SoftwareX 12 (2020) 100573

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

2D-VSR-Sim: A simulation tool for the optimization of 2-D voxel-based soft robots

Eric Medvet *, Alberto Bartoli, Andrea De Lorenzo, Stefano Seriani

Department of Engineering and Architecture, University of Trieste, Italy

ARTICLE INFO

Article history:
Received 23 January 2020
Received in revised form 23 July 2020
Accepted 24 July 2020

Keywords:
Evolutionary robotics
Soft robotics
Optimization
Learning

ABSTRACT

Voxel-based soft robots (VSRs) are robots composed of many small, cubic blocks of a soft material with mechanical properties similar to those of living tissues and that can change their volume based on signals emitted by the robot controller, i.e., by its brain. Designing the body and the brain of a VSR suitable for a specific task is a complex activity that requires suitable optimization heuristics. We here present a software, 2D-VSR-Sim, for facilitating research on the optimization of VSRs body and brain. 2D-VSR-Sim, written in Java, provides consistent interfaces for all the VSRs aspects suitable for optimization and considers by design the presence of sensing, i.e., the possibility of exploiting the feedback from the environment for controlling the VSR. We present the mechanical model employed by 2D-VSR-Sim and we experimentally characterize the computational burden of the simulation. Finally, we show how 2D-VSR-Sim can be used to repeat the experiments of significant previous studies and, in perspective, to provide experimental answers to a variety of research questions.

Code metadata

Current code version	v1.0.0
Permanent link to repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2020_14
Legal Code License	GNU GPL v3
Code versioning system used	git
Software code languages, tools, and services used	Java
Compilation requirements, operating environments & dependencies	JDK 11, dyn4j
Support email for questions	emedvet@units.it

1. Motivation and significance

Soft robotics [1–3] is a field of robotics that studies robots built from materials with mechanical properties similar to those of living tissues. Such *soft robots* have several advantages over their rigid body counterparts, such as, e.g., conforming to uneven surfaces, distributing stress over a larger volume, increasing contact time [4]. They may also be able to undertake tasks which would be impossible for a traditional, i.e., a rigid body, robot, such as passing through small apertures [5], non-invasively access human tissue [6], camouflage through transparency while still being able to move [7]. This potential comes at the cost of an increased complexity involved in designing the *body* of a soft robot and the corresponding controller, i.e., the *brain* of the robot. Such a design is often executed by means of optimization techniques [8,9].

In this context, a popular framework considers a kind of soft robots composed of many small, cubic blocks that can change their volumes over the time, according to control signals emitted by the robot controller. Such cubic blocks are called *voxels* and the corresponding soft robots are called *voxel-based soft robots* (VSRs) [10]. Optimization techniques are even more crucial for VSRs than for soft robots in general: there is a huge variety regarding the shape that can be assumed by a VSR, depending on how its voxels are distributed and assembled. Furthermore, determining the complex interactions that may emerge from the concurrent "behavior" of many small blocks is hard. Indeed, many applications of optimization to VSRs have been proposed, e.g., [5, 11–17]. Automation offered by optimization fostered research by allowing researchers and practitioners to focus more on the broad

* Corresponding author.
E-mail address: emedvet@units.it (E. Medvet).

https://doi.org/10.1016/j.softx.2020.100573

goal of optimization, rather than on the finer details about how to perform it.

In this work, we present a software, called 2D-VSR-Sim, for facilitating research in the optimization of VSRs. We designed 2D-VSR-Sim focusing mainly on two key steps of optimization: what to optimize and toward which goal. As a result, 2D-VSR-Sim offers a consistent interface (a) to the different components (e.g., body, brain, specific mechanisms for control signal propagation) of a VSR which are suitable for optimization and (b) to the task the VSR is requested to perform (e.g., locomotion, grasping of moving objects). We designed 2D-VSR-Sim without any specific optimization technique in mind. That is, 2D-VSR-Sim leaves researchers great freedom on how to optimize: different techniques, e.g., evolutionary computation or reinforcement learning, can be used on VSRs by researchers of different disciplines, e.g., robotics, artificial life, learning representations.

Some software frameworks originated from needs similar to ours, namely [18] (later wrapped in Evosoro [19]) and [20]. Other frameworks could be used for modeling and simulating VSRs, e.g., [21], but operate at a much lower abstraction level and require a larger design effort to the researcher. 2D-VSR-Sim differs from those frameworks because it offers an higher level of abstraction to the description of the VSRs that favors the task of defining what to optimize. In particular, we included by design the possibility for the VSR to *sense* itself and the environment: that is, the controller under optimization can use as inputs the current velocities, accelerations, rotations, etc., of each of the voxels. 2D-VSR-Sim allows the user (i.e., a researcher) to exploit those sensing abilities out-of-the-box, thereby saving the effort for modeling and implementing them in the simulation. A recent study showed that sensing the environment may be beneficial for obtaining a broader range of behaviors [15]. Moreover, sensing might lead to a sharper arising of the *embodied cognition paradigm*, according to which the complexity of the behavior of a robotic agent depends on both its brain and its body [22].

