Lightweight Physics-Based Models for the Control of Fluid-Mediated Self-Assembly of Robotic Modules*

Bahar Haghighat[a,*], Hala Khodr[b] and Alcherio Martinoli[c,**]

[a]Self-Organizing Systems Research Group (SSR), John A. Paulson School Of Engineering And Applied Sciences, Harvard, Cambridge, MA-02138, USA
[b]Computer-Human Interaction Lab for Learning & Instruction (CHILI), School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne (EPFL), CH-1015, Switzerland
[c]Distributed Intelligent Systems and Algorithms Laboratory (DISAL), School of Architecture, Civil and Environmental Engineering, École Polytechnique Fédérale de Lausanne (EPFL), CH-1015, Switzerland

ARTICLE INFO

Keywords:
programmable self-assembly
model-based design
physics-based models
ruleset controllers
distributed robotic systems

ABSTRACT

Self-assembling robotic systems constitute a subclass of distributed robotic systems that undertake the fundamental task of structure formation. These systems build desired target structures by putting their robotic modules together in a distributed and stochastic fashion, i.e., through a self-assembly process. The use of self-assembly as the underpinning coordination mechanism provides powerful means for structure formation across a variety of length scales as well as media. In particular, fluidic media have been shown to be very efficient enablers for small-scale self-assembly. In this paper, we consider a distributed robotic system consisting of multiple miniature robotic modules performing self-assembly in 2D, at the water-air interface. The course of the assembly process in the system culminating in a predefined target structure is shaped by the ruleset controllers programmed on the individual robotic modules, allowing only certain formations and ruling out others throughout the process. Designing control strategies relies heavily on accurate models of the system dynamics. Faithfully modeling such systems and their inter-module interactions involves capturing the hydrodynamic forces acting on the modules using typically computationally expensive fluid dynamic modeling tools. Such computational cost restricts the usability of the resulting models, particularly for the purpose of designing optimized controllers. In this paper, we present a new modeling approach and proceed by employing the resulting model for optimizing ruleset controllers. First, we show how the hardware and firmware of the robotic platform can be faithfully modeled in a high-fidelity robotic simulator. Second, we develop a physics plugin to recreate the hydrodynamic forces acting on the modules and propose a trajectory-based method for calibrating the plugin model parameters. Finally, we employ the resulting model and obtain automatically optimized ruleset controllers for given target structures.

1. Introduction

Self-assembly is defined as the reversible and spontaneous phenomenon of an ordered spatial structure emerging from the aggregate behavior of simpler preexisting entities, through inherently local and random interactions in the system. Self-assembling robotic systems have garnered significant interest for their robust performances in forming structures of varied complexities and at different length scales as well as their minimal design of constituting modules [1], [2], [3]. Among these systems, fluid-mediated self-assembling systems are of particular interest due to their capability for efficiently moving the modules around and providing interactions among them, particularly at small scales. It has been shown that fluids are highly efficient for moving sub-millimeter-scale particles [4].

A key factor in studying self-assembling robotic systems is developing models that accurately describe the dynamics of the underlying physical system where the self-assembly process takes place. Such models help in: (1) accurately predicting the performances (assembly rate and yield) of the distributed system, and (2) evaluating and optimizing control strategies, whether distributed (e.g., ruleset controllers programmed on the modules) or centralized (e.g., modulating environmental features such as the mixing forces deriving random interactions among modules), based on model predictions [5, 6]. Yet, relatively little effort has been devoted to modeling fluid-mediated self-assembling systems of robotic modules. Models with high level of abstraction are usually non-spatial and assume well-mixed systems. In fact, the implications of these assumptions are difficult to gauge due to the lack of appropriate and well calibrated modeling tools for such robotic platforms [7]. Furthermore, state-of-the-art robotic simulators such as Webots, Gazebo, or V-REP do not support fluid dynamics natively and need therefore to be either coupled with appropriate fluid dynamics simulation tools or to be augmented with appropriate plugins in order to faithfully capture the dynamics of the overall system. While coupling these robotic simulators with fluid dynamic simulation tools allows for capturing the hydrodynamic forces accurately, such accuracy comes at the cost of having a typically very computationally heavy model. Our goal here is thus developing a physics-based model which is computationally lightweight and thus reasonably affordable for creating large datasets of system trajectory under

This work has been entirely carried out at the Distributed Intelligent Systems and Algorithms Laboratory at EPFL with partial financial support by the Swiss National Science Foundation under the grant numbers 200021_137838/1 and 200020_157191/1.

*Corresponding author
**Principal corresponding author
bahar@seas.harvard.edu (B. Haghighat); hala.khodr@epfl.ch (H. Khodr); alcherio.martinoli@epfl.ch (A. Martinoli)
ORCID(s): 0000-0003-3348-4702 (B. Haghighat)

different control strategies. We then use the resulting model for predicting and evaluating system trajectories for given ruleset controllers in order to optimize their parameters. By doing so we create numerous datasets of sample system trajectories which are faithful to reality, hence exploiting the computational efficiency of the model realized while obtaining high fidelity to the real physical system.

Our model design method utilizes the Webots robotic simulator and involves developing a physics plugin for recreating the hydrodynamic forces acting in the system. The physics plugin is then calibrated such that the simulated and the real world systems trajectories match closely. Our calibration method employs a Particle Swarm Optimization (PSO) algorithm, and consists of minimizing the difference between the Mean Squared Displacement (MSD) data extracted from real and simulated trajectories of multiple robotic modules.

Our model-based control method then utilizes our resulting calibrated model of the system to evaluate given ruleset controllers and find optimized parameters. We create the rules in the ruleset using a framework introduced in our previous work [8]. Starting from a target structure, a ruleset which guarantees formation of the desired target structure is created. The structure of the ruleset controller, i.e. the constituting rules, is fixed at this point, however, the parameters of the ruleset, i.e. the probabilities associated with the execution of each rule, can be tuned for optimized assembly performance. Our ruleset controller optimization method utilizes as well a PSO algorithm and involves creating sample system trajectories by running the calibrated model while the robotic modules execute a given ruleset.

What is unique about our approach for developing optimized controllers in comparison to former contributions in the literature is that the evaluation of the controllers required by the optimization process is based on models calibrated to the dynamics of the real self-assembling system rather than abstract models as, for instance, in [9, 10]. The use of a metaheuristic optimization method for finding optimal ruleset parameters, a process typically requiring numerous evaluations of the system assembly performance for a given ruleset controller, further highlights the value of exploiting lightweight yet accurate physics-based models.

The paper is organized as follows: Section 2 describes the experimental fluid-mediated self-assembling robotic system used to collect the trajectories; Section 3 presents our approach for building the physics-based model of the system accounting for simple hydrodynamic effects such as drag, buoyancy, and, to some extent, fluctuations due to stirring the fluidic arena; Section 4 describes our method to calibrate the parameters of the physics-based model based on the comparison between the simulated and experimental trajectories of the robotic modules using a PSO algorithm. Section 5 presents our approach for exploiting the calibrated models in order to optimize ruleset controllers for achieving given target structures leveraging again a PSO algorithm. Finally, Section 7 summarizes our contributions, discusses the future research paths, and concludes the paper.

2. Experimental System

The experimental setup consists of a circular water-filled tank equipped with peripheral pumps, an overhead camera, an overhead projector, a wireless node communicating with the robots for programming and data logging, and a workstation (see Figure 1) [11]. The Lily robotic modules are not self-locomoted, they are instead stirred by the flow field produced by the pumps. Each Lily robotic module is endowed with four Electro-Permanent Magnetic (EPM) latches, one at each side, used for connecting and communicating to neighboring Lily modules. The tank is approximately 0.6 m in diameter and 0.3 m in depth, and has seven inlets perpendicular to the wall which are endowed with a small insert piece to deviate the flow by about 15 degrees, creating a flow field with both radial and circular components. While the perpendicular flow components instigate irregular trajectories and induce collisions in the middle of the tank, they exhibit dead spots around the wall. The tangential components, however, generate a circular field, giving rise to regular closed trajectories which do not favor collisions but eliminate dead spots. To minimize any interference with the surface flow, the outlets are all placed at the bottom of the tank. Each pump's flow rate can be continuously controlled up to 9 l/min, allowing for a variety of flow fields and corresponding induced trajectories. To monitor the evolution of the system, we use an overhead camera to track a passive marker located at the top of each robot using SwisTrack [12]. The positions of the markers are logged at a rate of approximately 30 Hz.

