A Two-Stage Relational Reinforcement Learning with Continuous Actions for Real Service Robots

Julio H. Zaragoza and Eduardo F. Morales

National Institute of Astrophysics, Optics and Electronics,
Computer Science Department,
Luis Enrique Erro 1, 72840 Tonantzintla, México
{jzaragoza,emorales}@inaoep.mx

Abstract. Reinforcement Learning is a commonly used technique in robotics, however, traditional algorithms are unable to handle large amounts of data coming from the robot's sensors, require long training times, are unable to re-use learned policies on similar domains, and use discrete actions. This work introduces *TS-RRLCA*, a two stage method to tackle these problems. In the first stage, low-level data coming from the robot's sensors is transformed into a more natural, relational representation based on rooms, walls, corners, doors and obstacles, significantly reducing the state space. We also use Behavioural Cloning, i.e., traces provided by the user to learn, in few iterations, a relational policy that can be re-used in different environments. In the second stage, we use Locally Weighted Regression to transform the initial policy into a continuous actions policy. We tested our approach with a real service robot on different environments for different navigation and following tasks. Results show how the policies can be used on different domains and perform smoother, faster and shorter paths than the original policies.

Key words: Relational Reinforcement Learning, Continuous Actions, Robotics

1 Introduction

Nowadays it is possible to find service robots for many different tasks. Due to the wide range of services that they provide, service robots usage has been increased, in recent years, in places like houses and offices. However, their complete use and acceptance will depend on its capability to learn new tasks.

Reinforcement Learning (*RL*) has been widely used and suggested as a good candidate for learning tasks in robotics. However, the use and application of traditional *RL* techniques has been hampered by four main aspects: (1) vast amount of data produced by the robot's sensors, (2) large search spaces, (3) the use of discrete actions, and (4) inability to re-use previously learned policies.

Robots are normally equipped with laser range sensors, rings of sonars, cameras, etc., which produce a large number of readings at high sample rates creating problems to many learning algorithms. Large search spaces produce very long training times which is a problem for service robotics where the state space is

continuous and a description of a state may involve several variables. Researchers have proposed different strategies, normally based on a discretization of the state space with discrete actions or with function approximation techniques. However, discrete actions produce unnatural movements and slow paths and function approximation techniques tend to be computationally expensive. Also, in many approaches, once a policy has been learned to solve a particular task, it cannot be re-used on similar tasks.

In this paper, *TS-RRLCA* (*Two-Stage Relational Reinforcement Learning with Continuous Actions*) a two stage method that tackles these problems, is presented. In the first stage, low-level information from the robot's sensors is transformed into a relational representation to characterize the set of states describing the robot's environment. The policies learned with this representation framework are transferable to other similar domains. We also use Behavioural Cloning [2], i.e., traces, to induce only a subset of relevant actions per state accelerating our policy learning process. This stage produces, in few iterations, a relational control policy with discrete actions. In the second stage, the learned policy is transformed into a relational policy with continuous actions through a fast Locally Weighted Regression (*LWR*) process.

The learned policies were successfully applied to a simulated and a real service robot performing navigation and following tasks with different scenarios and goals. It is shown how the continuous actions policies are able to produce smoother and shorter paths than the original relational policies and how the performance's quality of the tasks is similar to those performed by humans.

This paper is organized as follows, Section 2 presents related work. Section 3 describes a process to reduce the data coming from the robot sensor's. Section 4 introduces our relational representation to characterize states and actions. Sections 5 and 6 describe, respectively, the first and second stages of the proposed method. Section 7 shows experiments and results and Section 8 concludes and suggests future research directions.

2 Related Work

Recently, there has been an increasing interest in addressing adaptation for control tasks in robotics. In [8], a method to build relational macros for transfer learning in robot's navigation tasks is introduced. A macro consists of a finite state machine, i.e., a set of nodes along with rulesets for transitions and action choices. In [3], a proposal to learn relational decision trees as abstract navigation strategies from example paths in presented. These approaches use relational representations to transfer learned knowledge and use training examples to speed up learning, unfortunately, they consider only discrete actions.

