used, and its content
`stage_active()`	The active stage
`motor_17hs19_2004s1()`, `board_bg431b_esc1()`, `ctrl_default()`, `motion_default()`	Parameters

Model

Function	Purpose
`sim_closed_loop(mot, brd, c, ref, refp, T, locked, st)`	The prototype in closed loop
`ctrl_period_init()`, `ctrl_period()`	One period of the controller, shared by `sim_closed_loop()` and MIL
`stepper_simulate(mot, src, srcp, t, x0, locked)`	The motor alone, with a voltage source
`motion_init()`, `ref_motion()`	The motion generator
`ref_current_step()`, `ref_const_speed()`, `ref_speed_ramp()`, `ref_angle_step()`	Other references
`bridge_avg()`, `bridge_switching()`, `bridge_leg_error()`	Power stage models
`stall_init()`, `stall_observe()`, `stall_check()`, `stall_status()`	The stall detector

8.4 The plant in C

`sil/c/sstep_plant.[ch]` holds the power stage, the motor and the ADC of the plant block in portable C, with no Scilab header. `sil/c/sstep_xcos.c` only wraps it. Other targets can compile the same file.

Function	Purpose
`sstep_plant_init()`	Sets up the model: rotor at rest, no current, outputs off. Returns 0, or -1 for an invalid parameter
`sstep_plant_set_compare()`	Takes the compare values; they drive the legs in the next period

Function	Purpose
`sstep_plant_period()`	Advances one PWM period, with a load torque
`sstep_plant_adc_sample()`	ADC counts of the present state: iA, iB, Vbus
`sstep_plant_theta()`	Mechanical rotor angle, unwrapped
`sstep_plant_leg_error()`	Leg voltage error of the table at a leg current

Build option	Default	Meaning
`SSTEP_PLANT_FLOAT`	0	1: single precision, for an MCU with an FPU; 0: double, as in Xcos
`SSTEP_PLANT_SWITCHING`	1	0: leaves the switching stage out, so no `qsort` is needed

`stage.off` selects what "outputs off" means: `SSTEP_PLANT_OFF_ZERO_VOLTAGE`, as the prototype and Xcos do, or `SSTEP_PLANT_OFF_OPEN`, open windings. The electrical angle is kept in [-π, π) with a separate turn count, so a single-precision build does not lose accuracy over long runs.

8.5 Folders

Folder	Contents
`params/`	Motor, board, controller and motion parameters, and the default stage file
`model/`	Model functions: equations, controller prototype, stall detector, reports, settings
`sil/`	C blocks and plant (`sil/c/`), build of the library; compiled files in `sil/build/`
`xcos/`	Diagrams, their generators, the window and the scenarios; blocks in `xcos/blocks/`
`tests/`	Test scripts, runner and reference data
`tools/`	Python twin and export of the motor parameters; documentation tools in `tools/docs/`
`demos/`	Scripts with plots
`docs/`	This documentation

CHAPTER 9

Verification

This chapter describes how the model is checked and how to run the checks. The final full run gave **433 PASS, 0 FAIL, 0 SKIP** with both backends, on Windows 10 with Scilab 2027.0.0, the default stage file and Stepper-C at commit `3ef0f7e`.

9.1 How the model is checked

Check	Against	Tests
Motor and control behavior	Theory: time constants, power balance, ripple, spin-down	T00 to T13, B01
Scilab model	Python twin, written separately, with reference data in `tests/reference/`	X01 to X04
C library in Xcos	Scilab prototype, sample by sample	S01 to S06
Lab diagrams and report	The specified limits, labels and scenarios	S07 to S11
MIL	SIL	M01
Window and settings	The specified order and messages	C01 to C03

9.2 Running the tests

```
WScilex-cli -quit -f tests/run_all.sce      # Windows
scilab-cli -quit -f tests/run_all.sce       # Linux
```

The exit status is 1 when a test fails. The log starts and ends with the library folder and the stage file in use. A full run took 386 minutes.

Task	How
Run a group from the window	Group All, SIL, MIL or Model, then **Run tests** and **Refresh**
Run selected tests	Set `SSTEP_TEST_FILTER`, a regular expression over the file names; see below
Run the second backend	`run_all.sce` starts it by itself, through `tests/run_sil_backend.sce`

For example, to run C01 and T13:

```
SSTEP_TEST_FILTER="C01|T13" scilab-cli -quit -f tests/run_all.sce
```

In PowerShell, first set `$env:SSTEP_TEST_FILTER = "C01|T13"`. A filter that matches no file runs nothing and still reports `0 failure(s)`, so check that the log lists the tests you expected.

