SCS typical workflow

In this document we provide a brief review of the software used and the steps involved in a usual workflow using SCS to develop and test a model.

# Table of Contents

[1 Table of Contents 2](#_Toc42545764)

[2 What is SCS? 3](#_Toc42545765)

[3 Software requirements 3](#_Toc42545766)

[4 Usual workflow 4](#_Toc42545767)

[4.1 Create a maven project and a git repository 4](#_Toc42545768)

[4.1.1 Creating/importing a maven project using Eclipse 4](#_Toc42545769)

[4.1.2 Typical folder structure 4](#_Toc42545770)

[4.1.3 Adding SCS as a dependency 4](#_Toc42545771)

[4.1.4 Sample project 5](#_Toc42545772)

[4.2 Create a model 5](#_Toc42545773)

[4.3 Create a parametrized experiment 5](#_Toc42545774)

[4.4 Create a configuration file 6](#_Toc42545775)

[4.5 Running the simulations 6](#_Toc42545776)

[4.5.1 Using eclipse to run a single simulation 6](#_Toc42545777)

[4.5.2 Using a terminal to run a single simulation 6](#_Toc42545778)

[4.5.3 Executing parallel simulations in the CIRCE. 7](#_Toc42545779)

[4.6 Post-process results 7](#_Toc42545780)

# What is SCS?

SCS is a light-weight framework for defining and performing robot experiments. The framework is meant to provide easy ways do develop high- and low-level robot control logic, to define parametrized experiments with the robot(s), and to execute multiple simulations in parallel to acquire data for statistical testing.

SCS was developed to have a minimal footprint and to adapt to the requirements of different projects. Thus, by default SCS only provides a bare-metal 2d-kinematics simulator, but it can easily be extended to use any physics simulator.

# Software requirements

The following list indicates software either required by SCS or usually used along to run experiments.

1. **Java[[1]](#footnote-1)** 10 or later  
   Language in which SCS is written, and thus, in which your model will have to be written in.
2. **Maven[[2]](#footnote-2)**

Maven is a tool that simplifies building and distributing java projects, although it is best described in its “getting started” website[[3]](#footnote-3), which we highly recommend reading through. Maven is optional, but it is highly recommended since (among other things) it makes it extremely easy to manage project dependencies (like SCS) and the deployment process. We highly suggest getting familiar with the basics of maven[[4]](#footnote-4), also note that most Java IDEs support maven by default, so unless you plan to use it from a terminal, you will not need to install it.

1. **Eclipse**[[5]](#footnote-5) or **Intellij**[[6]](#footnote-6)

Both Eclipse and Intellij are optional Java IDEs. We highly recommend using one of them. In SCS we use Eclipse mainly for historical reasons, but it seems the community is shifting towards Intellij due to its impressive auto complete AI. Either of these IDEs will be fine and will come with maven installed.

1. **CIRCE**[[7]](#footnote-7) **account**

Performing experiments with SCS usually involves running hundreds or thousands of simulations to collect data for statistical testings. To our knowledge USF has two main computer clusters: CIRCE and GAIVI[[8]](#footnote-8). CIRCE is a larger multi-purpose cluster, while GAIVI is smaller cluster focused on GPU usage. In the past, SCS has only been used with CIRCE so we only provide information for this cluster. To use CIRCE, you will have to request access as instructed on their website and read all its usage and policies documentation.

1. **SLURM[[9]](#footnote-9)**

In order to use CIRCE, you will need to submit your jobs using SLURM, the tool used by CIRCE to manage job scheduling. On its website, CIRCE provides a beginner’s guide to SLURM usage[[10]](#footnote-10).

1. **Python 3[[11]](#footnote-11)**Python 3 is an optional scripting language used by SCS. In a usual workflow, python 3 is used to generate the parameter tables required to run simulations with SCS and to post process/plot the data generated.
2. **Git**[[12]](#footnote-12)  
   Tool required to manage the lab’s repositories. We assume the reader is already familiar with its usage.

# Usual workflow

In this section we describe the usual workflow to develop and test a model using SCS.

## Create a maven project and a git repository

The first step in the usual workflow is to create a java project to develop the model. To do so we suggest creating a maven project as it will simplify the workflow later. If the project is to be hosted on the lab’s github, ask the current lab administrator to create a repository for you.

