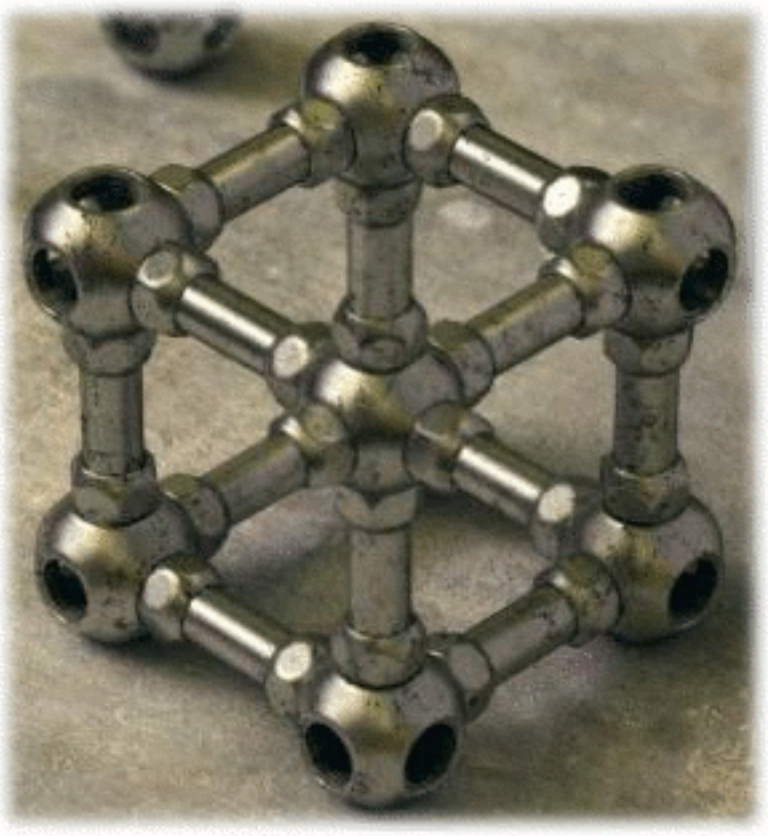

Navigation towards a goal position: from reactive to generalised learned control [1]

**Author(s):**

Valdinei Freire da Silva

Antonio Henrique Selvatici

Anna Helena Reali Costa

[1]This work was supported by Fapesp Project LogProb, grant 2008/03995-5, São Paulo, Brazil.

Navigation towards a goal position: from reactive to generalised learned control



2011 J. Phys.: Conf. Ser. 285 012025

(http://iopscience.iop.org/1742-6596/285/1/012025)

# Navigation towards a goal position: from reactive to generalised learned control

**Valdinei Freire da Silva**
Laboratório de Técnicas Inteligentes – LTI, Escola Politécnica da Universidade de São Paulo, Av. Prof. Luciano Gualberto, trav.3, n.158, Cidade Universitária São Paulo - Brazil
E-mail: valdinei.freire@gmail.com

**Antonio Henrique Selvatici**
Universidade Nove de Julho, Rua Vergueiro, 235, São Paulo, Brazil
E-mail: antoniohps@uninove.br

**Anna Helena Reali Costa**
Laboratório de Técnicas Inteligentes – LTI, Escola Politécnica da Universidade de São Paulo, Av. Prof. Luciano Gualberto, trav.3, n.158, Cidade Universitária São Paulo - Brazil
E-mail: anna.reali@poli.usp.br

**Abstract.** The task of navigating to a target position in space is a fairly common task for a mobile robot. It is desirable that this task is performed even in previously unknown environments. One reactive architecture explored before addresses this challenge by defining a hand-coded coordination of primitive behaviours, encoded by the Potential Fields method. Our first approach to improve the performance of this architecture adds a learning step to autonomously find the best way to coordinate primitive behaviours with respect to an arbitrary performance criterion. Because of the limitations presented by the Potential Fields method, especially in relation to non-convex obstacles, we are investigating the use of Relational Reinforcement Learning as a method to not only learn to act in the current environment, but also to generalise prior knowledge to the current environment in order to achieve the goal more quickly in a non-convex structured environment. We show the results of our previous efforts in reaching goal positions along with our current research on generalised approaches.

## 1. Introduction

A key task in building mobile robots is navigation, i.e., moving from an initial position to the target one. When knowledge about constraints (obstacles) and dynamics are considered to be known, the navigation problem can be addressed analytically from the point of view of control theory. However, a different approach must be used if unknown environments are considered.

