Robotica: (2020) volume 38, pp. 48–68. © Cambridge University Press 2019
doi:10.1017/S0263574719000468

N-learning: An Approach for Learning and Teaching Skills in Multirobot Teams

Luís Feliphe S. Costa†, Tiago P. Nascimento‡*, Rosiery da S. Maia¶ and Luiz Marcos G. Gonçalves†

† *Department of Computer Engineering and Automation, Universidade Federal do Rio Grande do Norte, Natal, Brazil. E-mails: lmarcos@dca.ufrn.br, luis.feliphe.pb@gmail.com*
‡ *Department of Computer Systems, Universidade Federal da Paraiba, Paraiba, Brazil*
¶ *Department of Informatics, Universidade do Estado do Rio Grande do Norte Mossoró, Brazil. E-mail: rosiery@gmail.com*

(Accepted March 10, 2019. First published online: April 16, 2019)

SUMMARY
We propose the N-learning practical approach for teaching and learning behaviors in a multirobot system, which is performed through mandatory behavior acquisition based on interactions between the robots at execution time. The proposed methodology can be used to self-program the robots of a team by programming only a single robot with a set of codes that contain behaviors to be transferred and used by other robots as necessary. These codes are implemented in a modular fashion. An advantage of our approach is that when a team of robots is required to perform a specific mission, the set of behaviors required to accomplish that mission can be implemented only once in a single robot or in a distributed fashion. Then, these distributed behaviors are transferred to each of the other robots in the team according to their demand, without the need to reprogram them by hand since the robots in the team can share them autonomously. As an application example, a human critic can teach (or program) only one or a few robots, and these robots are thus able to exchange knowledge with the other team members since they have been preinstalled to run the N-learning system basics. Simulated and real robot experiments are performed to demonstrate the feasibility and validation of our approach.

KEYWORDS: Multirobot system; Robotic leaning; Autonomous behavior; Knowledge acquisition.

1. Introduction

We propose a new approach toward autonomous learning and teaching behaviors in a multirobot system (MRS) at execution time. The key idea of N-learning is to increase the individual capabilities of robots in a team through interactions with other teammates. N-learning allows the robots to learn new behaviors that are represented by a large block of codes at runtime, providing the behavior acquisition through information exchange with teammates. The interactions are messages that are exchanged by the robots at runtime. These messages allow the robots to communicate with other robots to receive behaviors. They are transmitted within the local network. This approach is useful when each group of robots is programmed for a specific task. Eventually, a robot can break, and with N-learning, it is possible to reallocate a robot to work in a different group, with a different, previously unknown behavior. Another problem is raised when new tasks are needed in another environment and a robot must be reallocated to a new group. The new robot in this new group would learn the new task through the other robots. At any moment, we can insert a new robot with new skills in this environment, and through N-learning, we are able to propagate new behaviors at runtime

* Corresponding author. E-mail: tiagopn@ci.ufb.br

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

to the other robots. In these cases, programming the robots with all skills is not feasible because the decision for new objectives occurs when the robots were previously programmed. N-learning allows the team to be more flexible and to receive new behaviors at runtime. An approach with the same name is presented in the work of Mansfield et al.[1] However, their work uses a reinforcement learning algorithm to interact with the environment and addresses the state space explosion caused by additional agents in an MRS.

Therefore, our main contribution presented through the N-learning approach is the possibility of sharing (learning and teaching) new behaviors at execution time between the robots in the MRS. Our approach is useful for MRSs because the behaviors can be freely shared between robots at execution time. The robots do not need to be reprogrammed to acquire new behaviors. Another advantage is that the behaviors are learned according to the need of the robots in a decentralized fashion. Nonetheless, we notice that it is also possible to use an approach where a leader is necessary, for example, a patrolling mission. Learning can occur as a progressive task and increases the global team's behavioral capacity. A second major contribution is the unification of the behavior-based robotics (BBR) formalism for MRSs and the improvement of this formalism by adding the learning and teaching, behavior managing, and knowledge concepts.

Our approach follows the work of Brooks,[2] which introduces BBR that can be understood as an architecture that works with a set of behaviors that can be possibly executed by a robot. A selector component is used to choose the best behavior for a situation. The advantage of this architecture is that it can be modular, and the behaviors can be created to solve a specific problem. These behaviors in N-learning are pieces of code with data about the execution that can be shared throughout the MRS at runtime.

Many definitions exist for *behavior* in the robotics literature. Here, we adopt the definition from Wahde,[3] which introduces *behavior* as a set of actions that must be executed to achieve a goal. For example, the *avoid obstacle* behavior can use the following actions: *stop*, *bypass the obstacle*, and *start moving in a different course*. The nonexistence of a general definition of *behavior* makes it difficult to compare the results from different authors.[3] Additionally, it is common to find different implementations of behaviors. Some works have two behaviors for navigation and obstacle avoidance, while others treat them as a single behavior. The main idea of N-learning is to create an individual artificial entity that can learn behaviors autonomously. Therefore, it is not necessary to transfer the entire set of behaviors (the complete filled brain) to another robot, but rather only some needed behaviors. This approach avoids implementation errors and allows for the scalability of teams.

According to Stone and Veloso,[4] behaviors can be internal or external. Internal behaviors are responsible for changes in the internal structures of the robot, while external behaviors interact directly with the real world generating changes in it. This definition is useful because it allows for the creation of behaviors that are focused on cognitive tasks. Furthermore, our approach also presents a formalism that uses graphs as the underlying structure for computationally representing the behaviors. In these graphs, edges represent dependencies between the behaviors when they need to be executed simultaneously. All the robots have the main structure, named *behavior manager*, that is responsible for choosing which set of behaviors are to be executed. The *behavior manager* is also responsible for managing all sets of behaviors and learning and teaching new behaviors.

The advantage of using N-learning is that it enables robots to learn behaviors at runtime. Our approach allows the robot to perform new tasks that were not previously programmed. N-learning can be useful in two main situations. The first is when many different groups of robots are programmed to perform different tasks. In this case, if some robot stops working and it is necessary to reallocate a robot from another group to supply the broken one, then N-learning can be used to provide the new robot with the knowledge and behaviors of the group. A second situation is when the robots within a group can generate new behaviors at runtime to improve their own behaviors or to create a new one. For example, a robot can find a more efficient way to perform a path or find a path for a new place in the environment. In this second case, N-learning makes it possible to share these new behaviors, increasing the capabilities of the group. In both cases, it is not possible for the robot to start with all available behaviors of the environment. In the first case, the robot could have many useless behaviors if it started with all of them because it is programmed to perform a specific function. In the second situation, the behavior is generated at runtime as a result of some processing or interaction of the robot with the environment.

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

2. Related Works

Learning is important to allow robots to become capable of performing more complex missions. Many works in the literature create models that enable robot learning. Brooks argues that problem-solving behavior, language, expert knowledge and application, and reasoning become simple once the essence of being and reacting is available.[5] In fact, he believes that mobility, acute vision, and the ability to carry out survival-related tasks in a dynamic environment provide a necessary basis for the development of true intelligence.

The attention-driven preconscious perception for socially interactive agents (ASIA) is a model created by Raza et al.[6] that uses consciousness as a basis for learning. The main focus is to improve machine communication with subjective learning and social human interaction. However, ASIA does not focus on learning through robot interactions. Another model present in the literature that is also based on consciousness is the Conscious Attention-based Integrated Model (CONAIM).[7] It uses an attention system for perceiving the environment and comprises sensory memory, feature maps, saliency maps, attention maps, and others. In contrast, our approach uses behaviors to perform tasks, and these behaviors can be combined to solve complex missions. In the same direction as N-learning, CONAIM can provide learning at runtime. However, we propose a different approach in which the acquisition of behaviors with teammates is the main contribution.

Sharing information between robots is a very important issue in MRSs. The related literature focuses on sharing information to guide the execution of tasks in a collaborative and cooperative way.[8–10] Other works in the literature are inspired by nature.[11–13] However, in these works, the robots already have the basic actions preprogrammed. The learning of actions at runtime is proposed by Maia and Gonçalves[14] with the introduction of the intellectual development model (IDEM-MRS). It is based on the theories of Vygotsky[15] and Piaget.[16] The main feature is that IDEM-MRS uses social interactions to provide learning. However, it does not apply IDEM-MRS to robotic systems.[17]

An architecture with three layers is proposed by Rockel et al.[18] The first layer is designed to be a multiagent system (MAS), the last layer is an MRS, and the middle layer is a connection between the first and the last layer. It is the robot system abstraction layer that is responsible for managing the transition between the synchronous interface to the robot hardware and an asynchronous interface provided by the MAS. In their work, complex tasks can be divided into subtasks distributed to multiple agents. In contrast, our approach proposes the division of behaviors into composite behaviors that can be executed together, which also implies the reuse of code by the behaviors. Other architecture layers were proposed by Muller and Pischel.[19] One of the layers proposed by them is the cooperation component, which has to devise a joint plan. This approach can be useful to handle conflicts between teammates. However, this approach does not focus on transferring the behaviors properly to the teammates. The work of Lyons and Hendriks[20] proposes an architecture that allows for planning as an incremental adaptation of a reactive system. The planner component changes the reactor component at runtime to improve the behavior of the robot. In contrast, the behaviors are not freely shared in the MRS.

