Multi-Agent Systems [H02H4a]
Simulation tool of a programmable self-assembly in a robot swarm

Geoffroy Herbin, r0426473

June 6, 2020

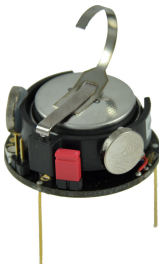
Abstract

Simulation should not be opposed to experimentation when developing an engineering system. Rather, simulation should support the design decisions, anticipate the problems, and lead faster towards good results. It can be particularly challenging to build a simulation tool that takes into account uncertainties, and can be used in a reliable way to predict or understand the results of real experiments. It should not prevent the building of such a tool, but encourage it. This report presents a simulation tool developed to support real experiments of tens to hundreds of tiny robots moving together in order to build shapes.

1 Introduction

Kilobots are tiny cheap robots developed in the objective of facilitating real life experiments on swarm intelligence algorithms. The authors of [5] report a system using a thousand of Kilobots, that are able to build complex two-dimensional one-connected shapes, only through local interactions and rules. The authors insist on two contributions: the first is the Kilobot's low-cost design, introduced in [4], and the second is the self-assembly algorithm, based on SDASH, introduced in [7], and adapted and proven in [6].

The Kilobot's design is unconventional and answers different challenges, inherent to swarm intelligence – and large groups of individuals – study. The functional requirements for such a robot are the availability to approximate holonomic motion, communicate with neighboring robots, measure distance to communicating robot, basic computation capabilities and internal memory. The main non-functional requirements are induced by the strong need of scalability: low cost, large-scale operability (switch-on/off, charging, ...). To fulfill those requirements, a Kilobot moves using two tiny vibration motors, and communicates with neighbors through infrared LED. A Kilobot costs about 14$, which makes it possible to use in large-swarm real life experiment.



a Kilobot

The self-assembly algorithm consists in a finite state automaton based on three primitives: *edge-following*, *gradient-formation*, and *localization*. Those are depicted in Appendix A.1, Figure 13. Edge-following allows a Kilobot to move along the edge of a group of stationary robots by measuring distances from nearest stationary neighbors. Gradient-formation allows each Kilobot to have a geodesic distance from the source – one of the seeds – which initiates this gradient value. Thanks to the localization, the Kilobots swarm forms a local coordinate system only through local communication and distances to neighbors.

At the start of the experiment, all the robots receive the algorithm, a black and white image of the shape to assemble, and the size (or scale) of this shape. Four seeds are then disposed next to the initial packed group of Kilobots. This operation initiates the process by indicating the origin of the local coordinate system, the source of the gradient, and it defines where the shapes should be formed. The seeds will not move for the rest of the process. Using the gradient value, a Kilobot will leave the swarm, and start edge-following. At some point in the process, the Kilobot will have entered the shape to assemble, and either finds itself about to leave the shape, or meets a robot with the same gradient value. If one of those conditions is met, the Kilobot stops for the remainder of the process.

This was a high-level and contracted view of the algorithm and working principle, and the reader is advised to look up the reference article and supplementary materials. Illustration of the different steps are given in Appendix A.1, see Figure 13.

While the algorithm itself is proven correct under idealized conditions, several uncertainties are elicited by the authors: the motion is not holonomic as the Kilobot continues going forward when rotating, the communication is lossy, there are errors on the neighbors' distance measurements, Kilobots don't all have the same speed, and the probability of a rare event (e.g. a Kilobot experiences a breakdown) increases with the amount of robots in the swarm. The authors also refer to the difficulty of studying the algorithm in a simulated environment, specifically because of the uncertainties, hard to model. As a result, they report long experiments and corrections/modifications of the self-assembly algorithm to compensate for those differences. For instance, as the Kilobots don't all have the same speed, the authors detail a protective mechanism that makes all robots staying away from each others.

The results obtained in [5] are impressive and demonstrate the capabilities of both the algorithms and the Kilobot. There is however no simulation environment presented, that would allow evaluating the ideal version, the impact of the real-life characteristics, and the effects of the different uncertainties. The benefits of such simulation environment are not to be demonstrated anymore. If they never replace completely a real life experiment, they allow early detection of numerous problems. It reduces the total resources needed for the development of such system, in time and in hardware. Other authors in [2] also report the implementation of a simulated environment, with in particular integration of finer physical properties. Their code and result have not been looked at, and are considered out of the scope of the current report. More generally, this work has not been inspired by any of the current existing simulation framework.

This work does not intend to *completely* model the system described in [5]. Rather, it firstly intends to reproduce a close-to idealized version, yet considering some real-life characteristics. Secondly, it shows how to elaborate on this version in order to model, implement and study the uncertainties described: non-holonomic displacement, speed uncertainties, distance measurement variation, and rare events.

2 Model

This section describes the specifications, the decisions, and the hypothesis considered in order to model the real life experiment of [5]. The self-assembly algorithm and the primitives are not detailed, as the intent is to scrupulously follow recommendations of [6]. This work heavily relies on good comprehension of those two

references. To support the reading, the state diagram of a robot, directly written according to [6], is given in Appendix A.2, Figure 14.

2.1 Overview

Modeling a perfectly ideal system would have a limited interest, and the findings would be similar to the correctness proof of the algorithm. It can be argued that such an idealized model would serve as a checkpoint for the model's implementation correctness. This is not the path followed by this study: examples and figures from [5] and [6] allow skipping this complete ideal model to directly integrate realistic behaviors as non-holonomic movement and the heuristic of trilateration. Yet, the implementation will detail shortcuts that can be taken in order to get closer to idealized behavior.

The model includes the Kilobot itself, called *robot* or *agent*, and the *world*, encompassing the whole experience. In real life, the world is limited to the experiment environment and includes the table, the charging system, the programming system, the cameras to monitor, etc. In the simulated environment, the world is limited to a two-dimensional space. A Kilobot is modeled as its center point, moving in the world. The parameters are defined with respect to the robots' center: for instance, the distance to closest neighbor is expressed as the distance between the robots' center. Although represented as points, the robots have an intrinsic circular shape of diameter equal to 1. This gives a multiplication factor of 32.5 between the spatial units from [5] and [6], and the modeled world. Rigorously, the unit of the model, *world* unit, is therefore not $[mm]$, but match a $1/32.5[mm]$. For simplicity, the text omits units when referring to simulated world.

As the goal is to reliably reproduce the results of the real life experiments, several modeling decisions are imposed:

- the world needs to be continuous: the robots move freely in two dimensions, without a grid restriction. This corresponds to classical (x, y) coordinates system.
- the robots perform the actions simultaneously. In real life, the robots move freely independently of the others: this needs to be reproduced,
- an individual robot knows its forward direction,
- the communication happens only with (close) neighbors, allowing the gradient computation, the localization, and the edge-following procedures.
- initially, all non-seed robots are constrained to be in a connected configuration inside an hexagonal lattice with spacing 2π, where every robot is at a distance greater than $3r$ from any point in the desired shape, where r is the radius of a robot.

2.2 Robot displacement

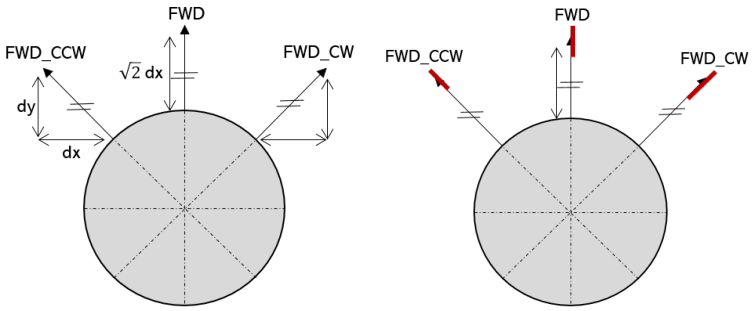
The algorithm proof in [6] assumes holonomic displacement: a robot does not move in (x, y) when changing its direction. This is not the Kilobot behavior, which only approximates this displacement, as described in [4]. The model integrates this uncertainty, and only three movements are possible: moving forward, clockwise, or counterclockwise. This is represented in Figure 1(a). Everytime the robot moves, it moves by a predefined distance, along one of the three directions. This is a discretized version of the real behavior, described in [4]: a Kilobot moves forward at $1cm/s$, and rotates 45 $^\circ/s$. The resulting model parameters are the length of one displacement *step*, directly mapping the real life speed of the Kilobot. There are two benefits of such modeling: first, it gets close to real-life behavior where a Kilobot continues moving forward when rotating; second, it offers an easy way to incorporate speed uncertainty, appropriately using the parameters. Time step and speed are discussed in section 2.3.

