Continuous vs. Discrete Asynchronous Moves: a Certified Approach for Mobile Robots

Thibaut Balabonski, Pierre Courtieu, Robin Pelle, Lionel Rieg, Sébastien Tixeuil, Xavier Urbain

Continuous *vs.* Discrete Asynchronous Moves: a Certified Approach for Mobile Robots

Thibaut Balabonski,[1] Pierre Courtieu,[2] Robin Pelle,[1] Lionel Rieg,[3] Sébastien Tixeuil,[4,5] and Xavier Urbain[6]

[1] LRI, CNRS UMR 8623, Université Paris-Sud, Université Paris-Saclay
[2] CÉDRIC – Conservatoire national des arts et métiers
[3] Yale University, New Haven, CT, USA
[4] Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, FR-75005, France
[5] Institut Universitaire de France
[6] Université Claude Bernard Lyon-1, LIRIS CNRS UMR 5205, Université de Lyon
[7] Contact author: Xavier.Urbain@lri.fr

Abstract Oblivious Mobile Robots have been studied both in continuous Euclidean spaces, and discrete spaces (that is, graphs). However the obtained literature forms distinct sets of results for the two settings.
In this paper, we explore the possibility of transforming results obtained in one model into results for the other one. Our approach focuses on certified results using the COQ proof assistant.

1 Introduction

Networks of mobile robots captured the attention of the distributed computing community, as they promise new application (rescue, exploration, surveillance) in potentially harmful environments. Originally introduced in 1999 by Suzuki and Yamashita [39], the model has been refined since by many authors while growing in popularity (see [27] for a comprehensive textbook). From a theoretical point of view, the interest lies in characterising, for each of these various refinements, the exact conditions under which a particular task can be solved or not.

In the model we consider, all robots are anonymous and operate using the same embedded program through repeated Look-Compute-Move cycles. In each cycle, a robot first "looks" at its environment and obtains a snapshot containing some information about the locations of all robots, expressed in the robot's own self-centred coordinate system, whose scale and orientation might not be consistent with the other robot's coordinate systems (or even with the same robot's coordinate system from a previous cycle). Then the robot "computes" a destination, still in its own coordinate system, based only on the snapshot it just obtained (which means the robot is oblivious, in the sense that its behaviour is independent of the past history of execution). Finally the robot "moves" towards the computed destination.

Different levels of synchronisation between robots have been considered. The weakest [27] (and most realistic) is the asynchronous model (ASYNC), where each robot performs its Look, Compute and Move actions at its own pace, which may not be consistent with that of other robots. The strongest [39] is the fully synchronous model (FSYNC), where all robots perform simultaneously and atomically all of these three steps. An intermediate level [39] is called semi-synchronous (SSYNC), where the computation is organised in rounds and only a subset of the robots are active at any given round; the active robots in a round performing exactly one atomic Look-Compute-Move cycle.

The general model is agnostic to the shape of the space where the robots evolve, which can be the real line, a two dimensional Euclidean space, a discrete space (*a.k.a.* a graph), or even another space with a more intricate topology. To date, two independent lines of research focused on *(i)* continuous Euclidean spaces, and *(ii)* graphs, studying different sets of problems and using distinct algorithmic techniques.

Continuous vs. *discrete spaces* The core problem to solve in the context of mobile robot networks that evolve in bidimensional continuous spaces is *pattern formation*, where robots starting from distinct initial positions have to form a given geometric pattern. Arbitrary patterns can be formed when robots have memory [39,12] or common knowledge [28], otherwise only a subset of patterns can be achieved [41,29,44]. Forming a point as the target pattern is known as *gathering* [39,16,36,2,15], where robots have to meet at a single point in space in finite time, not known beforehand. The problem is generally impossible to solve [39,16,36] unless the setting is fully synchronous [2] or robots are endowed with multiplicity detection [15]. Recently, researchers considered tridimensional Euclidean spaces [43,42,40], where robots must solve *plane formation*, that is, land on a common plane (not determined beforehand) in finite time. It turns out that robots cannot form a plane from most of the semi-regular polyhedra, while they can form a plane from every regular polyhedron (except a regular icosahedron). In the context of robots evolving on graphs, typical problems are *terminating exploration* [14,34,20,22,25,26,21], where robots must explore all nodes of a given graph and then stop moving forever, *exclusive perpetual exploration* [4,19,8,10,9], where robots must explore all nodes of a graph forever without ever colliding, *exclusive searching* [18,19,7], where robots must capture an intruder in the graph without colliding, and *gathering* [19,31,32,33,11], where robots must meet at a given node in finite time, not determined beforehand.

Although some of the studied problems overlap (*e.g.* gathering), the algorithmic techniques that enable solving problems are substantially different. On the one hand, robots evolving in continuous spaces may typically use fractional distance moves to another robot, or non-straight moves in order to make the algorithm progress, two options that are not possible in the discrete model. On the other hand, in the asynchronous continuous setting, a robot may be seen by another robot as it is moving, hence at some arbitrary position between its source and destination point within a cycle, something that is impossible to observe in the discrete setting. Indeed, all aforementioned works for robots on graph consider that their moves are atomic, even in the ASYNC setting, which may seem unrealistic to a practitioner.