9.3 Testing another stage file

Select the file in the window, or set `SSTEP_STAGE_FILE`, and run the tests. Simulations and tests use it, and the log names it.

- A file that is not valid counts as a failure, and the tests do not fall back to the default silently.
- Test X04 computes its reference with the Python twin for the active file, which needs `SSTEP_PYTHON` or a Python in `PATH` with numpy and scipy. Otherwise it reports SKIP.
- Tests that compare against numbers from the default stage can fail with a stage of other values. For example, S05 fails when dead time and diode voltage change while the leg error table stays the same, because the averaged and the switching model then describe different stages. Decide whether a failure comes from the new stage values before treating it as a defect.

Selected tests (`C0|X04|T10|S01`) ran on another stage file. A full run was made only on the default one.

9.4 Python twin

`tools/` holds a separate implementation of parts of the model, in Python with numpy and scipy. It has no stall detector, so the X tests run with detection off.

```
python tools/ref_model.py                # motor tests
python tools/ref_ctrl.py                 # current loop tests
python tools/ref_motion.py               # motion tests
python tools/ref_switching.py            # switching stage tests
python tools/ref_ext.py                  # T08, T11, T12
```

Add `--write` to the first four to regenerate `tests/reference/*.csv`. With another stage, `ref_switching.py --stage <file> --out <file>` writes the reference elsewhere.

9.5 Test list

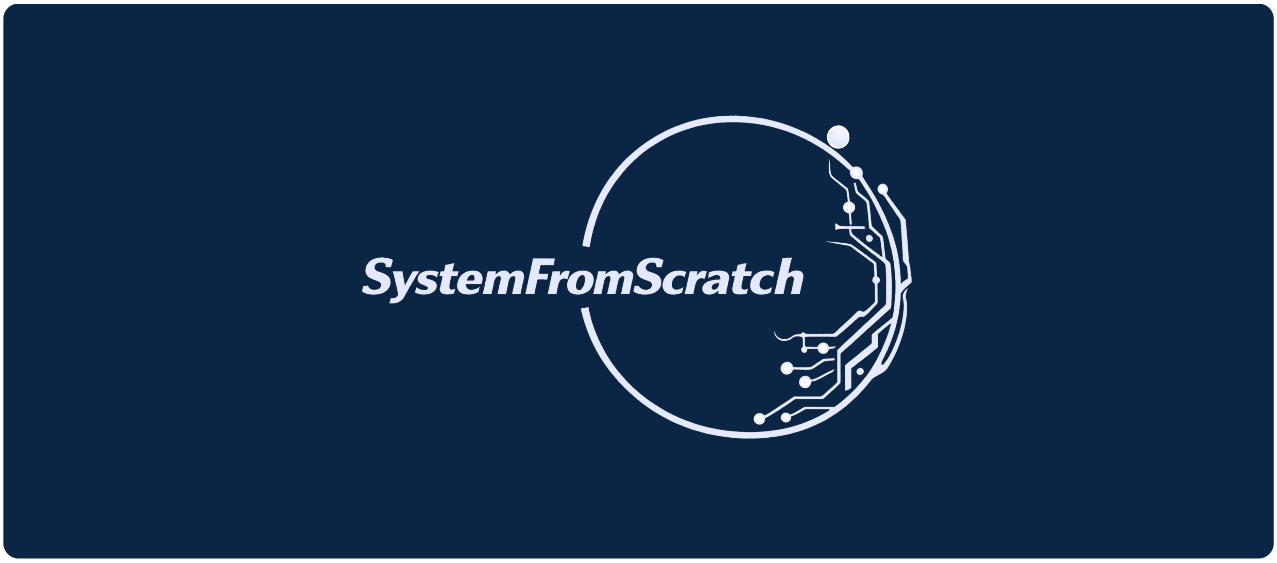
Test	Stimulus	Check
T00	Random states, current vector	Power balance, stable rotor position
T01	Constant voltage on phase A	Steady-state current V/R, locked and free rotor
T02	Voltage step, locked rotor	Current rises with τ = L/R
X01	Rotating voltage, 5 Hz electrical	Scilab equals the Python twin
B01	Random voltage requests, both topologies	Duties 0 to 1, direction kept, exact when feasible, largest amplitude
T03	Microstepping 2 A, 10 Hz electrical	A leads B by 90°, amplitude, rotor follows the field
T04	Current step 0 → 2 A, locked rotor, both topologies	Overshoot below 5 %, settling (2 %) below 1 ms, error below 0.1 %
X02	Current step, shared leg	Discrete loop equals the Python twin
T05	Move 2 rev, 5 rev/s, 50 rev/s²	Limits kept, target exact, load angle below 45°