A limitation with respect to reality is the lack of continuity in the rotation angle: $\pm 45^\circ$. Model-wise, the extension of the agent's movements to a finer grained version, closer to continuous rotation range, is trivial. However, considering the detailed algorithm in [6], such a refinement is not required: the edge-following procedure does not specify a specific angle, only a direction. As a result, this three-direction modeling is considered sufficient in the context of this work.

2.3 Time

In real life, the time notion is considered continuous. It means that the refresh of Kilobot's local position, or its gradient value is performed at a relative high frequency. In particular, this ensures computation required for the localization, trilateration using a greedy approach (see [6], to converge. In the simulated environment, the notion of time is discretized in steps. All the robots need to be activated at each time step, and not one per step, so that multiple robots can perform actions during one step. This better matches reality. Moreover, a time step should be small enough to properly approximate continuous scale. This directly sets constraints on the speed.

Speed definition In [4], an experiment including 25 Kilobots reports a communication rate of 240 five-byte packets per second, and that the distance sensing occurs during any communication. Assuming that this



(a) Only three direction possible (b) Impact of speed variation

Figure 1: *Robot displacement*

corresponds to current problem setup, it means a Kilobot receives 240 information about the distance of its neighbors 240 times per second, while moving at a speed of $6.5[mm/s]$, as defined in [6]. Assuming that when it edge-follows, a robot has 10 neighbors on average, one execution step of the simulation makes the robot moving by a distance d, for 10 distances received. This distance d has the same value as the nominal robot speed, because the time unit is the step. The link between Kilobot's speed and robot's speed is therefore established:

$$\begin{aligned} speed_{robot} &= 6.5\frac{mm}{s} \times \left(240\frac{distances}{s}\right)^{-1} \times 10\frac{distances}{step} \\ &\approx 0.27\frac{mm}{step} \\ &\approx 0.008\frac{\texttt{world}\ distance}{step} \end{aligned}$$

The speed of a robot is rounded to 0.01.

2.4 Uncertainties

In order to limit the scope of the tool for the context of this work, three uncertainties are modeled: the speed variation, the uncertainty on the distance measurements, and rare events.

Speed Variation It can be considered as an intrinsic characteristic of each Kilobot. An individual robot always moves with a constant speed, and this speed depends on internal properties and hardware proficiency. The model integrates this variation as a multiplication factor applied on the robot nominal (or supposed) speed. That is, the length of the displacement as in Figure 1(a) is multiplied by a factor, which is a robot randomly assigned parameter. Most Kilobots are expected to have similar speed, varying only by a little percentage, and only a few showing large differences. This corresponds to a normal distribution, with mean and standard deviation chosen to reflect realistic behavior. Each robot therefore has a speed variation factor sampled from $\mathcal{N}(1,\, 0.1^2)$. The mean and deviation are parameters of the model. The impact of the speed variation on the initial supposed speed is represented in Figure 1(b), where the bold red bars indicate the range of the possible displacement, per step. In particular, all directions are subject to the same speed variation.

This modeling matches the description in [6], although it's probably an over-simplification of real behavior: one could expect the current in battery to become lower, hence the overall speed to decrease it time. The model could integrate such behavior as an extra product term, dependent on the time step (exponential or linear decay, for instance). Also, one could imagine to have two modeling factors, one for dx, one for dy. These have not been studied further in the context of this work.

Distance measurement A Kilobot heavily relies on distance measurements to edge-follow, and to compute its gradient value. Uncertainty on the measurements comes from four sources: the electronic sensitivity itself, the anisotropic distance sensing (the distance measured from A to B differs than from B to A) due to sensor orientation and intrinsic performance, random noise, and messages loss. Rather than trying to model those four different sources, [4] reports a mean sensing accuracy of $\pm 2mm$ with a precision of less than $1mm$. As well detailed in [1], precision is related to how a measurement is reproducible, while accuracy states how close a measurement is to a target value. Only the accuracy matters in this modeling. It follows an additional term, sampled from a uniform distribution $\mathcal{U}(-0.062, 0.062)$[1], simply added to the distance returned.

[1]this is actually a conservative assumption: a normal distribution would be acceptable, as it is possible to make sure the sensors work in the best condition most of the time, hence reducing the accuracy uncertainty

Rare event It is considered that a rare event corresponds to a robot motors breakdown, and that a robot does not restart after a breakdown. Many sources discuss this type of modeling, see [8], which can lead to complicated simulation. This work follows a pragmatic approach: if a random sample from $\mathcal{U}(0,1)$ is higher than a configurable threshold (i.e. $\theta = 0.95$), it follows a motor breakdown. As a consequence, the robot will not move anymore, but still communicates with neighbors.

3 Implementation

This section details the key points of the model's implementation. Similarly to previous section, the details of the algorithms won't be repeated if they don't represent particular interest. The full code is located in a private github repository, accessible upon request to the author[2].

3.1 Overview

The implementation of the model follows the pseudo-code given in [6], and the state diagram in Figure 14. It uses the Object Oriented paradigm.

The system is implemented in Python, using MESA library (see [3]) to set up the multi-agents environment. It has been selected as regards to the programming language, the steep learning curve, and the features available that comply with model specification:

- Continuous space: allowing the robot to move on float values. MESA includes built-in functions to move a point in such space,
- Step-wise scheduling: when created, the agents are added to a MESA object `scheduler` that will take care of their activation at each time step,
- Simultaneous agent activation: the `scheduler` activates all the agents at each time step.

Furthermore, MESA offers some interesting tools for logging, as an export to csv files of defined variables. It has been extensively used for post-processing – one of the main interests of simulation environment. For the distance computation and mathematical representation, the `numpy` library is used.

3.2 World

The world is a `World` object, inheriting from `Model` class from MESA. It represents a 2D continuous space, with attributes `width` and `height`. Visually, the origin of the world is always represented on the lower left. When created, the world will set up the continuous space, create the robots, and add them to its `scheduler`. During simulation, the `scheduler` object will simultaneously calls each agent's `step()` method, to compute the changes for that particular step, then `advance()`, to apply the changes computed. This behavior is facilitated by MESA library, hence the proficiency of code reuse.

3.3 Robots

The robots are `Robot` objects, inheriting from MESA `Agent` class. They have multiple attributes that allow reproducing the behavior of a real Kilobot. The complete description is not done here – the reader is advised to look up details in the class itself – but the key elements are summarized.

A robot as a unique identifier in the whole world. This is a simplification of real experiment local identifier, but it does not alter the model nor results. A robot has an orientation and position in the world, `pos`, and a position in the local coordinate system, `_s_pos`. The initial `pos` is set at the creation of the robot, to comply with initial conditions requirements from [6]. `_s_pos` is computed at each step using the localization primitive, while the `pos` is updated after a robot movement.

If the robot is a `Seed`, inheriting from `Robot`: its behavior is much simpler: it remains static, and has both `pos` and `_s_pos` set at the beginning of the simulation run.

The local interactions between robots are implemented as simple function calls, and allow a robot to get the list of neighbors (robots closer than a desired distance), the distance to a neighbor, if a neighbor is stationary, and the gradient value of a neighbor. Robot's implementation regarding local interactions mimic Kilobot's capabilities – with the exception of the local ID.

Finally, the three primitives `edge_follow()`, `compute_gradient()` and `localize()` allow to build the finite state machine. The `localize()` method incorporates the opportunity to by-pass the heuristic used, either by always setting up `_s_pos` perfectly, as in an ideal world, or using an optimization algorithm to minimize the trilateration equation. The choice of the `localize()` behavior is selected through a parameter `TRILATERATION_TYPE`.

The attribute `state` is used along the simulation to trigger specific actions, as per the finite state automaton implemented.

[2] All the figures are issued using the program developed, unless specified otherwise

3.4 Initial set up

A key aspect of the simulation run is the initial set-up. To ensure the model specification, the hexagonal lattice is computed before any robot creation and packed in a list. The `world` then attributes the appropriate position to each `robot` created. This enforces the correct positioning of the four seeds, firstly created, and the robots according to the lattice. Two set up are used, as represented in Figure 2: either similar to the reference [5] ideal case, or more general realistic case.