Related works Designing and proving mobile robot protocols is notoriously difficult. Formal methods encompass a long-lasting path of research that is meant to overcome errors of human origin. Unsurprisingly, this mechanised approach to protocol correctness was successively used in the context of mobile robots [9,20,6,1,35,16,5,37,2,38,3].

In the discrete setting, model-checking proved useful to find bugs (usually in the ASYNC setting) in existing literature [6,23,24] and formally check the correctness of published algorithms [20,6,37]. Automatic program synthesis [9,35] can be used to obtain automatically algorithms that are "correct-by-design". However, those approaches are limited to small instances with few robots. Generalising to an arbitrary number of robots with similar approaches is doubtful as Sangnier *et al.* [38] proved that safety and reachability problems become undecidable in the parameterised case.

When robots move freely in a continuous bidimensional Euclidean space, to the best of our knowledge the only formal framework available is the Pactole framework.[1] Pactole enabled the use of higher-order logic to certify impossibility results [1,16,3] as well as certifying the correctness of algorithms [17,2], possibly for an arbitrary number of robots (hence in a scalable manner). Pactole was recently extended by Balabonski *et al.* [3] to handle discrete spaces as well as continuous spaces, thanks to its modular design. However, to this paper, Pactole only allowed one to express specifications and proofs with the FSYNC and SSYNC models.

Our contribution In this paper, we explore the possibility of establishing a first bridge between the continuous model and the discrete model in the context of autonomous mobile robots. Our position is that the

[1] http://pactole.lri.fr

continuous model reflects well the physicality of robots evolving in some environment, while the discrete model reflects well the digital nature of autonomous robots, whose sensors and computing capabilities are inherently finite. For this purpose, we consider that robots make continuous, non atomic moves, but only sense in a discrete manner the position of robots. Our approach is certified using the COQ proof assistant and the Pactole framework.

In more details, we first extend the Pactole framework to handle the ASYNC model, preserving its modularity by keeping the evolving space and the robots algorithm both abstract. This permits to retain the same formal framework for both continuous and discrete spaces, and the possibility for mobile robots to be faulty (even possibly malicious *a.k.a. Byzantine*). Then, as an application of the new framework, we formally prove the equivalence between atomic moves in a discrete space (the classical model for robots evolving on graphs) and non-atomic moves in a continuous space *when robot vision sensors are discrete* (that is, robots are only able to see another robot on a node when they perform the Look phase), irrespective of the problem being solved. Our effort consolidates the integration between the model, the problem specification, and its proof that is advocated by the Pactole framework.

Pactole and the formal developments of this work are available at `http://pactole.lri.fr`.

2 The asynchronous Look-Compute-Move model

The complete lack of synchronisation makes reasoning in the ASYNC model particularly error prone. Nevertheless, being the most realistic model, it is widely used in the literature. In this section, we describe how to include the ASYNC model in the Pactole framework.

The formalisation of the Look-Compute-Move model in Pactole for FSYNC and SSYNC has been described in [1,17,2]. We briefly recall what we need here, and emphasise what characterises the ASYNC model.

2.1 Configurations

Locations. The notion of location is a parameter of the Pactole framework and is left abstract in this section, as it depends on the nature of the space in which the robots evolve. In Section 3, we present two different spaces based on graphs, one in which the robots are only located on vertices of the graph, and the other in which the robots can also be located on edges.

Configurations associate a conformation to a robot. In the original Pactole model, robots were mapped to locations only. To reflect in ASYNC the lack of synchronisation and of uniformity of robots' actions, and to add generality to the model, we enrich configurations to map robots' id to a *conformation* (`RobotConf`) consisting of the current location, and information about movement: namely source and target locations. This allows for some robots to move while others are looking or computing.

```
Record Info : Type := { source: Location ; target: Location}.
Record RobotConf := { loc :> Location; robot_info: Info }.
Definition configuration := identifier → RobotConf.
```

We may now consider robots to be in two possible states summarized in Fig. 1: (1) an Idle state, and (2) a Moving state. An idle robot is ready to start a new cycle with a simple Look/Compute action performing the usual Look and Compute phases. Merging these two actions is justified by the fact that the computation is based on the snapshot taken during the Look action only, thus its result cannot be changed by any other event taking place after the Look action. A robot is considered to be moving whenever its current and target locations are different, and becomes idle again when it reaches its target location (thus an idle robot that decides not to move stays idle).

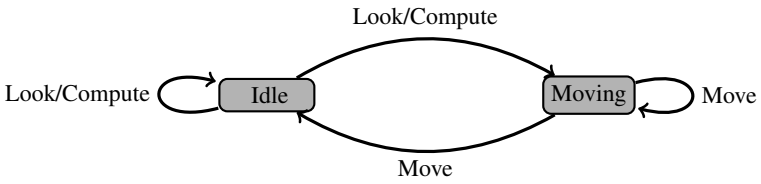

Figure 1. States and actions of the robots

Spectra and Robograms. We call the embedded program the robots use to define their moves a *robogram*. It consists of a function `pgm` that simply returns a destination location when given a perception (*spectrum*) of the environment and the robot's perception of its current location. Spectra inhabit an arbitrary type that is part of the description of the model and contributes to its genericity. Indeed, depending on the robots' capabilities, the perception may not be as accurate as the complete configuration: anonymous robots cannot see names, they may lack detection of multiplicity, frames of reference may not be shared, vision can be limited, etc. The forbidden information is pruned from the configuration, using the function `Spect.from_config` which returns a *spectrum*, the first input of the robogram's `pgm`.