Test	Stimulus	Check
T06	Load step at 1 rev/s: 0.25 and 0.5 N·m	Synchronous at the expected angle below Kt·I, slip detected above
T07	Reversal +4 → −4 rev/s	Synchronous, deceleration limit kept, final speed
T08	Plant R ×0.9 to 1.3, L ×0.8 to 1.2, J ×5, load 0.15 N·m	Overshoot below 10 %, settling below 2 ms, error below 1 %; ramp to 4 rev/s synchronous
T09	8 rev/s, both topologies	Shared leg reaches the voltage limit first, both synchronous
X03	Move 160 microsteps	Motion generator and loop equal the Python twin
T10	Switching stage: hold 2 A, ±2 A with dead time, 4 rev/s	Ideal switches equal the averaged model, ripple and dead time error equal the formulas, synchronous, peak below the trip
X04	Switching stage, shared leg, 4 rev/s	Scilab equals the Python twin
T11	Angle step; speed sweep with detent; 5, 8, 11 rev/s	Natural frequency ±5 %, detent resonance ±10 %, oscillation below 1° and 5° below the voltage limit, above 20° beyond it
T12	Unpowered rotor, Tc 0.01 N·m, 50 rad/s	Spin-down time from the formula, ±1 %
T13	Stall scenarios in the prototype	Must trip: stall flag in the first fault, 0 to 70 ms after the pull-out. Must not trip: no fault, below 2 full steps from the command
S01	SIL: current step, locked rotor	Library in Xcos equals the prototype, currents below 5 mA
S02	SIL: move 640 microsteps	Currents below 5 mA, commanded angle identical, rotor below 0.05 microstep
S03	SIL: 8 rev/s, both topologies	Current amplitude as the prototype
S04	SIL: run 4 rev/s, switching stage	Currents below 5 mA, peak as the prototype
S05	SIL: run 4 rev/s, averaged with leg error against switching	Commanded vA within 40 % of the ideal-leg difference, amplitude below 2 mA
S06	SIL: the T13 scenarios on both stages, FIXED and FLOAT	Same verdicts; same fault within one slow period, currents below 5 mA while synchronous
S07–S09	Lab: signal inputs equal the table; out of range, sines; through a fault	Currents below 1e-9 A, values and flag as specified, the run reaches `T_end`
S10	Lab: four superblocks, file read back, columns 1–19 equal `stepper_sil`, scenarios	Below 1e-9, every scenario as expected
S11	Report	Every panel has title, unit and legend; first fault named
M01	MIL against SIL	Currents below 5 mA, commanded position below 1e-6 microsteps, same fault within 1 ms, input limits, refused configurations
C01–C03	Settings, library folder, stage file, window and its callbacks	Order and messages as specified

9.6 Known gaps

- Everything above is simulation. The model has not been validated against a measured motor in this manual, and several motor parameters are derived or assumed.
- All results come from Windows. The Linux path of `sil_build()` was not run for this manual.
- M01 does not cover sine inputs or the table mode in MIL.
- The look of the window and the diagrams has no automatic test.

Author & Company



Stepper-Lab is designed and maintained by **Anton Vynohradov** and published by **SystemFromScratch®** — secure embedded systems and compliance-ready engineering, from concept to certification.

Author

Anton Vynohradov · ORCID 0009-0005-6965-0911 (orcid.org/0009-0005-6965-0911)

Embedded Systems Engineer · Secure Product Architect · Founder of SystemFromScratch®

Embedded systems built from scratch — from bare metal to Linux, from first schematic to certification.

Anton designs and builds reliable, secure embedded products end to end: firmware, board support packages and drivers, RTOS and Embedded Linux platforms, hardware bring-up, and the security and compliance work needed to ship them. Anton is based in Poland and works remotely worldwide.

Area	Focus
Embedded software	Bare-metal, RTOS and Linux firmware in C/C++, built to be maintained for years
BSP & drivers	Bootloaders, Zephyr drivers and Linux kernel work for MCUs, SoCs and SoC FPGAs
Security & compliance	Secure boot, threat modelling, MISRA C/C++, EU Cyber Resilience Act (CRA) readiness
Industries	Industrial IoT, automotive, robotics, medical and consumer electronics

Technology: C · C++ · Python · Zephyr RTOS · FreeRTOS · Embedded Linux · Yocto · Buildroot · ARM Cortex-M · STM32 · Microchip FPGA · RISC-V · CAN / ISO-TP · ROS 2 · Docker · Git · CI/CD · MISRA C/C++ · EU CRA

SystemFromScratch®

SystemFromScratch® was founded by Anton Vynohradov. The company combines deep embedded engineering with cybersecurity and certification-focused product development, for teams in industrial IoT, automotive, robotics, medical and consumer electronics that need a partner who understands the whole stack — silicon, firmware, operating system and the regulations that govern them.

