To appear in *Advanced Robotics*
Vol. 00, No. 00, 0000 0000, 1–16

Full paper

Humanoid Robot Imitation through Continuous Goal-Directed Actions: An Evolutionary Approach

Santiago Morante* [a], Juan G. Victores[b], Alberto Jardón[c] and Carlos Balaguer[d]

[a,b,c,d] *Robotics Lab research group, Universidad Carlos III de Madrid (UC3M), Av. Universidad 30, Leganés 28911, Madrid, Spain.*

(*0000*)

Humanoids can learn motor skills through the Programming by Demonstration (PbD) framework, which allows matching the kinematic movements of a robot with those of a human. Continuous Goal-Directed Actions (CGDA) is a framework that can complement the paradigm of robot imitation. Instead of kinematic parameters, its encoding is centered on the changes an action produces on object features. The features can be any measurable characteristic of the object like color, area, etc. The execution of actions encoded as CGDA allows a robot-configuration independent achievement of tasks, avoiding the correspondence problem. By tracking object features during action execution, we create a trajectory in an n-dimensional feature space that represents object temporal states, allowing generalization, recognition and execution of action effects on the environment. Experiments have been performed, using a humanoid robot in a simulated environment. Evolutionary computation was used for joint parameter calculation of a humanoid robot. The objective is to generate a motor trajectory which leads to a feature trajectory equal to the objective one. In one of the experiments, the robot performs a spatial trajectory based on spatial object features. In a new experiment, the robot paints a wall by following a color feature encoding.

Keywords: learning from demonstration; goal-directed imitation; robot learning; evolutionary computation; motor primitives

1. Robot Imitation

The field of robot imitation has been dominated by motor parameter reproduction [1]. This approach has been called *programming by demonstration* (PbD) [2] or *learning from demonstration* (LfD). These methods encode an action by recording the joint motor parameters of a demonstrator when performing the action, and then applying different machine learning techniques to extract a generalization. The demonstrator can either be the guided robot itself, or an external agent.

Some authors have questioned whether motor imitation alone is useful for humanoid robotics. Schaal asked in [3] as an 'outstanding question' to be answered: *How can the intention of a demonstrated movement be recognized and converted to the imitators goal?*. Additionally, [4] summarized the three key concepts of imitation: 'what to imitate', 'how to imitate', and 'when to imitate'. The Continuous Goal-Directed encoding of tasks aims to fulfill mainly the 'what to

* [a]Corresponding author: Santiago Morante Cendrero. Email: smorante@ing.uc3m.es. Telephone: +34 916246241

[b]Juan González Victores. Email: jgcvicto@ing.uc3m.es. Telephone: +34 916248813

[c]Alberto Jardón Huete. Email: ajardon@ing.uc3m.es. Telephone: +34 916246242

[d]Carlos Balaguer Bernaldo de Quirós. Email: balaguer@ing.uc3m.es. Telephone: +34 916249426

imitate' concept. We will also give some insights for the 'how to imitate' with our particular encoding.

We believe that imitation could be improved, and some of the stated problems solved, by taking the action consequences in the environment more into account. These consequences will be the goals of the task (a 'paint' action will modify the *color* of an object that is painted). When looking at how nature faces the problem of imitation, psychology indicates that the human brain encodes actions as end-goals. For example, when children imitate others grasping a person's ear, they tend to imitate the action goal (which ear to grasp) rather than the kinematic aspects of the action (which hand is used to perform the grasping) [5]. This behavior seems to be related with the mirror neuron system [6]. However, in usual robot imitation, there is a lack of codification of action effects, and only the kinematic aspects are considered. This fact limits flexibility in action execution. The robot can, effectively, repeat the same movements, but the action goals remain a mystery for it [4].

Let us overview some representative examples of PbD. In [2], a human demonstrator performs a task several times (e.g. hitting a ball) using a robotic arm. Positions, orientations and velocities of the arm are recorded, and the number of representative states of the action are estimated with Hidden Markov Models (HMM). HMM are used to handle spatio-temporal variabilities of trajectories across several demonstrations. Finally, and in order for the robot to execute the trajectory, Gaussian Mixture Regression (GMR) is used to create a regression function using previous HMM states. This reconstructed trajectory is the one the robot reproduces to imitate the human movement. Another common technique used, along with HMM [7] [8], is Gaussian Mixture Models (GMM) as in [9] [10].

We define goal directed actions as those in which the only parameters analyzed are the ones belonging to the elements affected by the action. When talking about goal directed actions in robotics, a goal encoding is found in [1], where they extract goals as relevant features that appear most frequently from a demonstrated dataset, this is, the invariants in time. This framework was extended in [11] where, despite they learn the kinematic trajectory to perform actions, they encode action goals, replicating [5]. In the robotic experiment, during the demonstration, the robot tries to extract a set of invariant constants. Later, the robot computes the trajectory that best satisfies the constraints. Another example is [12], where an object must be grasped and then placed at one of two presented targets that have different heights. There is a bridge shaped obstacle in the path. Depending on the height of the bridge, the object must be grasped differently and through a different path.

There are no exclusively continuous goal directed actions references in literature, to the authors' knowledge. A relatively near work [13], uses a combination of object spatial and demonstrator hand movement tracking. In [13], they build a system with a set of primitive actions (inverse models). When a human demonstrator performs an action, they continuously track the object and the demonstrator's hand spatially through time. At the same time, they run all inverse models during action stages to find the best performance of each model in each stage. Finally, they construct a high-level inverse model composed by those selected primitives, being able to imitate the action goal with similar spatial movements. The object tracking is only used to identify grasping and releasing stages. Tani's group developed an inverse model by training a multiple timescale recurrent neural network to match robot motor torques with the position of an object in [14]. Similarly, vision is only used to track the spatial position of the object.

Goal directed actions are interesting because they enable a robot configuration independent way to encode and execute actions. This will allow our system to avoid the correspondence problem (the difference in the kinematic model of the demonstrator and the learner [15]). Some authors have tried to avoid the correspondence problem by creating a common reward space between the human demonstrator and the robot [16]. In our case, we use the feature trajectory, which is demonstrator independent, to encode the imitation. Our work aims to allow features beyond spatial ones, such as changes in color or object deformations. Other paradigms, like PbD, rely on full body capture systems for humanoid robot programming. This requirement may be

Table 1. Main differences between PbD and CGDA paradigms.

