Received October 7, 2019, accepted October 26, 2019, date of publication November 4, 2019, date of current version November 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2951013

Online Learning and Teaching of Emergent Behaviors in Multi-Robot Teams

LUIS FELIPHE S. COSTA[1], TIAGO P. DO NASCIMENTO[2], (Member, IEEE), AND LUIZ MARCOS G. GONÇALVES[1], (Member, IEEE)
[1]Department of Computer and Automation, Universidade Federal do Rio Grande do Norte (UFRN), Natal 59064-741, Brazil
[2]Systems Engineering and Robotics Lab (LaSER), Department of Computer Systems, Universidade Federal da Paraíba (UFPB), João Pessoa 58051-900, Brazil

Corresponding author: Tiago P. Do Nascimento (tiagopn@ci.ufpb.br)

ABSTRACT In this manuscript, we propose an approach that allows a team of robots to create new (emergent) behaviors at execution time. Basically, we improve the approach called N-Learning used for self-programming of robots in a team, by modifying and extending its functioning structure. The basic capability of behavior sharing is increased by the catching of emergent behaviors at run time. With this, all robots are able not only to share existing knowledge, here represented by blocks of codes containing desired behaviors but also to creating new behaviors as well. Experiments with real robots are presented in order to validate our approach. The experiments demonstrate that after the human-robot interaction with one robot using Program by Demonstration, this robot generates a new behavior at run time and teaches a second robot that performs the same learned behavior through this improved version of the N-learning system.

INDEX TERMS Multirobot leaning, behavior-based robotics, knowledge transference, emergent behavior.

I. INTRODUCTION

Brooks [1] was the first researcher to propose the concept of behavior-based robotics (BBR). This paradigm can be understood as a framework that uses a set of behaviors used by a group of robots. In BBR, a behavior selector chooses the appropriate behavior according to the current situation. The advantage of our approach is that the proposed architecture is modular-based, solving each problem separately by applying one or more behaviors. A behavior can be external when interacting directly with the environment, or internal when resulting in changes in the internal structures of a robot [2]. With this definition, we can create behaviors focusing on cognitive tasks [3].

The first time that the transferring (learning and teaching) of pre-programmed behaviors was proposed was in the work of Costa et al. [4], through the approach called N-learning. In the N-learning approach, behaviors are blocks of code with information about the execution of a specific maneuver or action, which can be shared throughout the multirobot team at execution time. The main objective of the approach is to enable a group of robots to share knowledge through their interactions. The knowledge is represented here as one or more behaviors that enable the robot team to adapt to situations that are not previously taught in its initial programming. According to the authors [4], N-learning allows a robot to share one or more behaviors that it contains to another mate, which in turn eliminates the need for programming all robots within a team with the needed behaviors. These behaviors can be transferred in a distributed fashion using the interaction between the robots in a group at run time. This marks the main advantage of their approach.

Our approach is not like machine learning approaches. As previously stated in [4], the N-Learning approach is a knowledge transference algorithm, not a machine learning algorithm. The process of learning is through the transference of blocks of code, and it does not use update processes to learn a behavior. However, machine learning approaches can be used together with N-Learning.

Hence, in the current work, we propose an extension of that previous work [4], in which the N-learning (now extended) is able to acquire new behaviors in execution time besides only using previously set behaviors as in that work, thus allowing the multirobot system (MRS) to eventually learn new, emergent behaviors. Therefore, we borrow some theory and implementations from the previous works, for the better understanding and implementation of the current work. This new version of N-Learning (extended) is useful for

The associate editor coordinating the review of this manuscript and approving it for publication was Saqib Saeed.

multirobot systems due to the free and autonomous share of knowledge (behaviors) between the agents of a multirobot system. Furthermore, the behaviors can be acquired at execution time, by demand, and in a decentralized manner. Therefore, learning occurs progressively and increases the global behavioral capacity of the robotic team. N-learning, which will be shortly described further, embeds a robot with a main structure called the *behavior manager* that is responsible for choosing which set of behaviors that are to be executed, for managing all sets of behaviors, for sharing behaviors, and for creating new behaviors at execution time. The advantages of our proposed approach over the traditional N-Learning are:

- There is no need to share all behaviors from one robot to another, but rather only the ones needed;
- it is scalable and also avoids implementation errors;
- the extension of N-learning also uses graphs as the underlying structure to represent behaviors computationally and, therefore, is compatible with ROS [5];
- behaviors can be created at execution time.

As an example, it is plausible to have a situation where a heterogeneous group of robots pre-programmed with different sets of behaviors that are used to a specific task, e.g. surveillance and cleaning, need to exchange some of these behaviors to surpass an unexpected problem. Another useful situation is when a robot of a multirobot team breaks. Through the use of N-learning, another robot can assume the task from the broken robot by learning the behaviors to perform the task of the broken robot. In any multirobot systems designed to perform different tasks simultaneously, one would eventually need to pre-program a new robot with a specific set of skills. Through N-learning, we enable the team of robots to propagate new or previously programmed, behaviors at run time to any robot. Thus, the main contributions of this work is twofold: first comes the extension of the N-Learning for allowing the creation of new behaviors at execution time, by one or several robots, and that can be shared between the other robots of the team; and second it comes the coupling of the N-Learning with the Program by Demonstration approach as a manner of teaching new behaviors.

Here, at a low-level, we use the motion control behavior based on nonlinear model predictive control, as developed by Nascimento et al. [6]. Some other approaches as the ones based on control schemes using time delay estimation (TDE) could be used [7]–[9], however, our approach is devoted to the manipulation of multi-mobile-robot systems (MMRS). In TDE, the designed control scheme tries to obtain the estimation of system dynamics, and therefore no system dynamic model information is required. Other schemes as AST and FONTSM [10] are also applied in TDE to ensure good control performance in both reaching and sliding mode phases, not present here. It is known that TDE with AST is more suitable for applications in control of cable-driven manipulators, presenting good robustness and high control precision are obtained in the reaching phase, while the boundary information of the lumped uncertainties will be no longer required [10]. The above control based on nonlinear model predictive control resolves the problem in our work as we have nonholonomic robots that perform in an indoor environment.

Nonetheless, to program all the robots in a multirobot system with all possible behaviors and skills is unfeasible. This is because the decision for new objectives generally occurs at execution time. Much of it comes from the theory of multi-agent systems but can be used unrestrictedly in robotic systems. Panait and Luke [11] presented, in their survey, the definition of multi-agent learning as being an application for learning machines to solve problems involving multiple agents. They present three types of non-collaborative learning approaches: supervised learning, unsupervised learning, and reward-based learning. The implementation of some machine learning methods with several agents involved is not trivial. Supervised learning, for example, has greater difficulty in developing a development of an element that can provide the correct response to the various agents of the team. For this reason, the reward method is preferable. The method of using the reward can be divided into two main parts: reinforcement learning and stochastic search. Reinforcement learning involves the use of reward functions as values of agent attitudes. Whereas a stochastic search involves the use of evolutionary algorithms.

Team learning and concurrent learning are two categories of learning for multi-agent systems [12]. In team learning, a single agent learns and improves interactively the behavior of the whole team. Whereas concurrent learning allows each agent to learn freely by themselves independent of the teammates. There are several domains for the application of collaborative learning. The following are some of the inherent problems of embedded agents presented in the work of Panait and Luke [11]:

- In the plunder problem, items are scattered on a map and the devices are transported to a certain location. Usually, the evaluation of the team is given by the time required to ship all items.
- In the warehouse problem, boxes are scattered on a map with obstacles (two-dimensional). Agents should organize the boxes in a predefined manner by pushing them.
- The robot soccer is a problem that consists of two multirobot teams competing to evaluate which can score more goals. There are several variations with this problem, where most of them can be found in RoboCup.
- In cooperative navigation, a group of robots must walk through a site in a minimal amount of time, without colliding with obstacles or other robots.
- Cooperative target observation: a team of robots must track multiple mobile targets.