A group of mobile robots learned without being reprogrammed in the work of Groth and Henrich,[21] where they propose a methodology called programming by demonstration (PbD). In PbD, a demonstration is performed in the symbolic level. Then, the learner uses predefined primitives that move the arm to generate complex tasks. In the experiment, a manipulator learns with another manipulator. A limitation is that the robot must have the predefined primitives. This same problem occurs in another work that uses programming by description.[22] In the work by Huang et al.[22] the robots learn two behaviors: open the door and clean a table. The behaviors are learned by the combination of basic behaviors, not by interactions. Works with PbD are very similar to N-learning. First, they learn new behaviors at runtime. Second, the description can be performed by a teammate. Then, a robot can teach other robots. However, the learned behaviors are combinations of primitive behaviors. In contrast, N-learning has the advantage of providing the learning of these primitives with the teammates.

Flocking and exploration also use learning approaches.[23–26] The learning approach is behavior-based and multiagent. The robots cannot generate new behaviors or learn them with teammates. The controller-based behavior in flocking allows for the global coordination of the team. It is composed of primitive behaviors that can follow a direction or keep away from other robots. The primitive behaviors are not shared with teammates.

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

Finally, the ALLIANCE approach proposed by Parker[27] is an architecture for task achievement. It separates high-level and low-level tasks, where the high-level tasks are used to reach goals. The low-level tasks are related to robot survival in the environment (as obstacle avoidance). The ALLIANCE approach was then improved and called ALLIANCE-ROS[28] in the ALLIANCE architecture. It provides an environment with Python programming frameworks, abundant Robotics Operating System (ROS) modules, fault-tolerance methods for multirobot task allocation, and behavior description and coordination templates. In our work, we also use ROS, but adopt an approach with the possibility of learning tasks at the time of execution.

None of the works in the literature provide learning basic behaviors at runtime between teammates through the exchange of block of codes. The work closest to ours is the MRS that allows for PbD. In the PbD approach, the learned behaviors are combinations of previously programmed behaviors, which means that when a group of robots is already performing their tasks, they cannot learn basic behaviors to perform new tasks. The program by description approach can be useful when robots need to learn observable behaviors. An example is when a robot arm performs a trajectory, and then, other robots observe and perform the same behavior. However, for nonobservable behaviors (works that just process information), it is not possible to apply the PbD approach. In contrast, N-learning provides this capability. Another advantage of N-learning is that, for a robot, it is not necessary to perform the behavior to teach another robot, the behavior is shared through interactions between robots, and it improves the time to share the behaviors.

3. The N-learning Formalism

The N-learning formalism is inspired by one of our previous works.[14] However, here, we improve it to allow for the modeling of BBR. Formally, let us start by considering a set R of available robots in the environment as shown in Eq. (1), where $n \geq 1$ and $n \in \mathbb{N}$. Consider a robot $r \in R$; it has a graph, as given by Eq. (2), to represent the behaviors that it has in the instant of observation t. The set $N(r)$ is not empty and has all nodes of the graph. The set $E(r)$ has unordered pairs that represent the edges between the nodes in $E(r)$.

$$R = \{r_1, r_2, \ldots, r_n\} \tag{1}$$

$$S(r) = \{N(r), E(r)\} \tag{2}$$

Next, to work with *behaviors*, we shall present Definition 1 that is based on the work of Wahde.[3] An example of a behavior is to avoid obstacles that can have the actions: stop, go around the obstacle, and go in a different direction. Our N-learning approach allows for internal and external behaviors as defined by Stone and Veloso.[4] The internal behaviors update the robot internal state, performing some processing or decision-making. In contrast, the external behaviors can affect the real world by using the actuators.

Definition 1 *Behavior***.** A behavior is a set of actions to be performed by a robot that aims to achieve some goal.

All possible behaviors available in the MRS are represented in the set of Eq. (3), where $i \geq 1$ and $i \in \mathbb{N}$, and i will increase only if new behaviors emerge in the MRS. Local interactions can generate these behaviors.[29] The N-learning formalization is found in Definition 2. The learning of behaviors by a teammate r increases the number of elements in $N(r)$.

$$B = \{b_1, b_2, \ldots, b_i\} \tag{3}$$

Definition 2 *Emergent behavior***.** An emergent behavior is a behavior that is created during runtime and is generated by other behavior.

If a robot r has the behaviors $b(r) = b_1, b_2$, a third behavior can be created by the robot in runtime as a result of execution of b_1 or b_2. At the end, robot r will have $b(r) = b_1, b_2, b_3$, where the b_3 behavior can be shared with teammates. An example of this kind of application is when a robotic arm learns some movement at runtime using the PbD (a human performs a movement, and a robot learns to repeat the same movement by observation). After learning the movement, it can create a new behavior for the movement and share it with its teammates. Then, the robots within the group will not need to learn the movement with the program by demonstration approach (watch the movement

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

again); they simply share the behavior. The specific application of Definition 2 and the experiment exemplifying this situation are currently being detailed in another work.

Another punctual definition is further necessary because a universal definition of *composite behavior* does not exist. Actually, some works represent a more complex behavior in a single behavior.[3] Other works present the same behavior through many simple behaviors that are executed simultaneously or serially. In contrast, N-learning can use both approaches due to the relations of dependencies between behaviors. A dependence is established when the behavior uses another behavior. Therefore, based on Definition 1, we introduce the composite behavior in Definition 3.

Definition 3 *Composite Behavior***.** A behavior that uses one or more behaviors as actions is named composite behavior.

The logical dependencies represent the missing requirements for the composite behaviors. Thus, to perform a behavior, the logical dependencies must be satisfied. They are represented by a behavior $b \in B$ as $LD(b) = \{b_1, b_2, \ldots, b_m\}$, where $m \geq 1$, $b \notin LD(b)$ and $m \in \mathbb{N}$. If b does not have dependencies, then $LD(b) = \emptyset$.

As we are working with learning and it assumes that knowledge is acquired by the robots, we introduce Definition 4. Thus, behaviors can be understood as information exchanged between robots; therefore, they can also be understood as knowledge.

Definition 4 *Knowledge***.** Knowledge is the information that is acquired by the robot or that is present in the robot from the start.

The set of nodes $N(r)$ in Eq. (2) represents all behaviors (internal and external) of the robot r. All robots must have the basic structure that is responsible for managing and acquiring the behaviors. We name this structure the *behavior manager*, and it is presented in Definition 5. A robot without this behavior (which is similar to the brain basics) cannot communicate with its teammates through N-learning, making it impossible to learn and to teach. The behavior manager is represented as the b_1 behavior and, then, as mandatory behavior, $\forall r \in R$, $b_1 \in N(r)$. The behavior manager also defines which robot it refers to by b_s^r, where s is a behavior and r is the robot.

Definition 5 *Behavior manager***.** The *behavior manager* is the main structure of N-learning responsible for managing a set of behaviors, acquiring new behaviors and selecting the behaviors to be performed.

Behaviors have physical features that must be available in order for them to be performed. The physical features can specify some hardware that is necessary for the execution of a behavior. These features are represented by a robot r_i, $i = 1, \ldots k$ as $F(r_i) = \{f1, f2, \ldots, f_j\}$, where $j \geq 1$ and $j \in \mathbb{N}$. When the robot r decides to perform a behavior $b \in N(r)$, it needs to know if it has the physical requirements. These requirements of b are in the set of Eq. (4), where f_i, $i = 1, \ldots, k$ is a physical requirement and $k \geq 1$ and $k \in \mathbb{N}$. Then, to perform b: $\forall f \in PR(b), f \in F(r)$.

$$PR(b) = \{f_1, f_2, \ldots, f_k\} \tag{4}$$

All physical features available in the MRS are in the set of Eq. (5), where f_l is an available feature in the MRS and $l \geq 1$ and $k \in \mathbb{N}$. Each physical feature of a robot r_i, $i = 1, \ldots, k$ must be in FE and, then, $F(r_i) \subset FE$.

$$FE = \{f_1, f_2, \ldots, f_l\} \tag{5}$$

Figure 1 shows an example of the behaviors in a robot. To improve comprehension, we include all the behaviors (B) of the MRS in the graph. White nodes represent behaviors that are part of the graph in $N(r_i)$, $i = 1, \ldots, k$. The disconnected nodes are behaviors that can be learned by r_i, $i = 1, \ldots, k$. Blue nodes are behaviors that are known. Red nodes are behaviors in the MRS that are unknown by the robot. It is possible to check the following configuration for the graph:

- $R = \{r_1\}$,
- $N(r_1) = \{b_1, b_2, b_3, b_4, b_5, b_6, b_9\}$,
- $E(r_1) = \{b_1b_2, b_1b_3, b_1b_4, b_1b_5, b_1b_6, b_1b_9, b_2b_3, b_2b_9, b_3b_4, b_4b_5, b_6b_7\}$,
- $B = \{b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, b_{10}\}$,
- $FE = \emptyset$.