Robots are physical agents that must perform their tasks autonomously in the real world. Depending on the type of the robot, different sensors input are available that must be processed so that meaningful information can be extracted. The performance of a robot is the result of its perception, given by the sensory inputs, and its control system, all integrated in a *robot architecture*. The interaction between the robot and the environment occurs through alternating

cycles of perception (acquiring sensory information) and action (choosing appropriate controls) [1].

If we consider mobile robots as autonomous agents, it is natural that the design of its architecture depends on the type of agents chosen to model the robot. Russel and Norvig [2] propose a classification of agents considering the level of abstraction of the information employed in the control of their actions. The information can represent knowledge about the environment, about the agent's objective, and about the agent itself.

Regarding knowledge about the environment, consider an agent whose actions are controlled by the current or previous short time inputs; we call it *pure reactive agent* [3, 4]. Such an agent can be programmed with condition-action rules, but due to its limited knowledge, the control can only be guaranteed to be suitable in a short period of time, although adequate long-term behaviour can be achieved, depending on task and the environment.

A more robust agent considers previous long-term inputs, constructing a model of the environment on most architectures; for this reason, we call it *model-based agent* [5, 6]. The model of the environment may rely on two classes of knowledge: knowledge about how the world evolves independently of the agent actions and knowledge about how its own actions affect the world.

Regarding knowledge about the agents's objective, instead of providing an agent with rules based only on sensor inputs (in the long or short previous time), it is possible to consider also an intrinsic objective state, which also interferes in the condition of condition-action rules. If such state models the agent's objective it can be used in order that the agent presents flexibility in dealing with changes in the world. If it is possible to model agent's objectives as a set of desired situations we call it a *goal-based agent* [7, 8]. In this case the agent can combine the knowledge about the results of its own actions in order to choose those that attain the goal (a desired situation). Planning techniques relies on models of the environment and on reasoning about future possible situations in order to provide an adequate control to attain its objective [9]. Goal-based agents divides control in two qualitative classes: controls that reach the goal and controls that do not reach the goal, however, depending on the task, it is also important to analyse each control quantitatively. This can be achieved by means of a more general performance measure that allows a comparison of the possible satisfaction level among different trajectories to a goal state. The agent can implement such measure through a utility function, which maps a sequence of states of the environment into a real number; that is why we call such agent *utility-based agent* [10, 11]. Utility-based agents are especially important in stochastic environments, where a goal cannot be guaranteed to be attained. In this case, the utility function provides a mean of pondering the probability of success with regard to the importance of the goals.

As for the agent himself, the agent can modify internal states autonomously in order to adapt to each environment; we call it *learning agent*. In addition to the model of the environment, the agent can also learn rules about how to control itself autonomously so that its objective can be attained. Reinforcement Learning [12, 13] is an interesting approach for learning how to control an agent in an unknown environment without knowledge about its dynamics, but requiring an appropriated state model of the environment, so that association can be made among states and actions.

In this paper we investigate the use of Relational Reinforcement Learning as a method to not only learn how to navigate in the current environment, but also to generalise the acquired knowledge in order to achieve the goal more quickly in an unknown non-convex but structured environment. In the next sections, we present our ongoing and planned efforts to combine all the previously presented architectures of agents in order to build a hybrid robot architecture that can reach goal positions fast in unknown environments.

Section 2 describes our first experiments with a pure reactive agent, where we combine simple motor schemas in order to pursue a goal position avoiding obstacles. Even if the robot cannot

reach the goal position in structured environment, it is highly effective in environments with convex obstacles. Section 3 describes our architecture to combine goal-based, utility-based and learning agent in order to better coordinate motor schemas so that performance is improved. Section 4 describes our algorithm to combine low level control (motor schemas) with high level control (Reinforcement Learning), where we show that although finding a goal-position randomly can be time expensive, an adequate control can be learned fast. In the direction of learning generic rules which can be optimal among similar structured environments, in section 5 we present our current efforts in applying Relational Reinforcement Learning to the navigation task of reaching a goal position.

## 2. REACT: a pure reactive architecture

The behaviours integrating our reactive architecture REACT [14, 15] are based on Arkin's motor schemas [1]. A motor schema is a behaviour encoded by the potential fields method that translates sensor readings into a movement vector directly and in a continuous fashion.

Each motor schema is composed of two modules: the perceptual schema and the encoding module. The perceptual schema is responsible for the sensorial processing, determining relevant information for controlling the robot from sensors input. The encoding module calculates a movement vector based on the potential fields method using the information returned by the perceptual schema.