3. Modeling Approach

In order to realistically recreate our self-assembling robotic system in simulation, we use Webots [13], an open source (since December 2018) physics-based high-fidelity robotics simulator which uses the Open Dynamics Engine (ODE) for simulating rigid body dynamics. Additionally, in order to simulate specific not natively supported physics such as complex fluid dynamics, it is possible to employ custom-designed physics plugins. Building our physics-based model within the Webots simulation framework comprised two main aspects. First, faithful recreation of the Lily robotic module's hardware and firmware features, and second, faithful recreation of the hydrodynamic forces acting on the robotic modules floating in the tank filled with water. The latest version of Webots supports a basic fluid node which allows for a simple uniform stream velocity, but is not capable of simulating the complex fluidic field in our experimental arena. This motivates the design of a specific physics plugin capable of capturing the complexity of the real fluidic field.

3.1. Recreating the Robotic Module

We recreate the Lily robotic module within the simulated world of Webots in several steps (see Figure 2). In the first step, we define the physical entity of the module. A CAD model of the external shell of the module was designed in

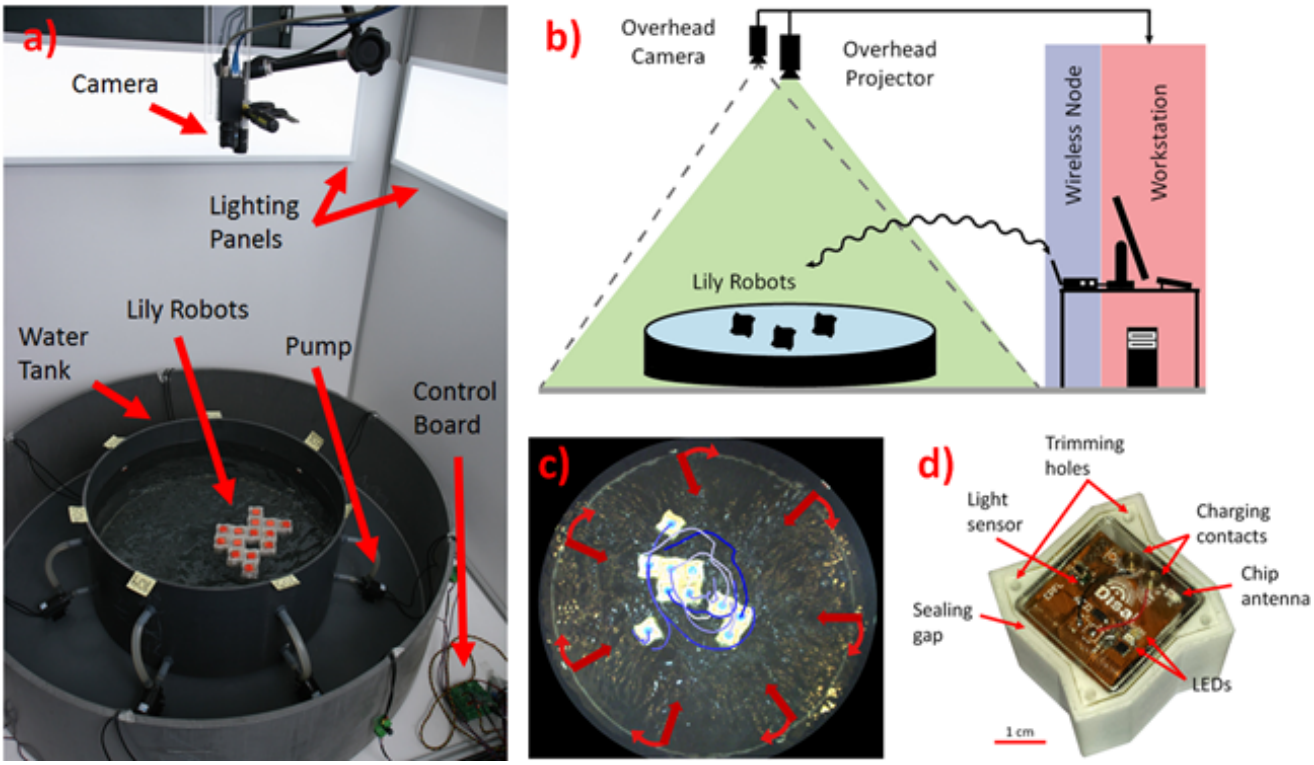
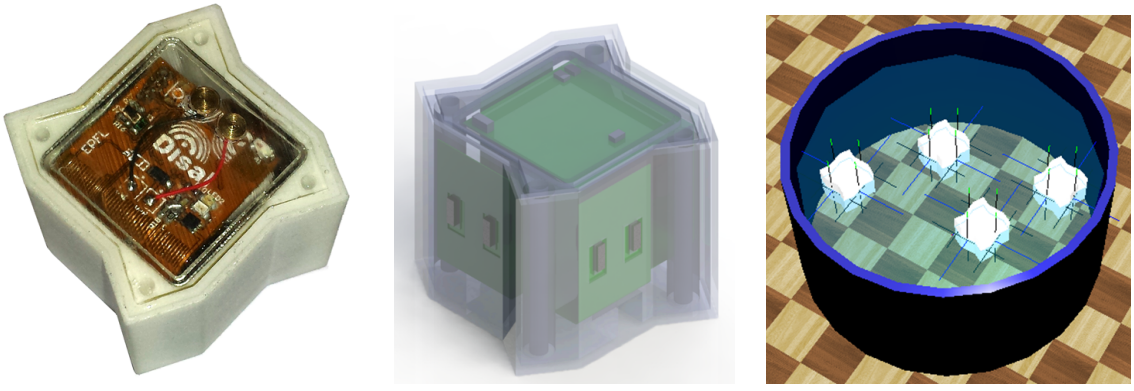


Figure 1: a) Image and (b) sketch of the experimental system. c) Visual tracking of Lily robotic modules. The red arrows indicate the pump flow. The blue lines indicate the history of the modules' trajectories. d) A Lily robotic module.



(a) (b) (c)

Figure 2: (a) A real Lily robotic module. (b) CAD design of the Lily robotic module exported from SolidWorks to Webots. (c) A sample world of simulated Lily robotic modules in Webots. The lines on the modules indicate axes on the EPM connectors.

SolidWorks and directly exported to Webots in the VRML V2.0 format. This defines the bounding object (i.e. bounding volume) associated with a Lily and is the one referred to by the ODE engine for simulating the collisions among modules. In the second step, a Lily robotic module PROTO was created. Within Webots, a PROTO allows for capturing all the features of a certain object within one PROTO container. The Lily PROTO was then augmented with the physical features of the Lily robotic modules. In particular, its bounding object as exported from SolidWorks, mass, and center of mass. A physical object in Webots has its associated linear and angular damping coefficients which are used to slow down an object. The rotational and linear speed of each object is reduced by the specified percentage (between 0.0 and 1.0) every second allowing for coping with simulation instability, all initially set to a default value of 0.5 each.

The Lily PROTO was then augmented with several *functionality nodes*, that is four connector nodes located on the sides to replicate the EPM latching mechanism, four emitter as well as four receiver nodes located on the sides with a range of 0.5 mm replicating the EPM inductive channel function, one light sensor node on the top, and an emitter node

as well as a receiver node located on the top with an infinite range to replicate the radio channel communication with the base station. The next step was then importing the Lily robotic modules' embedded controller software into the Webots simulated world. The low-level functionalities such as EPM communication through sending current pulses needed to be abstracted away and replaced by similar functions from Webots. However, the adapted controller maintains the same structure as the original one programmed on the real Lilies. The specific rulesets employed on the simulated and real modules are identical.

3.2. Recreating the Flow Field

In order to reproduce the complex flow field and the hydrodynamic forces acting on the Lilies in the real world, we use an approach inspired by the one in [7]. Our approach distinguishes from the one of [7] in two ways: (i) we capture trajectories of multiple floating objects rather than a single one, with the aim of capturing the effects of interactions of the floating objects which disturb the flow field, (ii) rather than brute force search, we employ a PSO algorithm to optimize the model parameters.