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

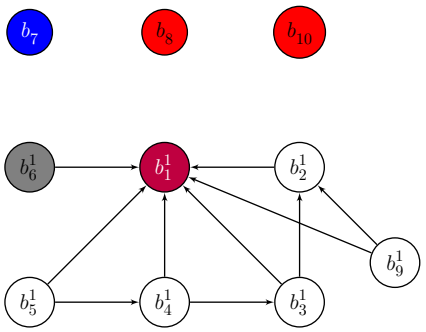


Fig. 1. A graph structure representing the behaviors of r_1. The connected behaviors that are not the behavior manager and composite nodes with missing dependencies are represented as white nodes. The behavior manager (b_1) is represented in purple. The composite behaviors with missing dependencies are in gray. The blue and red behaviors are not part of $N(r_n)$ and represent missing dependencies and unknown behaviors, respectively.

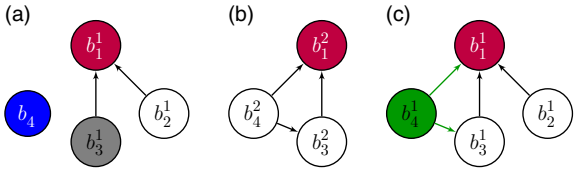


Fig. 2. Graph representation of learning behaviors. Robot r_1 (a) knows that b_4 exists but does not have it. In contrast, robot r_2 (b) has the b_4 behavior, and it is able to teach its teammates. In (c), robot r_1 learns b_4.

The behavior b_5 in Fig. 1 does not have logical dependencies. In contrast, b_2 has four dependencies, represented as $LD(b_2) = \{b_3, b_4, b_5, b_9\}$. In the graphical view, it is possible to note that dependencies are the successors nodes.

A behavior b_n with unsatisfied dependence is part of the set $N(r_n)$, but one or more of dependencies in $LD(b_n)$ are not in $N(r_n)$. Therefore, it cannot be performed by the robot. An example in Fig. 1 is the gray node b_6. It has a dependence to b_7 but $b_7 \notin N(r_n)$; then, robot r_n knows about the existence but does not have it.

The missing dependencies are represented by the set $MD(b_n) = b_1, b_2, \ldots, b_l$, where $l \geq 1$ and $l \in \mathbb{N}$. It represents all requirements of b_n that are not present in the robot, which means that the behavior b_n cannot be performed until $MD(b_n) = \emptyset$

3.1. Learning and teaching

When a behavior $b_n \in MD$, a robot r_n can ask teammates for it. An example is provided in Fig. 2(a). Robot r_1 has the behavior b_4 in MD and needs it to perform b_3. It sends a message to all robots in R asking for b_4. If any robot in R has b_4, it can teach r_1. In Fig. 2(b), the graph of r_2 shows that it has b_4. Then, the robot can teach r_1 based on its decision.

Learning is the process where robot r_n receives a behavior b_n from robot r_m, with $m \neq n$. This operation can be performed if $b_n \in N(r_n)$ and $b_n \notin N(r_m)$. In the case where the robot decides to teach many robots, $\forall r$ in RL, $\kappa \notin N(r)$, where RL is the set of all robots that are learning. Figure 2(c) shows an example of learning. Robot r_1 learns b_4 from r_2. After learning, $b_4 \notin MD(r_n)$ and b_4 become part of $N(r_n)$. In this case, no changes are performed in robot r_2.

3.2. Freedom of action

Aiming to become autonomous entities, the robots are free to change their strategy to define if it is necessary to acquire new behaviors, to teach some behavior to others, or to choose which behaviors to perform (execute behaviors). To allow for this feature, each robot has a function Eq. (6) with three variables that allow it to choose what to do (the current strategy), where $\alpha + \beta + \chi = 1$. The variables α, β, and χ are the probabilities of teaching others, performing a behavior and asking for a behavior, respectively. Note that these simple parameters introduced here can be used in a more elaborate function Ψ to allow for a degree of autonomy to the robots, including their use or a variation of it by

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

some high-level AI algorithm. The high-level AI algorithm is not the focus of this paper, but we are currently deepening this investigation in another work.

$$T = \Psi(\alpha, \beta, \chi) \tag{6}$$

The result of T indicates to the robot what to do. This result also indicates the current strategy of the robot. A robot can adopt a position of teaching, learning, and working, depending on its values. The process for action selection in the N-learning model is presented by Algorithm 1. However, T is an indicator that in some cases can be ignored. For example, when T indicates that a robot must teach but the robot does not have behaviors to teach, the T indicator is then ignored by the code due to this constraint. It can also be useful when the robot is a menace or in another specific situation (e.g., low battery or in danger of falling off the stairs); then, the robot can use a different action to avoid the risk.

Algorithm 1: Algorithm responsible for the selection of the action to be performed by the robot. It can teach, ask for a behavior or perform a behavior.

```
N <= Behaviors of the robot
b <= behavior
MD <= Missing dependencies

teaching <= alpha eq VI
perform  <= beta eq VI
ask      <= chi eq VI


begin
        action = random.randint(1, teaching + perform + ask);
        # teach some behavior
        if action <= teach:
                if (len (N) > 0):
                        b = random.choice (N)
                        send_behavior_to_teammates (b)

        # perform a behavior
        elif action <= (teaching + perform):
                #choose a random behavior
                b = random.choice (N)
                #Check the logical dependencies
                for dep in b.dependencies:
                        if dep not in N:
                                break; #not satisfied
                #check physical features
                for phy_req in b.physical_requirements:
                        if phy_req not in P:
                                break; #not satisfied
                perform_behavior (b)
        else:   #ask for behavior
                b = random.choice (MD)
                ask_teammates (b.name)
end
```

As introduced above, the choice of a specific behavior to be performed can be implemented by using different approaches, but this is not the focus of this work. For example, one could use a different algorithm that analyzes the history of execution with the data from sensors to choose the adequate behavior at runtime.

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

4. N-learning Implementation

An implementation of N-learning is provided for the validation of its formalism. The implementation is based on the ROS[30] using Python as the programming language. ROS is often used in the literature and has some particularities that are useful. It is multirobot, light, and provides libraries and tools for robotics developers. ROS also provides peer-to-peer communication between robots.

We are using an interpreted language because this is the programming paradigm that allows for the changing and reusing of codes, avoiding code recompilation when the robot learns a new behavior. Python has already implemented the communication with ROS. The RosPy library allows for Python applications to become ROS nodes and interact with the ROS environment. We use the Stage system, which is a multirobot simulator provided by ROS, to perform the simulations in this work, and the real robots experiments are performed directly in RosPy.

4.1. Behavior learning

In this subsection, we explain how we use Python language to implement behaviors using N-learning. Before performing an action, the robot needs to look within the list of behaviors that it is able to perform for the specific action. If the behavior is not on the list, the robot asks all teammates for it. The teammates look at their respective lists of behaviors and check if they have it. In the case where the robot has the behavior, it replies by sending the behavior to the robot that requested it. Then, the robot learns the new behavior and can perform it.

The behavior implementation should have two main information items as input: the actions to perform a task and the metadata associated with the behavior. Metadata are crucial for execution because they specify the physical and logical requirements necessary to perform a task. To represent the behavior, we create a class named *knowledge* that has a set of properties that represent the metadata associated with the behavior, including name, topics, description, model, file, dependencies, attributes, and type. These properties are defined in the following.

Property 4.1 *Name***.** The name represents a behavior in a unique way.

Property 4.2 *Topic***.** The topic specifies the necessary topics of the behavior.

The property *name* (4.1) is an identity that robots use to know which behavior they process. The topic's property (4.2) is a set of all ROS topics in the knowledge that needs to be used by a robot. For instance, if this knowledge uses /cmd_vel topic to move a robot, it must have knowledge of the topic's property. Such knowledge is responsible for knowing if a robot has physical features to perform a task because, in ROS, all sensors and actuators have topics to provide communication with ROS nodes. Such knowledge can also be used to distinguish a robot that has a specific physical feature in a system of heterogeneous robots.

Property 4.3 *Description***.** A short description of the behavior and how it works.

Property 4.4 *Model***.** A description of the robotic models in which a behavior works.

Property 4.5 *Dependencies***.** A specification of the necessary behaviors for the execution of a given behavior.

The *description* property (4.3) is a text that describes the behavior and how it works. The *model* property (4.4) is a list of robot models in which the behavior has already been tested. The *dependencies* property (4.5) is a list of all behaviors that are necessary to perform a given behavior. For example, if a robot wants to walk, this task could be divided into two behaviors: "TaskWalk," which is used to move robots to a specific pose in a map, and "WalkToRoom," which is a behavior with a set of coordinates that create a path to some specific room in a map. Therefore, the "WalkToRoom" uses "TaskWalk" as a dependency to move the robots through the coordinates until the robot arrives in that room. Dividing tasks can be a useful approach to avoid replication of code in a given behavior, thus providing smaller, atomic actions.

Property 4.6 *File***.** Specify the path and name of a file for a given behavior.

Property 4.7 *Attributes***.** A set of properties needed to execute a given behavior.

Property 4.8 *Type***.** Specify the type of a given behavior (internal or external).

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

The *file* property (4.6) has the path and the name of a behavior file in the computer. This file includes all the metadata and related instructions regarding how to perform a behavior. We choose a structure in which all of the behaviors are in the folder named "know." This makes it easy to find all of the behaviors of a given robot. The *attributes* property (4.7) is a set with all properties that are needed when a behavior is used by a robot. It is particularly useful when it is necessary to indicate the needed parameters to perform a given task. A common attribute in external behaviors is the name of the robot; without this attribute, all of the robots would publish on the same topic, thus making communication impossible. Finally, the *type* property (4.8) divides the behaviors into two main types, internal and external, as explained above.