Besides sensing, 2D-VSR-Sim differs from existing software tools also because it simulates a 2-D version of VSRs: operating in 2-D, rather than in 3-D, makes the search space in general "smaller" and hence potentially facilitates the optimization. On the other hand, optimized artifacts have no clear real counterparts. Indeed, some attempts to physically build VSRs are being made [10,23–25], at different scales and with different actuation mechanisms, but practicality is still limited. The model used by 2D-VSR-Sim internally is not tailored to any specific VSR implementation: different implementations could require significantly different models and no reference implementation has emerged yet. The modular structuring of 2D-VSR-Sim should facilitate the modeling of specific VSR properties, though, perhaps by means of specialized plugins. We plan to extend 2D-VSR-Sim to the 3-D case as future work.

Finally, 2D-VSR-Sim provides components for *visualizing* the simulated behavior of a VSR, which is very important in an exploratory research field of this kind. Moreover, this functionality has been obtained by the separation of concerns design principle, by exploiting the programming language features offered by Java, which greatly simplifies possible extensions of 2D-VSR-Sim.

2. Software description

2D-VSR-Sim is a simulator of one or more 2-D VSRs that perform a task, i.e., some activity whose degree of accomplishment can be evaluated quantitatively according to one or more indexes. The simulation is discrete in time, using a fixed time-step, and continuous in space: the position and configuration of each voxel of the VSR is updated at each time-step according to the mechanical model and to the VSR controller.

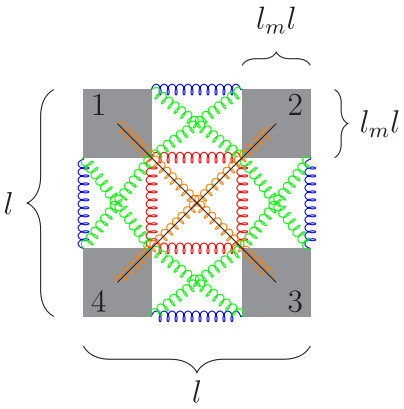

Fig. 1. The mechanical model of the voxel. The four masses are depicted in gray and numbered (for later reference) in black; the different components of the scaffolding are depicted in blue, green, red, and orange (see text); the ropes are depicted in black.

Table 1
Voxel configurable properties.

Description and symbol	Def. val.	Domain	Unit
Side length l	3	$]0, +\infty[$	m
Mass side length ratio l_m	0.3	$]0, 0.5]$	
Mass linear damping d_m^l	1	$[0, +\infty[$	
Mass angular damping d_m^ω	1	$[0, +\infty[$	
Mass m_m	1	$]0, +\infty[$	kg
Mass friction coefficient ρ_m	100	$[0, +\infty[$	
Mass restitution coefficient r_m	0.1	$]0, +\infty[$	
SDS frequency f_s	8	$[0, +\infty[$	Hz
SDS damping ratio d_s	0.3	$[0, 1]$	
Max area change ρ_A	0.2	$[0, 1]$	

2.1. Voxel model

In 2D-VSR-Sim a voxel is a soft 2-D block, i.e., a deformable square modeled with four rigid bodies (square *masses*), a number of spring–damper systems (SDSs) that constitute a *scaffolding*, and a number of *ropes*. SDSs and ropes have zero mass; ropes act as upper bounds to the distance that two bodies can have. Fig. 1 shows the mechanical model of a single voxel.

Most of the properties of the voxel model are *configurable* by the user, as explained below. The user can configure the scaffolding specifying a subset of the following groups of SDSs: (a) *side external*, one outer SDS connecting the two masses for each voxel side (blue in Fig. 1); (b) *side internal*, one inner SDS connecting the two masses for each voxel side (red in Fig. 1); (c) *side cross*, two crossing SDSs connecting the two masses for each voxel side (green in Fig. 1); (d) *central cross*, two crossing SDSs connecting the four masses (orange in Fig. 1). The presence of the ropes can be configured (enabled or disabled) by the user too. Table 1 shows the main parameters of the voxel model along with their default values and domains. Mass friction and restitution coefficients are used by the physics engine (see Section 2.6) while determining the effects of collisions of masses with other bodies (e.g., the ground).

By varying the values of the parameters, the user can impact on the properties of the material constituting the voxel. In particular, for impacting on the softness of the voxel, the user can operate on the scaffolding and/or on the SDS frequency f_s; with the former, the more the selected groups, the more rigid the voxel. After a few exploratory experiments, we set as default value $f_s = 8$ Hz and the scaffolding composed of all the groups.

The VSR can perform its task by varying the area of the composing voxels over the time, i.e., by actuating each voxel. In

the mechanical model of 2D-VSR-Sim, the actuation is obtained by varying the resting length of the SDSs. Given a control value $f \in [-1, 1]$, the resting length of all the SDSs of the voxel is instantaneously modified such that the voxel side becomes $l' = \sqrt{l^2(1 - f\rho_A)}$ where ρ_A is a parameter representing the maximum increase or decrease of the voxel area.