Depending on the space considered, the destination returned may be restricted, for instance to locations that are close enough to the starting location. The pack of theses possible constraints with the declaration of the function `pgm` constitutes what we call a *robogram*.

```
Record robogram := { pgm: Spect.t → Location.t → Location.t; (* + constraints *)}.
```

2.2 ASYNC executions

For all synchronisation models, an execution is a sequence of configurations, each of which is deduced from the previous one, based on the robogram and on a scheduler (called a *demon*) that assigns a change (or not) of conformation to each robot and which is considered as an adversary. To mimic this behaviour, our formal model does not introduce any extra information: execution steps are completely characterised through a transition function by: (1) the current configuration, (2) the demon's choices for the step (a *demonic action*), and (3) the considered protocol. Executions are simply streams of consecutive configurations for that function.

Demonic actions. Formally, each demonic action can request a moving robot to travel further towards its target, or an idle robot to initiate a new move. In each of these cases the demon provides its choices through the action: either the distance travelled along an ongoing move for a Move action, or a frame of reference for the perception of a robot for a Look/Compute action.

```
Inductive action {A} := Move (dist: A)                          (* moving distance *)
                      | LookCompute (Location.t → Iso.t). (* change of frame of ref *)
```

This choice (`Move` or `LookCompute`) is performed by the function `step`. When relevant, demonic actions also relocate Byzantine robots in an arbitrary way (the regular states and actions being *per se* irrelevant for these robots).

We have no control on the choices made by the demon, which is why we call it an adversary. It must nonetheless still make meaningful choices, which we model by the following constraint: only idle robots (that is, robots that are at their target location) may receive an order to look and compute.

```
step_LookCompute : ∀ robot robot_conf ref_change,
  step robot robot_conf = LookCompute ref_change
  → robot_conf.loc = robot_conf.robot_info.target
```

Transition function. One obtains successive configurations by running the robogram according the current demonic action and configuration. This is done by function `round` computing new conformations (`RobotConf`) in a configuration, for each robot identifier `r`, according to a demonic action `da`:

1. If `r` is Byzantine, it is relocated directly by `da` on `LookCompute` actions, and ignores `Move` ones.
2. Else, if `r` carries further its ongoing move (`Move` action), its current location is updated to the location it reached during this move (the way this reached location is computed may depend on the underlying space). In the diagram in Fig. 1, this corresponds to:
   - the Move transition from Moving to Idle when `r` reaches its target location,
   - the Move loop around Moving when `r` does not reach its target location,
   - a Move loop (not shown) around Idle if `r` was already at its target location.
3. Else, a new target location is defined as follows:
   (a) The local frame of reference provided by `da` is used to convert the configuration according to the relevant local point of view,
   (b) The resulting local configuration is transformed into a spectrum using `from_config`,
   (c) The obtained spectrum is passed as a parameter to the robogram, which returns the target location.
   (d) The target location is converted from the local frame to the global one.

   The robot's conformation is updated with the obtained location as new target, and with the current location as new source. In the diagram in Fig. 1, this corresponds to:
   - the Look/Compute transition from Idle to Moving when `r`'s current and target locations are different,
   - the Look/Compute loop around Idle when `r`'s current and target location are equal.

To define a full execution, the function `execute rbg d config` iterates `round` starting from configuration `config`, using robogram `rbg` and demon `d`. Note that a step in an ASYNC execution *does not always imply* a change in the multiset of inhabited locations, as some robots may undergo a change of state only.

3 Application: formal equivalence between discrete and continuous models

In a discrete setting, the simplest possible location type is discrete graphs where robots can only be located on vertices. A robogram takes as parameters a spectrum (perception) and a current location based on robots located on vertices, and returns a vertex as destination location. Travel along an edge is unnoticed as the target vertex is supposed to be reached instantaneously. Particularly simple, this model is convenient for reasoning; it may however be considered as rather artificial.

A more realistic point of view is given by continuous models, which take into account the *continuous* movements of the robots. We nevertheless restrict ourselves to *discrete observations*: each robot is only perceived as being close to some reference point. As a consequence, the space can still be seen as a graph (the graph of the chosen reference points) and the robots are always observed on the vertices. The movement of a robot between two vertices however is now continuous. The corresponding edge is parameterised by a travel ratio called `threshold`, which is compared to the position of a robot along the edge to determine whether the robot is perceived at the source or target vertex. Computed destinations are still vertices.

We propose formalisations for these two models in our formal framework, and prove formally their equivalence in the context of oblivious robots endowed with global strong multiplicity detection.