In Algorithm 2, it is possible to see the properties in the header of a control behavior. The control is an internal behavior that is used as a class by other behaviors. The properties are displayed as comments so that they have no influence on the behavior execution. The attribution symbol in the comment allows the behavior manager to identify a property and charge it in the memory.

Algorithm 2: Example of header for an N-learning control behavior.

```
#name=control
#model=turtlebot
#description=algorithm that calculates linear and angular velocities to go to (x, y)
#dependencies=none
#file=Control.py
#attributes=none
#type=internal

// implementation
```

In the current version of N-learning, we transfer the behaviors using a string *rostopic*. When the text arrives at the receptor, it is converted into a Python file with the name in the *file* property. Therefore, the learning at execution time is performed using ROS topics. Each behavior manager of each robot has access to an ROS topic named *knowledge*. When a robot decides to share a behavior, it converts the corresponding file into a string. Then, the behaviors send the string, publishing it as the ROS topic. All the receptors of the behavior have an implemented callback that is responsible for receiving the behavior and for choosing to learn or to reject the behavior. Note that when the robot already has the behavior, the learning is unnecessary.

4.2. Behavior manager

The basic behavior of N-learning is the behavior manager, which can also be considered to be composed of several tasks. Each robot has an instance of the behavior manager, and it is unique. In the BBR research field, this behavior manager can be thought of as the component responsible for the selection of behaviors. In contrast, in our system, it only involves the behavior manager. Since the behavior is developed to follow the N-learning approach using the header and the specs, it can be interpreted by any instance of the behavior manager that runs in each robot. To be part of N-learning, the robot must have the behavior manager. The main functions of the behavior manager are listed as follows:

1. Learn all initial behaviors that are available in the *know* folder.
2. Manage all behaviors in memory, including specifying where and how the corresponding knowledge will be recorded in memory. This information includes the file name, the metadata, and its instructions.
3. Choose to learn or not a behavior that is shared by a teammate.
4. Choose to teach a behavior.
5. Ask teammates for a behavior.
6. Choose to perform a task that includes checking if the requirements for that task are available and actually choosing which task to perform.

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

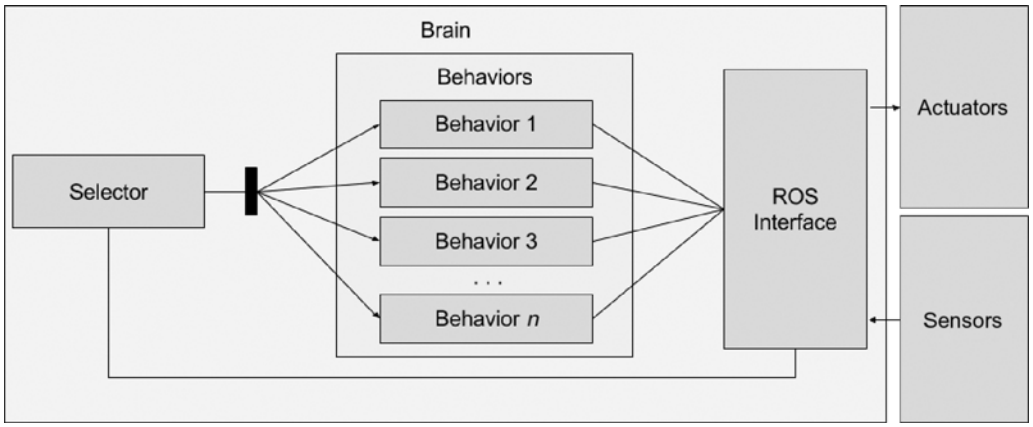


Fig. 3. The organization of the behavior manager in N-learning.

In our implementation, the behavior manager has the following as structures: the *behavior selector*, the *behavior database,* and the *interfaces* with sensors and actuators. Figure 3 shows how these structures are arranged inside the behavior manager.

Choosing to learn or to teach implies making a decision, and this can be performed using an artificial intelligence (AI) algorithm or by some random assertion. The AI algorithm can be, for example, a genetic algorithm where the fitness function can be calculated based on successful executions of behaviors. As this is not the topic of this work, in this current version of N-learning, our experiments use the random function and the approach based on roles. An additional approach that we adopt is using roles for each robot and performing learning guided by them. As mentioned above, one could develop an AI algorithm on top of our proposal for choosing the best behaviors to be performed (the approach that we are currently working on). Therefore, the current strategy of the robot is given by the parameters produced by random functions at this time. Thus, a robot can decide to learn rather than to execute the behaviors or to teach instead of learning. In another situation, when a robot wants to ask for a behavior, it can publish its name to the teammates. In the same way, when a robot receives this requirement, it can answer it or reject it. Their strategy will influence this matter.

5. Results

To verify and validate the proposed N-learning approach for MRSs, we have performed experiments with simulations and real robot scenarios. To initially test the N-learning approach, we used the Stage robotic simulator with an MRS. Next, for real experimentation, we use three Turtlebot robots.

5.1. Simulation results

We choose *Stage* as a multirobot simulator that allows for the construction of environments with robots, sensors, and objects in it.[31] The simulation aims to show that robots without behaviors can use N-learning to learn new behaviors throughout interactions with their teammates. To perform the simulations, we use a notebook Intel Core I5 1.6 GHz with 4 GB of memory. We use five instances of the behavior manager to control the robots, each one for a robot. The ROS master also runs in the same notebook. This simulation is available online (see online Experiment 1[32]). We have also created a map in Stage with five rooms and put five robots in it. Figure 4 shows the map with the robots. The robots are simulated versions of the Turtlebot robot, which is one of the robots that we have in our lab.

The next step is the creation of behaviors that allow the robots to walk in the environment (see Table I). The b_1 behavior is the behavior manager of the robot, responsible for managing the other behaviors. The b_2 behavior is named control, responsible for calculating the parameters for moving the robot to a specific point that is not yet reached in the map. Note that this is an internal behavior, not effectively moving the robot; rather, this behavior receives the localization parameters and answers with the linear and angular velocities that must be published in the robot actuators to move the robot through another behavior.

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

Table I. The behaviors used in simulation.

Name	Description
b_1	The behavior manager of the robot
b_2	Move a robot to a point in map
b_3	Knows in which room robot is in map
b_4	Move robot from room A to room B
b_5	Move robot from room A to room D
b_6	Move robot from room B to room C
b_7	Move robot from room C to room A
b_8	Move robot from room C to room B
b_9	Move robot from room D to room C
b_{10}	Move robot from room D to room E

Table II. The behaviors that each robot starts with in the simulation.

Robot	Behaviors
$r1$	$\{b_1\}$
$r2$	$\{b_1, b_3, b_4, b_5, b_6, b_7, b_{10}\}$
$r3$	$\{b_1, b_4, b_5, b_8, b_9, b_{10}\}$
$r4$	$\{b_1, b_3, b_6, b_7, b_8, b_9, b_{10}\}$
$r5$	$\{b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, b_{10}\}$

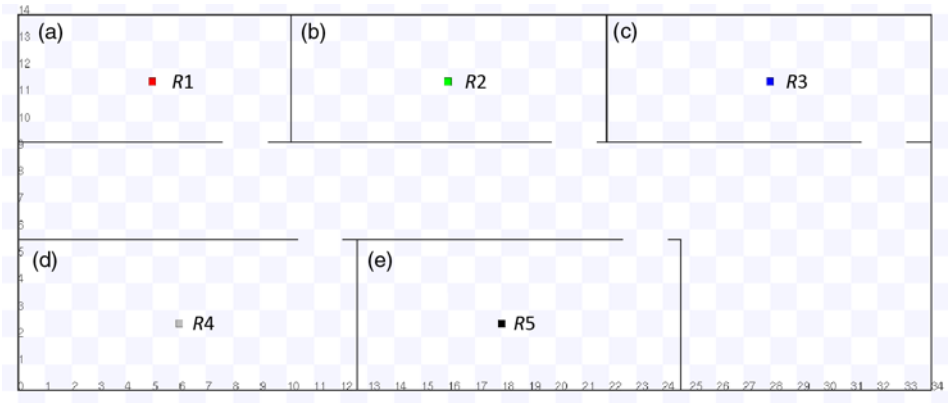


Fig. 4. The environment created with five rooms and five Turtlebots.

The b_3 behavior is also an internal behavior, and it is responsible for receiving the position of the robot and deciding in which room the robot is. Now, a robot can use a motion algorithm to leave a room if it knows the starting room and the location of the entrances; otherwise, it will make a wrong trajectory. These internal behaviors are used simultaneously with the composite behavior. The other behaviors are used to effectively move robots from a determined room to another, using b_2 and b_3.

To evaluate the learning capability in this simulation, only one robot starts with all behaviors (robot r_5). The four other robots in the simulation have only some behaviors (not all of them). The expected result for this simulation is that at the end of the simulation, all robots would be able to learn some behaviors, except the one that already has all behaviors (robot r_5). To make a robot start with no behavior, we remove the behavior file in the directory *know*. Table II shows the behaviors that each robot starts with.