2.2. VSR model

A VSR is modeled as a collection of voxels organized in a 2-D grid, each voxel in the grid being rigidly connected with the voxel above, below, on the left, and on the right. The connection between two voxels is modeled as two rigid joints connecting the centers of the masses on the common side. The rigid joint does not allow rotations of the masses around the connection points, nor variation in the distance between the two connected masses: in other words, two masses connected by a rigid joint are welded. Fig. 2 shows the mechanical model of an example VSR composed of 6 voxels.

The parameters of the voxels composing a VSR can have different values (with the exception of the side length l); a VSR can hence be composed of different materials.

2.3. VSR controller

The way a VSR behaves is determined by a *controller*. Whenever it is invoked, the controller determines, for each voxel v_i of the VSR, the control value $f_i \in [-1, 1]$ to apply. The control value is applied by the physics engine (see Section 2.4) and results in a change in the area of the corresponding voxel and hence a change in the shape of the VSR. The controller can be implemented by the user and 2D-VSR-Sim provides ample freedom in this respect: realizable controllers include those where f_i depends only on the current time t and those where f_i is the result of a possibly non trivial processing of current and previous states of the VSR and the environment.

A controller has access to several *sensors* for each voxel, hence giving the VSR the ability to sense itself and the environment. For each sensor s and each voxel v_i, the controller may use zero or more of the following: (a) the current value $s(t, v_i)$; (b) the average value $\frac{1}{n}\sum_{k=0}^{n-1} s(t - k\Delta t_c, v_i)$ of the last n readings (at times $t, \ldots, t-(n-1)\Delta t_c$); (c) the nth difference $s(t, v_i) - s(t-(n-1)\Delta t_c, v_i)$ between the current value and the $n-1$th reading. Available sensors allow to sense the current area of the voxel, the velocity of its center of mass, its rotation, and the fact that it is touching another body (e.g., the ground).

We implemented two controllers in 2D-VSR-Sim, one being stateless and not exploiting any sensor, the other based on a multilayer perceptron (MLP) and resembling the one presented in [15]. The two implementations can be used without writing any code and can be instantiated by setting the values of the parameters of the corresponding controllers, that we describe below.

The stateless, non-sensing controller simply applies to each voxel v_i a control value $f_i = f_i(t)$: the controller parameters consist hence in a grid of functions $f_i : \mathbb{R}^+ \to [-1, 1]$.

The MLP-based controller computes, at each invocation, the output $\boldsymbol{y} = f_{\text{MLP}}(\boldsymbol{x}, \boldsymbol{\theta})$ of a MLP with a user-defined architecture and weights $\boldsymbol{\theta}$ on an input $\boldsymbol{x}$. The size of the input layer m, $\boldsymbol{x} \in \mathbb{R}^m$ is implicitly defined by the user-defined sensors for each voxel of the VSR; the size of the output layer n, $\boldsymbol{y} \in [-1, 1]^n$ is equal to the number of voxels of the VSR. The control value f_i of a voxel v_i is set to the ith element y_i of $\boldsymbol{y}$. Optionally, a supplementary input can be set to the current value of a user-defined function of the current time, called driving function. The MLP-based controller parameters are hence: the grid of sensors to be used, the driving function, the MLP architecture, and the MLP weights $\boldsymbol{\theta}$.

2.4. Simulation

2D-VSR-Sim exploits an existing physics engine, dyn4j,[1] for solving the mechanical model defined by a VSR subjected to the forces caused by the actuation determined by its controller and by the interaction with other bodies (typically, the ground). We refer the reader to the documentation of dyn4j for the fine details about how physics is modeled.

The physics engine can be configured in several aspects. One that is particularly relevant is the time-step Δt that is used for numerically solving the model: in order to avoid numerical instabilities, Δt has to be small enough with respect to the largest SDS frequency f_s. For the same reason, we made 2D-VSR-Sim to not invoke the controller of the VSR at every time-step, but every $c + 1$ time-steps, $c \geq 0$ being the *control step interval*. After preliminary experiments, we set the default values for Δt to $\frac{1}{60}$ s (which is the default value of the underlying physics engine) and for c to 1, i.e., the controller is invoked every $\frac{1}{30}$ s.

2.5. Task

The goal of the optimization is represented in 2D-VSR-Sim as a *task* to be solved. In general, a task is a function that processes an input and gives an output: the common case is the one where the input is a description of a VSR (or of part of it) and the output is a measure of the degree to which that VSR accomplished the task. The latter can be based on any quantity made available by 2D-VSR-Sim and the underlying physics engine, e.g., position of the center of mass of the VSR, consumed energy, and so on.

We implemented one task in 2D-VSR-Sim that represents *locomotion*, i.e., a task where the goal of the robot is to travel as far as possible. We chose this task because it is the one on which the vast majority of previous studies put their focus. The implementation of this task can be used without writing any code and can be instantiated by setting the values of the parameters describe below.