Service	What it covers
Embedded software development	Bare-metal, RTOS and Linux firmware for performance, reliability and long-term maintainability
BSP & device drivers	Board support packages, bootloaders and low-level drivers for MCUs, SoCs, FPGAs and custom hardware
RTOS & Embedded Linux	Zephyr, FreeRTOS and Embedded Linux (Yocto, Buildroot): porting, configuration, integration, reproducible builds
Hardware bring-up & integration	Schematic review, first boot, validation and production support, including SoC FPGA and motor-control platforms
Cybersecurity & CRA readiness	Secure boot, firmware integrity, threat modelling and a secure development lifecycle for the EU Cyber Resilience Act
Compliance & consulting	MISRA C/C++, static analysis, code and architecture reviews, compliance roadmaps

Stepper-Lab is one of the open-source projects in the SystemFromScratch® portfolio: public engineering work that can be inspected line by line.

Contact

Channel	Address
Website	systemfromscratch.com
E-mail	avynohradov@systemfromscratch.com
GitHub	github.com/AntonVynohradov
GitHub organization	github.com/SystemFromScratch
LinkedIn	linkedin.com/in/vynohradov
ORCID	0009-0005-6965-0911 (orcid.org/0009-0005-6965-0911)
Location	Poland — working remotely worldwide

Building a connected product, bringing up new hardware, or preparing for the Cyber Resilience Act? Get in touch through systemfromscratch.com or by e-mail.

Support and feedback

- **Bugs and feature requests:** open an issue on GitHub (github.com/AntonVynohradov/Stepper-Lab/issues). Please include your Scilab version, the Stepper-C version, the backend, the scenario or context, and the log of `tests/run_all.sce` if a test fails.
- **Commercial support, integration and custom development:** contact SystemFromScratch® through systemfromscratch.com.

Trademarks and acknowledgements

SystemFromScratch® is a registered trademark. The Apache License 2.0 grants no rights to use the SystemFromScratch® name or logo; they may not be used to endorse or promote products derived from Stepper-Lab without prior written permission.

Motor, board and tool names (17HS19-2004S1, B-G431B-ESC1, STM32, Scilab, Xcos, ngspice) appear only to identify the hardware and tools the model describes or uses. STM32 is a registered trademark of STMicroelectronics International N.V.; Stepper-Lab is not affiliated with or endorsed by STMicroelectronics, any motor manufacturer or the owners of those tools.

APPENDIX A

Changelog

All notable changes to this project are documented in this file. The format is based on Keep a Changelog (keepachangelog.com/en/1.1.0/), and this project adheres to Semantic Versioning (semver.org/spec/v2.0.0.html).

A.1 [Unreleased]

Changed

- Public name: **Stepper-Lab** (file banners, window, README).
- `README.md` is short; its content moved to the documentation.
- `sil_report`: larger figure, legends inside the panels, scales from the run up to shortly after the first fault (`ylim` of `sil_report_data`).

Fixed

- `sil_ctrl_dir` finds a clone of the public repository, the folder `Stepper-C`, in `ext/` or next to this repository, before the old name `sfs-stepper-ctrl`.
- The window: the list of test groups starts on `All`; buttons look like buttons.

Added

- Documentation website (MkDocs Material, `mkdocs.yml`, `docs/`, published by `.github/workflows/documentation.yml`) and PDF manual (`tools/docs/build_manual.py`), from one set of sources. The version comes from the git tag.
- `tools/docs/export_figures.sce` writes the data of the result figures to `docs/assets/data/`; `tools/docs/diagrams.py` draws them; `tools/docs/check_examples.sce` runs the examples of the documentation.
- **stepper_lab** (`start_lab.sce`, `xcos/lab_diagram.sci`): a model for experiments, beside the unchanged `stepper_sil`. Superblocks Inputs, Controller, Plant, Outputs; any Xcos source drives the motion command (velocity or position), the run current and the load torque (`STEPPER_CTRL_IO`, `STEPPER_PLANT_IO`, `STEPPER_MONITOR_IO`); inputs out of range are limited, a fault does not end the simulation; five named scopes; `sil.values` columns 20 to 23; `sil_report(sil)` (`sil_report_data` without graphics).
- **MIL** (`xcos/stepper_lab_mil.zcos`, `STEPPER_CTRL_MIL`): the Scilab prototype in the same layout. `model/ctrl_period.sci` holds one controller period, which `sim_closed_loop` calls too, so the algorithm has one implementation.
- **Window** (`lab_window`): library folder, ngspice stage file, backend, MIL or SIL, scenarios (`lab_scenarios`, `lab_run`), comparison, tests in a second Scilab process. Settings per user in `SCIHOME/sfs_stepper_lab.json` (`lab_settings_*`).
- Library folder: `sil_ctrl_dir` also looks in the saved folder, after `SSTEP_CTRL_DIR`.