To improve the comprehension of how the behaviors are allocated by a robot, we create graphs that show how the robot's behavior managers start in the simulation. From the abovementioned description, b_1 is a behavior that manages all the other behaviors. Each graph has at least the b_1 node. Nodes that are not connected to b_1 represent behaviors that the robot does not have in its list of behaviors. Figure 5(a) shows the behaviors that robot r_1 starts with in this simulation. It is possible to see that

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

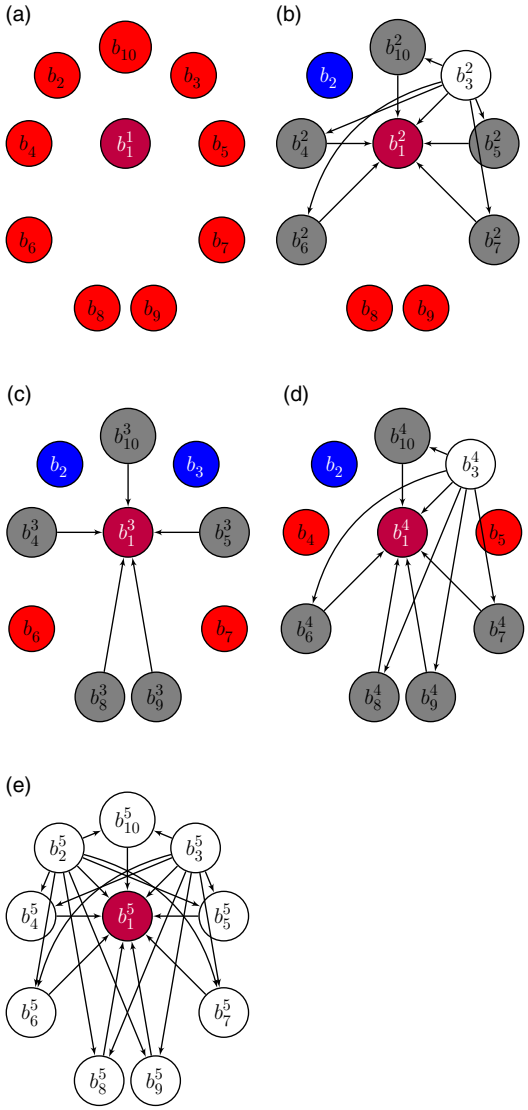


Fig. 5. The representation of behaviors in the simulation using graphs. (a) Robot r_1: it is possible to see that no edges exist between the b_1 and the others. (b) Robot r_2: this robot already has some behaviors, but it is impossible to walk without the control behavior (b_2). (c) Robot r_3: essential behaviors such as b_2 and b_3 are removed. Then, the robot cannot perform tasks to move through the rooms. (d) The behaviors of robot r_4. (e) Robot r_5: in contrast to the others, this robot has all behaviors of the multirobot system.

this robot does not have any behaviors except b_1, which means that it is impossible for r_1 to share some behavior and that it is inexperienced as a new robot that is joining the team. In contrast, it is possible for r_1 to learn any behavior from the MRS if a teammate decides to share a behavior.

Figure 5(b) shows the behavior manager of robot r_2. Notice that the behaviors b_2, b_8, and b_9 are removed. Only the b_3 behavior can be performed because the behaviors involving motion should have the behavior responsible for control (b_2). The robot knows the points that it should walk through to go to any room, but it does not know how to convert this into angular and linear velocities to apply them and effectively move. Despite not knowing b_2, the robot can still teach the tasks that use it, such as b_5 or b_7.

Figure 5(c) shows that robot r_3 starts without four of the behaviors. Note that two of them are crucial to perform the other tasks: the control (b_2) and the localization (b_3). Similar to the case of the r_2 robot, a robot without b_2 cannot walk. Additionally, without b_3, the robot does not know in what room it actually is. If the robot tries to perform b_9 without b_3, it will not know if the robot is in room D; then, the task will not be performed.

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

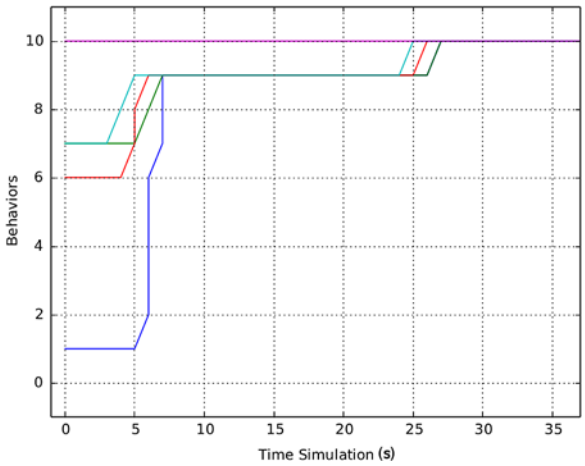


Fig. 6. Evolution of the robots in the simulation. It is possible to see that all the robots learned all the behaviors available in the multirobot system.

The fourth robot tested in the simulation has its behaviors as shown in Fig. 5(d). The robot starts without three of the behaviors, $\{b_2, b_4, b_5\}$. A similar situation has already been described above, where the robots are not capable of executing some behaviors because they do not have their logical requirements. Nonetheless, note that it is possible to use b_3 as an internal behavior, but it will not actuate in the robot or with the MRS.

In contrast to the other robots, robot five (r_5) has all behaviors available in the MRS. It has physical features to perform all tasks, but it cannot because it does not satisfy the conditions to perform them. There is no behavior in the robot that allows it to move from room E to another room. Nonetheless, as this robot knows all behaviors that the others do not, it could be a potential teacher to the teammates.

We calculate the time window spent by the robots to increase their number of behaviors until they encompassed all possible behaviors of the MRS. At the end of the simulation, all robots learn the totality of behaviors of the MRS. The robots could learn and perform the tasks when the requirements for the tasks are satisfied. For example, robots r_1, r_2, r_3, and r_4 start without the walk capacity. However, at the end of the simulation, all of them have this capacity. However, robot r_5 does not have this capacity because it is in room E (there is no behavior for leaving this room).

To check if the learning skill is working in all robots, we use the log of the robots to generate a plot to show how the behaviors increase with time. Figure 6 shows that all robots learn through time, acquiring a total of nine behaviors, which is the totality of existing behaviors in this simulation. A similar graph is used in the work of Lyons and Hendriks.[20] However, they do not use the behaviors versus time as a metric. They use the number of adaptations. The adaptations are changes made by the planner in the reactor of their architecture. These changes improve the behaviors performed by the robot.

In the performed simulation, it is possible to evaluate the learning of the robots. At the start, they are not capable of performing some tasks, but after learning these tasks with their teammates, they could execute the tasks such as control, path planning and localization. This concept has generated an MRS with robots that increase their capacities of performing tasks and that makes the team more cunning. Having all behaviors in the robot is not the purpose of our work, although it is possible (as in this simulation). The advantage of using N-learning is that by making it possible for robots to learn new behaviors at runtime, the robot is allowed to perform new tasks that it was not programmed to perform.

5.2. Real robot Experiment I

In the first real robot experiment, we basically use a notebook, three Turtlebots, and their laptops that are connected through a wireless network (see Fig. 7). In each laptop, we run an ROS bring-up process that is available in the ROS libraries and allows for collecting control information from the sensors in the Turtlebot. In the notebook, we use the ROS master that connects all ROS nodes. Three

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

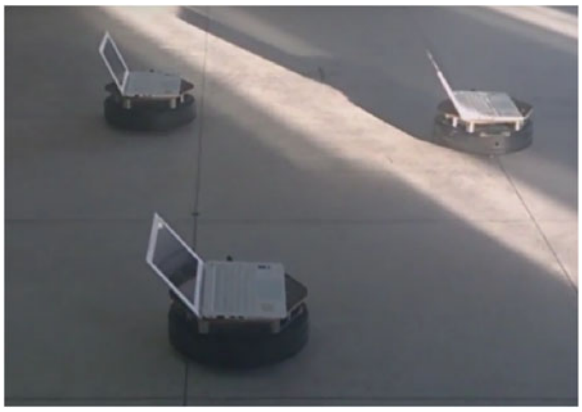


Fig. 7. The robots in Experiment I.

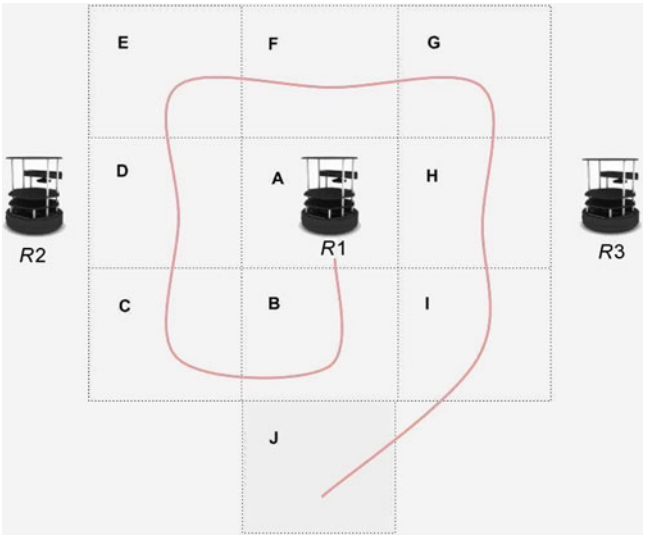


Fig. 8. The trajectory of the leader in Experiment I.

behavior managers are used here, one for each robot. To use many Turtlebots with the same ROS node, we provide a change in the MRS variable named ROS_NAMESPACE for each Turtlebot.