3.1 Discrete graphs

A formal model for graphs has been provided, and illustrated for SSYNC in [3] to which we refer for further details. Briefly, a graph is defined as a pair `(V, E)` of two sets, the vertices and the edges. Each edge has a source vertex and a target vertex, given by functions `src` and `tgt` respectively. A change of frame of reference is supported by a graph isomorphism (the type of which is written `Iso.t` in the formalisation). We want to extend this model by combining it with the ASYNC aspects presented above.

A graph `Graph` and a set `Names` of robots of some size `N` being given, we provide a model `DGF` in which the ASYNC notions described above are blended.

```
Module DGF (Graph : GraphDef)(N : Size)(Names : Robots(N)).
```

The locations are given by the set `V` of vertices of the graph.

Given a spectrum, a robogram computes as destination a location that must be reachable from (i.e., adjacent to) the current location of the robot. It is thus required that the target is linked through an edge to the current location. This is simply an additional constraint `pgm_range` to the definition of a robogram.

A moving robot travelling instantaneously between its source and target locations, the notion of travel distance degenerates into a Boolean choice: the robot either jumps to its destination, or stays at its current location. Hence the only effort in defining an ASYNC discrete graph in our formal model is to instantiate the parameter `A` in the definition of the demonic action with `bool`.

Further note that for technical reasons we will use, in our case study, a version of these discrete graphs enriched with a field `threshold` that will remain unused in the discrete case. This way both kinds of graphs will inhabit the same datatype, thus easing comparisons.

3.2 Continuous graphs with discrete observations

As in the discrete model, a graph and a set of robots being given, we provide a model `CGF` in which both ASYNC and continuous moves are embedded.

```
Module CGF (Graph : GraphDef)(N : Size)(Names : Robots(N)).
```

The type of locations is richer, and distinguishes two cases: a robot is either on a vertex of the graph (`OnVertex`) or at some position along an edge other than its source or target (`OnEdge`). A position along an edge is given by a *position* ratio p of its length such that $0 < p < 1$ (thus making actual lengths unnecessary in the model). We represent these ratios using arbitrary reals and a continuous bijection between reals and the interval $]0, 1[$.

```
Inductive location := OnVertex (l : Graph.V)
                    | OnEdge (e : Graph.E) (p : R).
```

Discrete observation is understood as a limitation (capability) of the robots' sensors. As such, it is naturally included in the spectrum. For example, with anonymous robots enjoying multiplicity detection, the spectrum of a configuration is based on multisets of locations, however it does not show robots locations with accuracy. Instead, each robot is seen at the "nearest" vertex: a robot located at some position ratio p along an edge is perceived at its source if p is less than the edge's `threshold`, and at its target otherwise. For this, it is sufficient to use the following projection function in the construction of a spectrum from a configuration whenever the position of a robot is looked up.

```
Definition LocC2D (locC : CGF.Location.t) : DGF.Location.t :=
  match locC with CGF.OnVertex l ⇒ l
                | CGF.OnEdge e p ⇒ if Rle_dec p (Graph.threshold e)
                                   then Graph.src e else Graph.tgt e
  end.
```

Thus the type of spectra is exactlty the same as in the discrete model. Note that we also require the returned destination to be a vertex in the additional constraints embedded in the definition of a robogram.

The parameter provided by the demonic action in a `Move` transition is more precise than in the discrete setting: it can be any *moving* ratio m in the interval $[0, 1]$. The transition function then interprets this moving ratio the following way:

- If the robot is on the source vertex of its ongoing move, $m = 0$ means staying there, $m = 1$ means going directly to the destination vertex, and $0 < m < 1$ means going at the corresponding position along the edge between the current vertex and the destination vertex.
- If the robot is at some position p on an edge, then it goes to the position $m + p$ on the same edge. In case $m + p \geq 1$ the robot goes to the target vertex.
- If the robot is already on the destination vertex, then it stays there.

For this model to make sense, the configurations must satisfy the following properties:

- The source and target locations of robots are vertices, with an edge going from the source to the target.
- If a robot is on a vertex, it is either its source or its target vertex.
- If a robot is on an edge, the latter has the same source and target vertices as the robot.

These properties are collected in a `good_conf` property, which is shown to be preserved by the transition function `round`.

```
Lemma good_conf_round: ∀ (config: CGF.Config.t) (rbg: robogram) (da: DGF.demonic_action),
  good_conf config → good_conf (round rbg da config).
```

Hence we restrict our initial configurations to configurations in which these properties hold, and this ensures that the configurations will remain well-formed in any execution.

3.3 Simulation of the discrete model in the continuous model

To prove that the discrete model and the continuous model with discrete observation are equivalent for oblivious robots with strong global multiplicity, we show that any given robogram produces the same executions in both models. We firstly establish in Theorem `graph_equivD2C` that for any "discrete" execution, there is a demon such that this execution can take place in the continuous with discrete observation context.

First remark that any robogram in one of the models can also be read as a robogram of the other model, thanks to the following facts:

- the first parameter of a robogram is a spectrum, and the types of spectra are the same in both models,
- the current position of the robot is always a vertex since the general model assumes that the robogram is applied only for idle robots, which are located on vertices,
- the destination returned by a robogram is a vertex.