The movement vector is analogous to a force derived from some potential function, usually generated by associating repulsive charges to the obstacles and attractive charges to the target position. At each instant, each motor schema calculates the forces generated by the interaction of the robot with the virtual potential field, and returns the resultant force as a movement vector.

Before calculating the final action, the influence of each motor schema must be coordinated. The potential fields method suggests a very simple way to do it: the multiplication of each motor schema response by a pondering weight. Robot actions are determined by means of weighing the movement vectors returned by each motor schemas and then performing their vectorial summation.

Encoded as motor schemas, the behaviours that integrate the REACT architecture are: avoid collision, move to goal, and move ahead. For each behaviour a pondering weight must be defined in order to coordinate the final action ($w_{AC}, w_{MTG}$ and $w_{MA}$).

The avoid-collision behaviour aims at avoiding the collision to the obstacles presented in the environment. The perceptual schema processes range readings in order to identify the location of the detectable obstacles. Then the movement vector is calculated analogously to the electrostatic force in Coulomb's law: repulsive charges are associated to each identified obstacle, generating move-away vectors with magnitudes that grow with the proximity to the obstacles. The behaviour response corresponds to the vectorial sum of the calculated vectors for all detected obstacles. The equations that determine the movement parameters for each obstacle are:

$$V(d) = \begin{cases} V_{AC}\, e^{\frac{S-d}{T}} & \text{for } d > S \\ V_{AC} & \text{for } d \leq S \end{cases} \quad \text{and} \quad \phi = \pi - \phi_{\text{rob-obst}}, \tag{1}$$

where $V$ is the response magnitude (speed), $d$ is the distance from the mass centre of the robot to the obstacle, $V_{AC}$ is the maximum speed allowed for the behaviour, $S$ is the stand off distance from the obstacle, $T$ is the scale constant for the exponential function, $\phi$ is the motion direction, $\pi$ is the mathematical number, and $\phi_{\text{rob-obst}}$ is the direction defined by the straight line that passes by the obstacle and the robot's centre of mass.

The move-to-goal behaviour aims at attracting the robot to a pre-determined location in the environment. The target position in the global coordinate system is informed by an external agent. The current robot position is determined by some localisation method. The resultant

motion direction is equal to the target direction, given by the straight line that passes by the robot's centre and the target location ($\phi = \phi_{\text{rob-targ}}$). The magnitude is given by a constant ($V = V_{MTG}$).

The move-ahead behaviour provides a certain trend for the robot of not changing its heading direction. No sensory information is used and the motion parameters are determined in a very simple manner: the magnitude is a constant ($V = V_{MA}$) and the direction is equal to the current robot heading ($\phi = \phi_{\text{robot}}$).

REACT can reach a goal very fast in small convex-obstacle worlds. If there are large obstacles to be avoided the robot may choose the most expensive direction to go, what will result in a significantly worse path. However, it does not matter how large an obstacle is, as long as it is convex, the goal will be reached. On the other hand, if there are non-convex obstacles in the world, the agent may get stuck in one of them. This happens because of the activation and deactivation of the avoid collision behaviour when inside a non-convex obstacle whose concavity is opposite to the goal.

A less restrict but essential point of view is regarding the robot performance. Depending on the parameters used in the motor schemas, the robot may perform poorly as evaluated by a given performance measure. In the next section we show how to tune the pondering weights in order to obtained a more balanced behaviour.

## 3. AAREACT: tuning reactive behaviours

AAREACT is a robot architecture that combines learning and reactive behaviours in order to not only reach the goal position, but also to improve a performance measure [15, 16]. The architecture consists of a learning apparatus which modifies parameters of the embedded REACT architecture based on the robot performance. The architecture schema is outlined in figure 1.

The learning part of the architecture is called *coordination layer*. Its role is that of adapting the values of the influence parameters that define the weight of each behavioural response in the resultant action depending on the environment situation. Defined by a suitable interpretation of the robot sensors, each situation must be mapped into a weight vector. Thus, the coordination module has to learn the best way to coordinate the behaviours regarding the current sensory data.

The situation of the environment is determined by the supervisor module present in this layer, while the critic module observes the robot performance. One should notice that both modules may use raw sensor data and the perception information determined by robots perception. However, while the supervisor module considers only current information, the critic module has an internal state being updated during the interval of one situation. When the situation changes, the critic module defines a reinforcement value that summarises how well it behaved while that situation was observed.