In all the above-mentioned problems, the use of N-learning is advantageous. A multirobot system heterogeneity can be used alongside N-learning by sharing different behaviors (or creating new ones) demanded by the raising of unexpected problems [4]. Most of the problems above can use solutions such as the Program by Demonstration (PbD) approach. Program by Demonstration is a technique used to teach robots new behaviors by demonstrating the task instead of reprogramming the robot by machine commands [13], [14]. The technique can be used both to learn how to perform a task done by a human or by a robot. The robot uses the sensors to analyze the task performed by an agent (a robot or a human) in a first step. Then, with the end of the analysis, the robot can replicate the behavior it observed [15]. There are, however, some limitations regarding this technique, for example, the robot can not completely learn new behaviors. The robot can only combine a set of behaviors that it has to replicate the attitude it has been taught. For example, a robot that analyzes how a robotic arm moves, could perform a similar movement, but using the behaviors that it had when it was programmed.

A recent example is the work of Park et al. [14]. The authors address the problem associated with path planning and impedance control that allows robots to manipulate physical interactions delicately. They proposed a path and impedance planning method for impedance control in a robot based on programming by demonstration through telemanipulation using a surface electromyogram. Park et al. [14] considered a task that requires quick and precise adjustment of path and impedance, that is, ball trapping. They implemented a teleoperated robot that can deliver an operator's impedance as well as a position during the ball trapping task to the slave side. The operators were asked to perform demonstrations of ball-trapping tasks using the implemented teleoperated robot, where the slave side is a vertical robot with one degree of freedom. The result showed that using the human demonstration, the robot could learn how to catch a dropped ball without rebounding.

Learning theories are used to explain how humans learn. Some known theorists are Piaget [16] and Vygotsky [17]. Maia and Gonçalves [18] made a connection between these theories and the multirobot systems (MRS) and multi-agent systems (MAS). Their approach was interesting for the N-Learning [4] and is also for our work, and we use here part of what was formalized by them. Among the important concepts that Maia and Gonçalves [18] addressed in their work, we can cite the Proximal Development Zone (ZPD), the Level of Real Development (LRD) and the Level of Potential Development (LPD). The LRD is the knowledge that an individual already possesses, it is his ability to solve problems. The LPD consists of the knowledge that the individual can learn and already has. ZPD is the knowledge that an individual can learn. In the direct application of these concepts onto robotics, LRD is the ability to solve a problem with the knowledge itself autonomously [18]. LPD is the difference between the ability to perform a task autonomously and the ability to perform it. ZPD is what the agent can perform. Maia and Gonçalves [18] also presented the mathematical formulation of these concepts in their work. Starting from the theory of Piaget [16], they explain the assimilation and accommodation in robotics. Assimilation as the acquisition of knowledge in a database, and accommodation as the development of a previously known task. Based on the mathematical formulation of these theories, Maia and Gonçalves [18] created the Intellectual Development Model for Multi-Robot Systems formalism (IDeM-MRS). In it are contained all the representations of robots, the environment, social theories, as well as their connections with an MRS. It was also defined as a set of states (through a state machine) that establishes the ability of a robot to perform a certain task based on its partial knowledge, total knowledge and ability to (physically) solve the task. Finally, the structure of this paper is as follows. The N-learning approach is described in the next section. Further, we describe the proposed extension in detail and show the results obtained with real robot experiments. Finally, we discuss the method and the conclusions are presented, at the end of the manuscript.

II. THE N-LEARNING APPROACH

As aforementioned, the N-learning formalism [4] was inspired by the work of Maia and Gonçalves [18], for allowing modeling of behavior-based robotics. Therefore, we compile here (from those works) the necessary theory for the understanding of the current work. Formally, N-Learning starts with a set R of robots that are available in the environment as depicted in Eq. (1). In the equation, $n \geq 1$ and $n \in \mathbb{N}$. A robot $r \in R$ has an associated graph to represent the behaviors that it has in the instant of observation t, as described by Eq. (2). $N(r)$ is not an empty set and it has all graph nodes. The $E(r)$ set is composed of unordered pairs, which represent the edges between the nodes of the set $E(r)$.

$$R = \{r_1, r_2, \ldots, r_n\} \tag{1}$$
$$S(r) = \{N(r), E(r)\} \tag{2}$$

Thus, a *behavior* can be defined as described in Def. 1.

Definition 1 (Behavior): Set of actions and/or tasks that can be executed by a robot to partly achieve some goal towards some mission objective.

An example is the set of instructions to be executed (as a behavior) for a robot to avoid some kind of obstacle. Notice that in this situation it must use some atomic actions such as stop, forward, and do some tasks such as go around the obstacle, and go in a different direction. As previously stated, the N-learning approach allows the execution of internal and external behaviors [2]. Internal behaviors are those that update the internal state of the robot, allowing it to perform some processing or decision making. With the external behaviors, the robot can actuate on the real world moving objects or so by using actuators.

The set of Eq. (3), where $i \geq 1$ and $i \in \mathbb{N}$, contains all of the behaviors that are available at a given time within the multirobot system. The parameter i should be increased if new behaviors emerge in the system. Notice that these emergent behaviors can be generated by local interactions between the robots or between robots and a human, at execution time. The formalization of emergent behavior inside the N-learning can be found in Def. 2. With this, the number of elements in $N(r)$ is increased by the acquisition of behaviors by a specific teammate r.

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

Definition 2 (Emergent behavior): A behavior that is created or acquired during run time and that is generated by another behavior.

If some robot r has in its set the behaviors $b(r) = b1, b2$, a new behavior can be created at execution time by the robot as a result of execution of $b1$ or $b2$. To this end, the robot r has a new set $b(r) = b1, b2, b3$, where the behavior $b3$ can be shared with its mates. This can be used for example in applications where a robotic arm learns some movement at run time using the programming by demonstration, where a human performs some movement and the robot learns and repeats the same movement, just by observing. After the acquisition of the new movement, a new behavior can be created for it and shared with the other team members. Thus, the other robots of the team will not need to learn the same movement again using the program by demonstration (PbD) approach, this can be done simply by sharing the just acquired behavior.

Another matter that is also necessary to introduce here is a universal definition of *composite behavior*. Some works present it as a more complex behavior in a single behavior [19]. Other authors present it as the same behavior through a set of simpler behaviors that are simultaneous or serially executed. Nonetheless, due to the relations of dependencies between the behaviors, the N-learning paradigm can use both approaches. It is just a matter of establishing a dependency whenever one behavior uses another one. Therefore, from the Definition 1, the composite behavior Definition (3) can be presented as:

Definition 3 (Composite Behavior): A behavior that uses one or more nested behaviors as its actions.

Notice that logical dependencies represent the missing requirements for the composite behaviors. Thus, to perform a behavior, these 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 $LD(b) = \emptyset$.

By working with the sharing of knowledge, and by assuming that knowledge is acquired by the robots, we thus introduce Definition 4. Thus, behaviors can also be understood as the exchange of information between robots. That is to say that behaviors can also be understood as **knowledge**.

Definition 4 (Knowledge): Information that is acquired by a robot or that is present in the robot memory at the execution start.

In Eq. (2), the set of nodes $N(r)$ represents all of the behaviors that are present at a given time in the robot r, internal and external ones. All robots must have some basic structure that is responsible for managing behaviors and acquiring new ones. This structure is named here as the *behavior manager*, and its essence is presented in Definition 5. This higher-level behavior has a functioning that is similar to the basics of a micro-program of an operational system (the brain basics). And a robot without it cannot communicate with its partners through the N-learning, being not possible to learn and/or else to teach. Further, the behavior manager will always be represented, in this work, as the b_1 behavior and also be set as a mandatory behavior $\forall r \in R, b_1 \in N(r)$. The behavior manager defines to which robot it refers by using a b_s^r notation, where s is the behavior and r is the robot id.