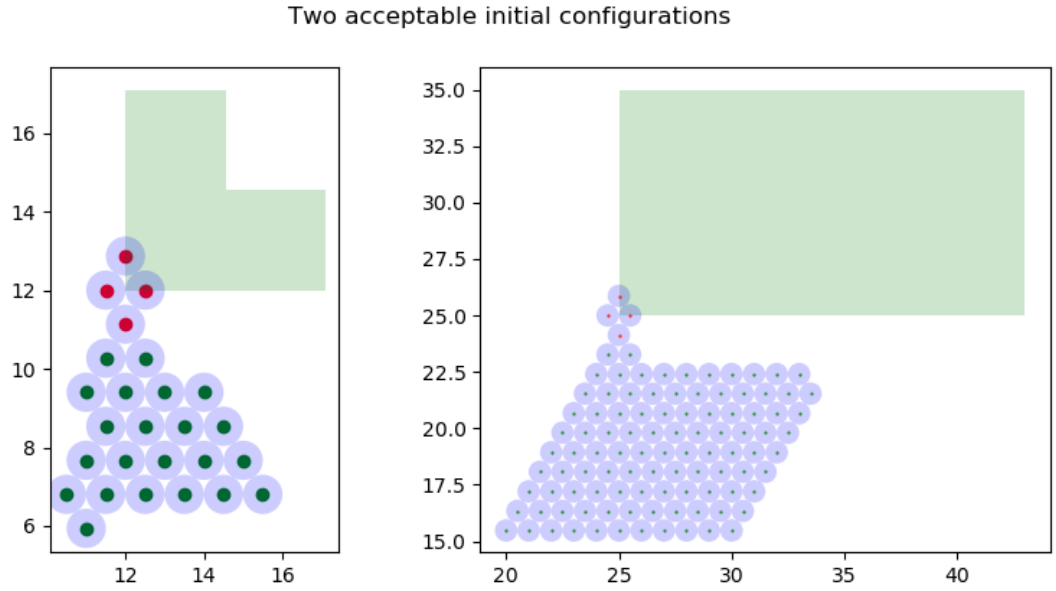


Figure 2: *Two valid initial configurations. Left, as in the reference article [5], Right, initial configuration for larger experiments. The axes express the global world coordinates. The points are the centers of seeds (in red) and robots (in green). The blue areas represent the robots. The green area represents the target shape.*

3.5 Ending state

At the end of the experiment, either all the robots could join the shape that may or not be complete, or there are too many robots to fill the shape. In this case, the already moving robots will continue edge-following, forever: that is a direct consequence of the algorithm implementation. To comply with the simulation need, an extra ending condition is added: if a robot crosses twice the mean y coordinate of the seeds, then the simulation run is stopped. Indeed, it implies a robot has performed a loop. Note that in order not to overload the computations, this condition is checked only every 1000 steps.

3.6 Simulation run

A simulation run matches usual multi-agent systems, with an initialization phase, and a repetition phase, until the maximum number of iterations is reached or a stopping condition is true. The sequence diagram in Appendix A.3, Figure 15, gives an overview of such a run. Logging is done through two channels: messages are printed on the console during a run, to monitor the evolution, and a `csv` file is created at the completion of each run. This file contains the positions in both coordinate system and the state of each robot, for each step.

3.7 Uncertainties

Their implementation follows the modeling described:

- **Speed variation**: implemented as an additional `intrinsic_speed_factor` attribute of a `robot`. The default value is 1 and is set by the `world` at the robot's creation. This factor is multiplied to the distance to perform, as described above.
- **Distance measurements**: the calls to get a distance are centralized in `get_distance` method. Every time this function is called, it samples a term and add it to the real value.
- **Rare event**: Every 500 steps, if a value randomly sampled is above the configured threshold, the parameter `intrinsic_speed_factor` is set to 0. As this parameter is not subsequently reset, it ensures the robot will remain static for the rest of the run, yet continuing being part of the simulation.

3.8 Parameters

The different parameters of the model implemented are summarized in Table 1. Some are a direct result of the the SDASH implementation [6], the others are related to the uncertainties and design decisions. The values have been computed in order to reliably match references' results. There are two additional user-defined simulation run parameters: the shape given to the robots and the number of robots.

Name	Typ. Value	Description	Formula
`K`	32.5	Division factor between distances in real simulated worlds	based of Kilobot size in mm
`DESIRED_DISTANCE`	1.615	Desired distance between the center of two robots, when edge-following. Real experiment reports 20[mm] between two neighbors, see [6], p 10.	$(2 * 16.25 + 20)/K$
`NEIGHBOR_RADIUS`	4.845	(Simulated) Communication range of the robot	$3 *$ `DESIRED_DISTANCE`
`G`	1.715	Distance to consider neighbors in gradient computation	$1.1 +$ `DESIRED_DISTANCE` $- 1$
`SPEED`	0.01	Distance achieved at each moving step (before uncertainty)	see section 2.3
`DIVIDE_LOCALIZE`	2	Division Factor used in trilateration greedy heuristic original = 4.	Experimental
`TRILATERATION_TYPE`	`real`	Type of trilateration to use in the localization primitive. See text.	`"real", "opt", "ideal"`
`STARTUP_TIME`	500	Number of steps to wait before leaving the `START` state. unit = `steps`	Experimental, see section 4.2
`INTRINSIC_MEAN`	1	Mean of the normal distribution used for a robot intrinsic speed variation factor	/
`INTRINSIC_STDDEV`	0.1	Standard deviation of the normal distribution used for a robot intrinsic speed variation factor	/
`DISTANCE_ACCURACY`	$6.15e-3$	Absolute value of the boundaries of the uniform distribution of additional distance term	2/`K`
`RARE_EVENT_THRESHOLD`	0.95	Threshold for uniform sample, above which a robot is down	/

Table 1: *Parameters used in the implementation of the simulation tool.*

4 Experiments

This section presents the experiments performed with the tool developed. First, the ideal case is reproduced. The influence of two hyper-parameters is then studied: the `START_UP` time, through the trilateration computing method, and the `DESIRED_DISTANCE` between robots. The uncertainties are introduced next: the speed variation, the distance measurement uncertainty, and rare event. Finally, a large scale simulation run is performed.

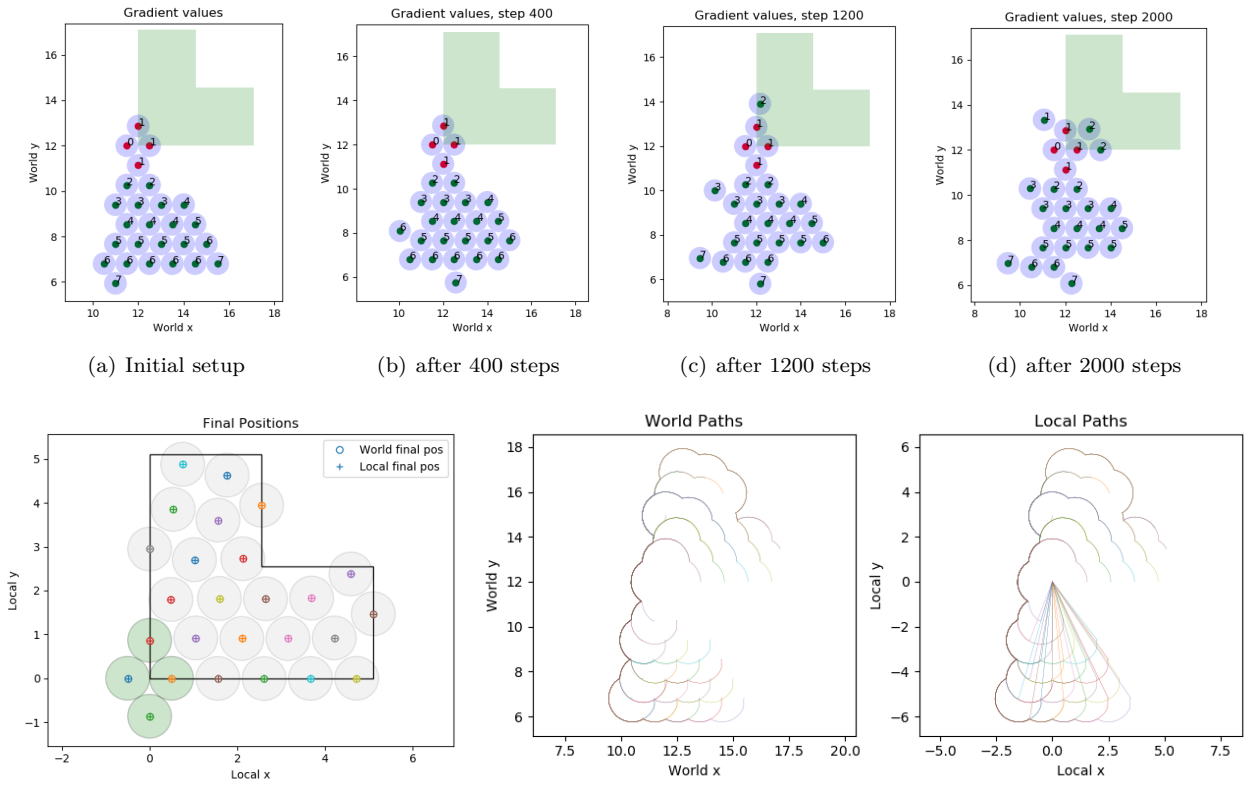
4.1 Reproduction of ideal case

The ideal process is reproduced. Besides the non-holonomic movements, no uncertainty is introduced. The experimental world has a height and weight of 25, and 22 robots (plus 4 seeds) are initially set in place. The shape to build is a `"L"`, with larger side of 5.02, and smaller size of 2.51. The parameters' values are fine-tuned in order to match visual results described in [5] and reminded in Appendix A, Figure 13. The parameters values are summarized in Table 2. Initially, the robots are placed as in Figure 2. Different states of the simulation are shown in Figure 3.