	PbD	CGDA
Objective of imitation	Spatial trajectories	Object feature states
Features tracked	Demonstrator's joint positions/velocities	Object's shape, area, color, coordinates, etc.
Strengths	Perfect kinematic imitation	Effects encoding
Weaknesses	Undefined goal to achieve	Undefined way to achieve the goal

difficult to achieve in everyday interaction with humanoid robots, and has a limited applicability to tasks such as painting a wall. Goal directed actions, instead, assume internal mechanisms to perform a demonstrated action by indicating the desired goals.

2. Continuous Goal-Directed Actions

Continuous Goal-Directed Actions are a way to encode the effects of an action when the action is demonstrated to a robot [17]. The CGDA framework is used for generalizing, recognizing and executing actions by their effects on objects. A continuous analysis generates a trajectory in an n-dimensional feature space, where n equals the number of tracked object features. Examples of tracked features could be color, area, weight, spatial positions, etc. The trajectories are discretized, and action repetitions lead to a point cloud through time. This point cloud is the subject of analysis of the developed techniques for recognition and execution. The main differences between the CGDA and PbD paradigms can be found in Table 1. We have developed a continuous tracking infrastructure to allow the learning of actions with relevant object feature intermediate states e.g. recognizing the rotation of a valve is unachievable without a continuous tracking infrastructure, because the final state of the valve could be the same as the initial, looking like no action has been executed.

A block scheme of CGDA can be found on Figure 1.

2.1 *Generalization*

When the trajectories resulting from tracking are considered in a discrete way, the action repetition create a point cloud in the feature space. For generalization purposes, we need to extract a representative n-dimensional trajectory of the point cloud from several repetitions. This process is composed by the following three steps.

(1) **Time Rescaling:** Each single action repetition is normalized in time (range $[0, 1]$). With this time rescaling, every action execution gets bounded by the same temporal limits, making the algorithm independent of the repetitions speed. All normalized trajectories are introduced in the same object feature space, forming a point cloud.

(2) **Average in Temporal Intervals:** To model the point cloud, we split it in N temporal intervals. The number of intervals is computed from the average duration of the original repetitions by fixing one interval per second. As the repetitions are normalized in time, the number of intervals allows preserving a notion of the action duration. The representative point p of each interval is extracted as $p = \frac{1}{p_{int}} \sum_{i=0}^{p_{int}} X_i$, where p_{int} is the number of points in the interval and X_i represent the vector of features for a point. The result is a vector of average features.

(3) **Radial Basis Function Interpolation:** Once we have the representative points of the point cloud, we have to join them to form a generalized action, i.e. an object feature trajectory we can consider as a generalization. In a robot joint space, an interpolation could create a jerky joint trajectory, so literature, e.g. [2], commonly uses regressors such as GMR. However, working in the object feature space, we perform an interpolation to assure the trajectories pass through the target points (which are the states of the object in an instant). We use a Radial Basis Function (RBF), which is an interpolation technique

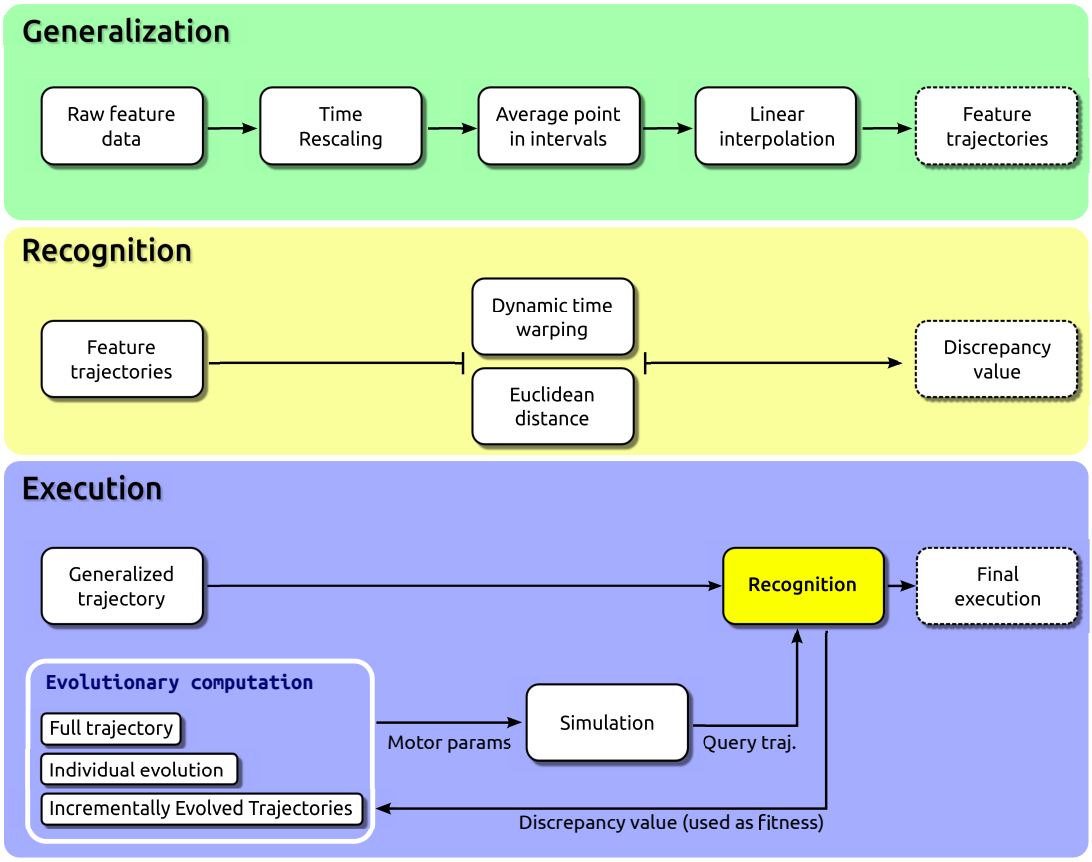

Figure 1. Block scheme of the Continuous Goal-Directed Actions (CGDA) framework.