The input of this task is a description of the VSR, in terms of its body (grid of voxels described by their parameters) and brain (controller). The output is given by one or more measures obtained by simulating the VSR that moves on the ground as, e.g., the distance traveled along the x-direction. Many aspects of the locomotion task can be configured, as, e.g., the roughness of the ground. Upon completion of the simulation, one or more measures can be taken: this allows to cast the optimization of a VSR for locomotion as a single- or multi-objective optimization problem. Measures that may be taken include the average velocity and the average of the sum of the squared control values of the voxels. For example, optimizing the controller of a VSR with a given body for the maximization of the former and the minimization of the latter corresponds to searching for a controller that is at the same time good in running and energy-saving.

2.6. Software architecture

2D-VSR-Sim is meant to be used within or together with another software that performs the actual optimization.

2D-VSR-Sim is organized as a Java package containing the classes and the interfaces that represent the models and concepts described previously. The voxel is represented by the `Voxel` class and its parameters can be specified using the `Voxel.Builder` class, using the builder pattern. The VSR is represented by the `VoxelCompound` class; a description of a VSR, that can be used for building a VSR accordingly, is represented by the `VoxelCompound.Description` class. A controller

[1] http://www.dyn4j.org/.

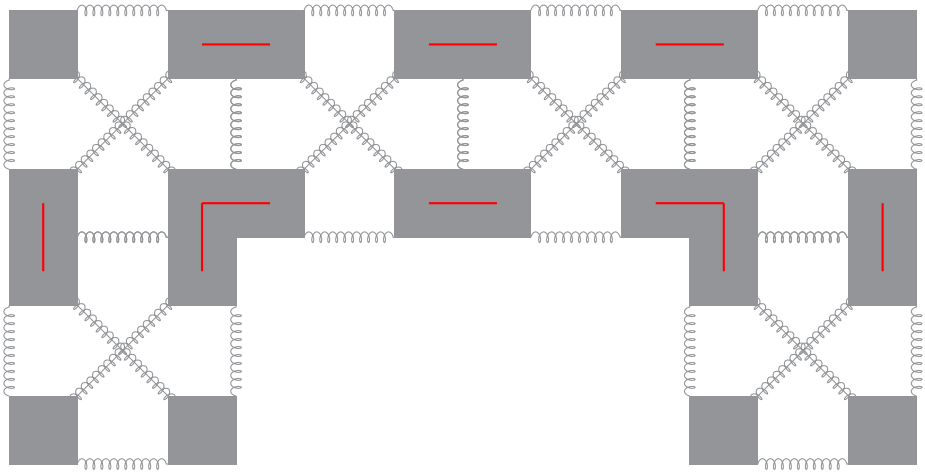

Fig. 2. The mechanical model of a VSR composed of 6 voxels. The masses, the SDSs, and the ropes are depicted in gray; the rigid joints connecting the voxels are depicted in red.

is represented by the interface `Controller`: a controller exploiting sensing can be realized by extending the abstract class `ClosedLoopController`, that takes care of collecting the sensor readings and makes them available to the inheriting class. Finally, a task is described by the interface `Task`.

Since VSRs are defined as grids of voxels, a class of particular importance in 2D-VSR-Sim is `Grid<T>`, that represents a 2-D grid of objects of type `T`: `Grid` may contain null objects, meaning that the corresponding positions are empty. For example, a `Grid<Voxel.Builder>` is used for specifying the body of a VSR, whereas a `Grid<SerializableFunction<Double, Double>>` is used for specifying the time functions, one for each voxel, that are the parameters of the stateless, non-sensing controller (represented by the class `TimeFunction` that implements the `Controller` interface).

2D-VSR-Sim provides a mechanism for keeping track of an ongoing simulation based on the observer pattern. A `SnapshotListener` interface represents the observer that is notified of progresses in the simulation, each in the form of a `Snapshot`: the latter is an immutable representation of the state of all the objects (e.g., positions of voxels, values of their sensor readings) in the simulation at a given time. We implemented two listeners using this interface. `GridOnlineViewer` renders a visualization of the simulated world within a GUI, whereas `GridFileWriter` produces a video file: both can process multiple simulations together, organized in a grid. The possibility of visualizing many simulations together can be useful, for example, for comparing different stages of an optimization.

Fig. 3 shows the graphical user interface (GUI) provided by `GridOnlineViewer`: four simulations of the locomotion task are shown with four different VSRs. On top of the GUI, a set of checkboxes allows the user to customize the visualization with immediate effect. Several measures can be visualized through the fill color of the voxel or other means; voxels SDSs and masses can be visualized too, as well as other useful information.

2.7. Impact of main parameters

We experimentally characterized the performance of 2D-VSR-Sim in terms of the impact of its main parameters. In particular, we considered two parameters – the scaffolding and the time-step Δt – and evaluated their impact on the simulation performance, i.e., how many simulation steps can be performed in the unit of time on a given computing machine. We also performed a mechanical characterization of the static and dynamic behavior of an assembly of simulated voxels: due to space constraints, we do not include here the results and refer the reader to [26].

We considered a VSR of $w \times 3$ voxels with the same properties (i.e., a sort of worm of length w) actuated by a stateless, non-sensing controller. The control value for a voxel at position x, y in the grid is given by $f_{x,y}(t) = \sin\left(-2\pi t + \pi \frac{x}{w}\right)$. The VSR moved on an uneven surface in the attempt of performing the locomotion task.