A spherical object has an isotropic drag coefficient, i.e. a constant value in all directions, while submerged in a fluid flow. As a result, a spherical object exhibits the same drag coefficient regardless of the angle of attack of the fluid field. When subject to a flow field, the drag force acts on any object floating in the field and is what makes the object move around with the flowing fluid. The drag force is a function of the velocity of the fluid field as well as the drag coefficient of the floating object (see Equation 2). Here we are interested in capturing the underlying flow field in our system. However, what we can easily measure by directly observing the system is the velocity field of objects floating and moving with the flow. Our approach involves inferring the flow field through observing the motion and visual tracking of spherical floating objects. The isotropic drag coefficient of the spherical objects simplifies computing the flow field based on their observed velocity field (see Equation 2). When we eventually consider having Lily robotic modules floating in the tank, their drag coefficient is considered as a free optimization parameter estimated through the automated optimization process (see Equation 5). We record the trajectory of floating spherical blocks (diameter of 3 cm), roughly the same size of a Lily robotic module, for three experiments with random starting positions and duration of 10 minutes each. For this, we use 24 ping pong balls whose weights are tuned such that the submersion level is similar to that of a Lily robotic module (25 mm below water level). The captured velocity fields acquired from different experiments are then augmented and discretized on a regular grid of 50 cells on each side, for our water tank of 60 cm in diameter. For each cell of the grid, the average and standard deviation of the observed velocity vectors are computed and assigned to that cell (see also Eq. 4). The fluid velocity field can then be computed considering the drag force. The value of the Reynolds number R_e determines the flow regime and the form of the drag force. The Reynolds number is a dimensionless value that measures the ratio of inertial forces to viscous forces and describes the degree of laminar or turbulent flow. The Reynolds number is calculated as below for the parameters of our system:

$$Re = \frac{\rho V L}{\mu} \simeq 6700 \tag{1}$$

This value of Reynolds number indicates a quadratic drag force as below:

$$|\vec{F}_{drag}| = \frac{1}{2}\rho A C |\vec{v}_{block} - \vec{v}_{flow}|^2 \tag{2}$$

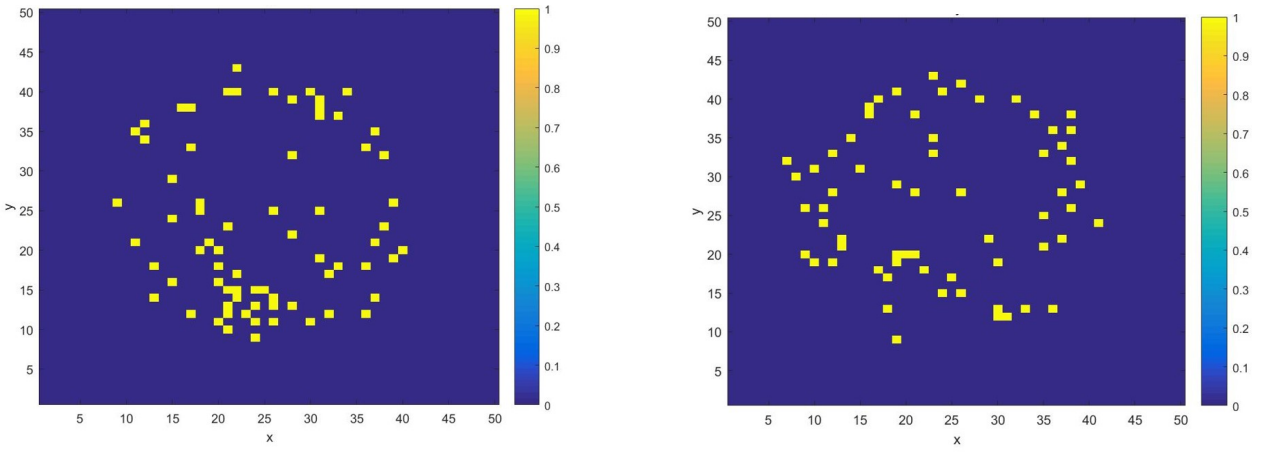
where $\rho = 10^3\ kg/m^3$ is the density of water, $V \simeq 20\ cm/s$ the experimentally-measured mean velocity of a ball, $L = 3$ cm the characteristic dimension, and $\mu = 8.90\ .\ 10^{-4}$ Pa.s the dynamic viscosity of water. The submerged area of a ball in the experimental system is approximately $A = 7\ cm^2$ (note that in simulation this value is measured by the physics plugin, we assume $A = 7\ cm^2$ for the computation of the flow field) and the drag coefficient constant in all directions $C = 0.47$. The velocity and acceleration of the ping pong balls are computed using the captured trajectory data. Considering the mass of a ball m, the flow velocity is then computed from Eq. 2 as below, considering $\vec{F}_{drag} = m\vec{a}_{drag} = m\vec{a}$, with $\vec{a}$ having components a_x and a_y in the 2D plane along the X- and Y-axis, respectively:

$$\vec{v}_{flow} = \vec{v}_{block} + \frac{m\vec{a}}{\sqrt{\frac{1}{2}\rho A C m \sqrt{a_x^2 + a_y^2}}} \tag{3}$$

A customized physics plugin is then designed for Webots so that an appropriate drag force is applied to a simulated Lily module based on the velocity of the module and the flow velocity at its location at each time instant. In order to account for rotational effects, the drag force is integrated over each face of the module. Each face is divided into $N = 10$ sections, and the drag force is computed for each section using Eq. 5.

For each cell j in the grid of a total of 2500 cells, we record the average μ_j and the standard deviation σ_j of the computed flow velocity vectors. We also test the normality of the distribution in each cell using the KS test. Results, shown in Figure 3, demonstrate that for the majority of the grid cells, the KS test failed to reject the hypothesis that the samples do not belong to a normal distribution with a confidence level of 95%. We can thus assume that the velocity at the location of each grid cell can be drawn from a normal distribution. Therefore, when a block falls in a given cell j, the physics plugin applies the corresponding flow velocity as below, where K_v is a free model parameter to be optimized.

$$v_{flow,j} = K_v \mathcal{N}(\mu_j, \sigma_j) \tag{4}$$



(a) (b)

Figure 3: Visualization of the KS test results on the captured flow velocity field along (a) X-axis, and (b) Y-axis. The points of higher temperature (yellow color) indicate the grid cells for which the hypothesis that the data points within the corresponding cell have a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ is rejected at a significance level of 5%.

We consider the drag force acting on the Lily robotic modules as below, where K_F is a free optimization parameter, estimating an appropriate value for the Lily modules drag coefficient. The area A on which the force acts is measured by the physics plugin.

$$|\vec{F}_{drag}| = K_F \frac{1}{2} \rho A |\vec{v}_{block} - \vec{v}_{flow}|^2 \tag{5}$$

The physics plugin also adds a stochastic force $F_s \sim \mathcal{N}(0, \sigma^2_{stoch})$ to the center of mass of each block in order to take into account the stochasticity and non-modeled effects as in [7]. The standard deviation σ_{stoch} defines our third optimization parameter. Moreover, we have two additional free parameters in Webots which are the linear and angular damping of an object: D_{linear}, $D_{angular}$. In summary, we will have a total of five free optimization parameters to be calibrated: the constants K_v, K_F, σ_{stoch}, D_{linear}, and $D_{angular}$.

4. Calibration Approach

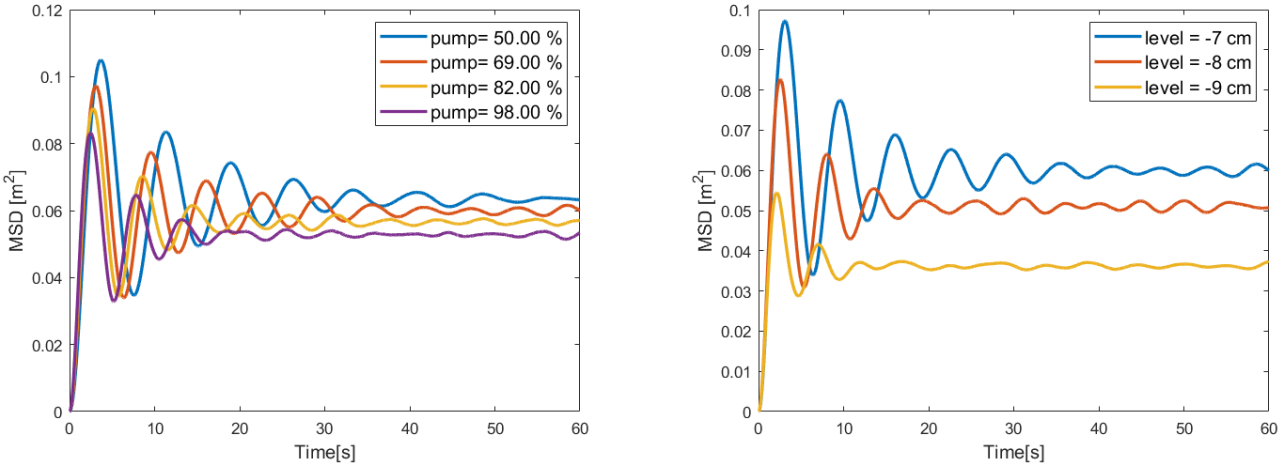
By definition, model calibration is an optimization process of adjustment of the model parameters to obtain a model representation of the processes of interest that satisfies pre-agreed criteria, typically expressed in the form of faithfulness metrics. We use the trajectories of the blocks floating on the simulated and real flow fields and refer to the MSD extracted from each data set for comparison. We then define our faithfulness metric to be optimized during the process as the error between the real and simulated MSD functions. We run a PSO algorithm in order to optimize the free model parameters, namely, K_v, K_F, σ_{stoch}, D_{linear}, and $D_{angular}$.