In [11], the authors introduced a method that temporarily drives a robot which follows certain initial policy while some user commands play the role of training input to the learning component, which optimizes the autonomous control policy for the current task. In [4], a robot is teleoperated to learn secuences of state-action pairs that show how to perform a task. These methods reduce

the computational costs and times for developing its control scheme, but they use discrete actions and are unable to transfer learned knowledge. A strategy to allow the use of continuous actions is to approximate a continuous function over the state space. The work developed in [5] is a Neural Network coupled with an interpolation technique that approximates *Q-values* to find a continuous function over all the search space. In [1], the authors use *Gaussian Processes* for learning a probabilistic distribution for a robot navigation problem. The main drawback of these methods is the computational costs and the long training times as they try to generate a continuous function over all of the search space.

Our method learns, through a relational representation, relational discrete actions policies able to transfer knowledge between similar domains. We also speed up and simplify the learning process by using traces provided by the user. Finally we use a fast LWR to transform the original discrete actions policy into a continuous actions policy. Next sections describe in detail the proposed method.

3 Natural Landmarks Representation

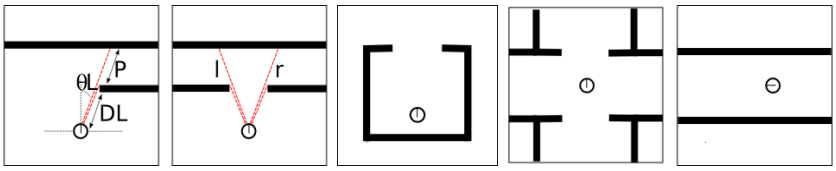
While performing a task, the robot senses and returns large amounts of data readings coming from its sensors. In order to produce a smaller set of meaningful information $TS\text{-}RRLCA$ uses a process based on [6, 10]. In [6], the authors describe a process able to identify three kinds of natural landmarks through laser sensor readings: (1) discontinuities, defined as an abrupt variation in the measured distance of two consecutive laser readings (Figure 1a), (2) corners, defined as the location where two walls intersect and form and angle and (3) walls, identified using the Hough transform. We also add obstacles identified through sonars and defined as any detected object between certain range.

A natural landmark is represented by a tuple of four attributes: (DL , θL, A, T). DL and θL are, respectively, the relative distance and orientation from the landmark to the robot. T is the type of the landmark: l for left discontinuity, r for right discontinuity (Figure 1b), c for corner, w for wall and o for obstacle. A is a distinctive attribute and its value depends on the type of the landmark, for discontinuities A is depth and for walls A is its length.

In [10] the data from laser readings is used to feed a clustering based process which is able to identify the robot's actual location such as room, corridor and/or intersection (the location where rooms and corridors meet). Figures 1c, 1d and 1e show examples of the resulting location clasification process.

Table 1 shows an example of the resulting data from applying this processes to the laser and sonar readings from Figure 2. The robot's actual location in this case is *in-room*.

The natural landmarks along with the robot's actual location, are used to characterize the relational states that describe the environment.



(a) Discontinuity detection. (b) Discontinuity types. (c) Room. (d) Intersection. (e) Corridor.

Fig. 1: Discontinuities attibutes and locations detected through a clustering processes.

Table 1: Indentified natual landmarks from the sensor's readings from Figure 2.