For each value of $w \in \{3, 6, \ldots, 42, 45\}$ (starting from $w = 45$), we performed 5 simulations lasting 60 s (simulated time). We executed the simulations in parallel (as `Callables` through the Java `ExecutorService` framework, one `Callable` for each core) on the Galileo partition of the CINECA HPC cluster, where each node is equipped with 2×18 cores based on 2.30 GHz Intel Xeon E5-2697 v4 (Broadwell) and with 128 GB RAM. We used OpenJDK 64-Bit Server VM (build 13 + 33) with the `-Xmx8G` option (i.e., with at most 8 GB) and repeated the procedure 10 times on 10 different HPC nodes, hence performing $5 \times 10 = 50$ simulations for each value of w. At the end of each simulation, we counted the average number of *simulated voxel steps per second (SVSPS)*, obtained as $\left(\frac{60}{\Delta t} 3w\right) \frac{1}{\tau}$, $\frac{60}{\Delta t}$ being the number of simulated steps, $3w$ the number of voxels, and τ the duration (wall time) in seconds of the simulation.

Fig. 4 shows the results in terms of SVSPS vs. the VSR length w for different configurations of the scaffolding (left plot) and for different values of the time-step Δt (right plot). It can be seen that 2D-VSR-Sim is able to perform approximately 20 000 SVSPS per core on the used machine—we remark that each simulation has been executed on a single core. Moreover, Fig. 4 shows that the number of SVSPS depends on the number of voxels: larger VSRs result in fewer SVSPS. The remarkably lower values of SVSPS for largest w values in both plots are related to how we executed the experiments: the Java VM took some time to warm up and delivered worse performance in the first executed simulations, that were the ones with $w = 45$.

Concerning the impact of the scaffolding, it can be seen that, as expected, the larger the number of simulated SDSs per voxel, the fewer SVSPS: we recall that E + C, E + I + C, All, and E + I + X correspond to 6, 10, 18, and 16 SDSs, respectively. Finally, concerning the time-step Δt, it can be seen that the lower Δt, the more SVSPS, though the differences are small. This finding can be explained by considering that the underlying physics engine is required to perform heavier computation when longer time-steps are performed. We recall, however, that the overall number of performed steps is inversely proportional to Δt: this makes, with other parameters being the same, convenient to prefer large values for Δt.

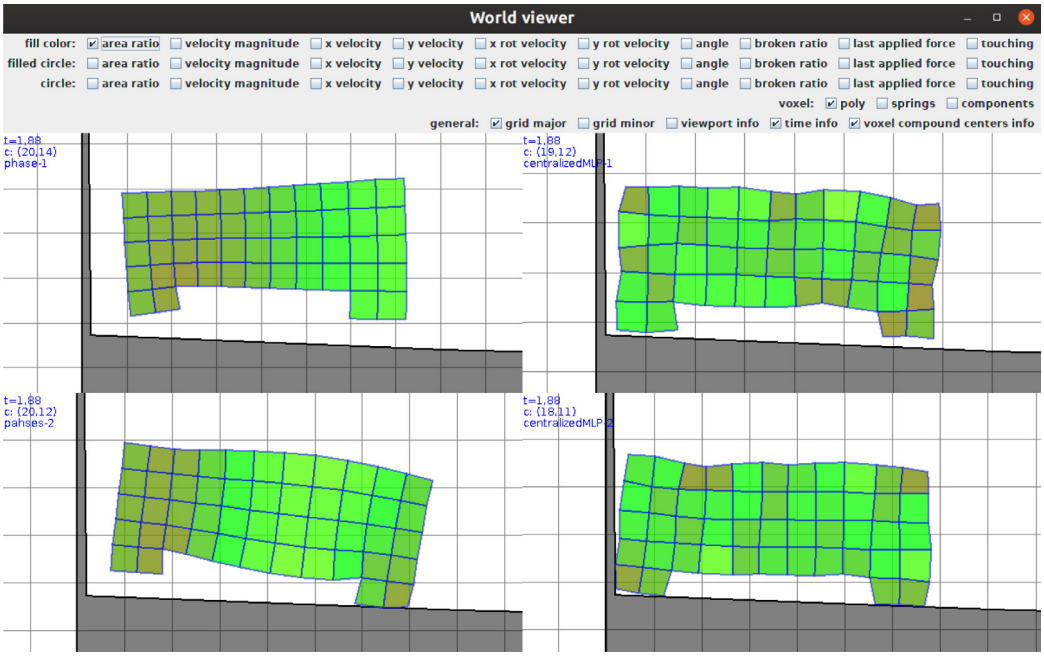

Fig. 3. The GUI provided by 2D-VSR-Sim for visualizing many simulations together (here four simulations in a grid of 2 × 2). In this (default) configuration, each voxel is filled with a color that is proportional to the variation of the voxel area: red means shrunk, green means no variation, yellow means enlarged. The ground is filled in dark gray.