based on measuring the influence of every known point over the queried point [18]. The RBF interpolation $f(x)$, which will become the final generalized trajectory, is mathematically expressed as a sum of radial basis functions:

$$f(x) = \sum_{i=1}^{N} w_i \, \phi(\|x - x_i\|) \tag{1}$$

Where N is the number of radial basis functions, equal to the number of intervals, and x_i represents the coordinates of each interval's known point. The radial basis function is denoted as ϕ, where the input parameter is the distance between the known point x_i and the queried point x, measured with L^2 norm. From the available radial basis functions, this linear one has been selected because we do not care about trajectory smoothness in the feature space. The coefficient w_i is the weight of a specific known point over the queried point x, and it is the value to be solved. As the interpolation is known at known points, the weight problem is solved as a set of N linear equation with N unknowns:

$$f(x_1) = \sum_{i=1}^{N} w_i \, \phi(\|x_1 - x_i\|)$$

$$\vdots$$

$$f(x_N) = \sum_{i=1}^{N} w_i \, \phi(\|x_N - x_i\|) \tag{2}$$

Once the interpolated function is returned, we consider this output as the generalized function of an action. Its physical meaning is how the state of the object, regarding its features, changes across the action performance.

A final generalization example, once this process is completed, can be seen in Figure 2.

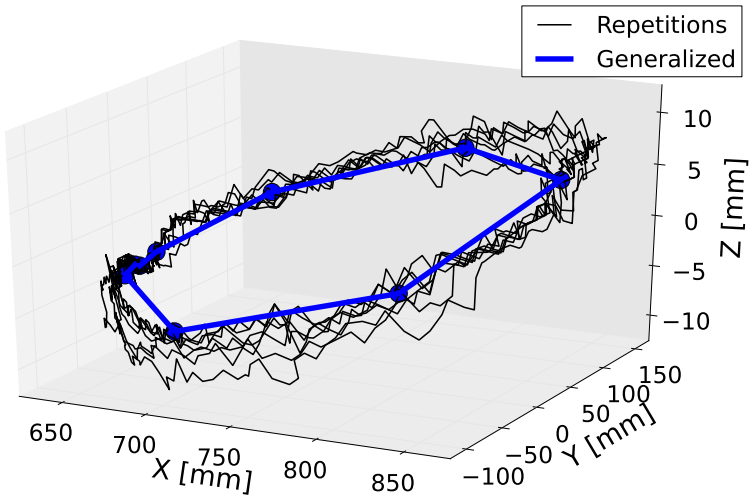

Figure 2. Plot representing a three feature trajectory. Black lines are training action repetitions. The blue line is the generalization of all the repetitions.

2.2 *Recognition*

The generalized trajectories can be used for recognition. Recognition can be performed by comparing a query trajectory with the previously computed generalized trajectories, returning the one with the highest similarity. This application can be useful for determining tasks as action sequences by splitting tasks into smaller and simpler actions. For explaining how we recognize actions, we assume we have a database of generalized actions, and an unknown action that we want to associate to the most similar of the database. We assume the recognition as the comparison of a query with our generalized trajectories, returning the one with the highest similarity.

Looking for flexibility, two different methods to perform recognition have been implemented. One of them is more flexible, but can lead to inaccuracies, and the other is more strict and limits the variations between the query and the generalization.

2.2.1 Dynamic Time Warping

As the query and the generalized actions are normalized in time, we can take t values along time for each action to compare them. The technique used for comparison is Dynamic Time Warping (DTW), which is an algorithm usually used to optimally align two temporal sequences

[19]. Our use of DTW is to compare two time-dependent sequences of points $X = \{x_1, .., x_N\}$ and $Y = \{y_1, .., y_M\}$ with $N, M \in \mathbb{N}$. To compare two elements, a local cost measure (a distance $d(x, y)$) is needed. A lower cost represents a bigger similarity of the sequences. Evaluating all pairs of points between the sequences, using in this case a L^2 norm, we obtain a cost matrix CM, with a size of $N \times M$:

$$CM = \begin{pmatrix} d(x_0, y_0) & \cdots & d(x_N, y_0) \\ \vdots & \vdots & \vdots \\ d(x_0, y_M) & \cdots & d(x_N, y_M) \end{pmatrix} \tag{3}$$

Once having this matrix, the goal is to find the lowest cost alignment path, which intuitively should run along the lowest cost cells. This alignment is called the warping path. DTW includes some constraints in the path calculation to assure a monotonic advance of the path and to assure that the first elements as well as the last elements are connected to each other. The total path cost $C_P(X, Y)$ is calculated as the sum of the local costs C:

$$C_P(X, Y) = \sum_{l=1}^{L} C(x_{nl}, y_{ml}) \tag{4}$$

Where L is the length of the path. For programming reasons, the path is usually calculated in an accumulated cost matrix, where each cell represents the cost of the correspondent pair (x, y) plus the cost to reach this cell (see Figure 3). In the accumulated matrix, the normalized cost $C_{P_{norm}}(X, Y)$ of the optimal path is expressed as:

$$C_{P_{norm}}(X, Y) = \frac{C(x_n, y_m)}{N + M} \tag{5}$$

In our case, we use this normalized cost of the optimal path as a measure of discrepancy between dimensions. As DTW is computed between two signals for one dimension only, we consider the total cost of alignment between two n-dimensional trajectories as the sum of the costs of the optimal paths of each dimension, obtaining a single score D:

$$D = \sum_{i=1}^{n} C_{P_{norm}}(X_i, Y_i) \tag{6}$$

This score is used as the measure of discrepancy between two trajectories in the n-dimensional space. In recognition, the trajectory with the smallest score is the one we consider the match.

2.2.2 Euclidean Distance

Another option to perform recognition, or in general, to measure the error between two trajectories, is to use the Euclidean distance. The query trajectory is normalized in time, in the same way it was for the generalized ones. This step allows us to take t values along time for each action (the query and the generalized) to compare them. The use of DTW is motivated for allowing a non-rigid measure of similarity between trajectories, as DTW is able to 'tighten' and 'widen' until both are best aligned. Unfortunately, DTW allows time displacements that can affect the order of execution of actions, resulting in task failure or performance decay. Therefore, Euclidean distance can be considered an improvement with respect to DTW as metric.