Name	Experimental Value
`DESIRED_DISTANCE`	1.05
`NEIGHBOR_RADIUS`	3
`G`	1.5
`TRILATERATION_TYPE`	`ideal`
`STARTUP_TIME`	10

Table 2: *Parameters in the reproduction of article results, in [5].*

Discussions The ideal case results are visually well reproduced. The snapshots reveal highly similar patterns to the reference results, which is the initial goal. The gradient values and behavior match expectations. Figure 3(e) shows that all the robots could find their spot in the shape: it stops the simulation. Each agent is represented as a `"o"` for its final world position, and a `"+"` for its final local position. In an ideal case, both perfectly match. If the robot is not, in the end, located where it thinks it lays, then there will be a difference between the two markers.



(a) Initial setup (b) after 400 steps (c) after 1200 steps (d) after 2000 steps

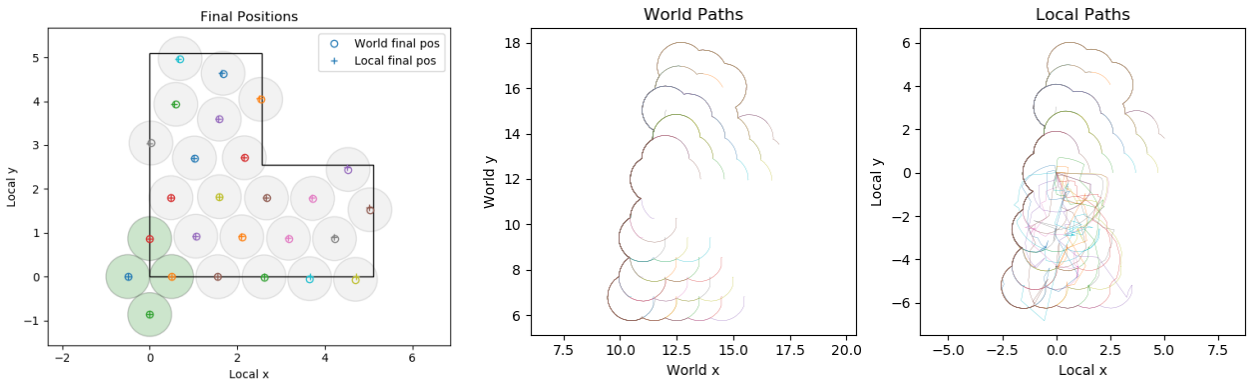
(e) End state (f) Paths

Figure 3: *Reproduction of article results, in [5], also shown in Appendix A.1*

Figure 3(f) introduces a visualization of the paths followed, in the world (on the left), or in the local coordinate system (on the right), as understood by each individual robot. The world paths correspond to expectations: circular orbits around the robots, same paths followed by all robots, ... Local paths show the `_s_pos` initially set at $(0, 0)$, then its fast convergence to ideal value, as per the `TRILATERATION_TYPE` used.

4.2 Influence of localization

In a perfect setup, the robot has directly perfect knowledge of its local position `_s_pos`. In a more realistic world, this is performed through trilateration and a greedy approach. The impact of this computing on the end-result can be assessed, modifying the parameter `TRILATERATION_TYPE`. It is expected that a more realistic setup will be slower to converge, and will lead to errors in end-positioning, visible through the robot markers. Figure 4 shows the results of simulation on the same setup as described, using `TRILATERATION_TYPE = "real"`. The graphical results for `TRILATERATION_TYPE = "opt"` is given in Appendix A.4, Figure 16. Table 3 compares the methods in terms of mean squared error (MSE) between the world and local positions, number of simulation steps, and relative time to complete.



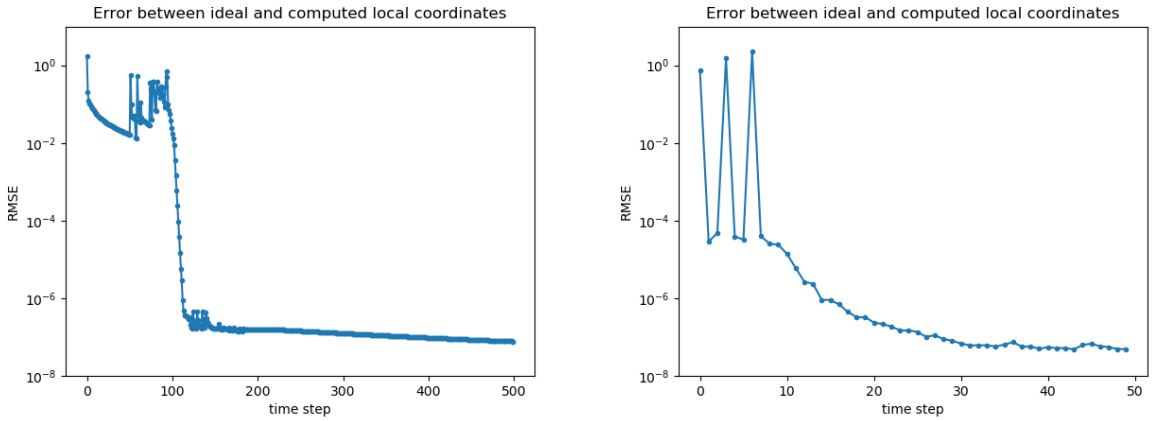
(a) End state (b) Paths

Figure 4: *Reproduction of article results, in [5], using the real trilateration method. Differences are visible between World positions and Local positions*

TRILATERATION_TYPE	End State MSE	Number of steps	Relative time to complete
ideal	$1.18e-32$	12300	1
opt	$4.9e-8$	12300	15
real	$5e-4$	12800	5

Table 3: *Parameters in the reproduction of article results, in [5].*

Discussions In the ideal case, the MSE is 0. Using an optimization method to compute the coordinates (`"opt"`), the graphical end-result is perfectly similar to the ideal case, as well as the number of steps to complete. The MSE is still very low, but the time to complete is much larger. The `real` trilateration experiment, using heuristic, shows discrepancies between the world and local positions, directly visible in MSE value. Those differences increase as the robots are further from the seeds: it matches reported real behavior. The number of steps to reach the end state is increased, but the global time to complete is reduced compared to the optimal experiment. This shows why the heuristic method is used in reality: using an optimization method is hard and resources demanding. The real local paths, in Figure 4(b), show the slow convergence of the local coordinates of each robot at the beginning. This convergence can be observed in terms of root mean square error (RMSE)[3] at each steps, and is reported in Figure 5(a). It takes about 200 steps for all the robots to have a local coordinate that has converged – with a remaining negligible RMSE. This value increases with the number of agents. Similar representation is shown in Figure 5(b) for the optimization computation: convergence is ≈ 10 times faster. These findings directly give the `STARTUP_TIME` parameter value, corresponding to the $60[s]$ startup time of the real-life experiment reported in [6].



(a) Real trilateration

(b) Optimization trilateration

Figure 5: *RMSE between ideal and computed local coordinated, per simulation steps*

4.3 Influence of the distance between robots

The distance between the robots when edge-following has a large influence on the process and on the final shape. The experimental world has a height and width of 50, and 25 robots (plus 4 seeds) are initially set in place. The shape to build is rectangle, of height 5 and width 10. Such an experimental world initial state is represented in Appendix A.5, Figure 17. The desired distance has two possible values: 1.615 and 1.3. All the other parameters are set as per Table 1. Results are shown in Figure 6.