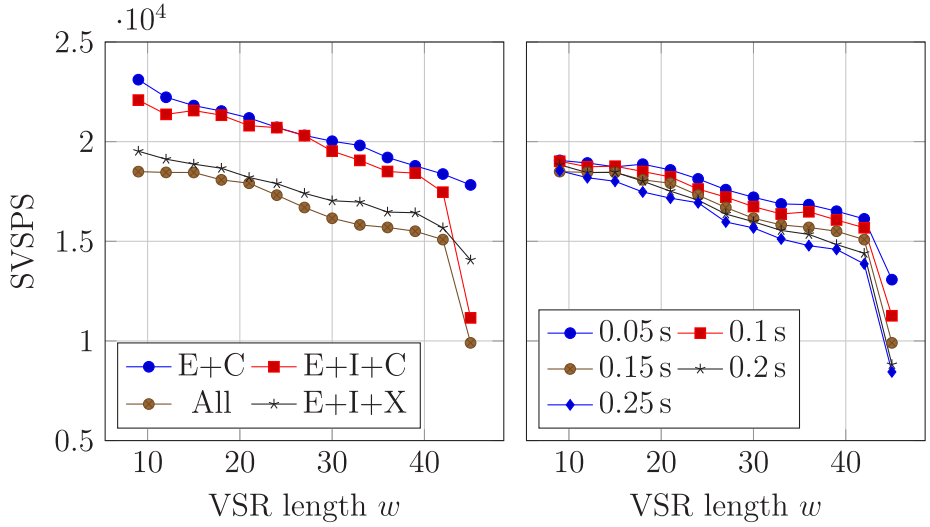

Fig. 4. Simulation performance of a worm-shaped VSR of $w \times 3$ voxels performing locomotion: number of simulated voxel steps per second (SVSPS) vs. the length w. On the left, for different scaffolding configurations; on the right, for different values of the time-step Δt.

3. Potential impact and illustrative examples

We designed 2D-VSR-Sim to facilitate research on the optimization of VSRs and we believe that, by achieving this goal, it may have a remarkable impact on many research fields (e.g., soft robotics, evolutionary computation, reinforcement learning). In order to corroborate this vision, we repeated the experiments carried out in three significant papers on this topic [10,14,15]—moreover, we highlight that 2D-VSR-Sim has already been used in [17]. Our intent was not to exactly reproduce the experimental results of the cited studies (also because they were obtained with 3-D VSRs and 2D-VSR-Sim work with 2-D VSRs), but instead to show how 2D-VSR-Sim can be used in a variety of scenarios for a variety of purposes. To this end, we performed our experiments in similar, but not identical, settings. In particular, we used here one single evolutionary algorithms (EA) for the two cases and adapted the representation of the solution, and hence the search space, to the specific case. We opted for a classic EA with random population initialization, fixed population size of 250 individuals, mutation and crossover for numerical vectors as genetic operators, tournament and truncation for reproduction and survival selection, respectively. We made the code of the experimental machinery using this EA together with 2D-VSR-Sim publicly available at https://github.com/ericmedvet/hmsrevo. We set $\Delta t = \frac{1}{30}$ s and used the same HPC machines of Section 2.7.

For brevity, we here report only the results that we obtained while repeating the experiments of [15]. In the cited paper, Talamini et al. proposed to design a VSR controller that can exploit the feedback from the environment – that is, sense it – in contrast with existing approaches (as, e.g., [10,14]) where the control values were simple functions of the current time. In order to verify

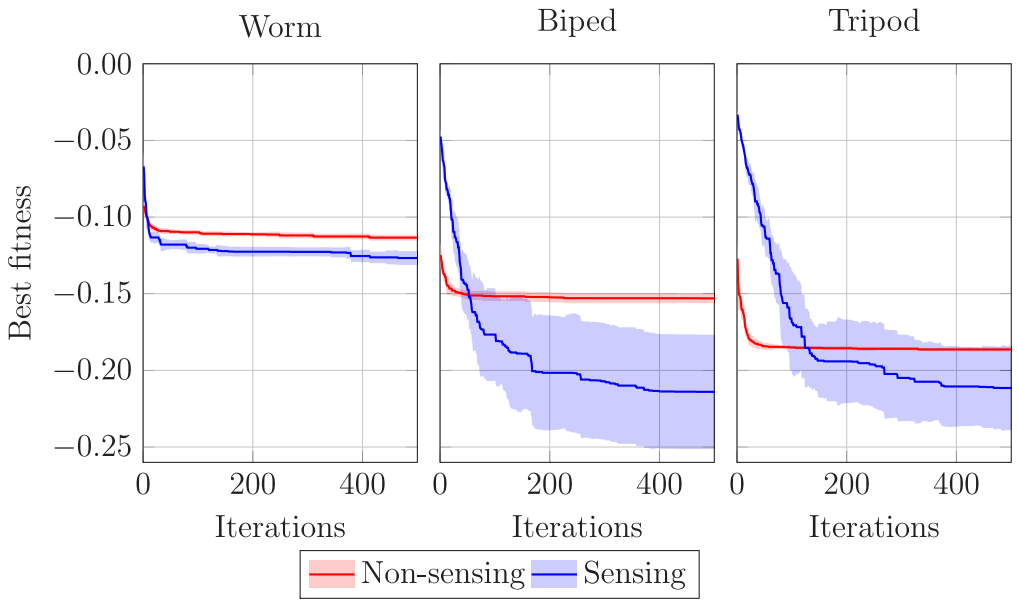

Fig. 5. Results of the sensing vs. non-sensing controller experiment inspired by [15]: fitness, i.e., average velocity, of the best individual during the evolution, one plot for each VSR shape. The shown value is the median across the 10 repetitions. The shaded area represents the interval defined by $\pm\sigma$, σ being the standard deviation across the repetitions.