Technically the types are different and a translation has to be applied to see a discrete robogram as continuous or a continuous robogram as discrete, but the translation is little more than just a cast.

We define a translation `ConfigD2C` from discrete to continuous configurations, and show that this translation relates any execution step in the discrete model with an execution step of the same robogram in the continuous model. Since for any given underlying graph the locations of the discrete model are a subset of the locations of the continuous model, the translation of the configurations is straightforward.

The property then reads as follows: for any robogram `rbg`, demonic action `da` and configuration c_1 in the discrete model, there is a demonic action `da'` in the continuous model such that the diagram in Fig. 2 is satisfied.

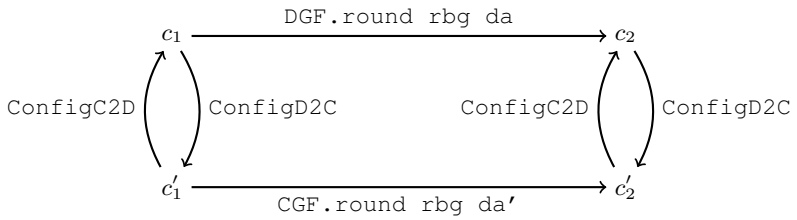

Figure 2. Bisimulation

```
Theorem graph_equivD2C: ∀ (c: DGF.Config.t)(rbg: DGF.robogram)(da: DGF.demonic_action),
 ∃ (da': CGF.demonic_action),
   ConfigD2C (DGF.round rbg da c)  ≡_CGF CGF.round (rbgD2C rbg) da' (ConfigD2C c).
```

The proof of this lemma requires to provide a demonic action `da'` in the continuous model, which is again obtained by quite a simple translation of the discrete action `da`. In particular, the boolean parameter associated to a move action is canonically translated to either `0` or `1`, and the conversion to the local frame of reference needs not be translated (since both models have the same underlying graph). Note that, since demonic actions are associated to constraints (namely `step_LookCompute`), the definition of a new demonic action requires a proof that these constraints are satisfied. Once this witness is provided, the proof amounts to reasoning by cases on the various parameters of the transition function: is the robot Byzantine or not? is the scheduled action a move or a new activation? is the parameter of the move `true` of `false`?

From this we deduce that any execution in the discrete model can be simulated in the continuous model. The reciprocal property, which is more complex, is detailed in the next section.

3.4 Simulation of the continuous model in the discrete model

Configurations in the continuous model can also be translated to configurations in the discrete model. The translation `ConfigC2D` uses the location projection function `LocC2D` already defined in the description of spectra in the continuous model.

This translation allows us to state a second simulation result, similar to the previous one but relating continuous executions steps to discrete ones (that is, reading the diagram in Fig. 2 from bottom to top).

```
Theorem graph_equivC2D: ∀(c': CGF.Config.t)(rbg: CGF.robogram)(da': CGF.demonic_action),
  CGF.good_conf c' →
  ∃da, ConfigC2D (CGF.round rbg da' c') ≡_DGF DGF.round (rbgC2D rbg) da (ConfigC2D c').
```

The definition of the witness `da` is subtler than in the previous lemma. The case where an idle robot is activated and computes a new destination (`LookCompute` action) is straightforward, since again we can use the same isomorphism. The `Move` case however cannot be treated using only the information in the continuous action `da'`: when a continuous demonic action provides a move ratio, we have to translate it into a boolean choice describing whether the move will end in the region of the source vertex or in the region of the target vertex. That is, we have to know whether the movement will pass the threshold or not. This requires knowing not only the demonic action `da'`, but also the configuration `c'`. The full definition then takes the following form:

```
Definition daC2D (daC: CGF.demonic_action) (confC: CGF.Config.t): DGF.demonic_action :=
{| DGF.relocate_byz := fun b ⇒ LocC2D (daC.relocate_byz b);
   DGF.step := fun robot robot_conf ⇒
```

```
(* Here we assume that {robot_conf} is the projection of {confC robot} *)
(* Consider the action given by the continuous demon... *)
match daC.step robot (confC robot) with
  (* a Look/Compute action is preserved, *)
  | CGF.LookCompute ref_change ⇒ DGF.LookCompute ref_change
  (* a Move action requires checking the current location of the robot. *)
  | CGF.Move m ⇒
    match (confC robot).loc with
      (* If the robot is on a vertex, then compare {m} to the threshold
         of the edge to the target vertex {e}. *)
      | CGF.OnVertex _ ⇒
        match (Graph.find_edge robot_conf.robot_info.source
                               robot_conf.robot_info.target) with
          | Some e ⇒ if Rle_dec m (Graph.threshold e) then DGF.Move false
                                                     else DGF.Move true
          | None ⇒ DGF.Move false
        end
      (* If the robot is on an edge do the same after adding the current
         position ratio to {m}. *)
      | CGF.OnEdge e p ⇒ if Rle_dec p (Graph.threshold e)
                         then if Rle_dec (m + p) (Graph.threshold e)
                              then DGF.Move false else DGF.Move true
                         else DGF.Move false
    end
end |}.
```

Again, the proof is by cases on all the parameters of the transition function, which are more numerous than in the previous case since the definition of the demonic action `da'` itself distinguishes many more cases.