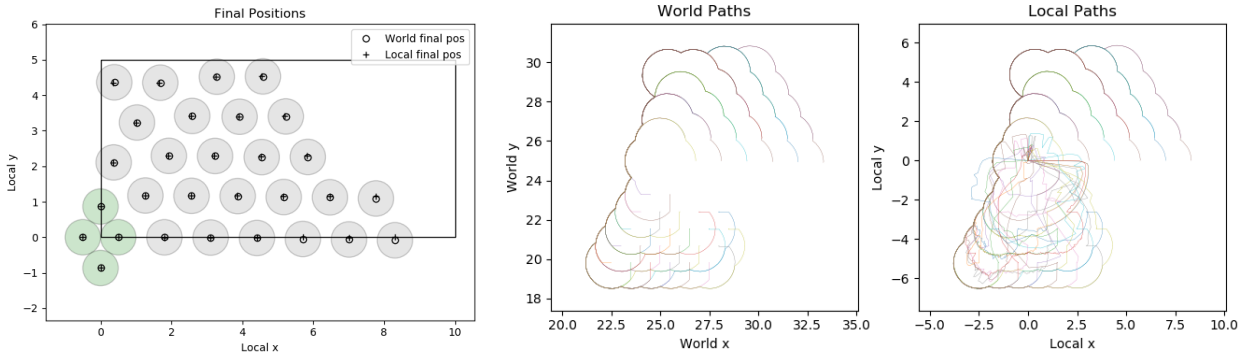
Discussions The end states vary: a small distance leads to all robots being inside the shape, incomplete, at the end of the run. A larger distance leads to a complete shape, and several remaining robots edge-following[4]. This situation triggers the added stop condition introduced in 3.5, due to a robot performing a loop. This is not implemented in the real experiment, which is manually stopped.

4.4 Speed Variation

The experimental world has a height and width of 50, and 25 robots (plus 4 seeds) are initially set in place. The shape to build is a rectangle of height 5 and width 10, as represented in Figure 17. All the other parameters are set as per Table 1. The speed variation is activated (see 3.7). The end state and paths are shown in Figure 7.

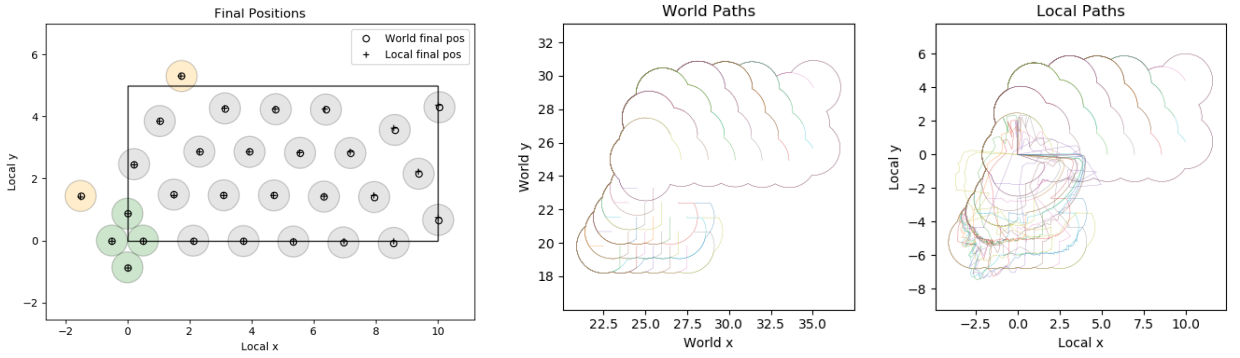
[3] RMSE instead of MSE to have a metric related to the distance on the world

[4] In the general case, some could also still be at their initial position, waiting to move



(a) End state, `DESIRED_DISTANCE = 1.3`

(b) Paths, `DESIRED_DISTANCE = 1.3`

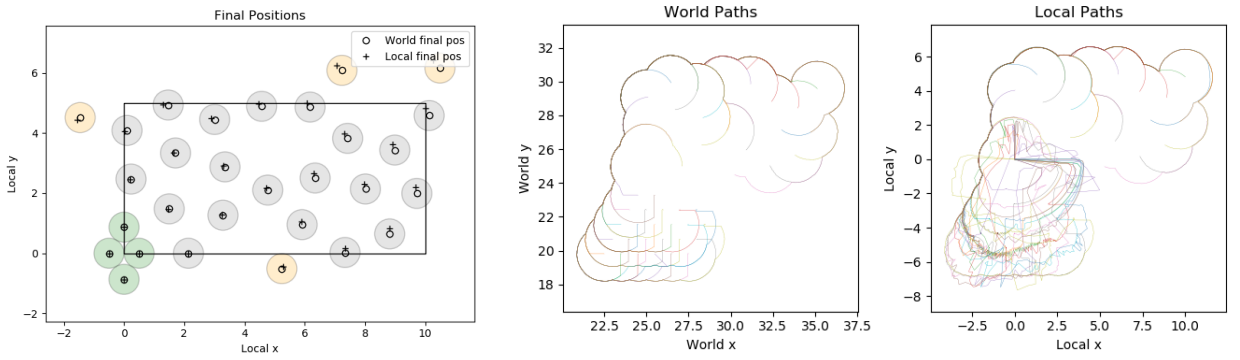


(c) End state, `DESIRED_DISTANCE = 1.615`

(d) Paths, `DESIRED_DISTANCE = 1.615`

Figure 6: *End states and paths for a rectangle shape self-assembly, using two `DESIRED_DISTANCE` parameter values. The end states vary: a small distance leads to an incomplete shape, and a larger distance leads to a complete shape, with some robots still edge-following (in orange)*

There are two expected effects. First, if the robot's speed is too high, it may have problem in computing its local coordinates and refreshing its gradient values. It will result in an error in positioning. Second, when robots don't move at the same speed, the moving robots will not remain equidistant on the shared path, and there is a risk of collision.



(a) End state

(b) Paths

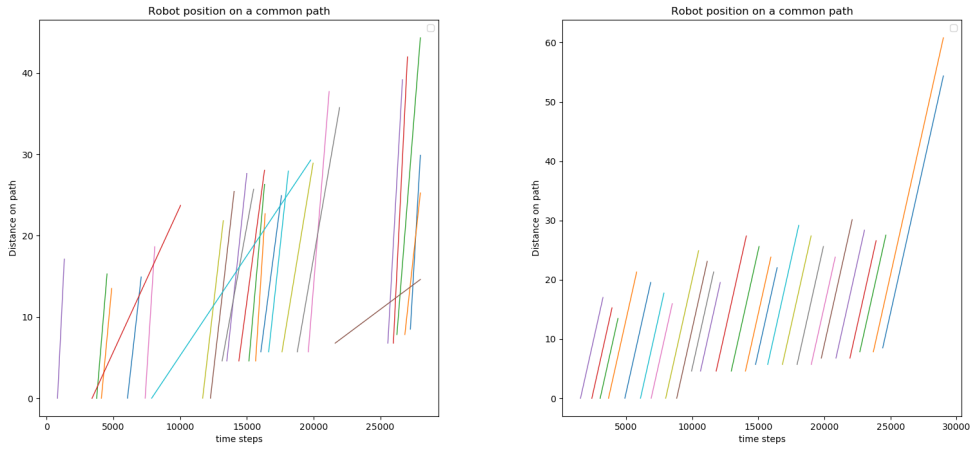
Figure 7: *End states and paths for a rectangle shape self-assembly, using speed variations.*

Discussions The first effect is observed in Figure 7. Several robots move too fast to refresh their internal state, and it follows an error in positioning. The overall shape is yet visible, and the impact seems limited. This effect can be considered as a simulation artifact if the processing speed of the Kilobot will not lead to such errors.

This graphical representation does not allows to see all the dynamic of the system. In order to observe the movement of each robot, it is possible project the robots' world (x, y) on a path, and plot the position of each robot on this *shared* path per time step[5]. Figure 8 shows such graphs, for the nominal case and the speed variation case.

[5]A quick explanation of how this graph is build is given in Appendix A.6

In the nominal case, represented in Figure 8(b), the robots move at the same speed. No two robots are at the same position for a given time step. The slopes of all represented trajectories are equal[6]. The two longest lines correspond to robots performing a loop. In the speed variation case, represented in Figure 8(a), the trajectories are not parallel anymore, and a higher slope indicates a faster robot. On this representation, each intersection corresponds to a collision in reality: a collision avoidance mechanism is required to prevent such cases. Slower robots also induce a slower start of the next robot to move, lengthening the experiment.



(a) Case with speed variations

(b) Nominal case

Figure 8: *Position of each robot on the shared path. A colored line represent the trajectory of one robot. Details of the graph construction in Appendix A.6*

There is no protective measure against speed variation in the tool implemented, and there is no mechanism preventing an agent from stepping over another robot. It comes clear from this analysis that collisions occur, and a collision avoidance mechanism is mandatory. In the real experiment, such mechanism exists: a robot stops if it is too close from its preceding neighbor (see [6]).