- ngspice stage file: `stage_load`, `stage_active` (`SSTEP_STAGE_FILE`, saved, default); `board_bg431b_esc1` and the tests use the active file. `tests/run_all.sce` prints the library and the stage file, and fails on a selected file that is not valid. X04 computes its reference with the Python twin when another file is active (`tools/ref_switching.py --stage --out`, `SSTEP_PYTHON`).
- `SSTEP_TEST_FILTER` selects tests in `tests/run_all.sce`. Tests C01 to C03, S07 to S11, M01.
- Stall detection of Stepper-C 1.1.0 in the prototype: back-EMF observer (`stall_observe`, fast loop), decision with the up/down counter and the learned offset at rest (`stall_check`, slow loop, integer millivolts), constants (`stall_init`) and status estimates (`stall_status`), with the equations of the library documentation. `sim_closed_loop` runs them with the motion generator when `c.stall_time_ms > 0` (fourth output `sd`: fault time, flags, per-period back-EMF estimates and counter); after a fault the outputs are off and the reference frozen, as in the library. Optional overcurrent trip of the library (`c.protect = 1`).
- `params/ctrl_default.sci`: `stall_time_ms` (0 = off), `stall_filter_us` 1000, `stall_speed` 1600, `stall_sync_pct` 30; `protect` 0.
- Averaged stage with the leg voltage error in the prototype (`model/bridge_leg_error.sci`, `brd.leg_error` in `sim_closed_loop`), the algorithm of `sstep_plant.c`, so that the prototype and the Xcos plant block can be compared on that stage.
- Tests T13 (prototype) and S06 (Xcos, FIXED and FLOAT backends; the other backend runs S01–S06 in a second Scilab process, `tests/run_sil_backend.sce`): must trip (stall at the hold current, load step above the pull-out torque, overload at start, blocked rotor), must not trip (enable and run → hold current steps, T05, T07, T09, T08 parameter uncertainty, T11 detent sweep and mid-band), on the averaged stage with the leg voltage error and on the switching stage; the Xcos run equals the prototype including the time of the fault. Scenarios in `tests/sim_stall_cases.sci`. Results in the README (Stall detection). Must-trip scenarios of a heavy axis and must-not-trip scenarios with heavy load and at the voltage limit, for the changes of the detector in Stepper-C (window on `e_d`, hold after a reversed back-EMF, stall flag with the overcurrent trip), ported to the prototype. The must-trip verdict asks for `SSTEP_FAULT_STALL` in the first fault, also with `SSTEP_FAULT_OVERCURRENT`.
- `sil/c/sstep_plant.[ch]`: the plant model (averaged and switching power stage, motor, ADC) in portable C, double or single precision (`SSTEP_PLANT_FLOAT`), the switching stage optional (`SSTEP_PLANT_SWITCHING`); outputs off as zero voltage or open windings. Used by the Xcos plant block and by builds for other targets.
- Averaged stage with the ngspice leg voltage error table (`brd.leg_error`, `stage_esc1_spice().leg_error`); test S05 against the switching stage.

- `tools/export_params.py`: `params/motor_*.json` from `params/motor_*.sci` (`--check` for CI); `params/motor_17hs19_2004s1.json`.

- Averaged two-phase hybrid stepper motor model (`model/`): state equations `[iA; iB; w; theta]`, back-EMF, torque with optional detent term, viscous friction, locked-rotor option; `stepper_simulate()` with lsoda.

- Parameters of the StepperOnline 17HS19-2004S1 (`params/motor_17hs19_2004s1.sci`).
- Sources `src_dc` and `src_open_loop` (phase voltages and load torque).
- Tests T00 (conventions), T01 (steady state V/R), T02 (time constant L/R) and X01 (cross-check with the Python twin), runner `tests/run_all.sce`.
- Python twin `tools/ref_model.py` and reference data `tests/reference/`.
- Demo `demos/demo_open_loop.sce`.