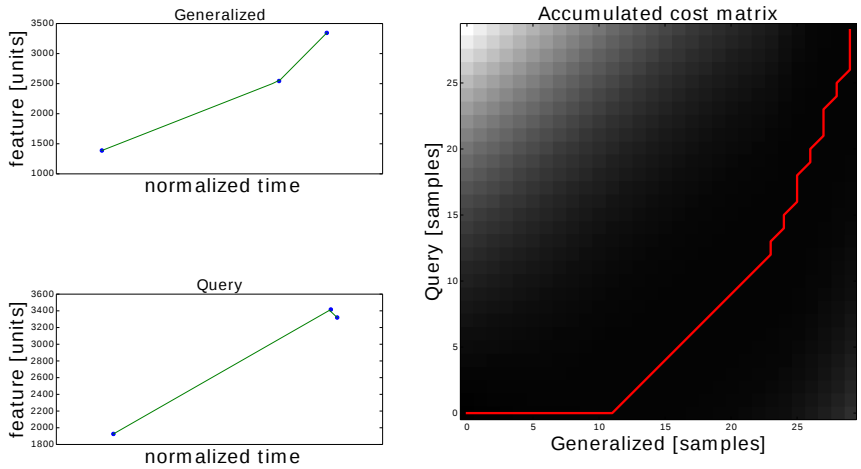

Figure 3. Example of accumulated cost matrix for two sequences. On the left side, the interpolated sequences are shown. The right side depicts the Dynamic Time Warping computation. White cells represent high cost, while dark cells are low cost ones. The red line is the lowest cost path.

Our aim when using Euclidean distance is to obtain a metric of discrepancy between two time-dependent sequences of points, namely $X = \{x_1, .., x_N\}$ and $Y = \{y_1, .., y_N\}$. We compute the total discrepancy $J(X, Y)$ between two n-dimensional trajectories as:

$$J(X, Y) = \|X - Y\| \tag{7}$$

The generalized trajectory with the smallest $J(X, Y)$ is the one we consider as the most similar. Let us recall that all the trajectories involved in this process of recognition are feature trajectories and not joint space trajectories. This implies that we are recognizing actions by what is happening to the object, and not comparing human movements.

2.3 *Execution*

Probably the most useful application for an imitation system is the robot execution of the learned, or generalized, action. We consider CGDA as both a way to encode actions to be recognized, and to be executed by robots. Tasks encoded as CGDA require a particular technique for performing the feature trajectory. As CGDA does not encode joint motor parameters but only object feature changes, the robot must be provided with a method for assuring the task accomplishment. Spatial trajectories are computable with inverse kinematics (IK), but if we want to extend this framework to other features like color or area and we also want them to be robot-configuration independent, we need to choose another method. The tested algorithm in this work is Evolutionary Computation (EC) applied on a humanoid robot in a simulated environment. Several evolutionary strategies have been developed for obtaining motor trajectories.

3. Evolutionary Strategies

In our previous work, we used a Full Trajectory Evolution strategy (depicted here for clarity). In this paper we improve this strategy accuracy with a new strategy called Individual Evolution. A third strategy, Incrementally Evolved Trajectories, has been developed for features which have a temporal dependency, which is the case of most non-spatial features.

3.1 *Full Trajectory Evolution*

This strategy, where all of the points of the full trajectory evolve and are evaluated simultaneously, was used in our previous work. The number of parameters to evolve in this case are $\text{DoF} \cdot N$, where DoF is the number of joints involved in the movement, and N is the number of points of the trajectory. For example, for a spatial trajectory using 3 joints ($\text{DoF} = 3$) in a feature trajectory of 5 points ($N = 5$), the number of parameters to evolve is 15. This strategy tends to makes the algorithm converge slowly, because the search space in this case usually becomes very large.

3.2 *Individual Evolution*

The second strategy, Individual Evolution, consists in evolving and calculating the fitness for each feature point in the trajectory individually, instead of evolving and evaluating the feature trajectory as a whole. This is valid for spatial trajectories, where the joint parameters for reaching a point do not depend on previous points. In this case the number of different evolutions to be performed is equal to N, and each of these evolutions must evolve a number of parameters equal to DoF. This second strategy can outperform the first one in time and fitness value, mostly due to the smaller search space to be examined in each case.

3.3 *Incrementally Evolved Trajectories*

For features which have a temporal dependency, which is the case of most non-spatial features, the Individual Evolution strategy is not useful, as the action may have, and usually has, a dependency on previous points. For instance, when painting a wall, and using a CGDA encoding of the task, the percentage of painted wall depends on the wall that was painted before.

For these situations, we have developed a third strategy called Incrementally Evolved Trajectories. In this case, we also evolve each point individually. The first point is evolved individually. After it converges, we start to evolve the second point, but in this case, the fitness evaluation is performed by sequentially executing both points (the previous first point and the current point). Once the second point converges, we start evolving the third point and for the evaluation we execute the three point trajectory. The same process is repeated for the remaining points. The pseudocode of this strategy is shown in Algorithm 1.

Algorithm 1 Incrementally Evolved Trajectories (IET)

```
1: procedure IET(X, ε)                ▷ X is the feature trajectory. ε is an error parameter.
2:     for i < numberOfPoints(X) do
3:         while fitness > ε do
4:             M_i ←evolve()
5:             f ←execute(M_[0,i])
6:             fitness ←recognize(f, X_i)
7:         end while
8:     end for
9:     return M
10: end procedure
```

This technique offers some advantages with respect to the alternative strategies. First, it reduces the search space for each individual evolution, as it only evolves one point each time. Second, this trick reduces also the time necessary to converge, and improves fitness value of the whole trajectory. In the experimental section, we will show how it is able to approximate different object feature trajectories.

Table 2. DTW cost matrix: test actions (lower case) vs. generalized trajectories (upper case). Bold numbers represent best values (minimum discrepancy).

	MOVE	ROTATE	CLEAN	PAINT
move	**229**	332059	290334	552055
rotate	389021	**7606**	325211	694049
clean	402555	304669	**1724**	44259
paint	497152	671078	25896	**1277**

4. Experiments

Several experiments have been performed. The first two have been performed to test recognition. The third one evaluates execution. It involves only spatial features, and the objective is to generate a robot motor trajectory which leads to a feature trajectory equal to the objective one. This feature trajectory is similar to a cleaning movement, and is encoded as a CGDA where the object tracked is in one of the robot's hands. The last experiment consists in having the robot paint a wall. The painting process is encoded as a CGDA, but this time the object tracked is the wall. The last two experiments are performed in simulation with a model of the humanoid robot Teo [20].