4.5 Distance Uncertainty

The experimental setup is similar to previous, see Figure 17. Distance uncertainty is activated, and all the parameters are as per Table 1. Four simulation runs are performed with varying `DISTANCE_ACCURACY`. The variations are expressed as a multiplication factor k of the nominal parameter `DISTANCE_ACCURACY`. Numeric results are shown in Table 4. A close-up of a path in the world coordinate system for $k = 1$ is given in Figure 9, and the end states for $k = 0.25$ and $k = 0.75$ are shown in Figure 10.

k	# steps until stop	MSE between world and local coord.	Localization RMSE after `STARTUP_TIME`
0.25	43000	0.0084	0.00408
0.50	46000	0.011	0.00816
0.75	150000	8.73	0.01224
1	stopped	/	0.01632

Table 4: *Distance accuracy experiments results*

Discussions Table 4 indicates that the positioning MSE and the initial localization RMSE (as introduced in 4.2) increases with the uncertainty. It also shows that the nominal `DISTANCE_ACCURACY` does not allow building the shape anymore, as the uncertainty introduced is too large: robots are lost. It results in erratic paths such as the close-up of Figure 9, and experiments that are lengthened. In Figure 10(c), the experiment stops and some robots have joined the shape, some believe they have but are in fact still on their way (gray robots outside of the rectangle), and others remain static as they don't realize the condition to start is true.

An error in the distance measurements directly affects the local coordinate system and the gradients. Because of the uncertainty introduced, the localization error stays high, and the trilateration cannot converge properly. It seems there is a threshold above which an initial localization error leads to undesirable experiment behavior.

In real system, there are different ways of solving this issue: one could filter the measurements using a moving average, as there is no sudden event foreseen. One could also make sure the sensors usually work according to their best performance: from a uniform distribution, the variation would then follow a normal distribution,

[6]Their length is not equal, because only a portion of the global path is retained. See Appendix A.6 for more details

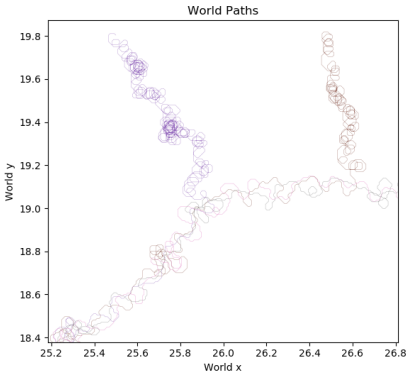
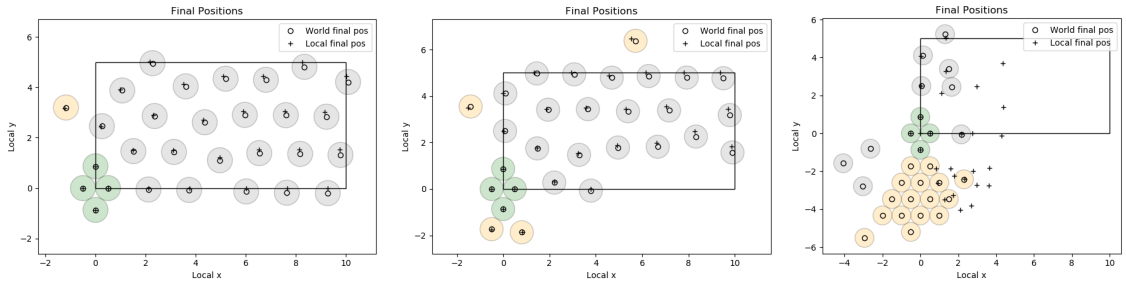


Figure 9: *Close-up of the path followed by robots when uncertainty on distance measurement.* $k = 1$



(a) $k = 0.25$ (b) $k = 0.50$ (c) $k = 0.75$

Figure 10: *End state for* $k = 0.25$*,* $k = 0.50$ *and* $k = 0.75$*. The distance uncertainty has not prevented the shape from being assembled for* $k = 0.25$ *and* $k = 0.50$*, but is too large in the case of* $k = 0.75$ *to reach the desired end state*

centered at 0. Such simulation results with $\mathcal{N}(0, \frac{2}{\mathrm{K}*3})$[7] indicate that the localization error remains lower: 0.00267 and 0.00996 for $k = 0.25$ and $k = 1$.

4.6 Rare Event Uncertainty

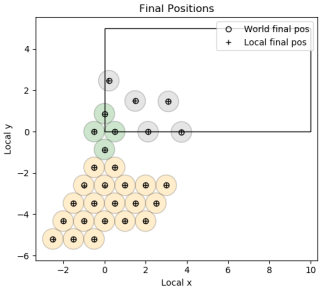


Figure 11: *End states after breakdown, threshold of* 0.80

The experimental setup is similar to previous, see Figure 17. Only the rare event uncertainty is activated. The experiment is run with two thresholds: 0.80 and 0.95. The sampling is performed every 1500 steps. Example of the end state is shown in Figure 11. The logs indicate respectively 11 and 14 robots down for the two thresholds considered. The experiments ended after the maximum allowed number of steps[8].

Discussions When a robot stops moving, it still communicates its state, its gradient, ... and prevents its neighbors from starting to move. As the experiment continues, more robots are subject to breakdown, and it results a complete and rapid "freeze" of the world. This shows that even with a low initial probability of having a breakdown, as the number of robots increases and the time goes on, the number of *rare* events increases. The authors of [5] report collaborative monitoring measures to prevent those occurrences. This simulation enlightens the relevance of such mechanisms.

[7] Corresponding to the uniform distribution, $\pm 3\sigma$

[8] 120% of the number of steps needed for the nominal run

4.7 Larger Scale

The system developed may be used at larger scale, to repeat and compare results from real experiments. Figure 12 shows the end state for a simulation run of 116 robots assembling a 19 by 11 rectangle. This corresponds to the experiments described in [6]. This simulation run did not include any uncertainty. The robots are perfectly packed, with remaining robots either still stopped, waiting to start, or edge-following. The discrepancy between the local and world markers tend to increase as the robots are further from the source, which is the expected real behavior.

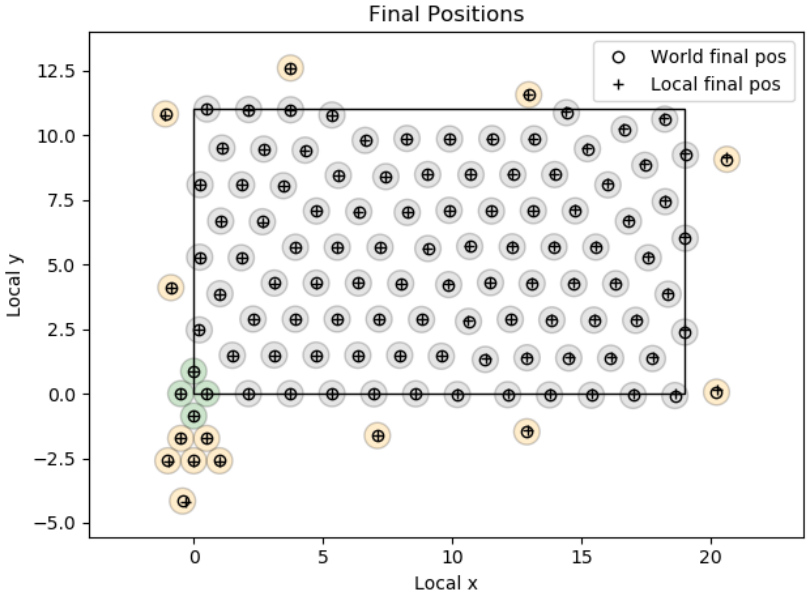


Figure 12: *Larger scale simulation run, reproducing end results of [6]*

5 Conclusion

This reports have presented a simulation tool for the Kilobots self-assembly algorithm, allowing to define an experimental setup that is capable of including Kilobot's uncertain behavior. The model and its assumptions have been presented, and several key points of the implementation have been detailed. The tool has also been used in different use cases that demonstrate behaviors reported by the authors of [5] and [6]. First, the ideal behavior has been properly reproduced. Second, the influence of two hyper-parameters, the distance between robots and the start-up time, have been discussed. Third, the uncertainty linked to the speed, the distance measurement, and rare event have been simulated, and their effect analyzed. Finally, a larger scale simulation run end result is given.

When used before physical experimentation, such a tool already indicates the preventive measures to implement to prevent any abnormal situation. It also allows defining hyper-parameters, such as the distance between robots, or the experiment start-up time. When used along the experimentation, it may help understanding the real observations.

To go further in the simulation, the tool could include mechanical properties of the robots, and implement the protective measures already defined, such as collision avoidance and collaborative monitoring.

References

[1] *Difference Between Accuracy and Precision*. URL: https://www.precisa.co.uk/difference-between-accuracy-and-precision-measurements/ (visited on 06/03/2020).

[2] Gregor HW Gebhardt and Gerhard Neumann. *The Kilobot Gym*. URL: https://projects.iq.harvard.edu/files/icra2018swarms/files/GebhardtEtAl_ICRA2018swarms.pdf.

[3] *Mesa Overview — Mesa .1 documentation*. URL: http://mesa.readthedocs.io/en/latest/overview.html (visited on 05/01/2020).