4.1. Optimization Algorithm

PSO is a population-based stochastic metaheuristic optimization technique originally introduced by Kennedy and Eberhart [14]. PSO is inspired by the social behavior of bird flocking or fish schooling, and represents a pool of candidate solutions as a swarm of particles moving in a multi-dimensional space and evaluated through a series of iterations. The pool of candidate solutions is then updated at the end of each evaluation iteration. The size of the swarm defines the number of candidates in the pool evaluated at each iteration. The information carried by each particle is a function of the dimension of the search space which in turn is defined by the number of parameters to be optimized through the metaheuristic search. The algorithm is initialized with a population of random candidate solutions and searches for optima by evaluating and updating the pool of solutions over iterations. While PSO shares many similarities with evolutionary optimization techniques such as the Genetic Algorithms (GA), the way candidate solutions are modified through iterations is totally different [15].

Pugh et al. showed that PSO could outperform Genetic Algorithms on benchmark functions and for certain scenarios of limited-time learning under the presence of noise [16, 17]. This motivates the choice of PSO for our particular optimization problem where stochasticity is an inherent feature of the underlying system. Multiple computationally efficient recipes for increasing the robustness of the PSO algorithm to noisy evaluations have been proposed in the literature [18, 19]. In this work, we have opted for a simple solution involving a re-evaluation and aggregation of the personal best performance of each particle as suggested in [17].

In PSO, the movement of particle i in dimension j depends on three components: the velocity at the previous step weighted by an inertia coefficient w, a randomized attraction to the particle's own personally best visited location over the previous iterations $x^*_{i,j}$ weighted by w_p, and a randomized attraction to the particle's neighborhood's best visited location over the previous iterations $x^*_{i',j}$ weighted



(a) (b)

Figure 4: (a) Effect of changing pump power on MSD. All experiments were done at the same water level, i.e. -7 cm. (b) Effect of water level on MSD. The pump power is fixed at 69%.

by w_n (Eq. 6a). r_1 and r_2 are random numbers drawn from a uniform distribution between 0 and 1. The particle position is then updated at each step using the updated velocity (Eq. 6b).

$$v_{i,j} := w \cdot v_{i,j} + w_p \cdot r_1 \cdot (x^*_{i,j} - x_{i,j}) + w_n \cdot r_2 \cdot (x^*_{i',j} - x_{i,j}) \quad (6a)$$

$$x_{i,j} := x_{i,j} + v_{i,j} \quad (6b)$$

The equation above is valid for both the canonical version and the noise-resistant one. Our approach to cope with stochasticity in our system is to evaluate each particle at each iteration multiple times. The total execution time for a PSO optimization procedure depends on four factors: population size (N_p), individual candidate evaluation time (t_e), number of iterations of the algorithm (N_i), and number of re-evaluations of each candidate solution's personal best within the same iteration (N_{re}). We evaluate all the particles in the swarm in parallel at each iteration, as a result the total time for the optimization procedure in our case is as below:

$$t_{total} = t_e \cdot N_i \cdot N_{re} \quad (7)$$

The individual candidate evaluation time t_e depends directly on the complexity of the model utilized in simulation and its computational efficiency.

4.2. Optimization Metric

Diffusion drives mixing in our system. In statistical mechanics, the MSD is a measure of the deviation of the position of a particle with respect to a reference position over time. It is the most common measure of the spatial extent of random motion, and can be thought of as measuring the portion of the space "explored" by the random walker. In the realm of biophysics and environmental engineering, the MSD is measured over time to determine if a particle is spreading solely due to diffusion, or if an advective force is also contributing. For instance, MSD analysis is a technique commonly used in colloidal studies to determine the dynamics of displacement of particles over time. The MSD is expressed as below.

$$< \Delta r^2(t) > = \sum_{k=0}^{n} < [R_k(t) - R_k(t_0)]^2 > \quad (8)$$

Where R is the position vector, and n is the total number of particles. The MSD is usually used to calculate the diffusion coefficient of a given system [20]. In order to verify that the MSD captures the change in the system dynamics, experiments of 10 minute length per water level and pump power configuration were conducted using 24 ping-pong balls. As indicated before, we use ping-pong balls for simplicity as they are symmetric and have an isotropic drag coefficient. We used three water levels in the tank, measured from the upper border as -7, -8, and -9 cm, respectively.

Figure 4 illustrates the MSD curves for different pump power and water level settings. Each pump has a maximum flow of 9 *l*/min at 100% power. First, we can notice that the MSD has a transient oscillating pattern followed by a convergence on a plateau. The oscillations are related to a situation where the blocks are affected by the stirring flow generated by the pumps and the frequency of the oscillations is related to the speed of circulation. On the other hand, the plateau indicates a maximum effective displacement explored by the blocks. As we can see in Figure 4(a), the higher the pump power, the higher the frequency of the oscillations and the lower is the plateau. This can be explained considering that when the pumps power is increased, the force pushing the blocks towards the center is higher and therefore the effective radius of the area explored by the blocks is smaller. Furthermore, by keeping the same pump power, but reducing

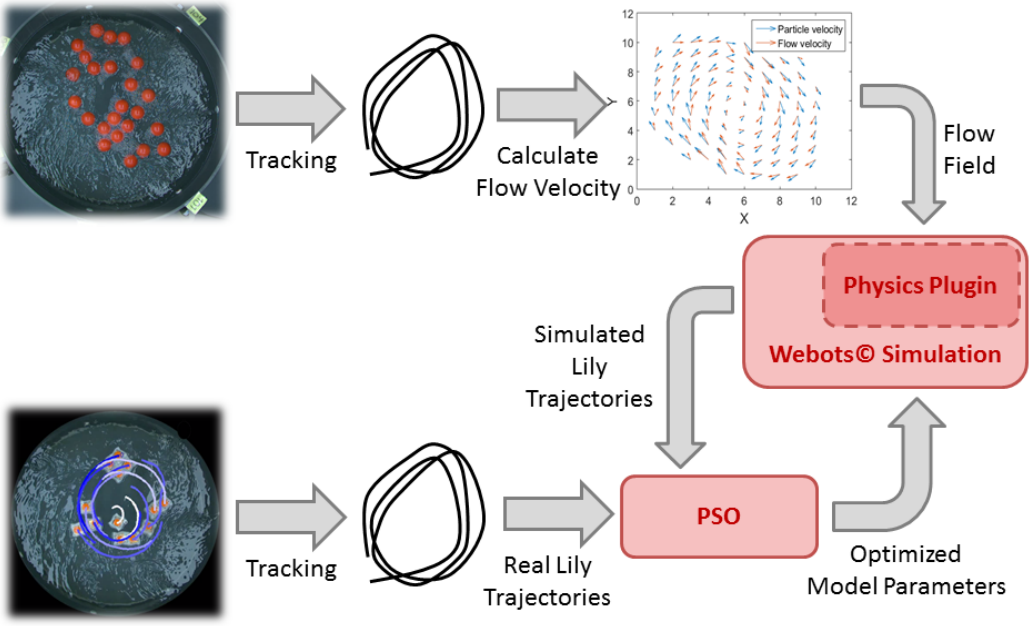


Figure 5: Calibration approach block diagram. There are three main components in the procedure, the trajectories of the real ping-pong balls, the trajectories of the real Lilies, and the simulated Lilies trajectories.

the water level, we can see a similar effect. Lowering the water level will change the relative alignment between the pumps' nozzles and the surface of the water, therefore increasing the fluidic force acting on the blocks' which in turn will decrease the effective exploration radius. The fact that the MSD of system trajectories is significantly different in different agitation modes in the system is a positive indicator that the MSD metric provides a reliable representation of system dynamics.