4.1 *Object Feature Trajectory Recognition*

The first experiment setup consists in a Kinect® pointed at a desktop tracking a colored marker. A demonstrator performs several actions using the marker in front of the camera. We connect the camera input with a computer vision library to measure marker features. We have given names to each basic action for simplicity in explanation, but semantics is not used in the process. The actions involve spatial movements (MOVE, ROTATE, CLEAN) and color changes (PAINT). Each action is described as follows:

- MOVE: Marker displacement of 30 cm in one straight direction.
- ROTATE: Rotation over the Center-of-Mass (CoM), on one axis, of 90 degrees.
- CLEAN: Keeping orientation fixed, movement of CoM over a circumference of 30 cm of diameter (1 revolution).
- PAINT: Keeping its spatial coordinates fixed, the marker is painted in a different color with a marking pen, until almost all of the area is covered.

We wanted to test our algorithms with small sets of repetitions, in order to prove its performance when there are few repetitions available. Seven repetitions of each basic action were recorded. Six of the repetitions were used to generate one generalized action, similar to common practices of PbD [10]. The final repetition of each set was used as a test action to be recognized. The tracked object features are: spatial location (X,Y,Z), area, HSV color (hue, value, saturation) and angle. Each test action is passed to the DTW recognition process, which compares it to each of the previously generated generalized actions. Results are shown in Table 2.

To measure the influence of all dimensions (relevant or not) over the relevant ones, we perform the same comparison using only spatial features, and ignoring the rest. The results can be seen in Table 3.

These results show that our assumption is correct, and the measure of discrepancy between similar feature trajectories is lower than between different actions. This makes the recognition algorithm to correctly associate all the test trajectories with their set. Table 2 and Table 3 show the influence of additional dimensions on the comparison. As more dimensions are used, the quality of the results decays.

Table 3. DTW cost matrix: reduced test actions (lower case) vs. reduced generalized trajectories (upper case). Bold numbers represent best values (minimum discrepancy). Only spatial features are used.

	MOVE	ROTATE	CLEAN	PAINT
move	**8.15**	10251.49	11428.03	4888.67
rotate	12836.77	**8.94**	10035.21	284.87
clean	12252.87	8977.23	**13.46**	5175.71
paint	4728.14	135.77	5021.54	**14.33**

Table 4. DTW cost matrix: Cartesian test actions (lower case) vs. Cartesian generalized trajectories (upper case). Bold numbers represent best values (minimum discrepancy).

	MOVE	ROTATE	CLEAN	PAINT
move	**0.0032**	0.1594	0.0448	0.1031
rotate	0.1563	**0.0123**	0.0939	0.0430
clean	0.0234	0.0323	**0.0003**	0.0371
paint	0.0486	0.0148	0.0300	**0.0004**

4.2 *Cartesian Space Trajectory Recognition*

During the previous experiment, we also measured the Cartesian positions (X,Y,Z) of the human demonstrator's arm joints: hand tip, wrist, elbow and shoulder. This data is incorporated in this experiment to highlight the differences in feature and Cartesian comparison. The demonstrator did not attempt to execute the actions in a specific kinematic way, or even equal between repetitions, the only aim was to accomplished the named actions. When comparing test and generalized Cartesian actions (following the same scheme as previously), we obtain Table 4.

As shown in Table 4, in this case, the system also recognizes the actions correctly, but the score differences are lower. In Table 2, the correct answer is 1 to 3 orders of magnitude lower, while in Table 4 results are all quite similar. This proves that enabling CGDA, we are allowing the demonstrator to focus on task completion, rather than focusing on the kinematic consistency.

4.3 *Executing a Spatial Task: Cleaning*

The objective is for the humanoid robot to execute a generalized trajectory extracted from a set of repetitions. This trajectory contains only spatial features (X,Y,Z). The action is the previously named as 'clean'. Due to the spatial encoding of the task, only the full evolution and the individual evolution strategy have been tested.

4.3.1 Full Trajectory Evolution

The generalized trajectory has a number of points that must be reached. To reduce the number of parameters to evolve, only 3 joints of one of the robot's arms are used. Fitness is evaluated when a full evolved joint trajectory is executed, by analyzing the features of a marker in the robot hand. The generalized goal action and the measured one are compared using the DTW recognition, and the score of discrepancy is used as the fitness value to minimize.

The termination condition is set to ten generations without improvement in the fitness value. The fitness evolution of our experiment can be found in Figure 4. The results can be seen in Figure 5.

4.3.2 Individual Evolution

In this case, each point of the generalized trajectory is the target value for an individually evolved strategy. An evolved point is considered valid enough if it has an error lower than 20 mm to the reference point. Once the evolution is finished, the evolved action is executed. Differently than in the previous case, the error is measured using Euclidean distance. For a nine point trajectory, EC took an average 53 ± 12 s to calculate the whole trajectory (for 30 trials

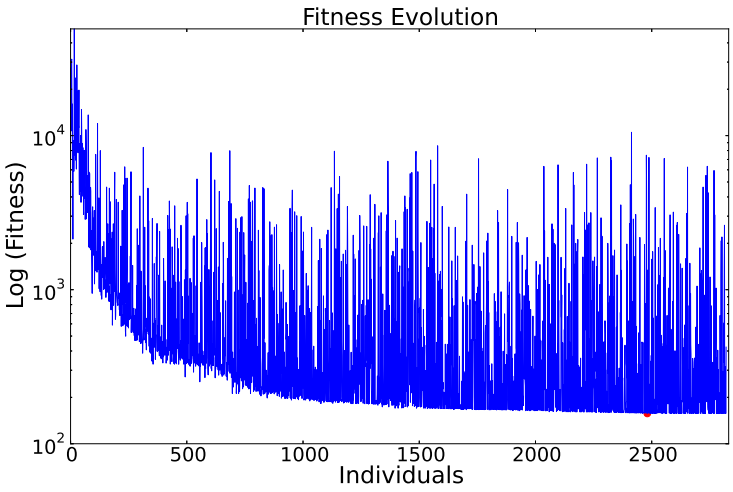

Figure 4. Fitness value through evolution. The red point is the minimum value achieved by evolution.