N	DL	θL	A	T
1	0.92	-17.60	4.80	r
2	1.62	-7.54	3.00	l
3	1.78	17.60	2.39	l
4	0.87	-35.70	1.51	w
5	4.62	-8.55	1.06	w
6	2.91	-6.54	1.88	w
7	1.73	23.63	0.53	w
8	2.13	53.80	2.38	w
9	5.79	-14.58	0.00	c
10	2.30	31.68	0.00	c
11	1.68	22.33	0.00	c
12	1.87	-170.00	0.00	o
13	1.63	-150.00	0.00	o
14	1.22	170.00	0.00	o
15	1.43	150.00	0.00	o

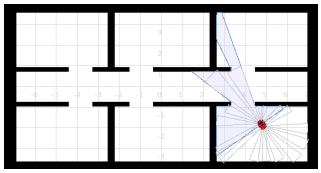


Fig. 2: Robot sensing its environment through laser and sonar sensors.

4 Relational Representations for States and Actions

In order to learn transferable policies, able to fulfill their goals under different conditions or even in different domains, a relational representation, where it is easy to encode the relative position of an agent with respect to a goal or to other objects in the environment, is used. The main idea is to represent states as sets of properties that can be used to characterize a particular state and which may be common to other states. A relational state (*r-state*) is a conjuntion of first order predicates. Our states are characterized by the following predicates which receive as parameters a set of values such as those shown in Table 1.

- *place*: robot's location. Evaluates if the received location value is valid. The valid values are *in-room*, *in-door*, *in-corridor* and *in-intersection*.
- *doors_detected*: orientation and distance to doors. A door is characterized by identifying a right discontinuity (r) followed by a left discontinuity (l) from the natural landmarks. The door's orientation angle and distance values are calculated by averaging the values of the right and left discontinuities angles and distances. The values used for door orientation are: *right* (door's angle between -67.5° and -112.5°), $left$ (67.5° to 112.5°), $front$ (22.5° to -22.5°),

back (157.5° to -157.5°), *right-back* (-112.5° to -157.5°), $right\text{-}front$ (-22.5° to -67.5°), $left\text{-}back$ (112.5° to 157.5°) and $left\text{-}front$ (22.5° to 67.5°). The values used for distance are: *hit* (door's distance between $0m.$ and $0.3m.$), *close* ($0.3m.$ to $1.5m.$), *near* ($1.5m.$ to $4.0m.$) and far ($> 4.0m.$).
- *walls_detected*: length, orientation and distance to walls (type w landmarks).[1] The possible values for wall's size are: *small* (length between $0.15m.$ and $1.5m.$), *medium* ($1.5m.$ to $4.0m.$) and *large* ($> 4.0m.$).
- *corners_detected*: orientation and distance to corners (type c landmarks).[1]
- *obstacles_detected*: orientation and distance to obstacles (type o landmarks).[1]
- *goal_position*: relative orientation and distance between the robot and the current goal. Receives as parameter the robot's current position an the goal's current position, though a trigonometry process, the orientation and distance values are calculated and then discretized.[1]
- *goal_reached*: Indicates if the robot is in its goal position. Possible values are *true* or *false*.

The previous predicates tell the robot if it is in a room, a corridor or an intersection, detect walls, corners, doors, obstacles and corridors and give a rough estimate of the direction and distance to the goal. Analogous to r-states, r-actions are conjuntions of the following first order logic predicates. These predicates receive as parameter the odometer's speed and angle readings.

- *go*: robot's actual moving action. Its possible values are $front$ (speed $>$ $0.1m./s.$), *nil* (-$0.1m./s.$ $<$ speed $<$ $0.1m./s.$) and *back* (speed $<$ -$0.1m./s.$).
- *turn*: robot's actual turning angle. Its possible values are *right* (angular speed $<$ -$0.1rad./s.$), *nil* (angular -$0.1rad./s.$ $<$ speed $<$ $0.1rad./s.$) and $left$ (angular speed $>$ $0.1rad./s.$).

Table 2 shows an r-state-r-action pair generated with the previous predicates which corresponds to values from Table 1. As can be seen, some of the r-state predicates (doors, walls, corners and obstacles detection) return the orientation and distance values of every detected element. The r-action predicates return the odometer's speed and angle values. These values are used in the second stage of the method. The discretized values, i.e., the r-states and r-actions descriptions, are used to develop a relational policy as described in the next section.