These two simulation results, taken together, mean that any execution in any of the two models (discrete or continuous) can be related to an equivalent execution in the other model.

4 Concluding Remarks

Our work established the first formal bridge between two previously distinct models for oblivious mobile robots. From a practical point of view, the formal equivavence we provide between the discrete model and the continous model with discrete sensors sheds new light about what is actually computable in real environments by limited capabilities robots. Furthermore, our work hints at possible new paths for future research:

- The first issue we plan to tackle is that of realistic sensing models for mobile robots. Actual robots endowed with omnidirectional 3D visibility sensors typically use a digital camera with a set of parabolic mirrors [13], which implies that the accuracy of the localization of a robot varies with the distance to its target robot. In our modeling, the `treshold` for a given edge e is the same for all participating robots, while a `treshold` that varies according to the distance of the observing robot to e would be more realistic. Adding this possibility to our framework is not difficult thanks to its modularity, but the equivalence proof is then likely to fail in the extended model.
- Another important long-term open question raised by our work is that of model equivalence beyond oblivious mobile robots. Our approach considers the equivalence of the executions and is hence agnostic of the actual problem being solved; it also enables Byzantine robots. It would be interesting to consider model equivalences with other classical distributed computing models (*e.g.* Problem A in robot model m with f faulty robots is equivalent to problem B in asynchronous shared memory model m' with f' faulty processes). A natural candidate case study would be the Consensus *vs.* Robot Gathering problem [30].

References

1. C. Auger, Z. Bouzid, P. Courtieu, S. Tixeuil, and X. Urbain. Certified Impossibility Results for Byzantine-Tolerant Mobile Robots. In T. Higashino, Y. Katayama, T. Masuzawa, M. Potop-Butucaru, and M. Yamashita, editors, *Stabilization, Safety, and Security of Distributed Systems - 15th International Symposium (SSS 2013)*, volume 8255 of *Lecture Notes in Computer Science*, pages 178–186, Osaka, Japan, Nov. 2013. Springer-Verlag.
2. T. Balabonski, A. Delga, L. Rieg, S. Tixeuil, and X. Urbain. Synchronous Gathering Without Multiplicity Detection: A Certified Algorithm. In B. Bonakdarpour and F. Petit, editors, *Stabilization, Safety, and Security of Distributed Systems - 18th International Symposium, (SSS 2016)*, volume 10083 of *Lecture Notes in Computer Science*, Lyon, France, Nov. 2016. Springer-Verlag.
3. T. Balabonski, R. Pelle, L. Rieg, and S. Tixeuil. A foundational framework for certified impossibility results with mobile robots on graphs. In *Proceedings of International Conference on Distributed Computing and Networking*, Varanasi, India, Jan. 2018.
4. R. Baldoni, F. Bonnet, A. Milani, and M. Raynal. Anonymous graph exploration without collision by mobile robots. *Information Processing Letters*, 109(2):98–103, 2008.
5. B. Bérard, P. Courtieu, L. Millet, M. Potop-Butucaru, L. Rieg, N. Sznajder, S. Tixeuil, and X. Urbain. Formal Methods for Mobile Robots: Current Results and Open Problems. *International Journal of Informatics Society*, 7(3):101–114, 2015. Invited Paper.
6. B. Bérard, P. Lafourcade, L. Millet, M. Potop-Butucaru, Y. Thierry-Mieg, and S. Tixeuil. Formal verification of mobile robot protocols. *Distributed Computing*, 29(6):459–487, 2016.
7. L. Blin, J. Burman, and N. Nisse. Exclusive graph searching. *Algorithmica*, 77(3):942–969, 2017.
8. L. Blin, A. Milani, M. Potop-Butucaru, and S. Tixeuil. Exclusive perpetual ring exploration without chirality. In N. A. Lynch and A. A. Shvartsman, editors, *Distributed Computing, 24th International Symposium (DISC 2010)*, volume 6343 of *Lecture Notes in Computer Science*, pages 312–327, Cambridge, MA, USA, Sept. 2010. Springer-Verlag.
9. F. Bonnet, X. Défago, F. Petit, M. Potop-Butucaru, and S. Tixeuil. Discovering and assessing fine-grained metrics in robot networks protocols. In *33rd IEEE International Symposium on Reliable Distributed Systems Workshops, SRDS Workshops 2014, Nara, Japan, October 6-9, 2014*, pages 50–59. IEEE, 2014.
10. F. Bonnet, A. Milani, M. Potop-Butucaru, and S. Tixeuil. Asynchronous exclusive perpetual grid exploration without sense of direction. In A. F. Anta, G. Lipari, and M. Roy, editors, *Principles of Distributed Systems - 15th International Conference, OPODIS 2011, Toulouse, France, December 13-16, 2011. Proceedings*, volume 7109 of *Lecture Notes in Computer Science*, pages 251–265. Springer-Verlag, 2011.
11. F. Bonnet, M. Potop-Butucaru, and S. Tixeuil. Asynchronous gathering in rings with four robots. In *Ad-hoc, Mobile, and Wireless Networks - 15th International Conference, ADHOC-NOW 2015, Lille, France, 2016*, Lecture Notes in Computer Science. Springer-Verlag, 2016.
12. Z. Bouzid, S. Dolev, M. Potop-Butucaru, and S. Tixeuil. RoboCast: Asynchronous Communication in Robot Networks. In C. Lu, T. Masuzawa, and M. Mosbah, editors, *OPODIS*, volume 6490 of *Lecture Notes in Computer Science*, pages 16–31, Tozeur, Tunisia, Dec. 2010. Springer-Verlag.
13. G. Caron, E. M. Mouaddib, and É. Marchand. 3d model based tracking for omnidirectional vision: A new spherical approach. *Robotics and Autonomous Systems*, 60(8):1056–1068, 2012.
14. J. Chalopin, P. Flocchini, B. Mans, and N. Santoro. Network exploration by silent and oblivious robots. In D. M. Thilikos, editor, *Graph Theoretic Concepts in Computer Science - 36th International Workshop, WG 2010, Zarós, Crete, Greece, June 28-30, 2010 Revised Papers*, volume 6410 of *Lecture Notes in Computer Science*, pages 208–219, 2010.
15. M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Distributed computing by mobile robots: Gathering. *SIAM J. Comput.*, 41(4):829–879, 2012.
16. P. Courtieu, L. Rieg, S. Tixeuil, and X. Urbain. Impossibility of Gathering, a Certification. *Information Processing Letters*, 115:447–452, 2015.
17. P. Courtieu, L. Rieg, S. Tixeuil, and X. Urbain. Certified universal gathering algorithm in $\mathbb{R}^2$ for oblivious mobile robots. In C. Gavoille and D. Ilcinkas, editors, *Distributed Computing - 30th International Symposium, (DISC 2016)*, volume 9888 of *Lecture Notes in Computer Science*, Paris, France, Sept. 2016. Springer-Verlag.
18. G. D'Angelo, A. Navarra, and N. Nisse. A unified approach for gathering and exclusive searching on rings under weak assumptions. *Distributed Computing*, 30(1):17–48, 2017.
19. G. D'Angelo, G. D. Stefano, A. Navarra, N. Nisse, and K. Suchan. Computing on rings by oblivious robots: A unified approach for different tasks. *Algorithmica*, 72(4):1055–1096, 2015.
20. S. Devismes, A. Lamani, F. Petit, P. Raymond, and S. Tixeuil. Optimal Grid Exploration by Asynchronous Oblivious Robots. In A. W. Richa and C. Scheideler, editors, *Stabilization, Safety, and Security of Distributed Systems - 14th International Symposium (SSS 2012)*, volume 7596 of *Lecture Notes in Computer Science*, pages 64–76, Toronto, Canada, Oct. 2012. Springer-Verlag.