Definition 5 (Behavior Manager): The main structure of the N-learning that manages a set of behaviors, eventually acquiring new behaviors and mostly selecting the behaviors to be performed.

Notice that behaviors have some physical features (or resources) that must be available for them to be performed. In the N-Learning (and also in this proposed work) the conditions (and resources) of the robots can be as available. Notice that some behaviors would not be performed if the robots do not have certain resources. Furthermore, with some deficits, the robots would not be allowed to perform certain behaviors, and thus a verification step is implemented to look for the necessary resources and conditions required by each behavior to be run. The physical features can specify some hardware that is necessary for the behavior execution, and are represented for 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 must know if the physical requirements are met. The requirements of b are represented 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: $\forall f \in PR(b), f \in F(r)$.

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

The physical features available in the system are all included in the set presented in Eq. (5), where f_l is an available feature 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 multirobot system 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 $r_i, i = 1, .., k$. Green nodes are behaviors that are known. Orange nodes are behaviors in the multirobot system that are unknown by the robot. It is possible to check the following configuration for the graph:

- $R = \{r_1\}$
- $N(r_1) = \{b_1, b_2, b_3, b_4, b_5, b_6, b_9\}$
- $E(r_1) = \{b_1\ b_2, b_1\ b_3, b_1\ b_4, b_1\ b_5, b_1\ b_6, b_1\ b_9, b_2\ b_3, \ldots b_2\ b_9, b_3\ b_4, b_4\ b_5, b_6\ b_7\}$
- $B = \{b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, b_{10}\}$
- $FE = \emptyset$

The behavior b_5 in Figure 1 does not have logical dependencies. In contrast, b_2 has four dependencies, represented as

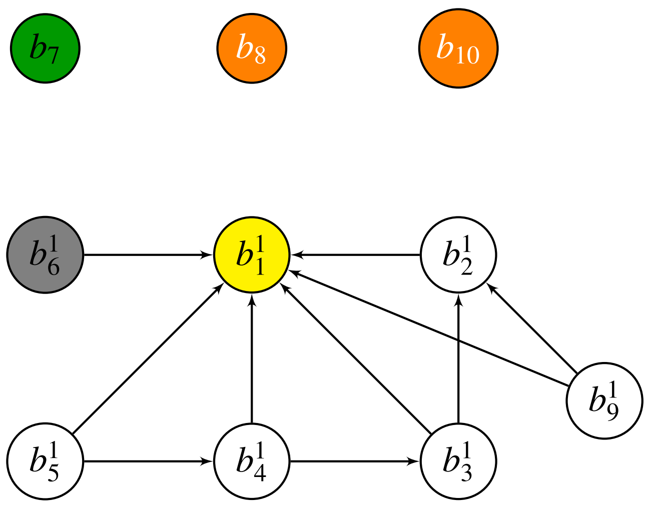


FIGURE 1. **Behaviors of the robot r_1. The composite nodes with missing dependencies are the white ones. The yellow node is the behavior manager (b_1). The gray node is the composite behavior with missing dependencies. The green and orange behaviors are not part of $N(r_n)$ and represent missing dependencies and unknown behaviors, respectively.**

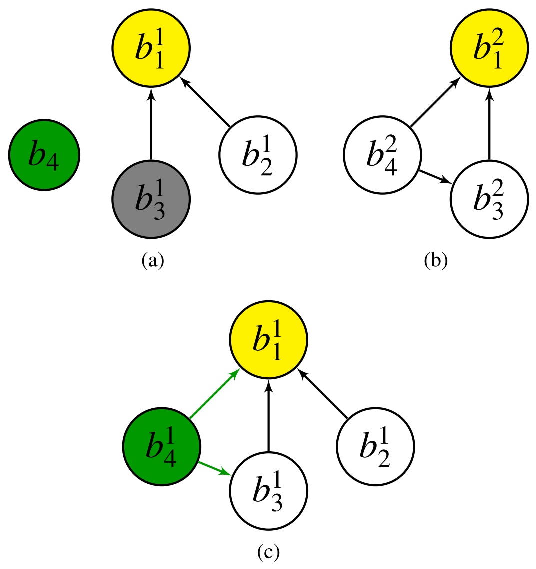


FIGURE 2. **Representation of the behavior sharing between robots. Robot r_1 (a) knows that there is a behavior b_4 that it does not have. Robot r_2 (b) has the behavior b_4, and it is able to share it with its teammates. Finally, r_1 receives the behavior b_4 (c).**

$LD(b_2) = \{b_3, b_4, b_5, b_9\}$. 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 Figure 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 $MD(b_n) = \emptyset$

A. SHARING BEHAVIORS

When an unknown behavior $b_n \in MD$ is necessary, a robot r_n can ask teammates for it. An example is provided in Figure 2a. The 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 Figure 2b, 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 a robot r_n receives a behavior b_n from the 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 2c shows an example of behaviors learning. The 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 the robot r_2.

B. FREEWILL

Freewill means the capacity of choosing. In our approach, choosing allows the robot to change the current strategy used. It, in turn, allows the robot to define the behaviors it must acquire, to transfer some of the behaviors to its mates, or to select the behaviors it will execute. This feature is only possible because each robot uses a function Γ as depicted bellow to measure the usability of the current strategy.

$$F = \Gamma(\theta, \phi, \eta) \tag{6}$$

This function uses three variables in which $\theta + \phi + \eta = 1$. The θ, ϕ, and η variables represent the possibilities of teaching, executing or requesting a behavior, respectively. These parameters are then used within a complex function Γ that enables the robots to increase their autonomy. This function also allows the use of high-level artificial intelligence (AI) algorithms. The AI algorithm is not the focus of this work but it is the topic of another research that is currently being performed by our group.

Therefore, the value of F gives the robot the direction of what action it must take, resulting in which strategy it must choose (i.e. sharing or receiving knowledge, executing the behavior, etc.). The use of F is advantageous in situations that the robot must change its course of action. For example when it reaches a depleted battery status or when it becomes having dangerous behavior due to malfunction actuators. Nevertheless, F is not always taking into account. The extended N-learning constrains its use in some cases. An example is when F informs the robot it must share a behavior that it does not have.

The introduction of Freewill is a start point in this work. We use a simple function Γ but it can be improved with more complex approaches. However, we are not focusing on this function here.

III. EMERGENT, GENERATOR AND MODEL BEHAVIORS

From Definition 2, one can notice that emerging behaviors are those that arise at run time. Mainly, these behaviors are due to the combination of behaviors in the robot. An example occurs with the Programming by Demonstration (PbD) technique, in which the robot observes the behavior of a human (or another robot) and tries to replicate such behavior. An example would be a robot with a robotic arm that observes the trajectory made by the arm of a human. Then, it will have a set of points to replicate the movement in the robotic arm. Using primitive behaviors that are already previously programmed, the robot can perform a combination of such behaviors to perform the same trajectory using a single control behavior to move the arm through the points. The previously proposed N-learning alone cannot act in this kind of scenario, because in this case knowledge is not passed from one robot to another [4], being this a drawback.

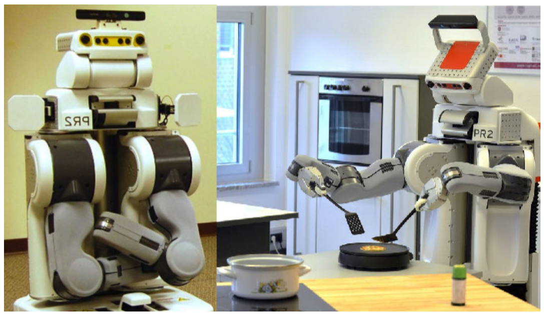
In the second scenario, we can use two robotic arms. The first robot observes the trajectory of the second robot and then replicates the movement. The Program by Demonstration would also work and the second robot could learn how to perform the movement. In this scenario, N-learning is also applicable. However, a possible question is whether it would be simpler to transfer to the second robot only one vector of coordinates, or the whole behavior. In a situation where the robotic arm performs the trajectory based on constraints, i.e. to avoid an obstacle during the movement, PbD would not be enough. In these cases, N-learning is a better alternative since it allows the transfer of behavior along with non-observable (internal) behaviors, including the conditions for changing the trajectory.