In this experiment, we create tasks to make it possible for the robots to participate in a patrol. Notice this is a different concept of task execution when compared with the above simulation. In this experiment, it is necessary to have a leader; thus, the robots are not fully independent of the others. In the experiment, the leader walks in the environment (Fig. 8), and when a teammate detects that the leader has passed by it, the patrol behavior is activated in order for the leader to go to the formation zone (Fig. 9). When the leader finishes walking in the scenario, all of the robots should walk in the formation starting from the formation zone (Fig. 9).

Figure 8 shows the environment of this experiment with the three robots in a randomly set initial position. The red line is the trajectory that the leader performs, while the teammates detect the leader that has passed them to activate the behavior that makes them go to the formation zone. When the robots are in the formation zone, they perform the trajectory that is presented in Fig. 9.

The description of all behaviors in the experiment is presented in Table III. They are somewhat similar to the simulation. We create a task named b_4 that is responsible for making the leader robot start the formation when it is in the J zone. Otherwise, the task makes the teammates walk to the formation zone near the J zone. Then, they follow the leader starting from the J zone.

As in the simulation, to test if the robots share the behavior in the experiment, we remove some behaviors from each robot. Thereby, we expect that the number of behaviors increases until all

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

Table III. The behaviors used in Experiment I.

Name	Description
b_1	The behavior-manager of the robot
b_2	Move a robot to a point in map
b_3	find which room robot is in map
b_4	Robots start to patrol
b_5	Move robot from room A to room B
b_6	Move robot from room B to room C
b_7	Move robot from room C to room D
b_8	Move robot from room D to room E
b_9	Move robot from room E to room F
b_{10}	Move robot from room F to room G
b_{11}	Move robot from room G to room H
b_{12}	Move robot from room H to room I
b_{13}	Move robot from room I to room J

Table IV. The behaviors that each robot starts with in Experiment I.

Robot	Behaviors
r_1	$\{b_1, b_3, b_6, b_7, b_8, b_9, b_{10}, b_{11}, b_{12}, b_{13}\}$
r_2	$\{b_1, b_2, b_5, b_6, b_7, b_8, b_9, b_{10}, b_{11}, b_{12}, b_{13}\}$
r_3	$\{b_1, b_4, b_5, b_6, b_7, b_8, b_9, b_{10}, b_{11}, b_{12}, b_{13}\}$

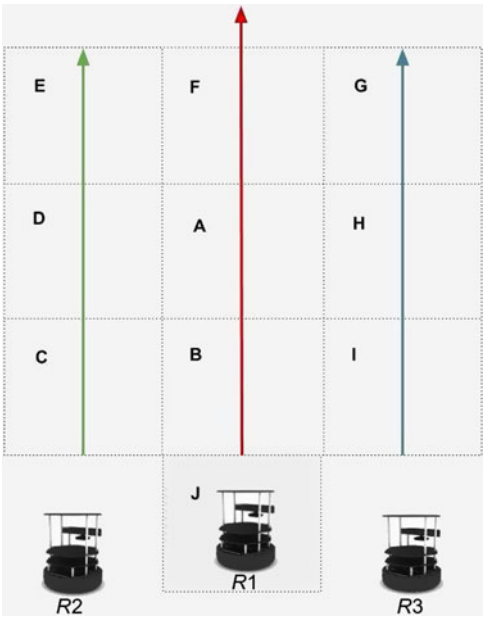


Fig. 9. The trajectory of robots when they are in formation in Experiment I.

robots acquire all available behaviors. Table IV shows the behaviors that each robot starts with in this experiment.

Similarly, we also generate graphs with the behaviors for each robot. They are displayed in Fig. 10(a)–(c). Figure 10(a) shows that robot r_1 does not have the b_2 behavior. This means that without learning this behavior, it is impossible to move around the zones. Nonetheless, after learning this b_2 behavior, it is necessary to learn behavior b_5 because the robot is in the A zone, and there is no other task that enables robot r_1 to move from this zone.

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

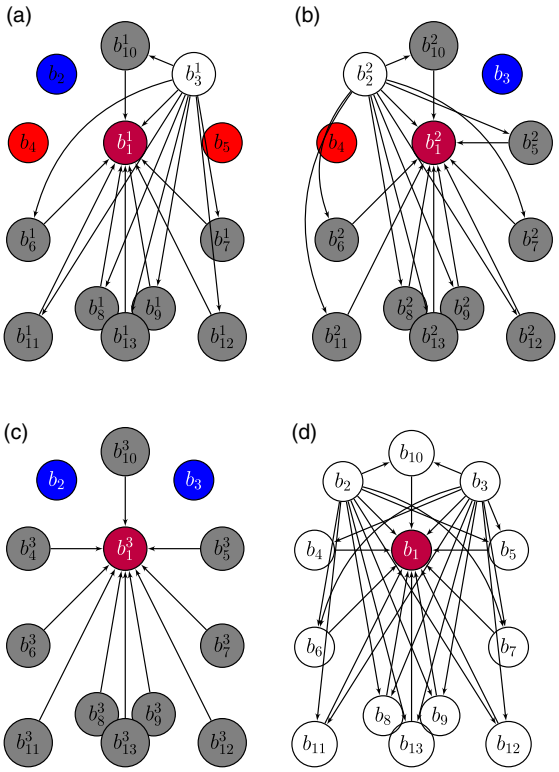


Fig. 10. The representation of behaviors in Experiment I. (a) The graph of initial behaviors of r_1. This robot has no way to walk yet, but when it learns b_2, it will be possible to perform tasks of movement through the zones. (b) Initial knowledge of r_2. It is possible to see the robot does not have the behavior (b_3). (c) The initial knowledge of r_3. (d) The final organization of behaviors to all robots in Experiment I.

Robot r_2 (see Fig. 10(b)) does not have b_3 and b_4 behaviors. Notice that even if it had the b_3 behavior, it could not walk. This happens because the robot is not in a specific zone. Therefore, it must learn b_4 and then wait for the leader to pass by the D zone to walk.

Figure 10(c) shows that robot r_3 does not have b_2 and b_3. In the same way as robot r_2, it cannot perform tasks involving motion because it is not in a zone from the map. When this robot learns b_2, then it will be possible to perform the task of following the leader and then go to the formation zone. This performed experiment is also available as an online resource.[33]

As expected, the robots could perform the patrol mission, and they are able to learn all the required behaviors necessary to perform the tasks involved in this mission. At the end of the experiment, each robot has acquired all available behaviors. For this reason, the graph representation of each robot is the same for all robots, as shown in Fig. 10(d).

The graph in Fig. 11 shows the behaviors of each robot. It is possible to check that the robots start with some behaviors, and then, these behaviors increase over time, which is possible because they learn the behaviors of the teammates.

This experiment shows that is it possible to transfer behaviors between robots at runtime, which is the major contribution of our work. It is useful when the system needs to change the role of some robot, for example, when two robots are performing a patrol outside the building. Due to a low battery, the robot responsible for indoor patrol requests one of the outside robots to perform it. The outside robot learns the behavior and performs the indoor patrol. When the outside robot was programmed, it was not responsible for indoor patrol, which is the reason it does not start with this behavior.

5.3. Real robot Experiment II

To show how N-learning works with different physical features and roles for each robot, we perform a second experiment. Similarly, we use three Turtlebots. However, their physical configuration has different physical features.

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

Table V. The physical aspects of the robots in Experiment II.

Robot	Camera	Laser scan	Kobuki
$r1$	False	True	True
$r2$	False	False	True
$r3$	True	False	True

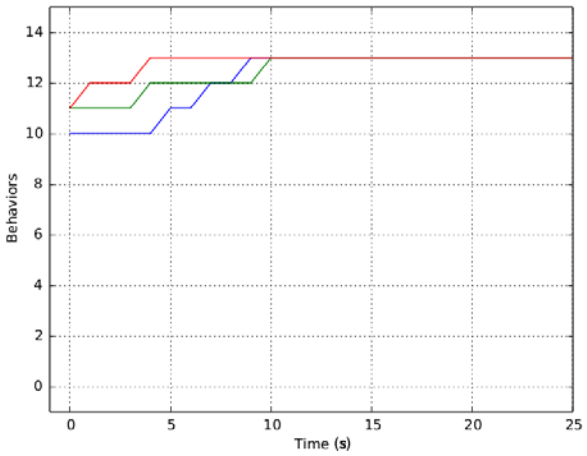


Fig. 11. The behavior acquisition in Experiment I.

The features available for each robot are presented in Table V. Kobuki is a robotic base responsible for the robot movement and sensor data acquisition. For instance, a robot with the Kobuki base can use a behavior for movement. The Kinect device provides many sensors. Here, we use the distance sensor and the camera as physical features. We do not use only one physical feature named Kinect because we can create behaviors that use just one type of sensor, which allows the robot to apply the same behavior in more than one device; for example, if the robot wants to capture an image, then it can do so using a Kinect or a different device, such as a camera, since the physical feature is the same (the topic and the type of data).