21. S. Devismes, A. Lamani, F. Petit, and S. Tixeuil. Optimal torus exploration by oblivious robots. In A. Bouajjani and H. Fauconnier, editors, *Networked Systems - Third International Conference, NETYS 2015, Agadir, Morocco, May 13-15, 2015, Revised Selected Papers*, volume 9466 of *Lecture Notes in Computer Science*, pages 183–199. Springer-Verlag, 2015.
22. S. Devismes, F. Petit, and S. Tixeuil. Optimal probabilistic ring exploration by semi-synchronous oblivious robots. *Theoretical Computer Science*, 498:10–27, 2013.
23. H. T. T. Doan, F. Bonnet, and K. Ogata. Model checking of a mobile robots perpetual exploration algorithm. In S. Liu, Z. Duan, C. Tian, and F. Nagoya, editors, *Structured Object-Oriented Formal Language and Method - 6th International Workshop, SOFL+MSVL 2016, Tokyo, Japan, November 15, 2016, Revised Selected Papers*, volume 10189 of *Lecture Notes in Computer Science*, pages 201–219, 2016.
24. H. T. T. Doan, F. Bonnet, and K. Ogata. Model checking of robot gathering. In J. Aspnes and P. Felber, editors, *Principles of Distributed Systems - 21th International Conference (OPODIS 2017)*, Leibniz International Proceedings in Informatics (LIPIcs), Lisbon, Portugal, Dec. 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
25. P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Remembering without memory: Tree exploration by asynchronous oblivious robots. *Theoretical Computer Science*, 411(14-15):1583–1598, 2010.
26. P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Computing without communicating: Ring exploration by asynchronous oblivious robots. *Algorithmica*, 65(3):562–583, 2013.
27. P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012.
28. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science*, 407(1-3):412–447, 2008.
29. N. Fujinaga, Y. Yamauchi, S. Kijima, and M. Yamashita. Asynchronous pattern formation by anonymous oblivious mobile robots. In M. K. Aguilera, editor, *Distributed Computing - 26th International Symposium, DISC 2012, Salvador, Brazil, October 16-18, 2012. Proceedings*, volume 7611 of *Lecture Notes in Computer Science*, pages 312–325. Springer-Verlag, 2012.
30. T. Izumi, Z. Bouzid, S. Tixeuil, and K. Wada. Brief Announcement: The BG-Simulation for Byzantine Mobile Robots. In D. Peleg, editor, *DISC*, volume 6950 of *Lecture Notes in Computer Science*, pages 330–331. Springer-Verlag, 2011.
31. T. Izumi, T. Izumi, S. Kamei, and F. Ooshita. Mobile robots gathering algorithm with local weak multiplicity in rings. In B. Patt-Shamir and T. Ekim, editors, *Structural Information and Communication Complexity, 17th International Colloquium, SIROCCO 2010, Sirince, Turkey, June 7-11, 2010. Proceedings*, volume 6058 of *Lecture Notes in Computer Science*, pages 101–113. Springer-Verlag, 2010.
32. S. Kamei, A. Lamani, F. Ooshita, and S. Tixeuil. Asynchronous mobile robot gathering from symmetric configurations without global multiplicity detection. In A. Kosowski and M. Yamashita, editors, *Structural Information and Communication Complexity - 18th International Colloquium, SIROCCO 2011, Gdansk, Poland, June 26-29, 2011. Proceedings*, volume 6796 of *Lecture Notes in Computer Science*, pages 150–161. Springer-Verlag, 2011.
33. S. Kamei, A. Lamani, F. Ooshita, and S. Tixeuil. Gathering an even number of robots in an odd ring without global multiplicity detection. In B. Rovan, V. Sassone, and P. Widmayer, editors, *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 of *Lecture Notes in Computer Science*, pages 542–553. Springer-Verlag, 2012.
34. A. Lamani, M. G. Potop-Butucaru, and S. Tixeuil. Optimal deterministic ring exploration with oblivious asynchronous robots. In B. Patt-Shamir and T. Ekim, editors, *Structural Information and Communication Complexity, 17th International Colloquium, SIROCCO 2010, Sirince, Turkey, June 7-11, 2010. Proceedings*, volume 6058 of *Lecture Notes in Computer Science*, pages 183–196. Springer-Verlag, 2010.
35. L. Millet, M. Potop-Butucaru, N. Sznajder, and S. Tixeuil. On the synthesis of mobile robots algorithms: The case of ring gathering. In P. Felber and V. K. Garg, editors, *Stabilization, Safety, and Security of Distributed Systems - 16th International Symposium, (SSS 2014)*, volume 8756 of *Lecture Notes in Computer Science*, pages 237–251, Paderborn, Germany, Sept. 2014. Springer-Verlag.
36. G. Prencipe. Impossibility of gathering by a set of autonomous mobile robots. *Theoretical Computer Science*, 384(2-3):222–231, 2007.
37. S. Rubin, F. Zuleger, A. Murano, and B. Aminof. Verification of asynchronous mobile-robots in partially-known environments. In Q. Chen, P. Torroni, S. Villata, J. Y. Hsu, and A. Omicini, editors, *PRIMA 2015: Principles and Practice of Multi-Agent Systems - 18th International Conference, Bertinoro, Italy, October 26-30, 2015, Proceedings*, volume 9387 of *Lecture Notes in Computer Science*, pages 185–200. Springer-Verlag, 2015.
38. A. Sangnier, N. Sznajder, M. Potop-Butucaru, and S. Tixeuil. Parameterized verification of algorithms for oblivious robots on a ring. In *Formal Methods in Computer Aided Design*, Vienna, Austria, Oct. 2017.
39. I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *SIAM Journal of Computing*, 28(4):1347–1363, 1999.