[4] M Rubenstein, C Ahler, and R Nagpal. "Kilobot: A low cost scalable robot system for collective behaviors". In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 3293–3298.

[5] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. "Programmable self-assembly in a thousand-robot swarm". In: *Science* 345.6198 (2014), pp. 795–799. ISSN: 0036-8075. DOI: 10.1126/science.1254295. URL: https://science.sciencemag.org/content/345/6198/795.

[6] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. "Supplementary Materials for Programmable self-assembly in a thousand-robot swarm". 2014. DOI: 10.1126/science.1254295. URL: https://science.sciencemag.org/content/suppl/2014/08/13/345.6198.795.DC1.

[7] Michael Rubenstein and Wei Min Shen. "Automatic scalable size selection for the shape of a distributed robotic collective". In: *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings* (2010), pp. 508–513. DOI: 10.1109/IROS.2010.5650906.

[8] Bart Van der Paal. "A comparison of different methods for modelling rare events data". 2014, pp. 1–75. DOI: 10.1080/01639625.1989.9967805.

A Appendix

A.1 Idealized algorithm

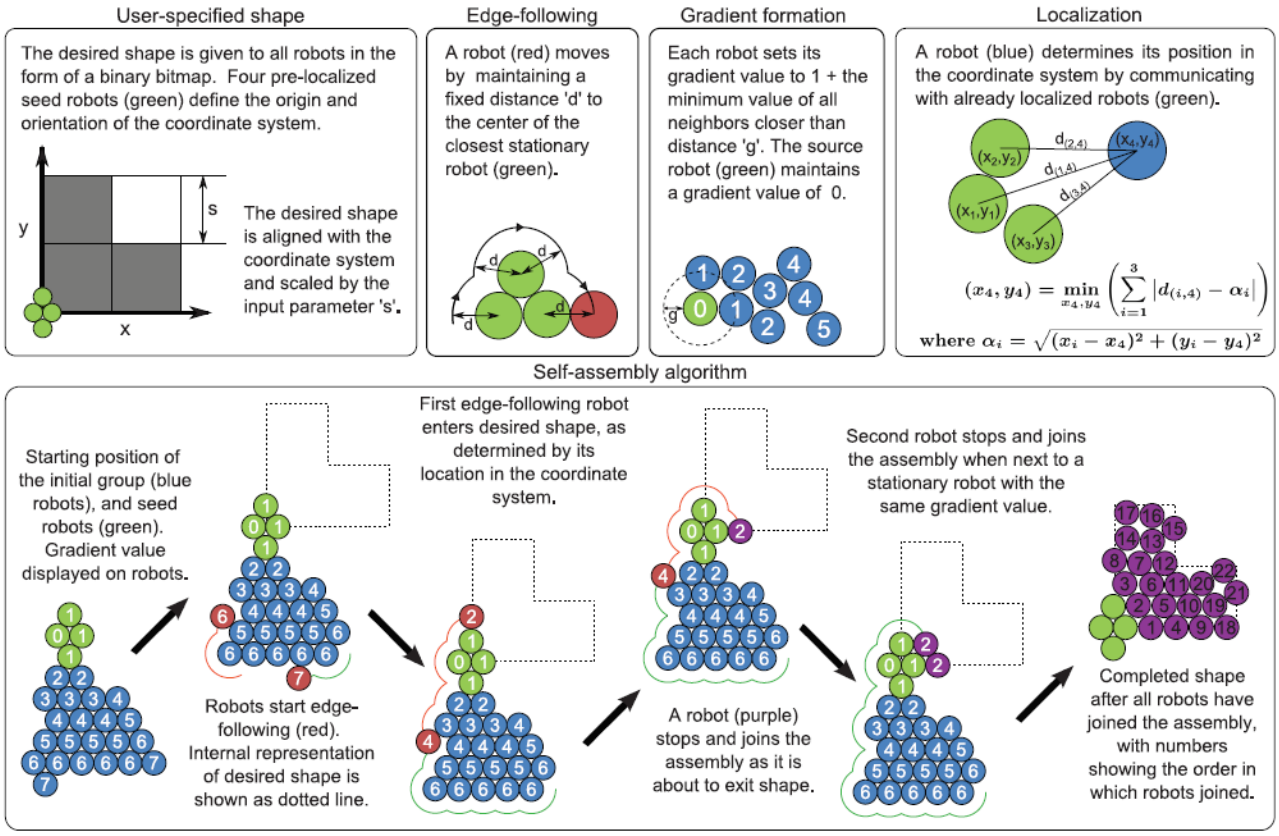


Fig. 2. Collective self-assembly algorithm. Top left: A user-specified shape is given to robots in the form of a picture. Top right: The algorithm relies on three primitive collective behaviors: edge-following, gradient formation, and localization. Bottom: The self-assembly process by which a group of robots forms the user-defined shape.

Figure 13: *Steps of the self-assembly algorithm, in an ideal case. source of the picture: [5]*

A.2 Self-assembly state machine

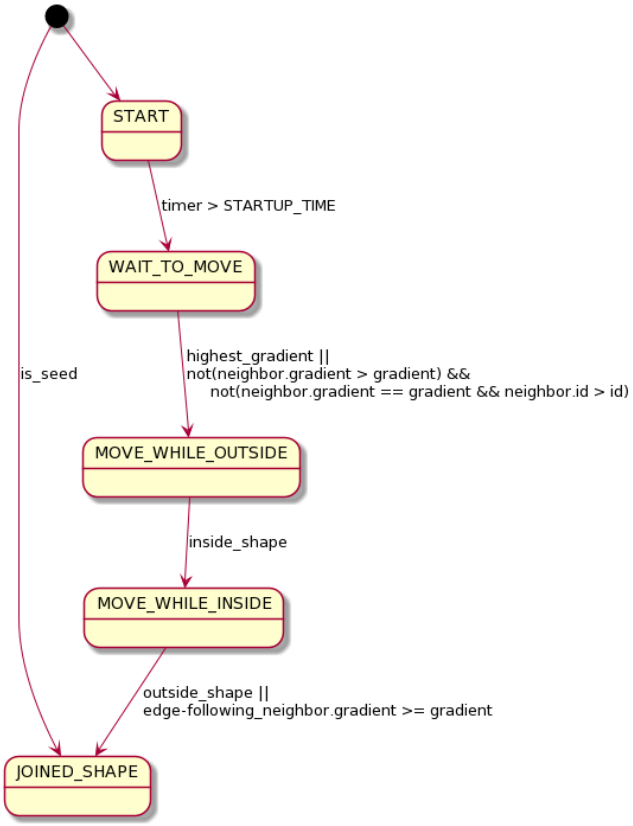


Figure 14: *State diagram of the self-assembly algorithm for a robot, using the three primitives.*