In the experiment, all robots have a Kobuki base and a Kinect. However, we chose to divide the Kinect device into two types of sensors: camera and laser scan. A robot with a camera and without a laser scan can take photos of an object and process it, but it cannot measure the distance to the observed object. In contrast, the robot with the laser scan can measure the distance to some object, but it cannot take photos in the environment. A robot without both is equivalent to a robot without Kinect ($r2$).

We define six behaviors for this experiment, and they are presented in Table VI. Each robot has a role that guides the main type of behavior the robot aims to perform. In this experiment, the $r1$ robot has the "leader" role, the $r2$ robot has the "follower" role, and the $r3$ robot has the "scout" role. This approach is useful in comparison to the random choice of behaviors from the first experiment. Each role has an associated behavior that the robot aims to perform. The leader role is associated with the patrol behavior, which allows it to walk in the environment. The follower role aims to perform the follower behavior b_3 (follower). In addition, the scout role wants to perform the behavior b_4 (scout behavior). In this version, the rules do not avoid learning behaviors; they avoid the execution of tasks that are not relevant to their respective role.

The roles will help the robots to perform a patrol. The leader robot ($r1$) must walk by the determined points, searching for a predefined object. In this experiment, the object is never found, but we introduced a false object into the environment. As it is patrolling an environment, it will call for the help of a follower robot ($r2$). If the leader robot ($r1$) finds some obstacle, it will ask the scout robot ($r3$) to capture an image of the obstacle. The request for help occurs because robot $r1$ cannot capture the image because of an obstacle. Its Kinect camera is not installed in the correct position for taking pictures of the environment, and thus, it is unable to use the camera as a physical feature. Although $r3$ identifies and recognizes the box, we must emphasize that image processing is not the focus of our

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

Table VI. The behaviors used in Experiment II.

Name	Description
b_1	The behavior manager
b_2	The patrol behavior
b_3	The control behavior
b_4	Follow the robot leader
b_5	Scout a given position (x, y, z)
b_6	Avoid some obstacle
b_7	Take a photo and process the image

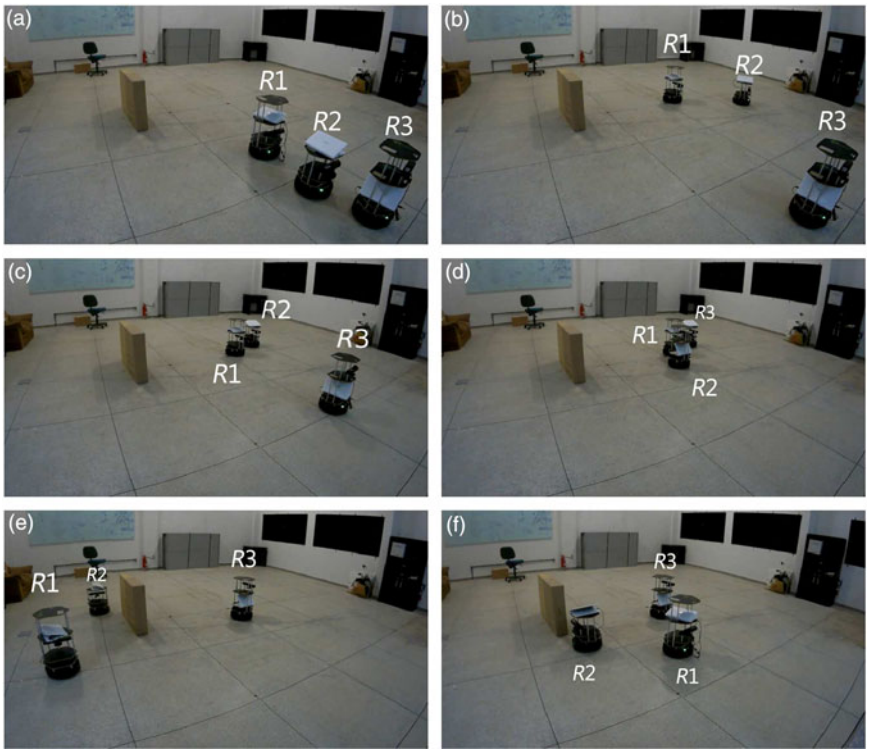


Fig. 12. Experiment II. (a) The robots in the initial configuration of the experiment. (b) The leader detects an obstacle and asks for help, while the follower moves to its final position. (c) The scout robot moves to a position that will enable it to take a photo of the obstacle. (d) The scout takes the photo from the box. (e) The leader and follower robots avoid the box. (f) The leader and follower robots return to the initial position.

work. Finally, the robots in the patrol must walk around the box (avoiding the obstacle) and continue visiting the points until they return to the start position.

We perform the experiment, and it is available as an online resource.[1] Figure 13(a) shows the path of the robots in the experiment, where the "x" in the plot represents the final position of the robot. The initial positions of the robots are presented in Fig. 12(a). The experiment starts with the leader beginning the patrol behavior, followed by the $r2$ robot. When $r1$ detects the obstacle in Fig. 12(b) and does not have a camera to capture the image, it asks robot $r3$ to perform it. The follower robot $r2$ gets into position (Fig. 12(c)), and then, robot $r3$ (Fig. 12(d) takes the photo after also getting into position. After identifying the obstacle (the false object), the leader moves around the box with the follower robot (Fig. 12(e)), and then, it finishes the patrol in the initial position (Fig. 12(f)).

Figure 13(b) presents the behaviors learned by the robots. All robots learned at least one behavior, but in contrast to the previous experiments, they do not learn all available behaviors. This occurred because the learning was directed by the roles of the robots. Table VII shows which behaviors were

[1] https://youtu.be/QEvG40NhC5Q.

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

Table VII. The initial and final behaviors of all robots in Experiment II.

Robot	**Initial behaviors**	**Final behaviors**
r_1	$\{b_1, b_2, b_3, b_4, b_5\}$	$\{b_1, b_2, b_3, b_4, b_5, b_6\}$
r_2	$\{b_1, b_6\}$	$\{b_1, b_3, b_4, b_5, b_6\}$
r_3	$\{b_1, b_7\}$	$\{b_1, b_3, b_5, b_7\}$

Table VIII. The initial and final memory occupation for each robot in Experiment II.

Robot	**Initial memory (KBytes)**	**Final memory(KBytes)**
r_1	940	944
r_2	916	940
r_3	916	932

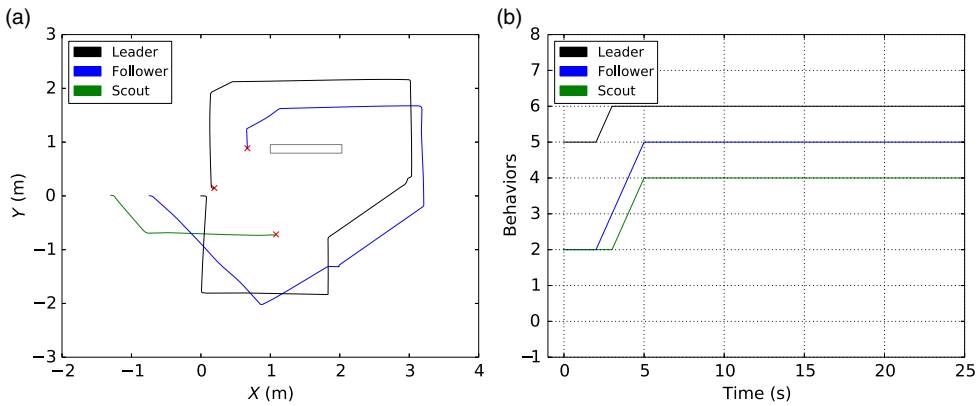


Fig. 13. Plots of Experiment II. (a) The path of the robots in Experiment II. (b) The learned behaviors in Experiment II.

learned by each robot. The leader learns to avoid obstacles and does not learn b_7. The $r2$ robot learns behaviors related to following the leader aligned to his role. The scout robot learns two behaviors, the "scout" and the "control," both related to its role. Control is a logical dependence of scout. At the end of the experiment, the patrol was completed, and the leader returned to the initial position.

In this experiment, we collected the information about the memory usage in the hard disk. Table VIII shows the initial and final memory occupation of the files in the disk. It is possible to see that the increase in memory is not significant in the hard disk (500 GBytes) for the number of behaviors that we use, which demonstrates that the scalability of our approach does not imply an increase in the computational cost.

The robots performed the patrol by sharing behaviors among themselves, guided by their roles. In this experiment, we also adopt physical features, allowing the robots to perform (or not) a behavior based on their features.

6. Conclusion and Future Work

We propose the N-learning approach as a working system that provides autonomous sharing (learning and teaching) of behaviors in an MRS. Our approach is best applied when there is a change in the mission directive. In these situations, new behaviors must be incorporated in the MRS during execution time, which was not necessary during the preprogrammed phase or the initial mission. During execution time, a robot that has a useful behavior can share this behavior with its teammates, the other robots that do not have the behavior. This allows for robot adaptability on task performances in dynamic missions.