if the ability of sensing actually allows to obtain more effective VSRs, the authors of [15] considered the locomotion problem, three VSR shapes, and optimized the controller parameters using an EA. They adopted the stateless, non-sensing controller of [14] and its representation as a comparison baseline.

We here considered three shapes similar to the ones used in [15]: (a) a worm of 4×1 voxels; (b) a biped with 4×1 voxels as trunk and two single-voxel legs at the extremes; and (c) a tripod with a 5×1 voxels as trunk and three single-voxel legs, two at the extremes and one in the middle. For the baseline controller, the representation is the same of the previous experiment and the search spaces are $\mathbb{R}^4$, $\mathbb{R}^6$, and $\mathbb{R}^8$, respectively for the worm, biped, and tripod. For the sensing controller, we used the MLP-based controller described in Section 2.3 with no inner layer and the 6 inputs corresponding to voxel area and velocity sensors. We set the driving function to $\sin(2\pi t)$; this resulted in the MLP being defined by $(6n + 1 + 1) \times n$ weights, n being the number of voxels. The corresponding search spaces are hence $\mathbb{R}^{104}$, $\mathbb{R}^{228}$, and $\mathbb{R}^{400}$ for the three shapes.

We used the average velocity as the fitness of the individuals and we performed 10 executions of the EA for each pair composed of a controller type (sensing or non-sensing) and a shape (worm, biped, or tripod).

Fig. 5 shows the results as the median value (across the 10 repetitions) of the fitness of the best individual during the evolution, one line for each of the two controller cases, one plot for each shape. It can be seen that the sensing controller is always more effective, after some optimization effort, in accordance with the findings of [15]. There are some differences in the amount of improvement that sensing delivers among the three shapes which could be explained, at least in part, by the different increases of the size of the search space.

We visually inspected the behaviors of some of the optimized VSRs and verified that, as found in the cited paper by systematically analyze the behaviors, sensing apparently produces more interesting behaviors. For example, we discovered that for the worm shape some sensing controllers resulted in a sort of jumping behavior.

4. Conclusions

We presented 2D-VSR-Sim, a software for simulating a particular kind of soft robots, called VSRs, composed of many simple voxels. 2D-VSR-Sim allows researchers to focus more on what to optimize rather than on how to model a VSR. In particular, 2D-VSR-Sim may boost research concerning how VSRs can exploit perception and modularity to improve their effectiveness. We highlight how the goal of this work is to provide software to make VSRs optimization easy for researchers and practitioners, and not to simulate a real robot.

We showed that 2D-VSR-Sim is highly versatile and can be easily tailored to a variety of experimental scenarios. To this end, we used 2D-VSR-Sim for repeating the experiments of some significant studies on VSRs. We think that many interesting research questions still exist that can be tackled by experimenting with VSRs and optimization. We think that, by reducing the burden for the experiments needed to validate answers to those questions, 2D-VSR-Sim may become a valuable tool for researchers in different disciplines.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We thank Giulio Fidel for his contribution to the testing and validation of 2D-VSR-Sim. The experimental evaluation of this work has been done on CINECA HPC cluster within the CINECA–University of Trieste agreement.

References

[1] Rus D, Tolley MT. Design, fabrication and control of soft robots. Nature 2015;521(7553):467. http://dx.doi.org/10.1038/nature14543.

[2] Kim S, Laschi C, Trimmer B. Soft robotics: a Bioinspired Evolution in Robotics. Trends Biotechnol 2013;31(5):287–94. http://dx.doi.org/10.1016/j.tibtech.2013.03.002.