- Discrete current loop (prototype of the Stepper-C fast loop): `current_pi_gains`, `current_pi`, `current_pi_update` (back-calculation anti-windup), one-period computation delay, back-EMF feed-forward, `sim_closed_loop` with an RK4 plant.
- Averaged power stage `bridge_avg` for two H-bridges and for three legs with a common leg (B-G431B-ESC1), voltage vector limited with its direction kept.
- Parameters `board_bg431b_esc1`, `ctrl_default`; references `ref_current_step`, `ref_const_speed`.
- Tests B01 (power stage), T03 (microstepping), T04 (current step), X02 (cross-check of the closed loop with `tools/ref_ctrl.py`).
- Motion generator (prototype of the Stepper-C slow loop): `motion_init`, `motion_slow` (trapezoidal profile with one-period look-ahead braking, MOVE / RUN / STOP commands, hold current), `ref_motion` (angle interpolated every fast period); parameters `motion_default`.
- Tests T05 (move with speed ramp), T06 (load step, loss of synchronism), T07 (reversal), T09 (voltage limit of both topologies), X03 (cross-check with `tools/ref_motion.py`).

Changed

- `stepper_motor_rhs` uses the new `stepper_motor_f`, the single place of the state equations.
- Reference functions keep a state: `[r, st] = ref(t, p, st)`; `sim_closed_loop` returns the final state and records the reference angle `phi` (column 12).
- Motion generator replaced by the bit-exact twin of the Stepper-C integer algorithm (`motion_step`, `motion_slow`, `ref_motion`, helpers `sim_int.sci`): exact discrete braking, integer angle interpolation, hold current after periods without motion. `tools/ref_motion.py` matches `src/sstep_motion.c` over 3 052 885 periods.
- T05 and T07 check the limits in the integer units of the library.
- Software in the loop in Xcos: `start_sil.sce` compiles Stepper-C (`sil_build`, FIXED or FLOAT) and opens `xcos/stepper_sil.zcos` (generated by `sil_diagram`). Blocks `STEPPER_CTRL` (the C library), `STEPPER_PLANT` (averaged power stage, motor, ADC; one period computation delay), `STEPPER_MONITOR`; computational functions in `sil/c/sstep_xcos.c`; scenario in the diagram context; results in the variable `sil`.
- Board parameters for the SIL model (trip levels, PWM period, ADC scaling); `sil_ctrl_rpar`, `sil_plant_rpar`, `sil_check_rpar` (configuration checked by the library itself).
- Tests S01–S03: the Xcos diagram against the prototype.
- Switching power stage (`model/bridge_switching.sci`, `stepper_period_sw`): centre-aligned PWM legs, dead time inserted at every turn-on, body diode conduction during the dead time, Rds(on); the motor is integrated between the switching instants. Selected with `brd.stage = "switching"` in `sim_closed_loop` (third output: current extremes and mean phase voltages per period) and in the Xcos plant block.
- Board parameters `stage`, `rds_on`, `vd`, `sw_nstep`.
- Python twin `tools/ref_switching.py`; reference `tests/reference/x04_*.csv`.
- Tests T10 (switching stage: ideal switches equal the averaged model, analytic ripple, dead time voltage error, 4 rev/s with dead time), X04 (Scilab vs Python), S04 (Xcos plant with the switching stage vs prototype). Demo `demos/demo_switching.sce`.
- `ref_speed_ramp` (constant current, linear speed ramp).
- `params/stage_esc1_spice.sci` / `.json`: switching stage values generated by the ngspice model (`sfs-stepper-power-spice`); `board_bg431b_esc1` and `tools/ref_switching.py` use them (effective dead time 133 ns, leg resistance 3.4 mΩ, body diode 0.73 V). New field `brd.dead_timer_ns`.

- Dry friction `Tc tanh(w / w_c)` in the mechanical equation (`stepper_motor_f`, Python twin, Xcos plant block); motor fields `Tc`, `w_c` (default 0, 0.05 rad/s).

- `sim_closed_loop`: optional `c.mot`, the motor the controller is tuned for (gains, feed-forward), different from the plant.
- `ref_angle_step`; `ref_speed_ramp` takes an optional load torque `p.TL`.
- Tests T08 (parameter uncertainty), T11 (natural frequency, detent resonance, mid-band instability), T12 (dry friction); Python twin `tools/ref_ext.py`.