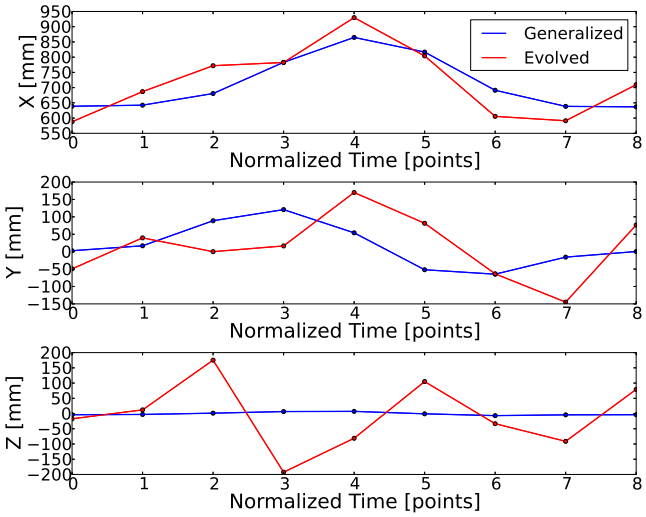

Figure 5. Unidimensional temporal plots of generalized reference (blue), and the object feature space trajectory from executing the EC winner joint position trajectory (red). The Z dimension gives the worst results, the system was not able to reduce the error in this dimension.

in an average computer), approximately 6 s to calculate each point. A comparison between the generalized trajectory and the evolved one can be found in Figure 6.

The evolved trajectories are bounded within the 20 mm limit with respect to the reference. The whole sets of trajectories (original repetitions, generalization and evolved) represented in a 3 dimensional figure, can be seen on Figure 7.

These results outperform those of the previous strategy, and also can be improved by imposing an acceptable error lower then 20 mm, but it would require more time to converge. The evolved full trajectory is depicted in Figure 8.

4.4 *Executing a Non-Spatial Task: Painting*

In the second experiment we want to make the humanoid robot paint a wall, without teaching it the necessary motor parameters. In the simulated scenario, there is a wall in front of the

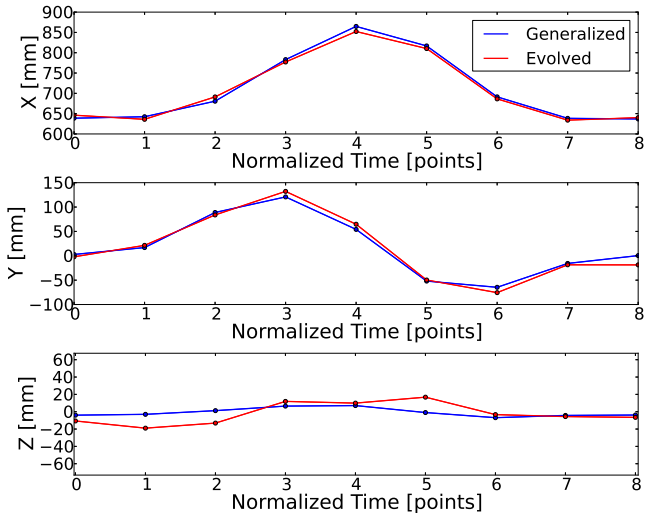

Figure 6. Unidimensional temporal plots of the generalized action, and the trajectory obtained by executing the EC winner joint parameters. The trajectories are quite similar for all cases. In the case of Z dimension, which may look like a less well-defined result, it is important to check its units. In none of the points the error is bigger than 20 mm.

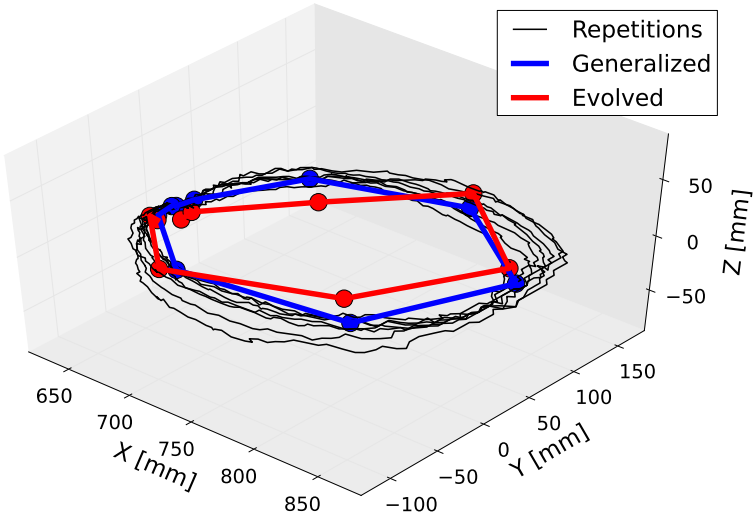

Figure 7. Plot representing a three feature trajectory. Black lines are training action repetitions. The blue line is the generalization extracted from all the repetitions. The red line is the evolved trajectory. Note that none of the points of the evolved trajectory has an error bigger than 20 mm with respect to its reference (the generalized).

humanoid robot, composed by 16 small squares, all with the same color. The final action objective is to change the color of all the squares. The generalized action is created synthetically and it has only one feature, which represents the percentage of painted wall. We suppose the percentage of the wall that is painted increases constantly with time, in 16 steps. In this case, due to the temporal dependency of the encoded feature, it is only feasible to test the incrementally evolved trajectory strategy.

4.4.1 Incrementally Evolved Trajectories

The generalized trajectory is used as the reference and EC is used to perform joint parameter evolution. In this case, every time the robot hand gets closer than 120 mm to a square, this square changes its color. This is a simplification to emulate the real action of sliding a paintbrush over a surface. Fitness is evaluated when a joint trajectory is executed, by analyzing wall features.