### Creating/importing a maven project using Eclipse

To create a maven project using Eclipse, click on “File > New > Other” and search for “Maven Project”, then follow the prompts on the screen. Use “com.github.biorobaw.scs\_models” as the group id and use the name of the project as the artifact id.

To import an existing project using eclipse click on “File > Import > Existing Maven Projects” and then follow the prompts on screen.

### Typical folder structure

The following represents a summary of the usual folder structure used in SCS projects:

* **src > main > java > com > github > biorobaw > scs\_models > YOUR\_PROJECT\_NAME** : project folder structure containing the code for the model following java and maven conventions.
  + **model** : folder containing the model code (robot control logic)
  + **tasks** : folder containing tasks to be performed in the experiment
  + **gui** : folder containing the code for the gui
* **experiments** : folder containing files to describe experiments and their results
  + **mazes** : folder containing xml files describing maze setups
  + **setups** : folder containing parametrized experiment xml setup files
  + **YOUR\_EXPERMENT\_SET\_N** : folder containing a set of experiments
    - **generatos** : folder containing scripts to generate the config files
    - **post\_processing** : folder containing code to processes/plot the experiment results
    - **logs** : folder which contains the results of the experiment
    - **run\_files.sh** : set of files for running the experiment.
    - **config\_files.csv** : set of files defining tables of parameters for parallel executions
* **scripts** : folder containing scripts used by all experiments
  + **circe\_cluster** : folder containing scripts to run in the cluster
  + **post\_processing** : folder containing scripts to process/plot experiment results
* **docs** : folder containing the documentation about the model
* **.gitignore** : file describing folders and files to be ignored by git
* **README.md** : file with the text to be displayed on the github’s web repository
* **pom.xml** : maven project’s pom file

### Adding SCS as a dependency

Once the maven project was created you will have to add SCS as a dependency to your project’s pom file. To do so, add the following lines under the “repositories” and “dependencies” tags respectively.

<!-- jitpack allows to add github repositories -->

<repository>

<id>jitpack.io</id>

<url>https://jitpack.io</url>

</repository>

<!-- add SCS with an adequate vesion number to the list of dependencies (example 4.0.0) -->

<dependency>

<groupId>com.github.biorobaw</groupId>

<artifactId>scs</artifactId>

<version>DESIRED\_SCS\_VERSION</version>  
</dependency>

Also, add the following plugin to the “plugins” tag in the pom file:

<plugin>

<artifactId>maven-assembly-plugin</artifactId>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>single</goal>

</goals>

</execution>

</executions>

<configuration>

<descriptorRefs>

<descriptorRef>jar-with-dependencies</descriptorRef>

</descriptorRefs>

</configuration>

</plugin>

Adding the plugging will allow to generate a jar containing all dependencies when executing maven’s “package” command.

### Sample project

SCS provides a sample project which can be used as a template for your own project[[13]](#footnote-13). If copy-pasting, be sure not to copy any git files other than the “.gitignore”, update the name of the project and the contents of the pom file. Finally, update any other files as required.

## Create a model

Once the project has been created and that you have added SCS as a dependency, the next step is to create a model to control a robot. To do so extend SCS’s class “Subject” which implements the high-level control logic of a robot. For an example, see section 4.1.4.

## Create a parametrized experiment

In SCS, a parametrized experiment is an xml file that defines sets of tasks to be performed by sets of robots and by the simulator itself. The file is parametrized because it uses the syntax $(VARIABLE\_NAME) to reference global variables that can be defined either in the variables section of the experiment file, by the simulator itself, or using a configuration file (see section 4.4). The latter allows to perform the experiment using different sets of parameters and compare the results.

Experiment files are usually located in folder “experiments > setups”, see section 4.1.4 for an example. The sample model describes all required and optional sections of a parametrized experiment file.

## Create a configuration file

A configuration file is a csv file with columns separated using white space. The file defines a table of parameters that will be used as global variables in the simulations. The first line in the file indicate the name of each column which coincides with the name of the global variable used. Each following line defines a set of values for the columns defined in the first line. When running a simulation, a configuration file is passed to SCS along with a row number to load the variables.