Changed

- Xcos plant block: 13 discrete states, output 2 adds the largest |iA|, |iB| of the last period; monitor output 5 (variable `sil`) has 12 columns (+ peaks).
- `sil_plant_rpar` adds the dry friction (P_TC, P_WC) and the power stage model parameters (P_STAGE..P_SW_NSTEP).
- Xcos plant block: thin wrapper of `sstep_plant.c`; the electrical angle is kept within one turn plus a turn count. `sil_plant_rpar` adds `P_NERR` and the leg error table after the load steps.
- `params/stage_esc1_spice.*` (generated by sfs-stepper-power-spice) carry the leg error table, its operating point and `"format": 1`.
- Controller block: the stall configuration in the parameter vector (`C_STALL_MS` .. `C_STALL_SYNC`, `C_NPAR` = 34), `sil_ctrl_rpar` takes it from `ctl` and the controller motor from `ctl.mot` (as `c.mot` in the prototype). Controller monitor output 12 values (+ back-EMF d/q, rotor velocity, load angle, detector level); monitor output 5 (variable `sil`) has 19 columns (+ state, fault flags, the five estimates).
- `test_S01_sil.sce` builds the backend of `sil_test_backend` (default FIXED).

Fixed

- `sil_build`: a rebuild compiles every file. The makefile does not follow the headers, so a changed structure with an old object file overwrote memory (S01 to S06 gave nonsense).
- `tests/run_all.sce` exits with status 1 on a failure also under `WScilex-cli` (mode `NW`).
- SIL with Stepper-C 1.1.0: `make_config()` left the new `stall_*` fields uninitialised, `sstep_init()` returned `SSTEP_ERR_PARAM` and S01–S05 failed; the configuration is now zeroed first.
- SIL build against Stepper-C 1.0.0: the monitor reads the setpoint through `sstep_internal.h` (the instance is opaque), MSVC compiles with `/std:c11` (`_Static_assert`), gcc with `-std=gnu11`.
- `sil_ctrl_rpar` / `sil_plant_rpar` with no command or load: Scilab's `zeros(0, n)` is `[]`, the column check failed.

APPENDIX B

License

Stepper-Lab is free and open-source software, licensed under the **Apache License, Version 2.0 (apache.org/licenses/LICENSE-2.0)**.

Item	Value
Copyright	© 2026 Anton Vynohradov
License	Apache License 2.0
SPDX identifier	`Apache-2.0`
Applies to	The model, the tools, the tests and this documentation

The SystemFromScratch® name and logo are registered trademarks; the license grants no rights to use them. They may not be used to endorse or promote products derived from Stepper-Lab without prior written permission.

B.1 Full license text

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      "control" means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      "Source" form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      "Object" form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      "Work" shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
      copyright notice that is included in or attached to the work
      (an example is provided in the Appendix below).

      "Derivative Works" shall mean any work, whether in Source or Object
```

```
      form, that is based on (or derived from) the Work and for which the
      editorial revisions, annotations, elaborations, or other modifications
      represent, as a whole, an original work of authorship. For the purposes
      of this License, Derivative Works shall not include works that remain
      separable from, or merely link (or bind by name) to the interfaces of,
      the Work and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, including
      the original version of the Work and any modifications or additions
      to that Work or Derivative Works thereof, that is intentionally
      submitted to Licensor for inclusion in the Work by the copyright owner
      or by an individual or Legal Entity authorized to submit on behalf of
      the copyright owner. For the purposes of this definition, "submitted"
      means any form of electronic, verbal, or written communication sent
      to the Licensor or its representatives, including but not limited to
      communication on electronic mailing lists, source code control systems,
      and issue tracking systems that are managed by, or on behalf of, the
      Licensor for the purpose of discussing and improving the Work, but
      excluding communication that is conspicuously marked or otherwise
      designated in writing by the copyright owner as "Not a Contribution."

      "Contributor" shall mean Licensor and any individual or Legal Entity
      on behalf of whom a Contribution has been received by Licensor and
      subsequently incorporated within the Work.

   2. Grant of Copyright License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      copyright license to reproduce, prepare Derivative Works of,
      publicly display, publicly perform, sublicense, and distribute the
      Work and such Derivative Works in Source or Object form.

   3. Grant of Patent License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      (except as stated in this section) patent license to make, have made,
      use, offer to sell, sell, import, and otherwise transfer the Work,
      where such license applies only to those patent claims licensable
      by such Contributor that are necessarily infringed by their
      Contribution(s) alone or by combination of their Contribution(s)
      with the Work to which such Contribution(s) was submitted. If You
      institute patent litigation against any entity (including a
      cross-claim or counterclaim in a lawsuit) alleging that the Work
      or a Contribution incorporated within the Work constitutes direct
      or contributory patent infringement, then any patent licenses
      granted to You under this License for that Work shall terminate
      as of the date such litigation is filed.