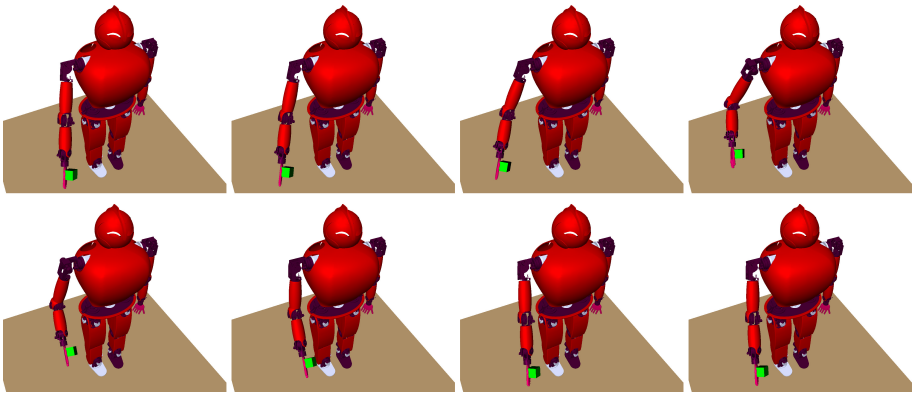

Figure 8. Action execution for a 'clean' command. Despite the trajectory is purely spatial, the objective of the evolution and the execution is to perform a feature trajectory obtained from the object tracker.

For a seventeen point trajectory, EC took 258 ± 7 s to calculate the whole trajectory (as an average of 30 trials in an average computer). It is important to notice that, as we are using the Incrementally Evolved Trajectories, the fitness evaluation gets more complex for each following point, as seen on Figure 9.

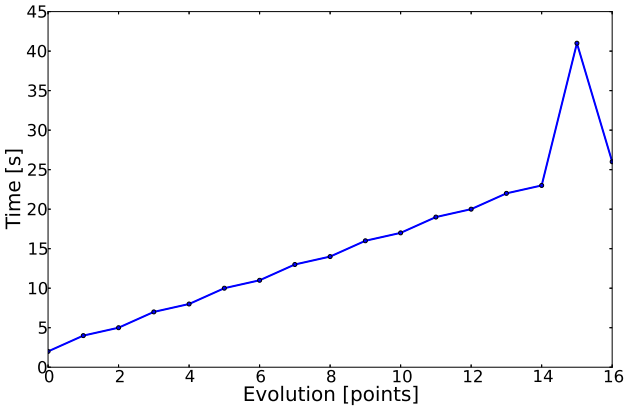

Figure 9. In this chart the time taken to compute each point is shown. Despite some final point (where the valid space is very restricted) the time of computation increases linearly.

Each point becomes more difficult to compute than the previous one because, every time a point changes the color of a square it reduces the next point valid space. The drastic increment in the 15th point is a possible effect that may appear in restricted valid spaces. As the squares are changing their color, the valid space for next computations is reduced, rising the probabilities of longer operation times. A comparison between the generalized trajectory and the evolved one can be found in Figure 10.

The trajectory obtained in this case is highly acceptable. One of the reasons behind the good convergence is the use of the incremental evolution strategy, as the evolutionary algorithm only has to focus on one fitness each time. One theoretical disadvantage of the strategy employed is that it may accumulate error of previous points in following evolutions. For instance, if the trajectory is supposed to reach a certain value in a certain point and EC is not able to reach it, it is probable that this point error will sum up to the error of the next evolved point as a constant in the fitness calculation. An example of the execution of an evolved action can be seen on Figure 11.

The criteria of success or fail for a given goal directed imitation, strongly depends on the action being imitated. In the case of the 'cleaning' task, we set an acceptable error for each feature

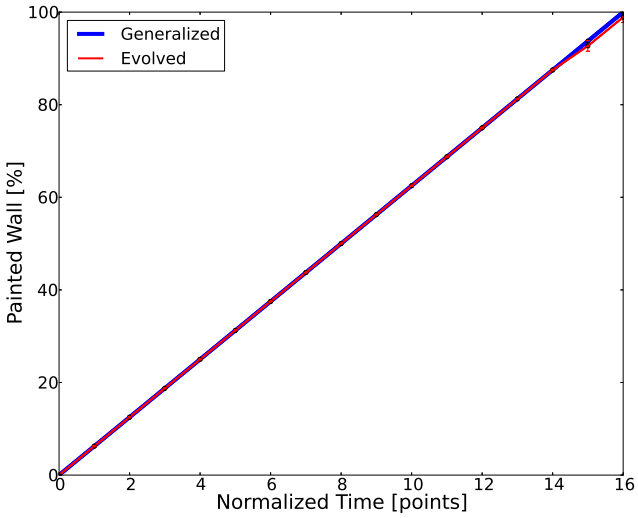

Figure 10. Unidimensional temporal plot of the generalized action, and the trajectory obtained by executing the EC winner joint parameters. As seen for a linear painting task (one wall square is painted in each step), the performance is very accurate.

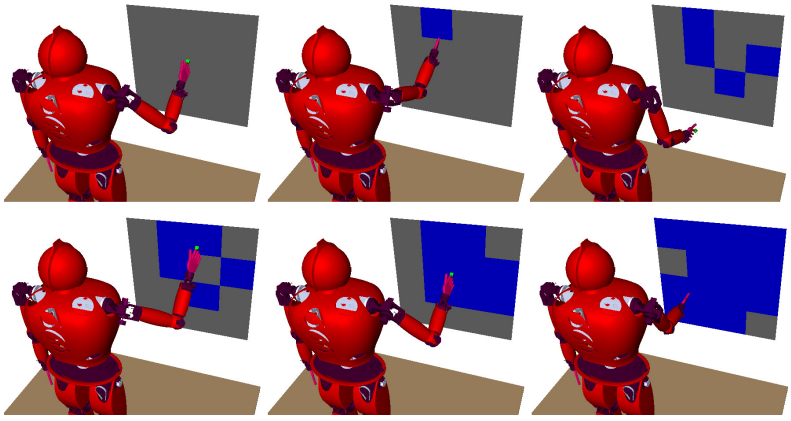

Figure 11. Action execution for a 'paint' command. Each square changes its color when the hand gets closer than a specific distance.

point. However, in the case of the 'painting' task, the criteria is related with the total number of squares painted.