5 TS-RRLCA First Stage

TS-RRLCA starts with a set of human traces of the task that we want the robot to learn. A trace ($\tau_k = \{f_{k_1}, f_{k_2}, ..., f_{k_n}\}$) is a log of all of the odometer, laser and sonar sensor's readings of the robot while it is performing a particular task. A trace-log is divided in frames; every frame is a register with all the low-level values of the robot's sensors ($f_{k_j} = \{laser_1 = 2.25, laser_2 = 2.27, laser_3 = 2.29, ..., sonar_1 = 3.02, sonar_2 = 3.12, sonar_3 = 3.46, ..., speed = 0.48, angle = 0.785\}$) at a particular time.

[1] The values used for orientation and distance are the same as with doors.

Table 2: Resulting r-state-r-action pair from the values in Table 1.

r-state	r-action
place(in-room), doors_detected([[front, close, -12.57, 1.27]]), walls_detected([[right-front, close, medium, -35.7, 0.87], [front, far, small, -8.55, 4.62], [front, near, medium, -6.54, 2.91], [left-front, near, small, 23.63, 1.73], [left-front, near, medium, 53.80, 2.13]]), corners_detected([[front, far, -14.58, 5.79], [front, near, 31.68, 2.30], [left-front, near, 22.33, 1.68]]), obstacles_detected([[back, near, -170.00, 1.87], [right-back, near, -150.00, 1.63], [back, close, 170.00, 1.22], [left-back, close, 150.00, 1.43]]), goal_position([right-front, far]), goal_reached(false),	go([nil, 0.1]), turn([right, 1.047]).

Once the set of traces has been given $(\tau_1, \tau_2, ...\tau_m)$, every frame in the traces, is transformed into natural landmarks along with the robot's location. This data is given to the first order predicates to evaluate the set of relations, i.e., generate the r-state and the r-action (as the one shown in Table 2). Every r-state-action pair is stored in a database (DB). Table 3 gives the algorithm for this Behavioural Cloning (BC) approach. At the end of this BC approach, DB contains r-state-r-action pairs corresponding to all the frames of the set of traces.

Table 3: Behavioural Cloning Approach.

Given a set of trace-logs in the form of frames
For each frame
Transform the low-level sensor data into natural landmarks and robot's location.
Use the natural landmarks and the robot's location to evaluate the r-state and r-action predicates.
$DB \leftarrow DB \cup$ new r-state-r-action pair.

As the traces correspond to different examples and they might have been generated by different persons, for the same r-state, several r-actions might have been performed. RL is used to develop a control policy that selects the best r-action in each r-state so the robot can acomplish its tasks.

5.1 Relational Reinforcement Learning

As means to select the best action for every state, an RL algorithm is applied. The RL algorithm selects, most of the time, the r-action that produces the greatest expected reward among the possible r-actions in each r-state. We used only the information from traces, so, only a smaller subset of possible actions, for

every state, will be considered which significantly reduces the search space. Table 4 gives the pseudo-code for the *rQ-learning* algorithm. This is very similar to the *Q-learning* algorithm, except that the states and actions are characterized by relations. Through this RL algorithm, the r-action that best serves to acomplish the task, is selected.

Table 4: rQ-learning algorithm.