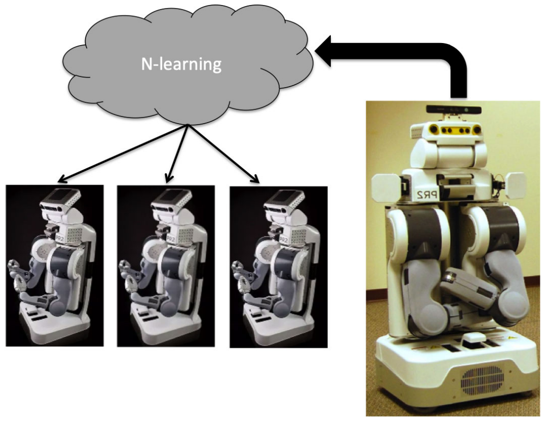
A third possible scenario occurs with a human teaching a robot (Fig. 3a). The robot would then share the knowledge gained with its teammates. The more robots using Program by Demonstration (PbD), the more time is required for teaching to all team members. By using PbD, the robot must first observe individually the teacher (a human or robot) acting (Fig. 3b). This repeated observation behavior causes a loss of time. In contrast, N-Learning is very useful in this situation if combined with Program by Demonstration. This combination is possible and it can create a behavior that is based on PbD and that uses N-Learning to disseminate the knowledge learned with the human to the whole team (Fig. 3c). Let us now create two important definitions.

Definition 6 (Behavior Generator): A behavior generator is a behavior responsible for generating new behaviors in the environment.

The Behavior Generator has a list of instructions that dictate which are the necessary tasks and in which manner the tasks must be arranged for the construction of new behaviors. A more complex Behavior Generator that is capable of taking into account which sensors are being used in a robot to learn a specific task through PbD will be studied in future works. It will be based on these sensors to dictate the order in which the tasks must be arranged in the creation of new behaviors.

(a) Robot Learns through PbD


(b) Robot teaches another robot through PbD



(c) Robot teaches several others through N-learning

FIGURE 3. Example of the difference between the PbD and the N-learning approaches.

Definition 7 (Model Behavior): Model behavior is the behavior used as the basis for generating new behaviors.

The model behavior is made specifically for each mission. Its specificity is due to the singleness of each mission which determines the uniqueness of each required new behavior. In turn, each new behavior is the set of pre-programmed tasks (functions) that are used by the Behavior Generator for the creation of new behaviors. One branch of future studies will also generalize the Model Behavior. In our case, the Model Behavior is based on the Surveillance Mission.

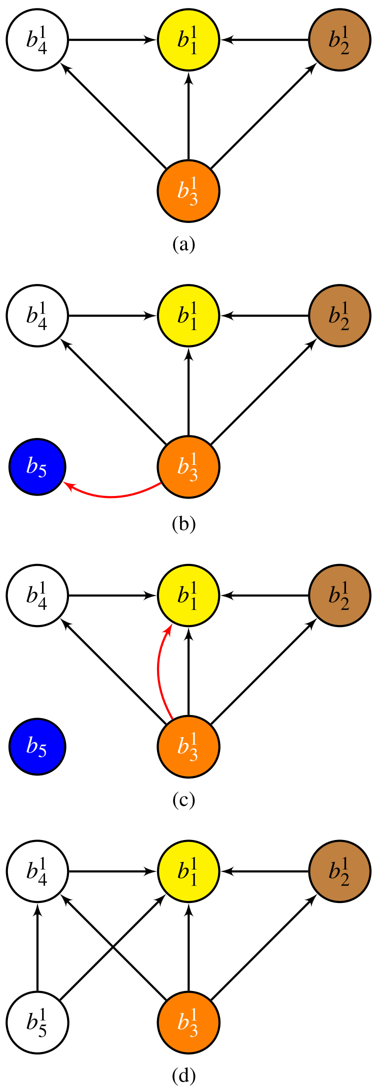


FIGURE 4. **These figures show the process of generating emerging behaviors. The model node is represented in brown attached to the generating behavior in orange. The blue node represents the behavior generated at run time. In (a) we have the initial configuration, then the behavior b_3 uses the model behavior b_2 with the information acquired to generate the behavior b_5 in the image (b). The behavior manager is notified about the generation of the pop-up behavior so that it can update the list of behaviors (image (c)). The emerging behavior is loaded, being shared or executed (image (d)).**

These two additional concepts that we propose here, are needed to better understand the creation of emerging behaviors. The first concept is generating a new behavior and is displayed in the definition 6. The generating behavior is responsible for getting the run time information that will be useful to the new behavior. The information can be a trajectory, a matrix representing a trained neural network, or any other useful information. Then the generator behavior would combine the information with a model behavior (definition 7) creating a new behavior. A graph representation of how an emergent behavior originates is shown in Figure 4. In Figure 4a the behavior b_3 is a generating behavior and has the model b_2 as a logical requirement. In Figure 4b the behavior b_5 is generated (red arrow) from the model and the data obtained at run time by b_3. Then, b_3 triggers the behaviors manager (red arrow) to reload the behavior files (Figure 4c), allowing the metadata from b_5 to be loaded. Once loaded, b_5 can be freely shared with teammates and executed (Figure 4d), provided that its dependencies are satisfied.

IV. EXPERIMENTS AND RESULTS

In this section, we present experiments with real robots that were performed at the Latin American Robotics Symposium and Competitions,[1] which took place in João Pessoa, Brazil, between 6th and 10th of November 2018 and other posterior experiments for showing the learning versatility of our approach, that were performed at our Laboratory. All experiments involved our Turtlebots platforms, equipped with a Netbook for running the robot software and ROS. In the first experiments, all of them start with our library package without any behavior. In the second experiment, the robots start with different behaviors that are supposed to have been learned in some way by each of them, in the case, we have taught them by demonstration.

In both experiments, we can assure that the performance of the second robot will be precisely the same as the first robot if all the smallest coordinates are passed. Our approach is a knowledge transference algorithm and the transference of the learned behavior is 100% the same intended to be transferred. The performance of the learning behavior by Program by Demonstration (PbD) is not the objective of our work, only the creation of new behavior, transference, and execution of this behavior.

A. EXPERIMENTS AT LARS 2018

The objective of this first experiment scenario is to test the combination of N-learning with PbD to generate emerging behaviors and to show that N-learning improves the PbD technique. Therefore, we set up two Turtlebot robots. The Turtlebot robot is composed of a tubular base with odometry and IMU sensors, and a Kinect camera on top of the robot. On the first Turtlebot, here called master robot, we implemented a follower-like behavior included in the ROS libraries. This behavior is responsible for searching the nearest point of the robot using the *LaserScan* and trying to keep the point always ahead of the robot and at a constant distance. When implementing this behavior, the start and stop functions of the robot were inserted using the *kobuki* base button of the Turtlebot. Thus, when clicking the button on the master robot, it starts to follow a person. When the button is triggered for the second time it stops the follower algorithm, and the robot itself. As said above, the motion control behavior also uses the nonlinear model predictive control, borrowed from the work of Nascimento et al. [6].