5. Discussion

In this section we discuss the different strategies presented and we will analyze the results obtained. We have used both Dynamic Time Warping and the Euclidean distance for recognition. Regarding the Euclidean distance results, there is a clear improvement in the error between the trajectories, when comparing to DTW. However, a numerical comparison would not be fair, e.g. comparing DTW to the distance error would be assuming that the Euclidean distance is the "correct" metric. On its influence on execution, this change forces the system to accomplish the feature variations in the correct order. An important addition is the possibility to accomplish non-spatial features, like color. In previous works [17], we were restricted to spatial characteristics, as we had not found a method with acceptable results. We have also introduced an evolution strategy called Incrementally Evolved Trajectories (IET). This method has allowed us improve the performance of execution for non-spatial features. IET shows an incremental time of compu-

tation for a sequence of points until the valid space becomes very restricted. In these situations, we have observed peaks in the computation time. Despite this fact, it allows experiments that were previously not feasible. At this stage of development, the motor performance, in terms of velocities and acceleration, are not taken into account. Our current focus is on developing a feasible goal directed framework. Adapting this framework to work with real robots will imply a refinement of the motor commanded signals, as well as determining the optimal number of required human demonstrations.

Regarding the results obtained in this paper, we have demonstrated that our CGDA approach is useful for execution even for non-spatial features. Thanks to the use of the Euclidean distance, the trajectories can be evolved obtaining a high degree of similarity, and this error is configurable for each evolution. Further analysis will be performed in order to assure that this method is applicable to other non-spatial features.

6. Conclusions

The main contribution of this work is enabling a robot to perform tasks involving spatial features (cleaning) and non-spatial features (painting) without previous knowledge of the robot kinematic parameters necessary to perform it. The Continuous Goal-Directed Actions (CGDA) analysis allows us to focus on task accomplishment, rather than on motor imitation. This approach is novel within the robot imitation paradigm. With CGDA, we are enabling a robot-configuration independent task accomplishment, also avoiding the correspondence problem.

Additionally, Incrementally Evolved Trajectories allows us to execute a feature trajectory with temporal dependencies, such as painting a wall. Despite this is not the only strategy we consider for the future, it reaches an acceptable degree of performance in execution of tasks encoded as CGDA. One of the main challenges of the CGDA approach is how to correctly perform complex non-spatial actions. This fact will probably lead us to use motor primitives combination of actions.

Currently, one of the main challenges of the CGDA approach is how to correctly perform more complex non-spatial actions. Future research lines will include the use of motor primitives combination strategies to perform these actions.

Acknowledgments

This work was supported by ARCADIA project (DPI2010-21047-C02-01), funded by CICYT project grant, and RoboCity2030-II-CM project (S2009/DPI-1559), funded by Programas de Actividades I+D in Comunidad de Madrid and co-funded by Structural Funds of the EU.

References

[1] Billard A, Epars Y, Calinon S, Schaal S, Cheng G. Discovering optimal imitation strategies. Robotics and autonomous systems. 2004;47(2):69–77.

[2] Calinon S, D'halluin F, Sauser EL, Caldwell DG, Billard AG. Learning and reproduction of gestures by imitation. Robotics & Automation Magazine, IEEE. 2010;17(2):44–54.

[3] Schaal S. Is imitation learning the route to humanoid robots? Trends in cognitive sciences. 1999; 3(6):233–242.

[4] Nehaniv CL, Dautenhahn K. Of hummingbirds and helicopters: An algebraic framework for interdisciplinary studies of imitation and its applications. In: Interdisciplinary approaches to robot learning. Vol. 24. World Scientific. 1999. p. 136.

[5] Bekkering H, Wohlschlager A, Gattis M. Imitation of gestures in children is goal-directed. The Quarterly Journal of Experimental Psychology: Section A. 2000;53(1):153–164.

[6] Rizzolatti G, Fadiga L, Gallese V, Fogassi L. Premotor cortex and the recognition of motor actions. Cognitive brain research. 1996;3(2):131–141.
[7] Calinon S, Billard A. Stochastic gesture production and recognition model for a humanoid robot. In: Ieee/rsj international conference on intelligent robots and systems (iros 2004). Vol. 3. IEEE. 2004. p. 2769–2774.
[8] Calinon S, Billard A. Recognition and reproduction of gestures using a probabilistic framework combining pca, ica and hmm. In: Proceedings of the 22nd international conference on machine learning. ACM. 2005. p. 105–112.
[9] Calinon S, Billard A. Incremental learning of gestures by imitation in a humanoid robot. In: Proceedings of the acm/ieee international conference on human-robot interaction. ACM. 2007. p. 255–262.
[10] Calinon S, Guenter F, Billard A. On learning, representing, and generalizing a task in a humanoid robot. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics. 2007;37(2):286–298.
[11] Calinon S, Guenter F, Billard A. Goal-directed imitation in a humanoid robot. In: Ieee international conference on robotics and automation (icra). IEEE. 2005. p. 299–304.
[12] Erlhagen W, Mukovskiy A, Bicho E, Panin G, Kiss C, Knoll A, Van Schie H, Bekkering H. Goal-directed imitation for robots: A bio-inspired approach to action understanding and skill learning. Robotics and autonomous systems. 2006;54(5):353–360.
[13] Johnson M, Demiris Y. Abstraction in recognition to solve the correspondence problem for robot imitation. In: Proceedings of towards autonomous robotic systems. Springer. 2004. p. 63–70.
[14] Yamashita Y, Tani J. Emergence of functional hierarchy in a multiple timescale neural network model: a humanoid robot experiment. PLoS Comput Biol. 2008;4(11):e1000220.
[15] Mohammad Y, Nishida T. Tackling the correspondence problem. In: Active media technology. Springer. 2013. p. 84–95.
[16] Gonzalez-Fierro M, Balaguer C, Swann N, Nanayakkara T. A humanoid robot standing up through learning from demonstration using a multimodal reward function. In: Ieee-ras international conference on humanoid robots (humanoids). 2013.
[17] Morante S, Victores JG, Jardón A, Balaguer C. Action Effect Generalization, Recognition and Execution through Continuous Goal-Directed Actions. In: Ieee international conference on robotics and automation (icra). 2014.
[18] Press WH. Numerical recipes 3rd edition: The art of scientific computing. Cambridge university press. 2007.
[19] Albrecht T. Dynamic time warping. In: Information retrieval for music and motion. Springer. 2009. p. 69–84.
[20] Martínez S, Monje CA, Jardón A, Pierro P, Balaguer C, Muñoz D. Teo: Full-size humanoid robot design powered by a fuel cell system. Cybernetics and Systems. 2012;43(3):163–180.