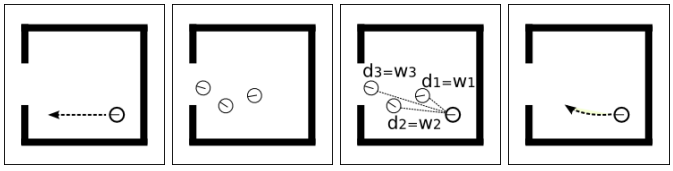
Initialize *Q(S,A)* arbitrarily (where S is an *r-state* and A is an *r-action*)
Repeat (for each episode):
  Initialize s
  (s represents the values of the sensor's readings of the robot in the current state)
  Transform s into natural landmarks and obtain the robot's actual location.
  $S \leftarrow$ *r-state(s)* % Send the natural landmarks and the location to
  the first order predicates to generate the correspondig r-state.
  Repeat (for each step of episode):
    Search the r-state registers, along with its corresponding discretized
    r-actions A, in DB, which equals the S discretized values.
    Choose an action A using a *persistently exciting* policy (e.g., ϵ-greedy).
    Take action A, observe *s'* (the new state) and r (the new state reward).
    S' ← r-state(s')
    $Q(S,A) \leftarrow Q(S,A) + \alpha(r + \gamma max_{A'} Q(S',A') - Q(S,A))$
    $S \leftarrow S'$
  until s is terminal

Through our relational representation, policies can be transfered to different, although similar domains or tasks. Even that learned policies can be re-used, the actions are still discrete. In the second stage, this discrete actions policy is transformed into a continuous actions policy.

6 TS-RRLCA Second Stage

This second stage refines the coarse actions from the discrete actions policy previously generated. This is achieved using LWR. The idea is to combine discrete actions' values given by that policy with the action's values previously observed in traces. This way the robot follows the policy, developed in the first stage, but the actions are tuned through the LWR process. What we do is to detect the robot's actual r-state, then, for this r-state the previously generated discrete actions policy determines the action to be executed (Figure 3a). Before performing the action, the robot searches in DB for all of the registers that share this same r-state description (Figure 3b). Once found, the robot gets all of the numeric orientation and distance values from the read DB registers. This orientation and distance values are used to perform a triangulation process. This process allow us to estimate the relative position of the robot from previous traces with respect to the robot's actual position. Once this position has been estimated, a weight is assigned to the robot from previous traces action's values. This weight depends on the distance of the robot from the traces with respect to the actual

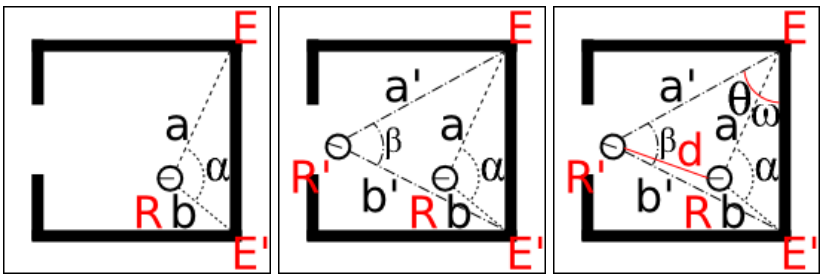
robot's position (Figure 3c). These weights are used to perform the LWR that produces a continuous r-action (Figure 3d).



(a) r-state and corresponding r-action (b) A trace segment (c) Distances and weights (d) Resulting continuos action

Fig. 3: Continuous actions developing process.

The triangulation process is performed as follows. The robot R in the actual r-state (Figure 4a), senses and detecs elements E and E' (which can be a door, a corner, a wall, etc.). Each element has a relative distance (a and b) and a relative angle with respect to R. The angles are not directly used in this triangulation process, what we use is the absolute difference between these angles (α). The robot reads from DB all of the registers that share the same r-state description, i.e., that have the same r-state discretized values. The numerical angle and distance values from these DB registers correspond to the relative distances (a' and b') from the robot of traces R' relative to the same elements E and E', and the corresponding angle β (Figure 4b). In order to know the distance between R and R' (d) through this triangulation process, equations 1, 2, 3 and 4 are applied.



(a) R robot's r-state and identified elements. (b) R' robot from traces. (c) Elements to be calcultated.