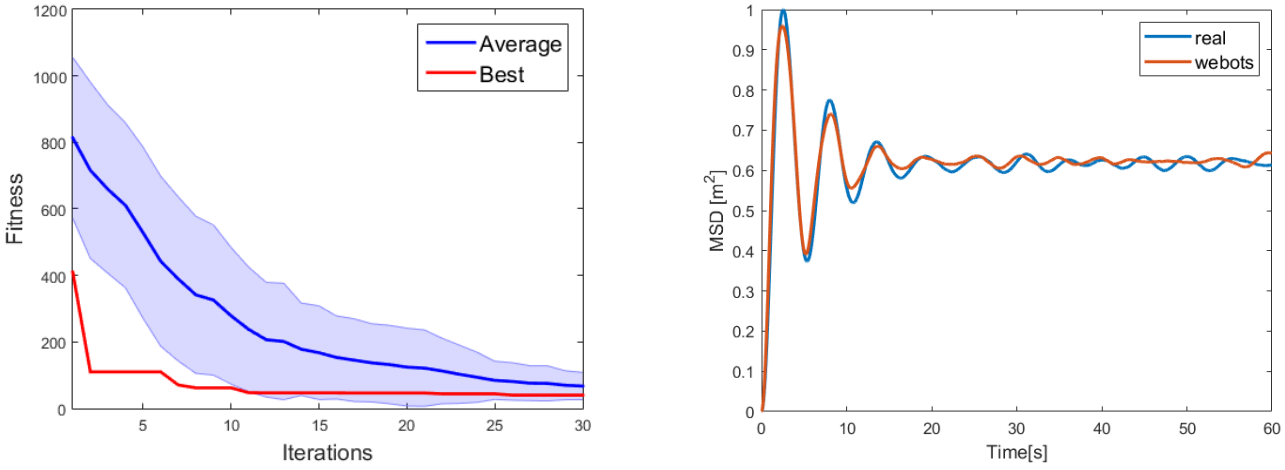
4.3. Optimization Procedure

As discussed earlier, the basic principle of the calibration is to match the simulated and real MSD curves in an attempt to faithfully reproduce the dynamics of the real system in simulation. There are five model parameters to be tuned in the calibration process; K_v defining a scaling factor for the randomness in the velocity field as described in Eq. 4, K_F defining a scaling factor in the drag force as described in Eq. 5, σ_{stoch} defining the standard deviation of the stochastic force field, and the two linear and angular damping coefficients used by Webots for any body mass expressed as D_{linear} and $D_{angular}$, respectively. As mentioned before, we use a PSO algorithm to fine tune these parameters so that real and simulated MSD curves are as aligned as possible. The optimization takes place only within a specific range for each of these parameters in order to ensure the stability of the simulation. The PSO parameters of inertia, personal best coefficient, and global best coefficient are set to 0.1832, 0.5287, and 3.1913, respectively, according to [21]. No particular attempt to optimize the PSO parameterization was carried out.

The fitness function is therefore the difference between the resulting MSD from simulation and the mean MSD measured on our real set-up. When computing the MSD value the trajectories of each of the floating blocks are aggregated, rather than being considered separately, and the resulting MSD curves are averaged. Mathematically, we can formulate as below, where N_s is the number of time steps per sample. Where $N_s = 2400$ in our case corresponds to 60 s emulated time with a simulation time step of 25 ms.

$$Fitness = \sum_{i=1}^{N_s} |MSD_{webots} - MSD_{real}| \tag{9}$$

Figure 5 depicts a block diagram of our calibration procedure. The flow field extracted from trajectories exclusively generated with ping-pong balls is used to create the flow field in simulation using the dedicated physics plugin explained in Section 3.2. We used 24 ping-pong balls, a water of -8 cm from the edge of the tank, and a pump power of 69 %. At this point the development of the flow field in simulation is frozen. Originally, we first applied the optimization to the case of floating ping-pong balls trajectories. The optimized parameters were then used as a starting point for the optimization involving the Lily robotic modules.



(a) (b)

Figure 6: Lily modules experiment:(a) Optimizing for the fitness function throughout the PSO algorithm iterations. (b) Comparison of simulated and real MSD data.

Table 1
PSO algorithm parameters and optimized physics-based model parameters for simulated and real experiments with ping-pong balls and Lily robotic modules.

Floating blocks	Dimensions	Iterations	Swarm size	K_v	K_F	σ_{stoch}	D_{linear}	$D_{angular}$
Ping-pong balls	5	30	47	1.24	2.266	195.25	0.3483	0.2476
Lilies	5	100	47	1.365	0.442	126.82	0.332	0.12

In this further optimization process, however, the objective was to match the MSD values obtained from experiments involving both real and simulated Lily robotic modules. The parameters used for these simulations are shown in Table 1. We later discovered that the optimization process was robust enough to allow for a procedural simplification. We therefore directly optimized the model parameters for the case of the real and simulated trajectories of Lily robotic modules, as depicted in Figure 5.

We used 15 Lily robotic modules for capturing real trajectories. The water level and pump power were the same as before, namely -8 cm and 69%, respectively. The PSO results as well as the resulting matching MSD are shown in Figure 6 and the used simulation parameters as well as the optimized model parameters are listed in Table 1. It can be seen that the MSD curves from the simulated and real Lily robotic modules have a close matching using the optimized model parameters. As a further validation step, we compared the normalized mean velocities in x and y directions as depicted in Figure 7, where the data was extracted from 10 minutes of experiment using 24 ping-pong balls.

5. Model-Based Control of Self-Assembly

Depending on the capabilities of the self-assembly building blocks and the controlability of the environment, a range of fully distributed to fully centralized control approaches may be employed. The control approaches aim to guide the self-assembly process towards building the desired target structure efficiently. The efficiency of a control approach can be measured considering two metrics, the rate and the yield of the controlled process. The assembly yield at each time is the number of copies of the target structure assembled at that time. The final yield is at a point in time when the experiment is halted. The assembly rate indicates the rate at which the process progresses towards building copies of the target structure. The problem of designing control strategies for self-assembling robotic systems has been traditionally studied in a very abstract way. In one of our previous contributions [8], we addressed the problem of designing ruleset controllers directly applicable to robotic modules rather suited only for abstract bodiless modules. Here, we aim to present a method for optimizing the parameters of such rulesets, namely the probability associated with the execution of each rule. To this end, we need to understand the functionality of the system as well as its dynamics and have accurate models of the system dynamics at hand. In our self-assembling system of Lily robotic modules, the process of assembling a given desired target structure can be influenced mainly by two factors. First, by tuning the program, i.e. the embedded ruleset controller, governing the assembly decisions of the robotic modules and second, by tuning the flow in the environment moving the robotic modules around.

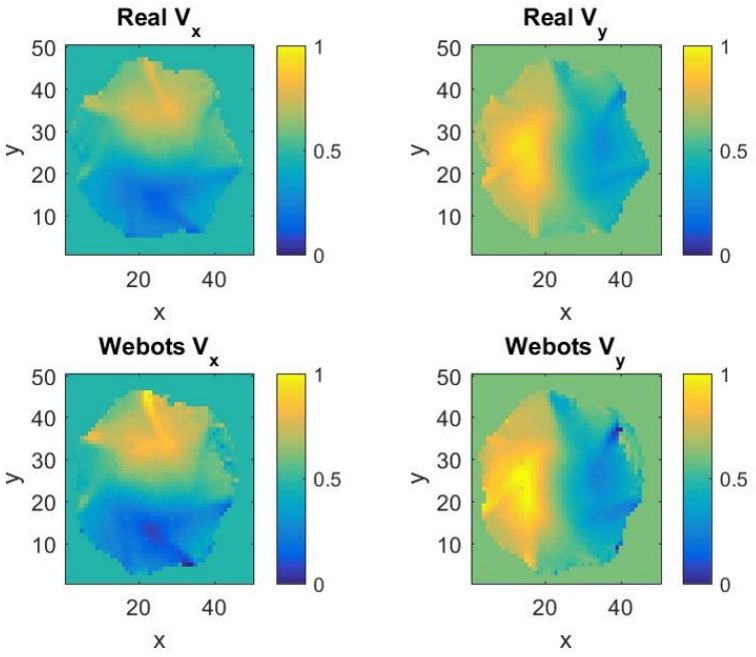


Figure 7: Comparison of real (top row) and simulated (bottom row) mean velocity. The data is extracted from 10 minutes of experiment using 24 ping-pong balls.

In the current work, we consider a fixed agitation mode in the system, i.e. the one for which the calibration procedure has been carried out as explained in the previous sections, and focus our optimization task on finding optimal ruleset parameters.

Forming a target structure by a swarm of Lilies involves several aspects. Given a target structure, an appropriate behavioral ruleset is programmed on all robots through wireless bootloading. The robots EPM latches are by default enabled, resulting in a default latching upon meeting another robot. Once latched, the EPM-to-EPM inductive communication channel is physically established. The robots then exchange their internal states and look for an applicable rule in their ruleset. The ruleset contains interaction rules similar to chemical reactions, with the left-hand side representing the current self-state and the neighbor-state, and the right-hand side representing the corresponding updated states. If no applicable rule is found, the robotic modules will unlatch by switching off their EPM latches; otherwise, they remain latched and update their internal states accordingly. Each Lily then updates the base station with its new internal state over the radio. This information can then be used to log the experiment and also as the ground truth for validating the models of the system. In addition to event-based reporting of their internal state, Lilies periodically communicate to the base station to check for pending commands such as a query about the battery voltage level or the internal state, a command for pausing the experiment, or a command for turning the robotic modules off. This scheme allows the robotic modules to spend most of their time in sleep mode or having the power-hungry radio transceiver off, thus resulting in an extended battery life. The commands from the base station can be as well used to modify the robotic moules' behavioral ruleset on the fly.