During the experiments, the visitors to the robotics fair were invited to interact with one of the robots, the one that has the follower algorithm pre-programmed. The interaction

[1]Robotica 2018 - http://www.robotica.org.br/

consists of making the robot to move through the environment by following the human in front of it while realizing distinct paths. Its key idea is to allow the participants to interact with the robot randomly. The audience participating in the experiment is diverse, i.e. children, young people, and adults. During the experiments, the robot is following a human while keeping the pose coordinates from where it passes in every t seconds. These positions are stored in a data structure until the behavior is finished. During the follower behavior performed by the master, only some major, important coordinates acquired in every t seconds were saved. N-Learning can record and perform the exact movements. However, this would imply a drawback of recording the smallest difference in displacement performed by the first robot. Therefore, we chose not to record all coordinates. After the interaction with the robot, the follower algorithm is deactivated. Now, it is necessary to create the new *Path Generator Behavior* based on the information acquired at run time from the master robot. This is performed by the master robot using model behavior. This behavior model serves to include the necessary instructions into the new behavior. The Behavior Manager knows exactly where to enter this new information to customize a new behavior. With the template loaded and the new information in memory, it is necessary to save a file with a new name, generated in a way that does not match the existing behaviors. For this purpose, *Generated Behavior* pattern is used. Then the new behavior is loaded into the behavior dictionary along with the meta-data and thus it can be shared with the second robot.

After the emergent behavior (the Path Generator Behavior) is created, it is transferred to robot r_2, here called supporting robot. The supporting robot also receives the acquired major coordinates and all other three previously programmed behaviors passed to the supporting robot through N-learning. Then, the supporting robot performs the same path generation behavior using the major coordinates passed along with the emergent behavior created by r_1. When the tracking of the path is finished, the experiment is terminated. Illustrative images of this experiment can be seen in Figure 5.

These experiments were carried out and it was possible to repeat it six times (in six different situations) during the symposium. The web server of the event was not available for the realization of these experiments. Therefore, an ad-hoc network was created to enable communication between the robots. The data were obtained from the odometry of the robot. The configuration of available resources differs between the robots. Robot r_1 uses all available resources while the Kinect camera from robot r_2 is deactivated. Table 1 shows the behaviors available at the beginning of each experiment. Robot r_1 had all behaviors previously programmed, except b_5, which will be generated at run time. No behavior beyond b_1 was present at the beginning of the experiment in robot r_2. The execution of the tasks is guided by the roles, where r_1 is the master and r_2 is the support robot. Their objectives are respectively, to generate a learned behavior taught by the human to be followed and to learn how to move with the behavior generated by the master. Finally, the data presented below do not reflect the true position of the robots, but the estimated position measured by their sensors and fused using an Extended Kalman Filter.

The first executed scenario (experiment e_1) has the robot motion chart shown in Figure 6a. The path in blue, which is traversed by the robot r_1, involved several curves and in the end, the robot was at a position very close to the initial one. In this experiment, more than one person has participated in

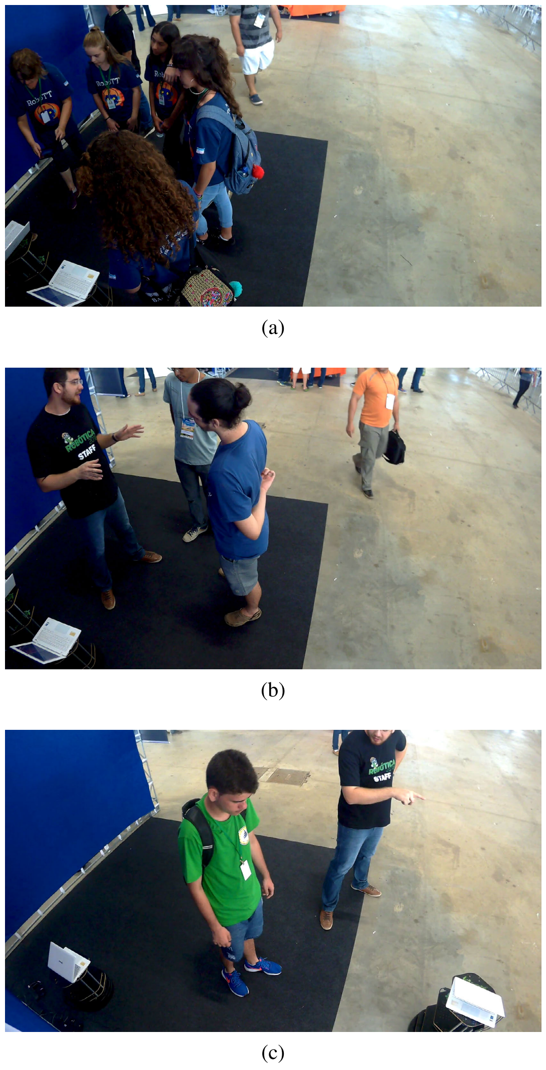


FIGURE 5. Experiment carried out at the Robotica Conference. It is possible to verify the diversity of people who participated in the experiments, which programmed the robots by demonstration.

TABLE 1. The behaviors of the master robot.

Behavior	Title
b_1	Behavior Manager
b_2	Person Follower
b_3	Model for generating new behavior
b_4	Trajectory tracking control
b_5	Path Generator (generated at run time)

the tests. The change of participants was straightforward only by positioning themselves in front of the robot so that the robot would follow. At the end of all human-robot interactions, the new behavior was generated and transferred to the supporting robot. From the graph shown in Figure 6a it can be noticed that the path was similar to r_1. At the end of the path, the robot tries to return to the starting position to make the path a second time, as it has no other behaviors to perform.

Some parts of the path made by r_2 are in a grid format, without making the curves perfectly. Some factors influenced this behavior. The first factor is that, as previously stated, only some sampled coordinates of the path are acquired, at every t seconds, in this case, every 4 seconds, which were saved by the master robot. This creates an acceptable margin of tolerance between the position of the trajectory performed by the master robot and trajectory generated by the supporting robot using the created behavior. We noticed that the difference in both trajectories is not large enough to generate large displacements. It is also interesting to notice that the sample points collected will vary according to the speed that the robot moves. If the followed person moves in a faster manner the robot will move faster, and thus fewer points will be collected. In slower trajectories, the robot gets more points. The same behavior can be seen in the second experiment presented in Figure 6b. This second experiment is also the result of several people interacting with the robot successively. The master robot generated again a path with several curves. The support robot was able to accomplish a similar trajectory reaching the goal of its role.

In experiment 6c, the graph shows a different behavior at the beginning of the trajectory. The configuration of the initial position of the robot in this experiment was different from the others, since the supporting robot did not start from the master robot initial position $pos_{r_2} \neq (0, 0, 0)$. As the behavior created by the master robot started at its initial position $pos_{r_1} = (0, 0, 0)$, r_2 had to move to that position to complete the path. At the end, the robot started the path again toward the starting position as in the first experiment (the **VIDEO** of experiment 6c was also performed[2]).

The fourth experiment in this first scenario that is shown in Figure 6d presents a simple path, with only one loop. The path that is taken by the supporting robot, also, does not perform the curves smoothly. However, at the end of the experiment, it could be noticed that the behavior has been performed successfully. In the fifth and sixth experiments, two loops were found in the path of the robots (Figure 6e and Figure 6f). Once again the supporting robot was able to follow the paths successfully.

When analyzing all 6 runs in this scenario, we can compare the total amount of time that the supporting robot needed to learn the emergent behaviors with the total amount of time that the master robot needed to learn the behavior from the programming by demonstration. The total amount of time the master robot needs to learn the behavior from

[2]https://youtu.be/lhgcI-j3VZo

TABLE 2. Time comparison table.