Fig. 4: Triangulation process.

$$\overline{EE'} = \sqrt{a^2 + b^2 - 2ab\ cos(\alpha)} : \ \textit{Distace between } E \textit{ and } E'. \tag{1}$$

$$\theta + \omega = arcsec(a'/\overline{EE'}) : \ \textit{Angle between } a' \textit{ and } \overline{EE'}. \tag{2}$$

$$\omega = arcsen(a/\overline{EE'}) : \ \textit{Angle between } a \textit{ and } \overline{EE'}. \tag{3}$$

$$d = \sqrt{a^2 + a'^2 - 2aa'\ cos(\theta)} :\ Distance\ between\ R\ and\ R'. \quad (4)$$

These equations give the relative distance (d) between R and R'. Once this value is calculated, a Kernel is used to assign a weight (w) value. This weight is multiplied by the speed and angle values of the R' robot's r-action. The resulting weighted speed and angle values, are then added to the R robot's speed and angle values. This process is applied to every register read from DB whose r-state description is the same as R and is repeated everytime the robot reaches a new r-state. The main advantage of our approach is the simple and fast strategy to produce continuous actions policies that, as will be seen in the following section, are able to produce smooth and shorter paths in different environments.

7 Experiments

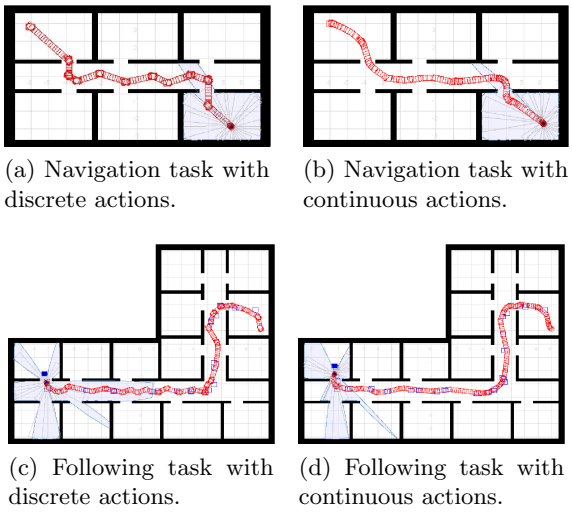
Experiments were carried out in simulation (*Player/Stage* [9]) and with a real robot[2]. Both robots (simulated and real) are equiped with a 180° front *SICK* laser sensor and an array of 4 back sonars (-170°, -150°, 150° and 170°). The laser range is $8.0m$ and for sonars is $6.0m$. The tasks referred in these experiments were navigating through the environment and following an object.

The policy generation process was carried out in the map 1 shown in Figure 5. For each of the two tasks a set of 15 traces was generated in this map. For the navigation tasks, the robot and the goal's global position (for the goal_position predicate) were calculated by using the work developed in [6]. For the "following" tasks we used a second robot which orientation and angle were calculated through laser sensor. To every set of traces, we applied our BC approach to abstract the r-states and induce the relevant r-actions. Then, *rQ-learning* was applied to learn the policies. For generating the policies, *Q-values* were initialized to -1, $\epsilon = 0.1$, $\gamma = 0.9$ and $\alpha = 0.1$. Positive reinforcement, r, (*+100*) was given when reaching a goal (within 0.5 m.), negative reinforcement (*-20*) was given when the robot hits an element and no reward value was given otherwise (*0*). To generate the continuous actions policy, LWR was applied using a Gaussian Kernel for estimating weights. Once the policies were learned, experiments were executed in the training map with different goal positions and in two new and unknow environments for the robot (map 2 shown in Figure 5c and map 3 shown in Figure 6). A total of 120 experiments were performed: 10 different navigation and 10 following tasks in each map, each of this tasks executed first with the discrete actions policy from the first stage and then with the continuous actions policy from the second stage. Each task has a different distance to cover and required the robot to traverse through different places. The minimum distance was $2m$. (Manhattan distance), and it was gradually increased up to $18m$.

Figure 5 shows a navigation (map 1) and a following task (map 2) performed with discrete and continuous actions respectively.

Figure 6, shows a navigation and a following task performed with the real robot, with the discrete and with the continuous actions policy.