5.1. Synthesizing Structures of the Rulesets

We employ a dedicated software framework which allows for automatically synthesizing rulesets which are directly programmable on the robotic modules [8]. The framework is based on an extended graph grammars formalism and generates rulesets based on a geometrical description of the desired target structure, a specified synthesis algorithm, and the geometry of the robotic modules. The synthesized rulesets include two types of rules, *forward* and *reverse* rules. The forward rules advance the assembly process forwards by having a larger assembly form as a result of the execution of the rule. The reverse rules break down an assembly into smaller ones, essentially reversing the effect of a forward rule. For any forward rule in the ruleset there exists a corresponding reverse rule, reversing the effect of that rule. The significance of the reverse rules is in providing a method for avoiding deadlock situations in the system, where as a result of the modules having only partial information about the state of the system, several partially built copies of the target structure may co-exist, essentially competing with one another rather than contributing to the completion of a copy of the target structure. Probabilistic execution of the reverse rules allows for avoiding deadlock situations and achieving the target structure with probabilistic guarantees[1]. It is these probabilities which we intend to optimize in the following sections.

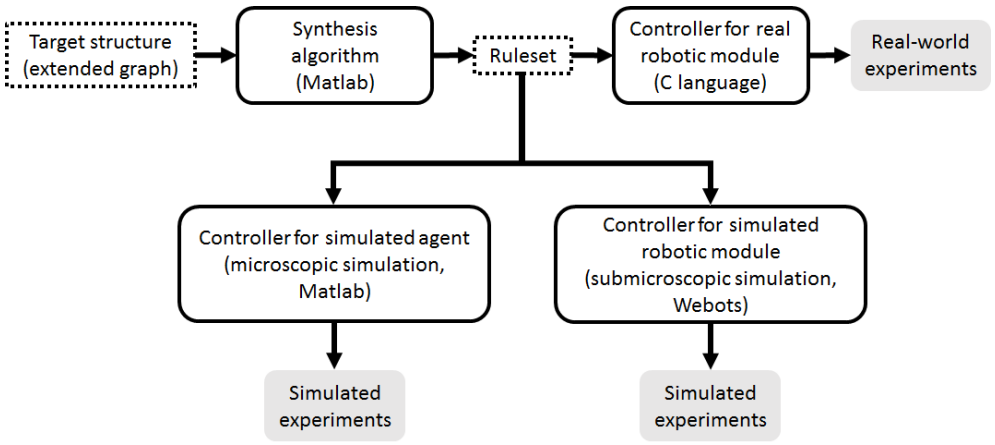


Figure 8: Diagram of the overall software framework. The "synthesis algorithm" block may utilize different rule synthesis algorithms. In the current work SingletonR synthesis algorithm has been used from [8].

Structurally, the synthesized rules describe two interacting modules' states using a combination of a control state variable and a relative hop number. The idea is that the robotic modules can only take part in a reaction defined by a certain rule if they have the right control state and are participating in the reaction with the appropriate orientation. Once a latching connector is engaged, the module communicates its internal state in the form of a relative extended label of $l = (l_a, l_h)$ with l_a being the module's control state and l_h being a relative hop number which represents the relative orientation of the currently engaged connector with respect to its predecessor, assuming a CCW hop convention. For a module with an internal state of (l_a, l_n) and N connectors, the relative hop number is calculated as $l_h = [(l_n - l_c) mod N] + 1$, where l_n and l_c are the indexes of the most recently and the currently engaged connectors, respectively. For an isolated module the connectors are anonymous in terms of interaction possibilities, thus $l_h = 0$.

Assuming that Lilies are all initialized with a state of $(0,0)$, the rulesets ϕ_-^S and ϕ_+^S shown below synthesized by the SingletonR algorithm in [8] allow for formation of a chain shape and a cross shape as depicted in Figures 9 and 11, respectively. The S index indicates a serial assembly strategy induced by the rulesets, meaning that each assembly step proceeds by having exactly one isolated robotic module joining the growing structure. The forward rules, r_1 to r_4, advance structure formation, while the backward rules, $\bar{r}_1$ to $\bar{r}_4$, allow for avoiding deadlocks.

The resulting rulesets are typically not easy to understand at first glance. Here, we provide additional explanations and visualizations in order to bring further intuition on their operation. Consider the ϕ_-^S whose SA progress course using five Lily robotic modules is visualized in Figure 9. The values of $l_h = 3$ on the Left-Hand-Side (LHS) of the rules dictate two hops on the link slots between the successive latching events, resulting in a linear structure considering the square-shaped modules. The reverse rules all have $l_h = 1$ at the LHS, indicating that the rule's corresponding interaction happens at the link slot engaged the latest.

$$\phi_-^S = \begin{cases} (0,0)\ (0,0) \xrightarrow{r1} (1,1)-(2,1) \\ (1,3)\ (0,0) \xrightarrow{r2} (3,1)-(4,1) \\ (4,3)\ (0,0) \xrightarrow{r3} (5,1)-(6,1) \\ (6,3)\ (0,0) \xrightarrow{r4} (7,1)-(8,1) \\ (1,1)-(2,1) \xrightarrow{\bar{r}1} (0,0)\ (0,0) \\ (3,1)-(4,1) \xrightarrow{\bar{r}2} (1,3)\ (0,0) \\ (5,1)-(6,1) \xrightarrow{\bar{r}3} (4,3)\ (0,0) \\ (7,1)-(8,1) \xrightarrow{\bar{r}4} (6,3)\ (0,0) \end{cases}$$

$$\phi_+^S = \begin{cases} (0,0)\ (0,0) \xrightarrow{r1} (1,1)-(2,1) \\ (1,2)\ (0,0) \xrightarrow{r2} (3,1)-(4,1) \\ (3,3)\ (0,0) \xrightarrow{r3} (5,1)-(6,1) \\ (5,4)\ (0,0) \xrightarrow{r4} (7,1)-(8,1) \\ (1,1)-(2,1) \xrightarrow{\bar{r}1} (0,0)\ (0,0) \\ (3,1)-(4,1) \xrightarrow{\bar{r}2} (1,4)\ (0,0) \\ (5,1)-(6,1) \xrightarrow{\bar{r}3} (3,3)\ (0,0) \\ (7,1)-(8,1) \xrightarrow{\bar{r}4} (5,1)\ (0,0) \end{cases}$$

Consider the ϕ_+^S whose SA progress course using five Lily robotic modules is visualized in Figure 11. Each square represents a Lily, labeled with its internal state l_a value in the middle. The most recent engaged link slot is indicated with a blue mark, while the relative hop numbers of l_h are shown on the modules' sides. For each Lily, numbering the slots always starts with $l_h = 1$ at the most recently engaged slot and follows a CCW convention. Note that the synthesis algorithms only generate the rules; appropriate probabilities should be associated with forward and reverse rules in order to allow the system to recover from deadlocks, while reliably forming the target.

5.2. Optimizing Parameters of the Rulesets

Assuming a stable final target structure, the probability associated with the backward rule $\bar{r}_4$ dissociating the fully formed target structure is set to zero. As a result, there are three ruleset parameters to be optimized, namely the p_1, p_2, and p_3, associated with $\bar{r}_1$ to $\bar{r}_3$ backward rules. The optimization is done independently for each of the rulesets

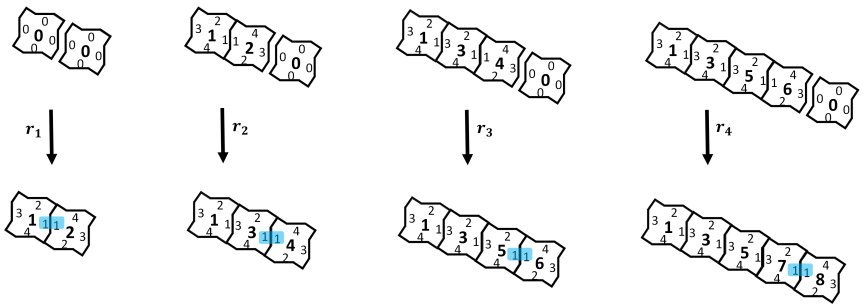
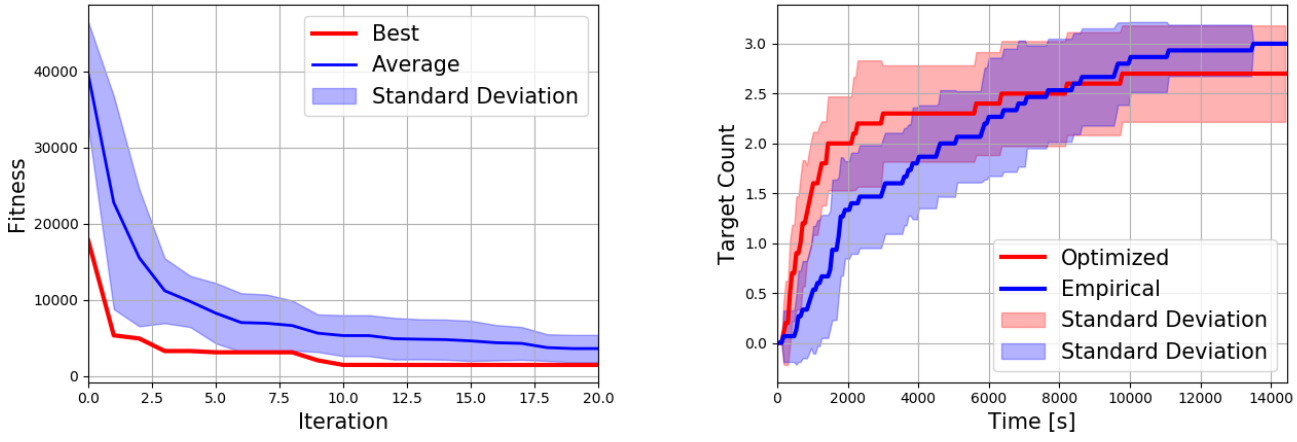


Figure 9: Progress of the SA process for the chain shape target structure employing ϕ^S_-. The latest engaged latching connectors on the modules are highlighted with a blue mark, while the relative hop numbering starting at the most recently engaged latching connector are shown on the sides of each module.