   4. Redistribution. You may reproduce and distribute copies of the
      Work or Derivative Works thereof in any medium, with or without
      modifications, and in Source or Object form, provided that You
      meet the following conditions:

      (a) You must give any other recipients of the Work or
          Derivative Works a copy of this License; and

      (b) You must cause any modified files to carry prominent notices
          stating that You changed the files; and

      (c) You must retain, in the Source form of any Derivative Works
          that You distribute, all copyright, patent, trademark, and
          attribution notices from the Source form of the Work,
          excluding those notices that do not pertain to any part of
          the Derivative Works; and

      (d) If the Work includes a "NOTICE" text file as part of its
          distribution, then any Derivative Works that You distribute must
          include a readable copy of the attribution notices contained
          within such NOTICE file, excluding those notices that do not
          pertain to any part of the Derivative Works, in at least one
          of the following places: within a NOTICE text file distributed
          as part of the Derivative Works; within the Source form or
          documentation, if provided along with the Derivative Works; or,
          within a display generated by the Derivative Works, if and
          wherever such third-party notices normally appear. The contents
          of the NOTICE file are for informational purposes only and
          do not modify the License. You may add Your own attribution
```

```
            notices within Derivative Works that You distribute, alongside
            or as an addendum to the NOTICE text from the Work, provided
            that such additional attribution notices cannot be construed
            as modifying the License.

         You may add Your own copyright statement to Your modifications and
         may provide additional or different license terms and conditions
         for use, reproduction, or distribution of Your modifications, or
         for any such Derivative Works as a whole, provided Your use,
         reproduction, and distribution of the Work otherwise complies with
         the conditions stated in this License.

      5. Submission of Contributions. Unless You explicitly state otherwise,
         any Contribution intentionally submitted for inclusion in the Work
         by You to the Licensor shall be under the terms and conditions of
         this License, without any additional terms or conditions.
         Notwithstanding the above, nothing herein shall supersede or modify
         the terms of any separate license agreement you may have executed
         with Licensor regarding such Contributions.

      6. Trademarks. This License does not grant permission to use the trade
         names, trademarks, service marks, or product names of the Licensor,
         except as required for reasonable and customary use in describing the
         origin of the Work and reproducing the content of the NOTICE file.

      7. Disclaimer of Warranty. Unless required by applicable law or
         agreed to in writing, Licensor provides the Work (and each
         Contributor provides its Contributions) on an "AS IS" BASIS,
         WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
         implied, including, without limitation, any warranties or conditions
         of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
         PARTICULAR PURPOSE. You are solely responsible for determining the
         appropriateness of using or redistributing the Work and assume any
         risks associated with Your exercise of permissions under this License.

      8. Limitation of Liability. In no event and under no legal theory,
         whether in tort (including negligence), contract, or otherwise,
         unless required by applicable law (such as deliberate and grossly
         negligent acts) or agreed to in writing, shall any Contributor be
         liable to You for damages, including any direct, indirect, special,
         incidental, or consequential damages of any character arising as a
         result of this License or out of the use or inability to use the
         Work (including but not limited to damages for loss of goodwill,
         work stoppage, computer failure or malfunction, or any and all
         other commercial damages or losses), even if such Contributor
         has been advised of the possibility of such damages.

      9. Accepting Warranty or Additional Liability. While redistributing
         the Work or Derivative Works thereof, You may choose to offer,
         and charge a fee for, acceptance of support, warranty, indemnity,
         or other liability obligations and/or rights consistent with this
         License. However, in accepting such obligations, You may act only
         on Your own behalf and on Your sole responsibility, not on behalf
         of any other Contributor, and only if You agree to indemnify,
         defend, and hold each Contributor harmless for any liability
         incurred by, or claims asserted against, such Contributor by reason
         of your accepting any such warranty or additional liability.

      END OF TERMS AND CONDITIONS

      APPENDIX: How to apply the Apache License to your work.

         To apply the Apache License to your work, attach the following
         boilerplate notice, with the fields enclosed by brackets "[]"
         replaced with your own identifying information. (Don't include
         the brackets!)  The text should be enclosed in the appropriate
         comment syntax for the file format. We also recommend that a
         file or class name and description of purpose be included on the
         same "printed page" as the copyright notice for easier
         identification within third-party archives.

      Copyright [yyyy] [name of copyright owner]

      Licensed under the Apache License, Version 2.0 (the "License");
      you may not use this file except in compliance with the License.
      You may obtain a copy of the License at
```

```
        http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License.
```