[3] Schmitt F, Piccin O, Barbé L, Bayle B. Soft Robots manufacturing: a Review. Front Robot AI 2018;5:84. http://dx.doi.org/10.3389/frobt.2018.00084.
[4] Chen A, Yin R, Cao L, Yuan C, Ding H, Zhang W. Soft Robotics: Definition and Research Issues. In: 2017 24th international conference on mechatronics and machine vision in practice (M2VIP). IEEE; 2017, p. 366–70. http://dx.doi.org/10.1109/M2VIP.2017.8267170.
[5] Cheney N, Bongard J, Lipson H. Evolving Soft Robots in Tight Spaces. In: Proceedings of the genetic and evolutionary computation conference. ACM; 2015, p. 935–42. http://dx.doi.org/10.1145/2739480.2754662.
[6] Sitti M. Miniature soft robots—road to the clinic. Nature Rev Mater 2018;3(6):74. http://dx.doi.org/10.1038/s41578-018-0001-3.
[7] Li P, Wang Y, Gupta U, Liu J, Zhang L, Du D, et al. Transparent soft robots for effective camouflage. Adv Funct Mater 2019;29(37):1901908. http://dx.doi.org/10.1002/adfm.201901908.
[8] Zhang H, Wang MY, Chen F, Wang Y, Kumar AS, Fuh JYH. Design and development of a soft gripper with topology optimization. In: 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS). IEEE; 2017, p. 6239–44. http://dx.doi.org/10.1109/IROS.2017.8206527.
[9] Coevoet E, Escande A, Duriez C. Optimization-based inverse model of soft robots with contact handling. IEEE Robot Autom Lett 2017;2(3):1413–9. http://dx.doi.org/10.1109/LRA.2017.2669367.
[10] Hiller J, Lipson H. Automatic design and manufacture of Soft Robots. IEEE Trans Robot 2012;28(2):457–66. http://dx.doi.org/10.1109/TRO.2011.2172702.
[11] Cheney N, MacCurdy R, Clune J, Lipson H. Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding. In: Proceedings of the genetic and evolutionary computation conference. ACM; 2013, p. 167–74. http://dx.doi.org/10.1145/2463372.2463404.
[12] Cheney N, Clune J, Lipson H. Evolved Rlectrophysiological Soft Robots. In: Artificial life conference proceedings. MIT Press; 2014, p. 222–9. http://dx.doi.org/10.1162/978-0-262-32621-6-ch037.
[13] Corucci F, Cheney N, Giorgio-Serchi F, Bongard J, Laschi C. Evolving Soft Locomotion in aquatic and terrestrial environments: Effects of material properties and environmental transitions. Soft Robot 2018;5(4):475–95. http://dx.doi.org/10.1089/soro.2017.0055.
[14] Kriegman S, Cheney N, Bongard J. How Morphological Development can Guide Evolution. Sci Rep 2018;8(1):13934. http://dx.doi.org/10.1038/s41598-018-31868-7.
[15] Talamini J, Medvet E, Bartoli A, De Lorenzo A. Evolutionary synthesis of sensing controllers for Voxel-based soft robots. In: Artificial life conference proceedings. MIT Press; 2019, p. 574–81. http://dx.doi.org/10.1162/isal_a_00223.
[16] Kriegman S, Walker S, Shah D, Levin M, Kramer-Bottiglio R, Bongard J. Automated Shapeshifting for function recovery in Damaged Robots. 2019, arXiv preprint arXiv:1905.09264.
[17] Medvet E, Bartoli A, De Lorenzo A, Fidel G. Evolution of distributed neural controllers for Voxel-based Soft robots. In: Proceedings of the genetic and evolutionary computation conference. 2020, p. 112–20. http://dx.doi.org/10.1145/3377930.3390173.
[18] Hiller J, Lipson H. Dynamic simulation of Soft Multimaterial 3D-printed objects. Soft Robot 2014;1(1):88–101. http://dx.doi.org/10.1089/soro.2013.0010.
[19] Kriegman S, Cappelle C, Corucci F, Bernatskiy A, Cheney N, Bongard JC. Simulating the evolution of Soft and rigid-body robots. In: Proceedings of the genetic and evolutionary computation conference companion. ACM; 2017, p. 1117–20. http://dx.doi.org/10.1145/3067695.3082051.
[20] Austin J, Corrales-Fatou R, Wyetzner S, Lipson H. Titan: A parallel asynchronous library for multi-agent and soft-body robotics using NVIDIA CUDA. 2019, arXiv preprint arXiv:1911.10274.
[21] Hu Y, Anderson L, Li T-M, Sun Q, Carr N, Ragan-Kelley J, et al. DiffTaichi: Differentiable programming for physical simulation. 2019, arXiv preprint arXiv:1910.00935.
[22] Pfeifer R, Bongard J. How the body shapes the way we think: a new view of intelligence. MIT press; 2006.
[23] Kriegman S, Nasab AM, Shah D, Steele H, Branin G, Levin M, et al. Scalable sim-to-real transfer of soft robot designs. 2019, arXiv preprint arXiv:1911.10290.
[24] Sui X, Cai H, Bie D, Zhang Y, Zhao J, Zhu Y. Automatic generation of locomotion patterns for soft modular reconfigurable robots. Appl Sci 2020;10(1):294. http://dx.doi.org/10.3390/app10010294.
[25] Kriegman S, Blackiston D, Levin M, Bongard J. A scalable pipeline for designing reconfigurable organisms. Proc Natl Acad Sci 2020. http://dx.doi.org/10.1073/pnas.1910837117.
[26] Medvet E, Bartoli A, De Lorenzo A, Seriani S. Design, validation, and case studies of 2D-VSR-Sim, an optimization-friendly simulator of 2-D Voxel-based Soft Robots. 2020, arXiv preprint arXiv:2001.08617.