[2] An *ActivMedia GuiaBot*, `www.activrobots.com`



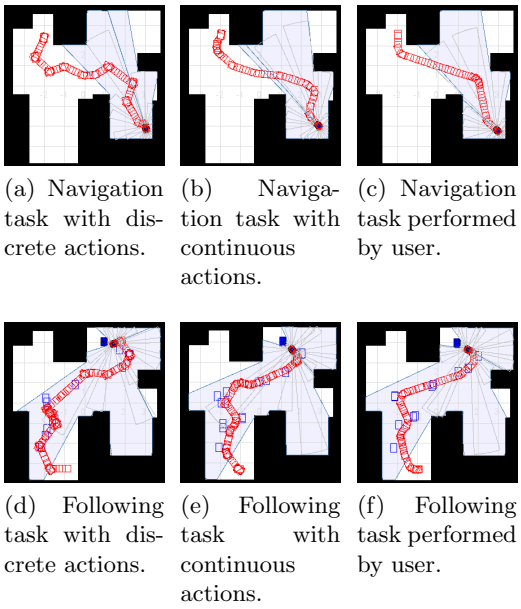
(a) Navigation task with discrete actions.

(b) Navigation task with continuous actions.

(c) Following task with discrete actions.

(d) Following task with continuous actions.

Fig. 5: Tasks examples from Maps 1 (size $15.0m. \times 8.0m.$) and 2 (size $20.0m. \times 14.0m.$).



(a) Navigation task with discrete actions.

(b) Navigation task with continuous actions.

(c) Navigation task performed by user.

(d) Following task with discrete actions.

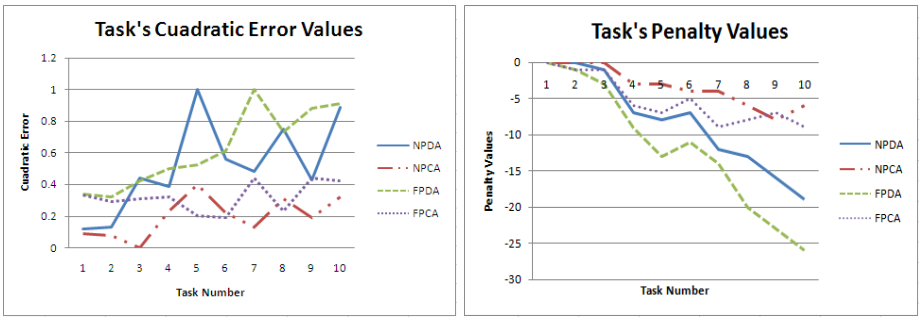
(e) Following task with continuous actions.

(f) Following task performed by user.

Fig. 6: Navigation and following tasks examples from Map 3 (size $8.0m. \times 8.0m.$).

If the robot reached an unseen r-state, it asks for guidance to the user. Through a joystick, the user indicates the robot which action to execute and the robot stores this new r-state-r-action pair. As the number of experiments increased the number of unseen r-states is reduced.

Figure 7a[3] shows results in terms of quality of the performed tasks with the real robot. This comparison is made against tasks performed by humans. All of the tasks performed in experiments with the real robot, were also performed by a human using a joystick (Figures 6c, 6f), and logs of the paths were saved. The graphic shows the normalized cuadratic error between these logs and the trayectories followed by the robot. Figure 7b shows results in terms of how much of the environment's free space, the robot uses. This comparison is made using the work developed in [7]. In that work, values were given to cells acordingly to its proximity to objects or walls. The closer the cell is to an object or wall the higher cost is given. Values were given as follows: if the cell is occupied ($0m.$ to $0.3m.$) a value of -100 was given, if the cell is close to an object ($0.3m.$ to $1.0m.$) a value of -3, if the cell is near ($1.0m.$ to $2.0m.$) a value of -1, otherwise 0. Cuadratic error and penalty values for continuous actions policies are lower than those with discrete actions.