A configuration file must define at least the following columns:

* “config” a name for a fixed set of parameters
* “experiment” path to an experiment file
* “group” name of a group defined in the experiment file
* “run\_id” integer id representing one simulation using the set of parameters defined by “config”. This value is required since a set of parameters must be executed multiple times in order to collect sufficient data to perform statistical tests.

Besides from the parameters mentioned above, the file must include all variables used by the experiment file which were not defined in the “variables” section of the experiment file.

The configuration might include any other used defined variables that will be loaded as global variables at runtime.

Configuration files are usually located in folder “experiments > YOUR\_EXPERMENT\_SET\_N”. See section 4.1.4 for an example.

## Running the simulations

Once all the above steps have been completed, now you can run a simulation. The process is described in the following subsections.

### Using eclipse to run a single simulation

To run a simulation using eclipse you will need to create a run configuration. To do so, go to “Run > Run configurations …”. Then double click on “Java Application” to create a new configuration. On tab “Main” set “Main Class” to “com.github.biorobaw.scs.Main”. Then, on tab “Arguments”, set “Program Arguments” using the following syntax:

**CONFIG\_FILE ROW LOG\_FOLDER CREATE\_FOLDERS**

**CONFIG\_FILE** is the path to the experiment config file to be used.

**ROW** represents the row of parameters to be loaded from the table defined in the config file.

**LOG\_FOLDER** represent the path to store the results of the simulation. It is usually set to “experiments > YOUR\_EXPERMENT\_SET > logs > EXPERMENT\_NAME”.

**CREATE\_FOLDERS** is a boolean that indicates whether to create the log folders if they don’t exist. It is usually set to true if running a single simulation, or false if running in the cluster.

### Using a terminal to run a single simulation

To run a simulation using a terminal, from your project’s base folder execute the following two commands.

mvn package

java -cp **CLASS\_PATH MAIN\_CLASS CONFIG\_FILE ROW LOG\_FOLDER CREATE\_FOLDERS**

CLASS\_PATH=target/**PROJECT\_NAME**-**PROJECT\_VERSION**-jar-with-dependencies.jar

MAIN\_CLASS=com.github.biorobaw.scs.Main

The rest of the arguments were explained in section 4.5.1.

### Executing parallel simulations in the CIRCE.

In order to use SCS in the cluster, first you will have to configure your CIRCE account by doing a local installation of the java version you are using and of any required python packages. Then you will have to clone your project and perform maven’s command “package”.

Finally, in order to execute a full experiment, you will have to create a SLURM sbatch file. Sample files can be found in the SCS sample project described in section 4.1.4, under the folder “scripts > circe\_cluster”. With small modifications, the sample files can be used for any project.

After all simulations have finished, it is always good to check that no simulations failed. To do so, you can use the files provided in the sample project to check whether all simulations have generated a specific log file.

If a simulation failed, you can check why if failed by checking slurm output files.

## Post-process results

The final step of the workflow is post processing and plotting the data generated by the simulations and performing any statistical tests. Although this step varies with every project, for consistency, the files to process/plot results should be stored in the folders described in section 4.1.2, so that all scs projects have the same structure and workflow.

but to keep consistency in between projects,

1. Java: <https://jdk.java.net/> [↑](#footnote-ref-1)
2. Maven official website: <https://maven.apache.org/index.html> [↑](#footnote-ref-2)
3. Maven getting started website: <https://maven.apache.org/guides/getting-started/index.html> [↑](#footnote-ref-3)
4. 5 minutes maven tutorial: <https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html> [↑](#footnote-ref-4)
5. Eclipse: <https://www.eclipse.org/> [↑](#footnote-ref-5)
6. Intellij: <https://www.jetbrains.com/idea/> [↑](#footnote-ref-6)
7. CIRCE: <https://wiki.rc.usf.edu/index.php/CIRCE> [↑](#footnote-ref-7)
8. GAIVI: <https://www.csee.usf.edu/gaivi/> [↑](#footnote-ref-8)
9. Slurm: <https://slurm.schedmd.com/documentation.html> [↑](#footnote-ref-9)
10. Slurm tutorial: <https://wiki.rc.usf.edu/index.php/Guide_to_SLURM> [↑](#footnote-ref-10)
11. Python 3: <https://www.python.org/downloads/> [↑](#footnote-ref-11)
12. Git: <https://git-scm.com/> [↑](#footnote-ref-12)
13. SCS sample project [↑](#footnote-ref-13)