Experiment	PbD Time (min:sec)	N-Learning Time (min:sec)
e_1	3:57	0:07
e_2	5:12	0:03
e_3	1:24	0:04
e_4	2:51	0:06
e_5	3:46	0:14
e_6	2:59	0:06

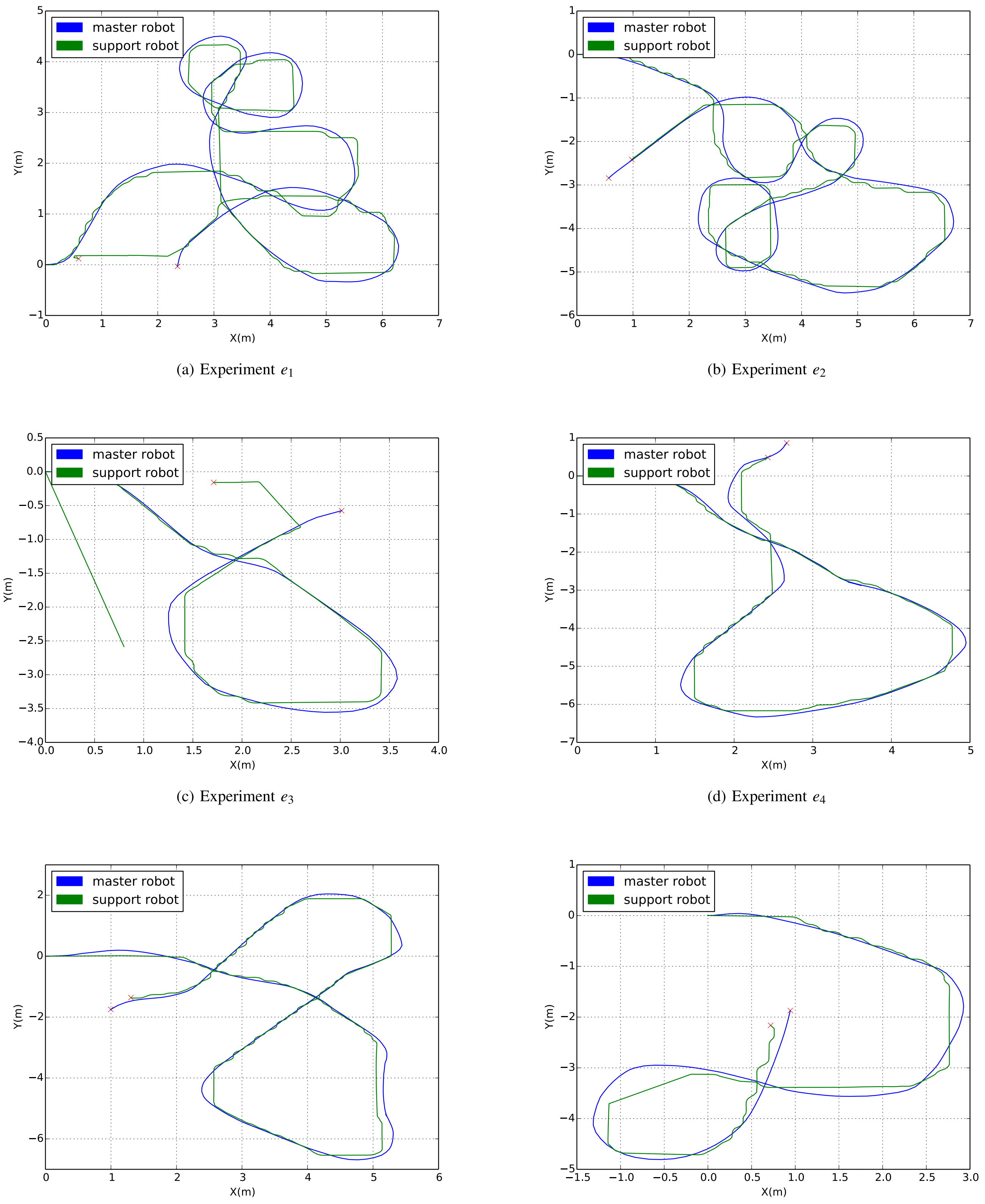
the PbD does not take into account the time the robot stays in a stopped position waiting for a person to be in front of the master robot so it can follow. This comparison is presented in Table 2 and indicates that the learning time using the PbD approach is greater than the learning time using the N-learning approach. We are also able to see that when a robot uses the Programming by Demonstration approach, it spends more time in learning a new behavior then when it uses the N-learning approach. This is expected since the followed human took some time to move and the robot also had to make the journey. However, if we consider the scenario where there are other robots in the system to learn the same path, N-learning is a tool that substantially reduces learning the time. At first, the robot learns according to the PbD time column, and from the second robot on, they would only need the time that N-learning uses to teach the behaviors into the master robot to the teammates. The approach is useful because if PbD time was used to teach the teammates, the final time would be much longer. Each robot would have to demonstrate the movements for each robot of the team that needed to learn.

For example, if we use ten robots want to learn a path like that of experiment e_3 which was the shortest learning time for PbD (and also the shortest path). The learning time with PbD (t_p) disregarding the time to go back to the initial position would be $t_p = 10 * 84 \longrightarrow t_p = 840s$. Using the N-learning approach, the learning time $t_n = 10 * 4 \longrightarrow t_n = 40s$. Considering the current implementation, one can use broadcast messages to teach all the individuals at once, but the study of how long the learning in this situation would last will be analyzed in future works. Considering that the time of a single broadcast message is not different from the time of a message for a single robot, we would have the time $t_n = 4s$.

B. CHANGING A PREVIOUSLY ACQUIRED BEHAVIOR

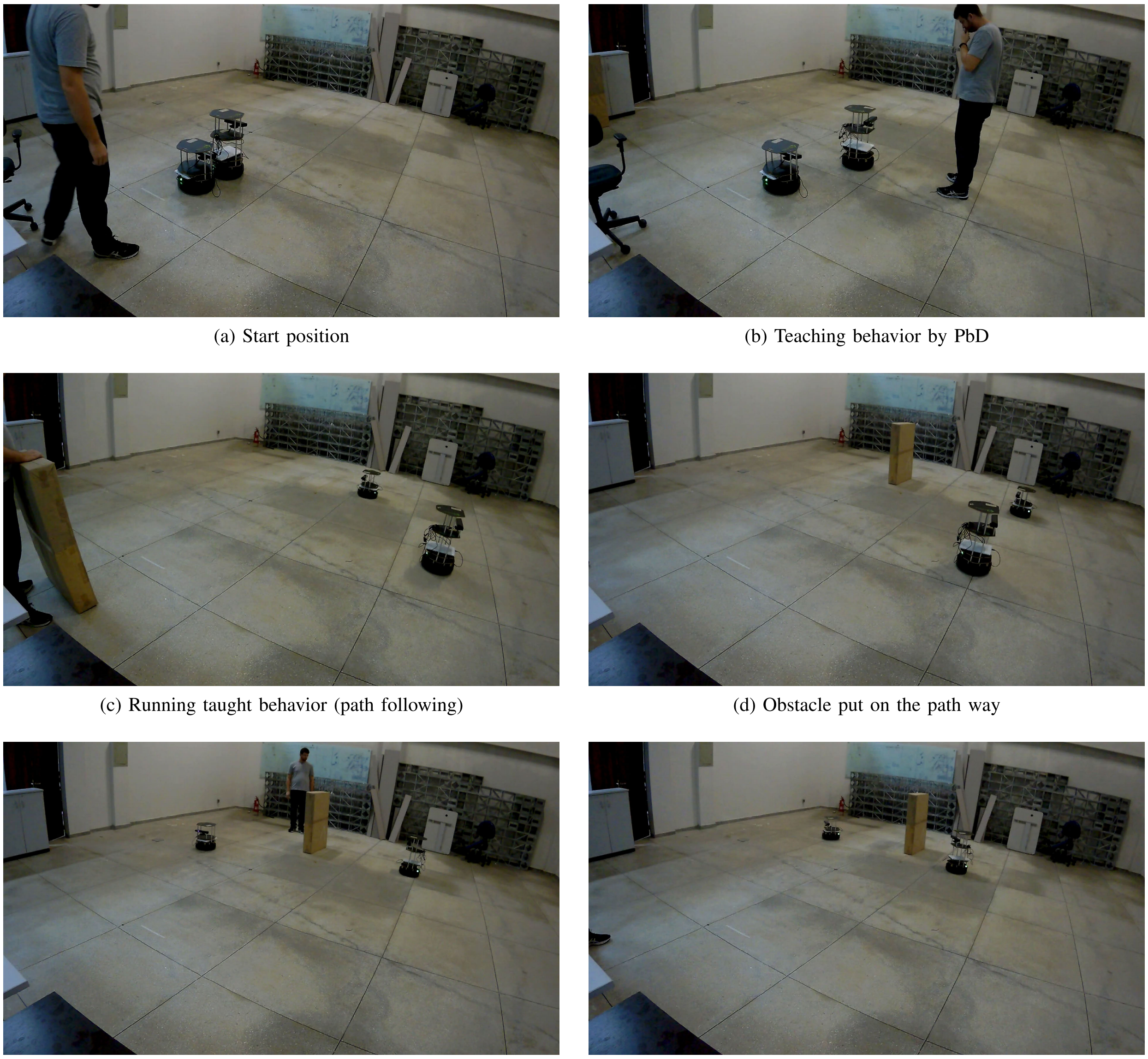
In the second row of experiments performed at our lab, two of our Turtlebots start at some initial position, as shown in Figure 7a. Differently from experiment 1, in this experiment we have two different Turtlebots. The first Turtlebot is taller and the camera is set in a higher position on the robot to better see the actions of the human teacher. The second robot has a normal height. Both robots are initially taught with some behavior using the PbD approach.