(a) (b)

Figure 10: Chain shape target structure comprising five Lilies: (a) Evolution of the fitness function throughout iterations of the PSO algorithm. (b) Comparison of ruleset performance for optimized vs empirically selected parameters.

and its corresponding target structure. The forward rules are chosen to be executed with probability 1, meaning that if there is a chance for the right configuration to form upon collision of two Lilies, it will form and the assembly will proceed.

Similar to the case of the model calibration, we use a PSO algorithm to optimize the ruleset parameters so that a fitness function is minimized. In order to cope with the stochasticity in the system, we use the same noise-resistant version of PSO as explained in Section 4.1. The PSO parameters of inertia, personal best coefficient, and global best coefficient are set to 0.1832, 0.5287, and 3.1913, respectively. No particular attempt to optimize the PSO parameterization was carried out. Each particle has a dimension of three corresponding to the three ruleset probability parameters. A total of 15 particles are evaluated over 20 iterations. As we use a noise-resistant variant of the PSO algorithm, each particle is evaluated twice and the fitness value is averaged. The total number of evaluations therefore sums up to 600. Each evaluation runs a Webots simulated world of the system with 15 Lily robotic modules programmed with the ruleset corresponding to the particle being evaluated. The maximum achievable yield is three for our target shapes of size five and the simulated world is evaluated over a period of 4 hours or 14400 seconds, denoted as T_{sim}. The system trajectory is thus evaluated for an overall duration of 2400 hours of simulated time. We run the simulations on a cluster, engaging 15 nodes at a time each node comprising an Intel Xeon E5620 processor of 8 cores, with each node running one instance of the simulated world. In this configuration, 600 evaluations of the system trajectory each for a duration of four hours takes a total of about 28 hours wall-clock time.

The fitness function that we employ aims to maximize the assembly rate while achieving the highest yield possible. It is defined as the sum over an array $\{\Delta T_i\}_{i=1}^{N}$ whose elements are initialized with T_{sim}, where N is the maximum yield or the total number of copies of the target structure achievable in the system. The n^{th} element in the array corresponds to the formation of the n^{th} copy of the target structure and is the time elapsed between the formation of the $(n-1)^{th}$ and n^{th} copies of the target, provided that the event occurs within the simulation duration. The first element stores the

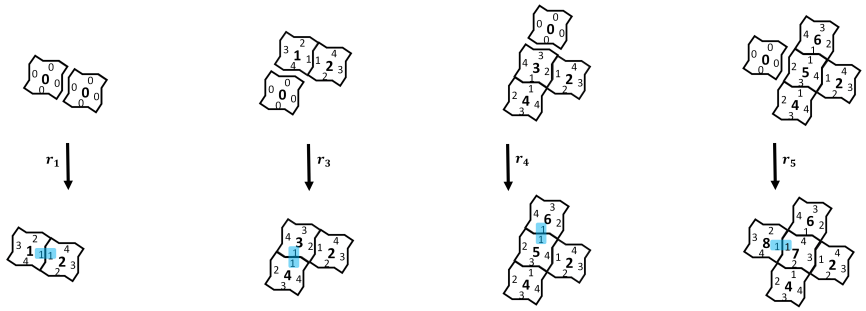


Figure 11: Progress of the SA process for the cross shape target structure employing ϕ_+^S. The latest engaged latching connectors on the modules are highlighted with a blue mark, while the relative hop numbering starting at the most recently engaged latching connector are shown on the sides of each module.

time until the formation of the first copy of the target since the beginning of the simulation. Mathematically, we can formulate the fitness function as below, where N is the maximum yield.

$$Fitness = \sum_{i=1}^{N} min(\Delta T_i, T_{sim}) \tag{10}$$

Figure 10 shows the results of the optimization process and a comparison of the optimized rule parameters with an empirically chosen set of parameters for the chain shape target structure. The evaluations of the best set of parameters and the empirically chosen set of parameters are carried out for 15 runs. The optimized set of parameters for $p = [p_1, p_2, p_3]$ as well as the empirically set ones are $p_o = [0.4796, 0.4203, 0.0016]$ and $p_e = [0.01, 0.005, 0.000025]$, respectively. The choice of p_e is simply based on the intuition that the more advanced the structure the more stable it must be and also that the formation of the dimers should be quite stable. It is interesting to note that while the optimized rule probabilities achieve a significantly higher assembly rate, the overall assembly yield is eventually higher for the case of empirically chosen parameters. The crossing of the two curves highlights two points. First, it highlights a feature of our specific choice of fitness function which seems to favor achieving a higher rate over actually building the maximum number of targets. A different fitness function might lead to optimized parameters which allow improved rate and yield compared to the empirical choice. Second, in the case that no such fitness function exists, this motivates employing a hybrid strategy where the two sets of rule probabilities p_o and p_e are used during the assembly process, depending on the number of copies of the target structure formed in the system.

Figure 12 shows the results of the optimization process and a comparison of the optimized rule parameters with an empirically chosen set of parameters for the cross shape target. The evaluations of the best set of parameters and the empirically chosen set of parameters are carried out for 15 runs. The optimized set of parameters for $p = [p_1, p_2, p_3]$ and the empirically set ones are $p_o = [0.4857, 0.4086, 0]$ and $p_e = [0.01, 0.005, 0.000025]$, respectively. Here again the choice of p_e is simply based on the intuition that the more advanced the structure the more stable it must be and also that the formation of the dimers should be quite stable. For the case of the cross shape target structure, the optimized set of parameters allows for a higher rate and yield compared to the empirically set parameters.

6. Discussion and Conclusion

In this paper, we addressed the problem of designing and calibrating accurate physics-based models of self-assembly of floating modules in a fluidic environment, subsequently exploited for designing model-based control strategies for the system. In particular, we considered the case of our fluid-mediated self-assembling robotic system built around the water-floating Lily robotic modules which rely on the surrounding environment for their mobility. The fluidic flow in the environment is created by several peripheral pumps on the fluidic tank. The self-assembly process in the system was guided towards achieving a global target structure in a distributed fashion by means of appropriate ruleset controllers programmed on the robotic modules. The robotic modules maintained an internal state corresponding to their local perception of the progress of the self-assembly process. The ruleset controller programmed on the modules regulated the outcome of the random interactions between two robotic modules and was characterized by specific rules as well as their associated parameters of execution. Given a desired target structure composed of several robotic modules, the rules, constituting the structure of a ruleset, were synthesized using a formal framework developed in one of our previous works. The framework essentially compiles the desired target structure into a set of assembly steps. Finding optimized ruleset parameters is then realized by exploiting the physics-based model of the system, ultimately resulting in a nonconventional model-based control approach.