40. Y. Tomita, Y. Yamauchi, S. Kijima, and M. Yamashita. Plane formation by synchronous mobile robots without chirality. In J. Aspnes, A. Bessani, P. Felber, and J. Leitão, editors, *21st International Conference on Principles of Distributed Systems, OPODIS 2017, Lisbon, Portugal, December 18-20, 2017*, volume 95 of *LIPIcs*, pages 13:1–13:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.
41. M. Yamashita and I. Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Theoretical Computer Science*, 411(26-28):2433–2453, 2010.
42. Y. Yamauchi, T. Uehara, S. Kijima, and M. Yamashita. Plane formation by synchronous mobile robots in the three-dimensional euclidean space. *J. ACM*, 64(3):16:1–16:43, 2017.
43. Y. Yamauchi, T. Uehara, and M. Yamashita. Brief announcement: Pattern formation problem for synchronous mobile robots in the three dimensional euclidean space. In G. Giakkoupis, editor, *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 447–449. ACM, 2016.
44. Y. Yamauchi and M. Yamashita. Pattern formation by mobile robots with limited visibility. In T. Moscibroda and A. A. Rescigno, editors, *Structural Information and Communication Complexity - 20th International Colloquium, SIROCCO 2013, Ischia, Italy, July 1-3, 2013, Revised Selected Papers*, volume 8179 of *Lecture Notes in Computer Science*, pages 201–212. Springer-Verlag, 2013.