This behavior is to perform a path following task (Figure 7b). Then, this behavior is shared between the robots



(a) Experiment e_1 (b) Experiment e_2 (c) Experiment e_3 (d) Experiment e_4 (e) Experiment e_5 (f) Experiment e_6

FIGURE 6. Trajectories of both robots during the experiments.



(a) Start position

(b) Teaching behavior by PbD

(c) Running taught behavior (path following)

(d) Obstacle put on the path way

(e) Taught emergent behavior (obstacle avoidance)

(f) Running new behavior (path following avoiding obstacle)

FIGURE 7. Experiment carried out at the Laboratory. The robots start at initial position (a) and are taught by the human (b) then a path following behavior runs in both of them (c). A box is put on the the path (d) and the human teaches the robot how to avoid it (e). At the end both robots know how to avoid the box (f) by running the new behavior that was shared by the robot that was taught.

that start following by themselves (Figure 7c) the learned trajectory in the environment, which is recorded as the smallest loop at Figure 8, somewhat between a circular or square shape, as if they were a group of children playing (going after each other).

Also, one of these robots has been programmed with a *modify PbD* behavior including the PbD itself that executes the follower algorithm to learn emergent behaviors. This *modify PbD* behavior has also been shared between the robots by N-Learning and it can be executed at any time as requested. In practice, as desired by some robot in answer to unexpected situations. E.g., if no one of the robots know how to proceed in some unexpected situation, then a request can be sent to a human and, this is mandatory, the robot has to wait for the teacher (human) to pressing some button so then it starts the follower (with PbD) behavior in order to learn the new emergent behavior, on-line. In this case, this new behavior is a piece of the path that avoids an obstacle, that is taught by the human until he presses the stop button again.

A situation in which this can be triggered is, for example, when the visual system detects an obstacle on the pathway as shown in Figure 7d. In this situation, the *modify PbD* behavior started by the human takes the previous behavior (path) that is being executed (in the first robot that detects the box) and changes the part of the path by the piece that is being taught by PbD (Figure 7e). This piece changes the path in between the position where the robot stopped until the human determines its end, pressing the start and stop buttons for those.

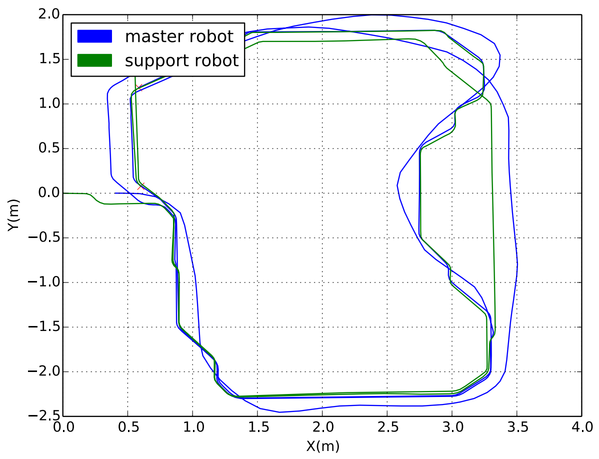


FIGURE 8. **Experiment carried out at the Laboratory. It is possible to verify that the path followed in the old behavior was modified by one of the robots that shared this behavior to the other, thus changing the behavior of both according to the new path taught to the robot. The learning of the new (emergent) behavior for this part of the path is automatically triggered by the visual system that detected a box just in the middle of the old path, and that none of the robots know how to deal with.**

To create the new part of the path (seen at right in Figure 8), the robot follows the human while keeping the pose coordinates from where it passes at every t seconds, in the same way as in the PbD above. These positions are stored in a data structure until this new behavior is finished. After finishing, the follower algorithm is deactivated (by pressing the same button) and now the old behavior is updated including this new piece of trajectory on it, taking as initial position the point where the robot asked for help and as final position the current robot position. Thus, the *Path Generator Behavior* was enhanced for this to be possible, based on the information acquired at run time from the human teacher. Notice that this updated behavior can be shared between other robots that, in practice, can avoid the obstacle (Figure 7f). The old and new trajectories can be seen surrounding the obstacle that was put on their way was avoided at the right of Figure 8.

Notice that this approach can be generalized with any behavior that is currently running and that eventually generates some deadlock situation. This is somewhat inspired by a child's natural behavior that stops some action and asks for help on what to do next, in the case of some problem appearing.

Concerning memory usage, in the N-Learning paradigm (and in the improvement proposed here) the robots just need to have enough memory to run a set of behaviors that is strictly necessary for operation in a given environment. However notice that this set does not need to have all the behaviors that some robot can perform, including the ones for other tasks in another completely different environment. This is especially useful in constraint hardware such as embedded systems of Unmanned Aerial Vehicles that have limits to memory storage for navigation purposes. Thus, only behaviors that are necessary for a given mission can be acquired by sharing mechanisms, requested by any robot. With the current approach, we can even allow these behaviors to be enhanced on-line, as seen in the second experiment. We could verify that completely new behavior can be created or acquired in some way and shared between the robots with the current enhancement.

V. CONCLUSION AND FUTURE WORKS

We have extended the approach called N-Learning, which was originally devised for learning and teaching of behaviors in a multirobot team, to allow the on-line acquisition of emergent behaviors. We have redesigned and implemented a new basic central behavior that allows the robots to acquire new behaviors and sharing them with their teammates. This is done through interactions between the robots and/or with a human, at execution time. This approach can be applied by a team of robots for self-programming through the acquisition and sharing of knowledge, here represented by a block of programs that contain the desired behaviors.

This capability can be useful in situations where the team of robots should accomplish a specially devised mission requiring a specific set of behaviors that can now be programmed only a single time at a specific robot or various robots, in a distributed way. Our initial motivation was to compare with the Program by Demonstration approach since the first experiment, which is clear with the second set of experiments. We could not find in the literature any other work that shares entire blocks of behaviors between robots which makes it hard to compare at such level. This was the contribution of the original N-Learning, which has been enhanced in this work.