The primary and main contribution of this work is introducing a method for designing and calibrating a lightweight physics based model of a fluid-mediated self-assembling system of robotic modules. To this end, we first recreated our robotic modules' hardware and firmware in a simulated world using the Webots robotic simulator. We then developed a dedicated physics plugin to apply appropriate

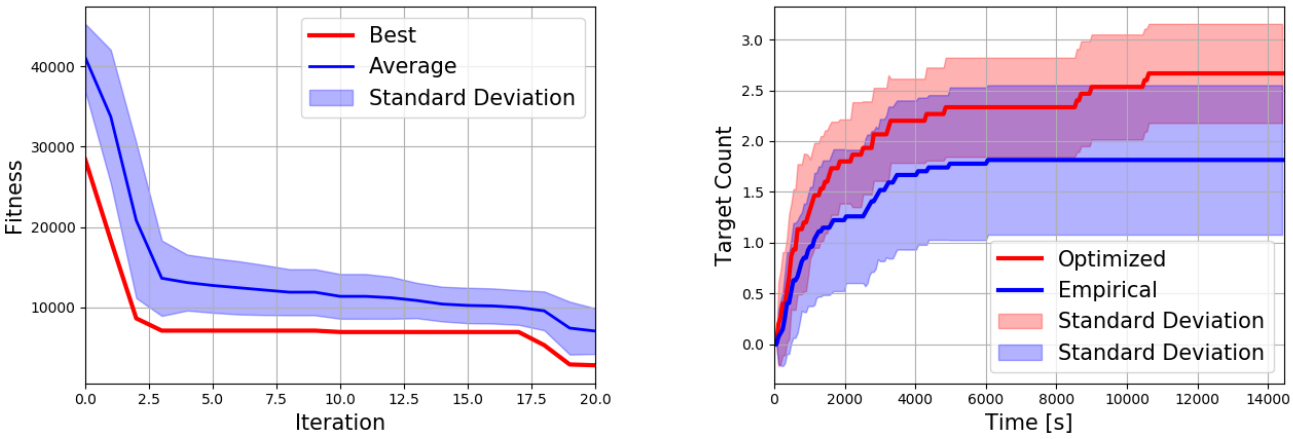


Figure 12: Cross shape target structure comprising five Lilies: (a) Evolution of the fitness function throughout iterations of the PSO algorithm. (b) Comparison of ruleset performance for optimized vs empirically selected parameters.

hydrodynamic forces on the simulated robotic modules. In particular, we introduced a novel method for automatically calibrating the model parameters employing a PSO algorithm. The secondary contribution of this work is using the calibrated physics-based model of the system in order to optimize ruleset controllers for the robotic modules in order to achieve a desired target structure. This was motivated by the fact that the physics of the system, in particular, the mixing created by the agitation in the environment, plays a key role in the progress of the assembly process. As a result, the performance of a ruleset controller can only be reliably evaluated and subsequently optimized by observing the real physical system or a faithful simulation model of it.

The strengths and limitations of our proposed approach can be summarized as follows. For a given agitation mode in the fluidic arena, our model design and calibration approach results in a computationally lightweight but accurate model of the system which can in turn be used to evaluate the efficiency of control strategies. The reduced computational cost of the model makes it particularly suitable for use in combination with metaheuristic optimization procedures where numerous evaluations of the system trajectory are necessary. Our current method for optimizing the ruleset controller considers first synthesizing the rules based on the given desired target structure and then generating optimized rule probabilities through exploitation of a previously calibrated physics-based model. In general, for a given target structure several assembly strategies, or equivalently, several rulesets guaranteeing the creation of the target structures exist. By fixing the ruleset prior to performance evaluation under realistic conditions we are probably limiting the overall assembly performance. A more general approach for optimizing ruleset controllers would be to simultaneously optimize the actual rule ensemble (among the multiple candidate rulesets ensuring the creation of the targeted structure) and its parameters.

Future work will be followed on several fronts. In addition to investigating the simultaneous optimization of ruleset selection and rule parameters mentioned above, we intend to leverage the same modeling approach to investigate and optimize more sophisticated control strategies involving a combination of distributed and centralized control for the robotic modules and the agitation, respectively. Finally, last but not least, a validation of our optimized control solutions with real Lily robotic modules will be needed.

CRediT authorship contribution statement

Bahar Haghighat: Conceptualization of this study, Software, Methodology, Writing - Original draft preparation. **Hala Khodr:** Software, Methodology. **Alcherio Martinoli:** Writing - Original draft preparation.

References

[1] V. Ganesan, M. Chitre, On stochastic self-assembly of underwater robots, IEEE Robotics and Automation Letters 1 (2016) 251–258.

[2] B. Haghighat, M. Mastrangeli, G. Mermoud, F. Schill, A. Martinoli, Fluid-mediated stochastic self-assembly at centimetric and submillimetric scales: Design, modeling, and control, Micromachines 7 (2016) 138.

[3] M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, G. Chirikjian, Modular self-reconfigurable robot systems [grand challenges of robotics], IEEE Robotics & Automation Magazine 14 (2007) 43–52.

[4] L. Jacot-Descombes, Fluid-mediated Self-assembly of MEMS Microcapsules for Liquid Encapsulation and Release, Ph.D. thesis, 2013.

[5] L. Matthey, S. Berman, V. Kumar, Stochastic strategies for a swarm robotic assembly system, in: IEEE International Conference on Robotics and Automation, 2009, pp. 1953–1958.

[6] T. P. Pavlic, S. Wilson, G. P. Kumar, S. Berman, Control of stochastic boundary coverage by multirobot systems, Journal of Dynamic Systems, Measurement, and Control 137 (2015) 034504.

[7] E. Di Mario, G. Mermoud, M. Mastrangeli, A. Martinoli, A trajectory-based calibration method for stochastic motion models, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011, pp. 4341–4347.
[8] B. Haghighat, A. Martinoli, Automatic synthesis of rulesets for programmable stochastic self-assembly of rotationally symmetric robotic modules, Swarm Intelligence 11 (2017) 243–270.
[9] E. Klavins, S. Burden, N. Napp, Optimal rules for programmed stochastic self-assembly., in: Robotics: Science and Systems, 2006.
[10] N. Bhalla, P. J. Bentley, C. Jacob, Evolving physical self-assembling systems in two-dimensions, in: International conference on evolvable systems, 2010, pp. 381–392.
[11] B. Haghighat, A. Martinoli, Characterization and validation of a novel robotic system for fluid-mediated programmable stochastic self-assembly, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2016, pp. 2778–2783.
[12] T. Lochmatter, P. Roduit, C. Cianci, N. Correll, J. Jacot, A. Martinoli, Swistrack-a flexible open source tracking software for multi-agent systems, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008, pp. 4004–4010.
[13] O. Michel, Webots: Professional mobile robot simulation, Advanced Robotic Systems 1 (2004) 39–42.
[14] J. Kennedy, R. Eberhart, Particle swarm optimization, in: IEEE International Conference on Neural Networks, 1995, pp. 1942 – 1948 vol.4.
[15] A. Engelbrecht, Particle swarm optimization: Where does it belong?, in: Third IEEE Swarm Intelligence Symposium, 2006, pp. 48–54.
[16] J. Pugh, Y. Zhang, A. Martinoli, Particle swarm optimization for unsupervised robotic learning, in: IEEE Swarm Intelligence Symposium, 2005, pp. 92–99.
[17] J. Pugh, A. Martinoli, Distributed scalable multi-robot learning using particle swarm optimization, Swarm Intelligence 3 (2009) 203–222.
[18] E. Di Mario, I. Navarro, A. Martinoli, Analysis of fitness noise in particle swarm optimization: From robotic learning to benchmark functions, in: IEEE Congress on Evolutionary Computation, 2014, pp. 2785–2792.
[19] E. Di Mario, I. Navarro, A. Martinoli, Distributed particle swarm optimization using optimal computing budget allocation for multi-robot learning, in: IEEE Congress on Evolutionary Computation, 2015, pp. 566–572.
[20] D. Frenkel, B. Smit, Understanding molecular simulation: From algorithms to applications (Academic, San Diego, 2002) (1997) 63–107.
[21] M. E. H. Pedersen, Good parameters for particle swarm optimization, Hvass Lab., Copenhagen, Denmark, Tech. Rep. HL1001 (2010).

Bahar Haghighat is a postdoctoral research fellow at the John A. Paulson School Of Engineering And Applied Sciences, Harvard university. She received her PhD degree in *Robotics, Control, and Intelligent Systems* in 2018 at the Swiss Federal Institute of Technology in Lausanne (EPFL). Bahar received her M.Sc. degree in Electrical Engineering majoring in Digital Electronics in 2012 and her double-major B.Sc. degrees in Electrical Engineering and Physics in 2010 from Sharif University of Technology, Iran.

Hala Khodr obtained her BE (2016) in Electrical and Computer Engineering from the American University of Beirut (AUB) and her MSc (2018) in Microengineering with a specialization in Robotics from the Swiss Federal Institute of Technology in Lausanne (EPFL). She then joined the Computer-Human Interaction in Learning and Instruction (CHILI) laboratory at EPFL in December 2018 as a Ph.D. student. Her research focuses on tangible swarm robots and their integration in classrooms for enhancing the learning experience.

Alcherio Martinoli has a M.Sc. in Electrical Engineering from the Swiss Federal Institute of Technology in Zurich (ETHZ), and a Ph.D. in Computer Science from the Swiss Federal Institute of Technology in Lausanne (EPFL). He is currently an Associate Professor at the School of Architecture, Civil, and Environmental Engineering and the head of the Distributed Intelligent and Algorithms Laboratory at EPFL. Before joining EPFL he carried out research activities at the Institute of Biomedical Engineering of the ETHZ, at the Institute of Industrial Automation of the Spanish Research Council in Madrid, Spain, and at the California Institute of Technology, Pasadena, U.S.A. His research interests focus on methods to design, control, model, and optimize distributed cyber-physical systems, including multi-robot systems, sensor and actuator networks, and intelligent vehicles.