A.3 Simulation run sequence

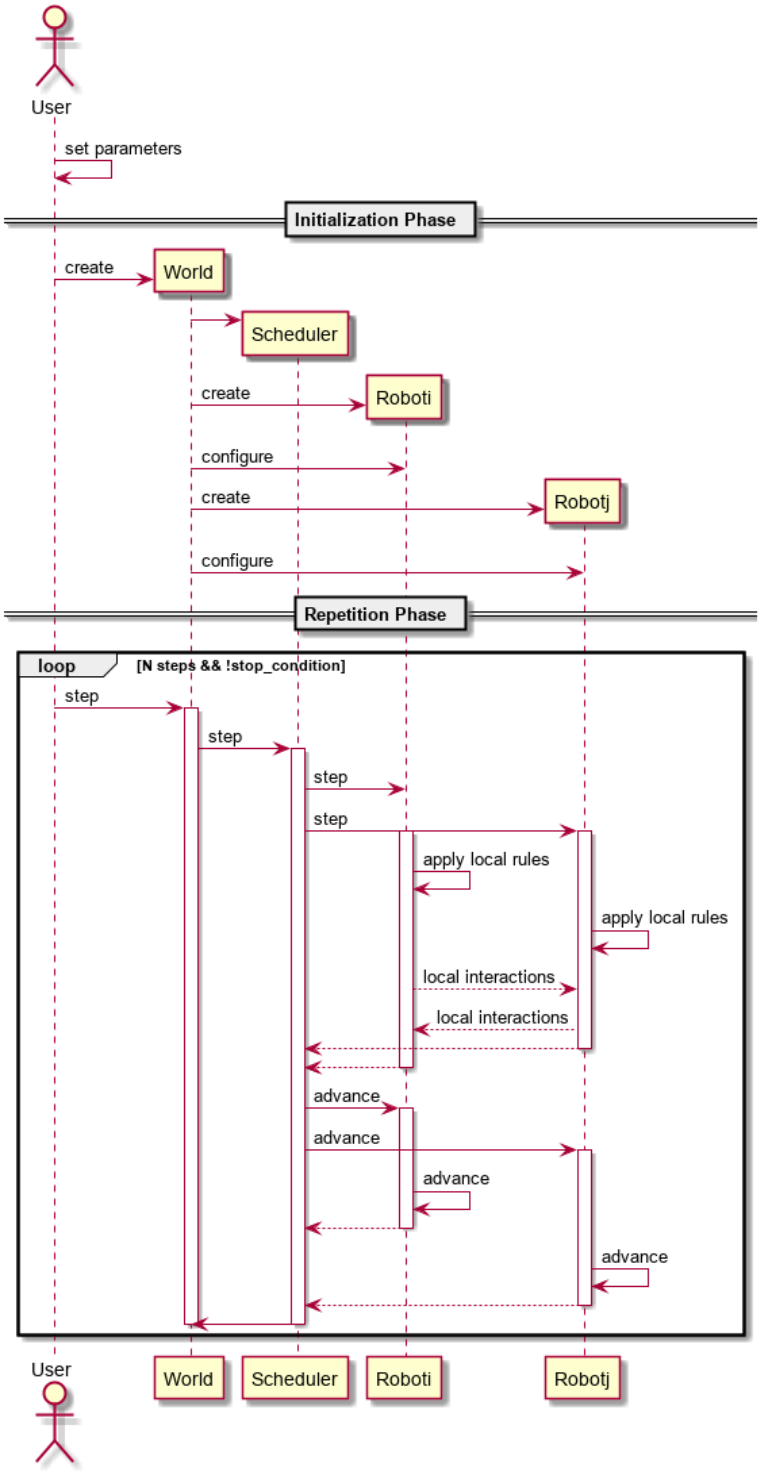
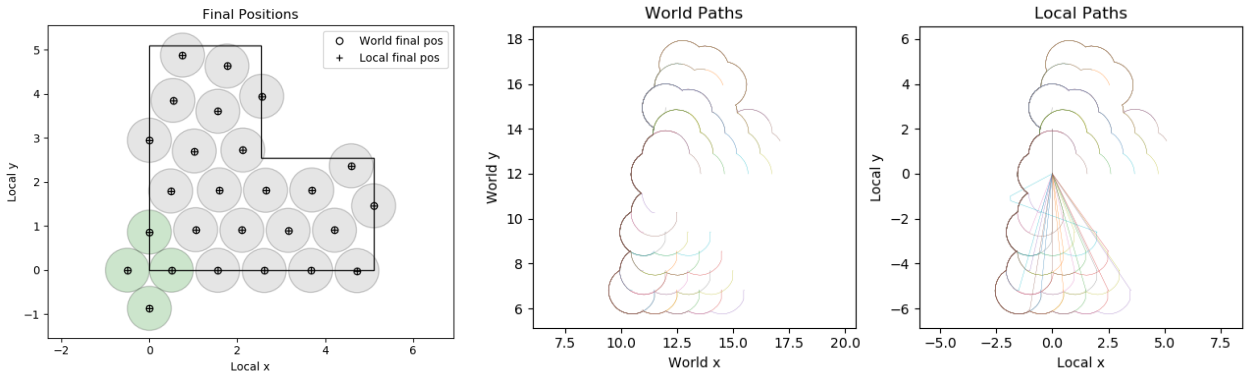


Figure 15: *Sequence of a simulation run. The user is informally mixed with a main process or any higher level function call.*

A.4 Optimal trilateration



(a) End state

(b) Paths

Figure 16: *Reproduction of article results, in [5], using an optimization method to compute local coordinate system. The end state and the world paths are highly similar to ideal situation, and local paths show the math beyond the method: the optimization problem first finds a local optimal symmetric to real robot position, then converge when agents starts moving.*

A.5 Experimental world - initial state

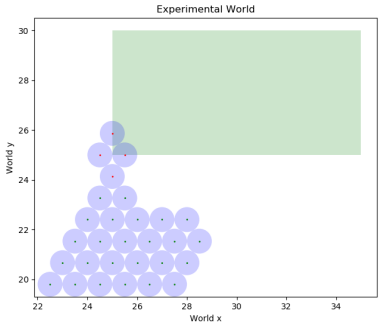


Figure 17: *Experimental world.*

A.6 Graph of the shared path

In the analysis of the experimental results, it may be useful to show the position of the robots on the shared (or common) path – the path that all robots take – per time steps. To build such graph, the following method is used, based on Figure 18.

Let's take as the common path only the portion of the robots' path starting from O, directed towards the seeds. The points S_i, $i \in [1, 5]$, belong to this common path. Each robot joins the common path in point S_i according to its initial position Pj, excepted for the robots with largest initial gradient values, that join this path in O.

Let's call d_j the cumulative distance traveled by robot j, and d_{ji}, the cumulative distance from each robot start until the shared path, along the path this robot follows. This is a line integral of the movement of a robot along its path, between its P_j and S_i.

Let's call d_{iA} the distance along the shared path between each S_i and A, a known reference point, and d_{OA}, the distance along the shared path between O and A. Let's also assume that the origin of the common path is O. The coordinate of S_i on the shared path is then $d_{OA} - d_{iA}$

If l_j designates the position of robot j on the shared path, it follows that $l_j = d_j - d_{ji} + (d_{OA} - d_{iA})$.

Applying this formula to the cumulative distance of each robot gives allows to compare the trajectories timewise, such as in Figure 8.

A.7 Localization initial errors

Complementary material related to the distance uncertainty. Figure 19 shows the localization error after `STARTUP_TIME` for $k = 0.25$ and $k = 0.75$. It remains at a high value.

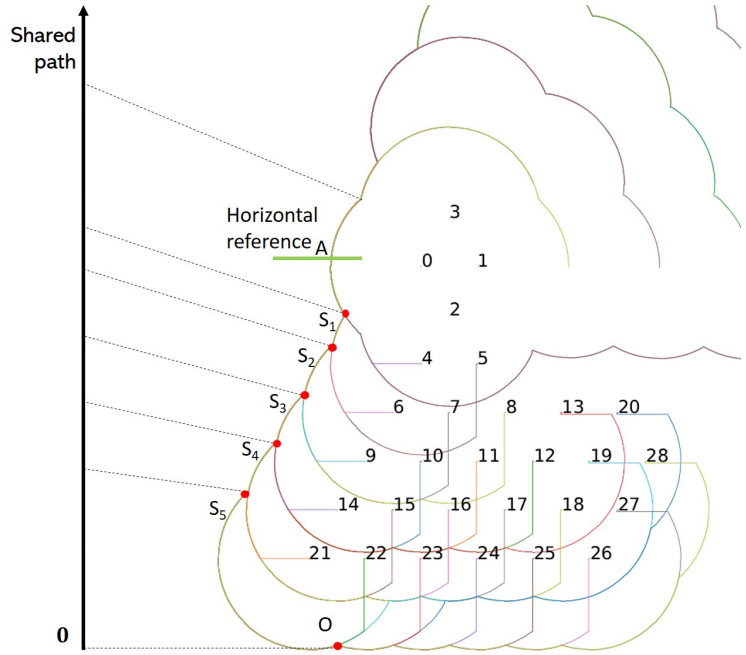


Figure 18: *Robots are indicated with their unique ID, at their starting position. The paths are shown for each robot. Points S_i denote where the robots joints the common path, and an horizontal* `A` *reference is set (as the mean y coordinate of the seeds) in order to have a common reference.*

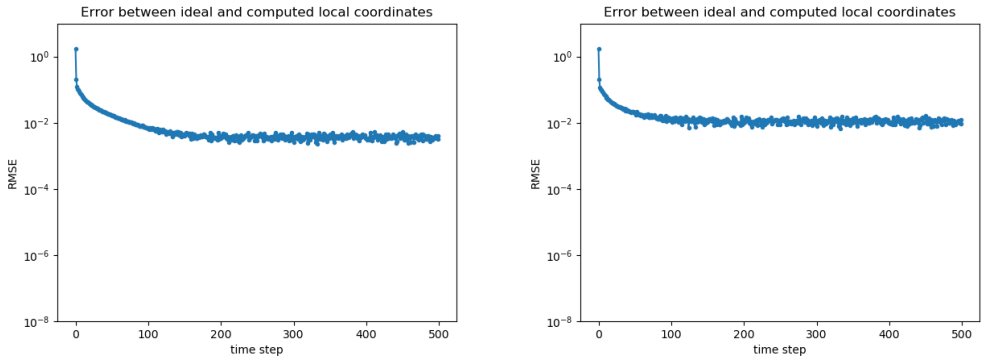


Figure 19: *Localization RMSE. Left,* $k = 0.25$*. Right,* $k = 0.75$