We have shown that the program by demonstration paradigm can be used in conjunction with our proposal to set new behaviors at execution time. These new behaviors can be acquired by any robot of the team that has resources capabilities for it and then be shared among the other robots of the team automatically, using the N-Learning, without necessarily reprogramming them. The advantage is to allow the team to have only one acquired (programmed or by demonstration) behavior and further, these (eventually distributed) behaviors to be transferred to the other robots according to their demand. This avoids reprogramming all of the robots, by demonstration or by hand, since the robots in the team can now acquire and share behaviors autonomously. Thus, the main contribution is that the basic behavior sharing capability [4] was extended in this manuscript by including the capacity of acquisition of new behaviors at execution time.

Experiments with real robots were performed to verify and validate our approach. In all the experiments the supporting robot r_2 was able to carry out a path similar to the master robot r_1, which has learned these paths as emergent behaviors. There are differences in the paths that can be improved by reducing the master robot sampling time t. The supporting robot was able to follow the path using the behavior created by the master robot successfully.

As future work, more complex behaviors involving vision as foveated attention [20], [21] for recognition and obstacle detection, visual recognition of marks [22], and navigation strategies based on occupancy elevation map [23] will be integrated to this system. Also, approaches based on AI are planned to be studied to allow the robots to be more self-programmed, and independent of the human teacher, the model of the emergent behavior.

ACKNOWLEDGMENT

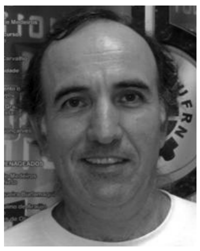
The authors would like to thank CAPES for the Ph.D. grant and CNPq.

REFERENCES

[1] R. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robot. Autom.*, vol. RA-2, no. 1, pp. 14–23, Mar. 1986.
[2] P. Stone and M. Veloso, *Task Decomposition Dynamic Role Assignment for Real—Time Strategic Teamwork*. Berlin, Germany: Springer, 1999, pp. 293–308.
[3] Y. Sheng, H. Zhu, X. Zhou, and W. Hu, "Effective approaches to adaptive collaboration via dynamic role assignment," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 1, pp. 76–92, Jan. 2016.
[4] L. Costa, T. Nascimento, R. Maia, and L. Gonçalves, "N-learning: An approach for learning and teaching skills in multirobot teams," *Robotica*, to be published, doi: 10.1017/S0263574719000468.
[5] (Aug. 22, 2019). *Robot Operating System*. [Online]. Available: http://wiki.ros.org/pt
[6] T. P. Nascimento, C. E. T. Dórea, and L. M. G. Gonçalves, "Nonlinear model predictive control for trajectory tracking of nonholonomic mobile robots: A modified approach," *Int. J. Adv. Robotic Syst.*, vol. 15, no. 1, pp. 1–14, 2018.
[7] Y. Wang, F. Yan, J. Chen, F. Ju, and B. Chen, "A new adaptive time-delay control scheme for cable-driven manipulators," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3469–3481, Jun. 2019.
[8] Y. Wang, K. Zhu, C. Bai, and F. Yan, "Adaptive super-twisting nonsingular fast terminal sliding mode control for cable-driven manipulators using time-delay estimation," *Adv. Eng. Softw.*, vol. 128, pp. 113–124, Feb. 2019.
[9] Y. Wang, B. Li, F. Yan, and B. Chen, "Practical adaptive fractional-order nonsingular terminal sliding mode control for a cable-driven manipulator," *Int. J. Robust Nonlin.*, vol. 29, no. 5, pp. 1396–1417, Mar. 2019.
[10] Y. Y. Wang, J. W. Chen, F. Yan, K. W. Zhu, and B. Chen, "Adaptive super-twisting fractional-order nonsingular terminal sliding mode control of cable-driven manipulators," *ISA Trans.*, vol. 86, no. 3, pp. 163–180, Mar. 2019.
[11] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Auton. Agents Multi-Agent Syst.*, vol. 11, no. 3, pp. 387–434, Nov. 2005.
[12] J. Xu, C. Tekin, S. Zhang, and M. V. D. Schaar, "Distributed multi-agent online learning based on global feedback," *IEEE Trans. Signal Process.*, vol. 63, no. 9, pp. 2225–2238, May 2015.
[13] A. Vakanski, F. Janabi-Sharifi, and I. Mantegh, "An image-based trajectory planning approach for robust robot programming by demonstration," *Robot. Auton. Syst.*, vol. 98, pp. 241–257, Dec. 2017.
[14] S. Park, W. Lee, W. K. Chung, and K. Kim, "Programming by demonstration using the teleimpedance control scheme: Verification by an semg-controlled ball-trapping robot," *IEEE Trans. Ind. Informat.*, vol. 15, no. 2, pp. 998–1006, Feb. 2019.
[15] M. J. A. Zeestraten, I. Havoutis, and S. Calinon, "Programming by demonstration for shared control with an application in teleoperation," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 1848–1855, Jul. 2018.
[16] J. Piaget, *Equilibration of Cognitive Structures: The Central Problem of Intellectual Development*. Chicago, IL, USA: University Chicago Press, 1985.
[17] L. Vygotsky, "Play and its role in the mental development of the child," *Sov. Psychol.*, vol. 5, no. 3, pp. 6–18, 1967.
[18] R. S. Maia and L. M. G. Gonçalves, "Intellectual development model for multi-robot systems," *J. Intell. Robotic Syst.*, vol. 80, no. 1, pp. 165–187, 2015, doi: 10.1007/s10846-015-0224-0.
[19] C. Lauretti, F. Cordella, E. Guglielmelli, and L. Zollo, "Learning by demonstration for planning activities of daily living in rehabilitation and assistive robotics," *IEEE Robot. Autom. Lett.*, vol. 2, no. 3, pp. 1375–1382, Jul. 2017.
[20] R. B. Gomes, B. M. de Carvalho, and L. M. G. Gonçalves, "Visual attention guided features selection with foveated images," *Neurocomputing*, vol. 120, pp. 34–44, Nov. 2013.
[21] F. F. Oliveira, A. A. S. Souza, M. A. C. Fernandes, R. B. Gomes, and L. M. G. Goncalves, "Efficient 3D objects recognition using multifoveated point clouds," *Sensors*, vol. 18, no. 7, p. 2302, 2018.
[22] L. A. V. Souto, A. Castro, L. M. G. Gonçalves, and T. P. Nascimento, "Stairs and doors recognition as natural landmarks based on clouds of 3D edge-points from RGB-D sensors for mobile robot localization," *Sensors*, vol. 17, no. 8, p. 1824, 2017.
[23] A. Souza and L. M. G. Gonçalves, "Occupancy-elevation grid: An alternative approach for robotic mapping and navigation," *Robotica*, vol. 34, no. 11, pp. 2592–2609, 2016.

LUIS FELIPHE S. COSTA received the master's degree from the Federal University of Paraíba in the Postgraduate Program in Informatics with a sandwich period at the Faculty of Engineering, University of Porto (FEUP), and the Ph.D. degree in electrical and computer engineering from the Federal University of Rio Grande do Norte, in 2018. He was involved in several research activities including multirobot systems, embedded systems, artificial intelligence, and distributed simulations. He was with the Distributed and Real Time Embedded Systems Lab (DARTES), FEUP.

TIAGO P. DO NASCIMENTO received the B.S. degree in mechatronics engineering from the College of Technology and Science, FTC, Brazil, in 2007, the M.S. degree in electrical engineering from the Universidade Federal da Bahia, Brazil, in 2009, and the Ph.D. degree in electrical and computer engineering from Oporto University, Portugal, in 2012. In 2013, he joined the Department Computer Systems, Universidade Federal da Paraiba, João Pessoa, Brazil, as an Assistant Professor.



LUIZ MARCOS G. GONÇALVES received the Ph.D. degree in systems and computer engineering from COPPE-UFRJ, Brazil, in 1999. He is currently a Full Professor with the Computer Engineering Department, Universidade Federal do Rio Grande do Norte, Natal, RN, Brazil. He has done research works in several aspects of graphics processing, including robotics vision (main interest), computer graphics, geometric modeling, image processing, and computer vision.