(a) Cuadratic error values (b) Penalty values

Fig. 7: Navigation and following results of the tasks performed by the real robot.

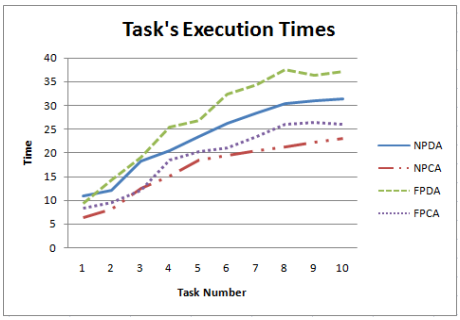


Fig. 8: Execution times results.

Policies developed with this method allow a close-to-human quality in the execution of the tasks and tend to use the available free space in the environment.

[3] **NPDA**: Navigation Policy with Discrete Actions, **NPCA**: Navigation Policy with Continuous Actions **FPDA**: Following Policy with Discrete Actions, **FPCA**: Following Policy with Continuous Actions.

Execution times (Figure 8) with the real robot were also registered. Continuous actions policies execute faster paths than the discrete actions policy despite our triangulation and LWR processes.

8 Conclusions and Future Work

In this paper we described an approach that automatically transformed in real-time low-level sensor information into a relational a representation which does not require information about the exact position of the robot and greatly reduces the state space. The user provides traces of how to perform a task and the system focuses on those actions to quickly learn a control policy, applicable to different goals and environments. LWR is then used over the policy to produce a continuous actions policy as the robot is performing the task. The actions performed in the traces are sufficient to deal with different scenarios, however, rare actions may be required for special cases. We would like to include an exploration strategy to identify such cases or incorporate voice commands to indicate the robot which action to take when it reaches an unseen state.

References

1. F. Aznar, F. A. Pujol, M. Pujol, and R. Rizo. Using gaussian processes in bayesian robot programming. *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, pages 547–553, 2009.
2. I. Bratko, T. Urbancic, and C. Sammut. Behavioural cloning of control skill. *Machine Learning and Data Mining*, pages 335–351, 1998.
3. A. Cocora, K. Kersting, C. Plagemanny, W. Burgardy, and L. De Raedt. Learning relational navigation policies. *Journal of Intelligent and Robotics System*, pages 2792–2797, Octuber 2006.
4. K. Conn and R. A. Peters. Reinforcement learning with a supervisor for a mobile robot in a real-world environment. *Proc. of the IEEE CIRA*, 2007.
5. C. Gaskett, D. Wettergreen, and A. Zelinsky. Q-learning in continuous state and action spaces. *In Australian joint Conference on AI*, pages 417–428, 1999.
6. S. F. Hernández and E. F. Morales. Global localization of mobile robots for indoor environments using natural landmarks. *Proc. of the IEEE 2006 ICRAM*, pages 29–30, Septemer 2006.
7. L. Romero, E. F. Morales, and L. E. Sucar. An exploration and navigation approach for indoor mobile robots considering sensor's perceptual limitations. *Proc. of the IEEE ICRA*, pages 3092–3097, 2001.
8. L. Torrey, J. Shavlik, T. Walker, and R. Maclin. Relational macros for transfer in reinforcement learning. *ILP*, pages 254–268, 2008.
9. R. Vaughan, B. Gerkey, and A. Howard. On device abstractions for portable, reusable robot code. *Proc. of the 2003 IEEE/RSJ IROS*, pages 11–15, 2003.
10. J. Herrera Vega. Mobile robot localization in topological maps using visual information. *Masther's thesis (to be publised)*, 2009.
11. Y. Wang, M. Huber, V. N. Papudesi, and D. J. Cook. User-guided reinforcement learning of robot assistive tasks for an intelligent environment. *Proc. of the The 2003 IEEE/RSJ IROS*, pages 27–31, October 2003.