Our proposal can also be used as a basis for robot cognition in combination with AI algorithms, augmenting the capability of behavior sharing to provide improvements in the overall intelligence of MRSs. Through simulation and the experiments performed on real robots, we observed that the

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

teammates have learned all available behaviors from the MRS, thus improving their own abilities. We use the composite behavior concept, which allows for the reuse of code of behaviors by the robots. The behavior manager manages all behaviors in a robot and is responsible for checking if the built-in (physical) resources of the robot enable it to perform a task.

A good characteristic of using our approach is that it allows for the use of different versions of control for each robot in a heterogeneous MRS. When using a heterogeneous MRS with different robot models, the robot would choose the model that is a better fit with respect to its robot type and the current behavior needed by using the *model* property. Another specific hardware device can be used as a physical feature, which is useful in a situation with heterogeneous robots that have some hardware in common. For example, if a Pioneer 3-AT and a Turtlebot are both using a Kinect sensor, then they can share behaviors between them using this hardware. However, this is a topic for our future work. Another topic for future work is the use of an AI technique that can improve our current approach, allowing the robots to optimally choose their behaviors in a better way. Such an approach can also provide a strategy to the robots during execution time. For instance, a robot might prefer learning rather than performing tasks during the start of its existence; then, when it reaches a threshold, it can change its strategy, diminishing learning in favor of performing missions and tasks. Importantly, note that the strategy can also be changed to learn according to the execution of tasks. For example, a task could be evaluated to determine whether it is well performed or not, thus making the robot search for other better manners of performing the given task.

Acknowledgments

This study was financed in part by the *Coordenao de Aperfeioamento de Pessoal de Nvel Superior – Brasil* (CAPES) – Finance Code 001, which supported the PhD Grant of Luis F. Costa at the Electrical and Computer Engineering Doctoral Program of the Federal University of Rio Grande do Norte, Brazil.

References

1. M. Mansfield, J. J. Collins, M. Eaton and T. Collins, "N-learning: A Reinforcement Learning Paradigm for Multiagent Systems," **In**: *AI 2005: Advances in Artificial Intelligence* (S. Zhang and R. Jarvis, eds.) (Springer, Berlin, Heidelberg, 2005) pp. 684–694.
2. R. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Rob. Autom.* **2**(1), 14–23 (1986).
3. M. Wahde, Introduction to Autonomous Robots (Copendium, Chalmers University of Technology, Goteborg, Sweden, 2016).
4. P. Stone and M. Veloso, "Task Decomposition and Dynamic Role Assignment for Real - Time Strategic Teamwork," *International Workshop on Agent Theories, Architectures, and Languages* (Springer, Berlin, Heidelberg, 1999) pp. 293–308. http://dx.doi.org/10.1007/3-540-49057-4_19
5. R. A. Brooks, "Intelligence without representation," *Artif. Intell.* **47**(1–3), 139–159 (1991).
6. S. A. Raza, A. Kanwal, M. Rehan, K. A. Khan, M. Aslam and H. M. S. Asif, "Asia: Attention Driven Pre-conscious Perception for Socially Interactive Agents." *2015 International Conference on Information and Communication Technologies (ICICT)*, Rennes, France (2015) pp. 1–8.
7. A. D. S. Simes, E. L. Colombini and C. H. C. Ribeiro, "Conaim: A conscious attention-based integrated model for human-like robots," *IEEE Syst J.* **11**(3), 1296–1307 (2016).
8. G. Corrente, J. Cunha, R. Sequeira and N. Lau, "Cooperative Robotics: Passes in Robotic Soccer," *2013 13th International Conference on Autonomous Robot Systems*, Lisbon, Portugal (2013) pp. 1–6.
9. Y. Gan, X. Dai and Q. Da, "Emulating Manual Welding Process by Two Cooperative Robots," *Proceedings of the 33rd Chinese Control Conference,* Nanjing, China (2014), pp. 8414–8420.
10. P. Singh, R. Tiwari and M. Bhattacharya, "Navigation in Multi robot System Using Cooperative Learning: A Survey," *2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)*, New Delhi, India (2016) pp. 145–150.
11. M. T. Daş, L. C. Dülger and G. S. Daş, "Robotic Applications with Particle Swarm Optimization (PSO)," *2013 International Conference on Control, Decision and Information Technologies (CoDIT)*, Hammamet, Tunisia (2013) pp. 160–165.
12. M. T. Khan, F. Nasir, M. U. Qadir and J. Iqbal, "Artificial Immune System Based Framework for Multi-robot Cooperation," *2014 9th International Conference on Computer Science Education (ICCSE)*, Vancouver, Canada (2014) pp. 50–55.
13. H. I. Lin and C. S. G. Lee, "Neuro-fuzzy-based skill learning for robots," *Robotica* **30**(6), 1013–1027 (2012).
14. R. S. Maia and L. M. G. Gonçalves, "Intellectual development model for multi-robot systems," *J. Intell. Rob. Syst.* **80**(1), 165–187 (2015). http://dx.doi.org/10.1007/s10846-015-0224-0.
15. L. Vygotsky, "Play and its role in the mental development of the child," *Sov. Psychol.* **5**(3), 6–18 (1967).

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press

16. J. Piaget, *The Equilibration of Cognitive Structures: The Central Problem of Intellectual Development* (University of Chicago Press, Chicago, 1985).
17. Z. Yan, N. Jouandeau and A. A. Cherif, "A survey and analysis of multi-robot coordination," *Int J. Adv. Rob. Syst.* **10**(12), 399 (2013). http://dx.doi.org/10.5772/57313.
18. S. Rockel, D. Klimentjew and J. Zhang, "A Multi-robot Platform for Mobile Robots; A Novel Evaluation and Development Approach with Multi-agent Technology," *2012 IEEE Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, Hamburg, Germany (2012) pp. 470–477.
19. J. P. Muller and M. Pischel, "Integrating Agent Interaction into a Planner-Reactor Architecture," *AAAI Workshop on Distributed AI*, Seattle, USA (1994) pp. 229–243.
20. D. Lyons and A. Hendriks, "Planning as incremental adaptation of a reactive system," *Rob. Auton. Syst.* **14**(4), 255–288 (1995). http://www.sciencedirect.com/science/article/pii/092188909400033X.
21. C. Groth and D. Henrich, "Single-Shot Learning and Scheduled Execution of Behaviors for a Robotic Manipulator," *ISR/Robotik 2014; 41st International Symposium on Robotics*, Munich, Germany (2014) pp. 1–6.
22. S. Huang, E. Aertbelien, H. V. Brussel and H. Bruyninckx, "A Behavior-Based Approach for Task Learning on Mobile Manipulators," *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*, Munich, Germany (2010) pp. 1–6.
23. E. Di Mario and A. Martinoli, "Distributed particle swarm optimization for limited-time adaptation with real robots," *Robotica* **32**(2), 193–208 (2014).
24. M. Dorigo and U. Schnepf, "Genetics-based machine learning and behavior-based robotics: a new synthesis," *EEE Trans. Syst. Man Cybern.* **23**(1), 141–154 (1993).
25. F. J. Mendiburu, M. R. A. Morais and A. M. N. Lima, "Behavior Coordination in Multi-robot Systems," *2016 IEEE International Conference on Automatica (ICA-ACCA)*, Curico, Chile (2016) pp. 1–7.
26. D. N. Ray, A. Mandal, S. Majumder and S. Mukhopadhyay, "Human-Like Gradual Multi-agent q-Learning Using the Concept of Behavior-Based Robotics for Autonomous Exploration," *2011 IEEE International Conference on Robotics and Biomimetics*, Karon Beach, Phuket, Thailand (2011) pp. 2725–2732.
27. L. E. Parker, "Alliance: An architecture for fault tolerant multirobot cooperation," *IEEE Trans. Rob. Autom.* **14**(2), 220–240 (1998).
28. M. Li, Z. Cai, X. Yi, Z. Wang, Y. Wang, Y. Zhang and X. Yang, "ALLIANCE-ROS: A Software Architecture on ROS for Fault-Tolerant Cooperative Multi-robot Systems" *Pacific Rim International Conference on Artificial Intelligence* (Springer International Publishing, Cham, 2016) pp. 233–242. http://dx.doi.org/10.1007/978-3-319-42911-3_19.
29. S. Forrest, "Emergent computation: Self-organizing, collective, and cooperative phenomena in natural and artificial computing networks," *Physica D: Nonlinear Phenom.* **42**(1), 1–11 (1990).
30. ROS.org, Ros (2015). http://wiki.ros.org/pt.
31. B. P. Gerkey, R. T. Vaughan and A. Howard, "The Player/Stage Project: Tools for Multi-robot and Distributed Sensor Systems," *Proceedings of the 11th International Conference on Advanced Robotics*, Coimbra, Portugal (2003) pp. 317–323.
32. L. F. Costa, Online resource - first simulated experiment (2017). https://www.dropbox.com/s/i4vk4san1bmlxs2/OnlineResource1.mp4?dl=0.
33. L. F. Costa, Online resource - second experiment in simulation (2017). https://www.dropbox.com/s/l2hlmfk17rqraxu/OnlineResource2.mp4?dl=0.

https://doi.org/10.1017/S0263574719000468 Published online by Cambridge University Press