The learning model adopted in AAREACT is Reinforcement Learning (RL). RL is a generic model towards autonomous agents. An agent must maximise a given performance measure through trial-and-error interactions with the environment. The performance measure is described by a reinforcement function [13]. The underlying model of several RL algorithms are Markovian Decision Processes (MDP). An MDP is described by a tuple $\langle \mathcal{S}, \mathcal{A}, P(s'|s,a), r(s,a) \rangle$, where $\mathcal{S}$ is a set of states (situations), $\mathcal{A}$ is a set of actions (vectors of weights), $P(s'|s,a)$ models the probability of transition to the state $s'$ when the action $a$ is chosen in the state $s$ and $r(s,a)$ is the reinforcement incurred. The sets of states and actions are considered to be known, whereas the probabilities $P(s'|s,a)$ and reinforcements $r(s,a)$ must be inferred directly from the environment.

The RL module maintains a function $Q(s,a)$ of the estimated utility of choosing a certain vector of weights to be sent to the Merging module in each possible situation of the environment. In every change of situation, the RL module gets the new situation and the reinforcement value from the supervisor and critic modules, respectively, and updates its $Q(s,a)$ by using an RL

algorithm. Then, in most cases, the parameters chosen to be passed to the reactive layer are those considered to have the greatest utility among all for the current situation. However, random actions must also be tried in order to learn improved control rules, when it is possible.

When the agent observes a situation $s$ and executes an action $a$, it obtains a response from the environment in terms of the reinforcement $r$ which can be a reward or a penalty. Then, a new situation $s'$ is perceived. Once in this new situation, the agent has to choose a new action $a'$. Being the most popular algorithm, Q-learning [17] update is done through:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a' \in \mathcal{A}} Q(s',a') - Q(s,a)). \quad (2)$$

where $r \in \mathbb{R}$ is the reinforcement signal, $\alpha \in ]0,1[$ is the learning rate, and $\gamma$ is the discount rate. Both $\alpha$ and $\gamma$ are project parameters previously defined.

If one decides to always trust in the current result of the learning process, the adopted strategy should be the greedy one which always chooses the action to which is associated the greatest value of utility $Q$ for the situation. However, a less confident strategy is more suitable to explore the possibility of actions, what is useful mainly in the initial phases of learning, when the agent is unlike to have enough experience to decide the best action for the current situation.

Analogously to Kalmár's work [18], the environment situation space is defined by a set of on/off features, abstracted from the sensors' data. The environment situation is then defined by a vector indicating the activation or not of each feature, called features vector. The features defined for AAREACT are based on the presence of obstacles between the agent and the target position, relative direction of the target point, and relative direction and distance of obstacles, as depicted in figure 2. The environment situation is defined by the vector of features, signaling which features are on and which are off. A change of situation occurs when one or more inactive features are activated, or also when one or more activated features are deactivated.

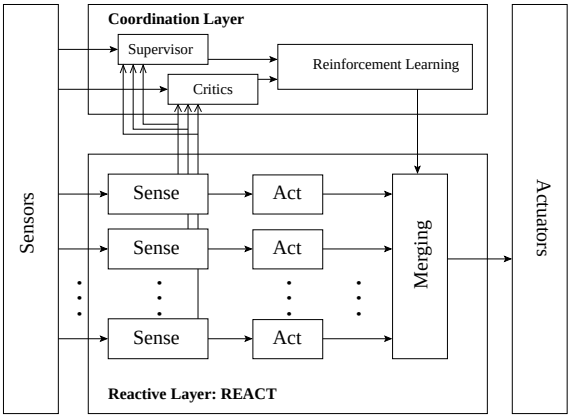


**Figure 1.** AAREACT outline.

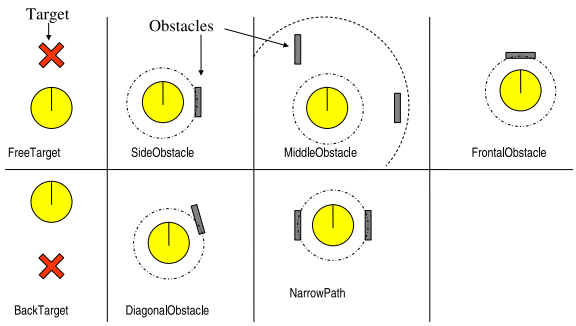


**Figure 2.** Illustration of the defined features. The robot is represented by the yellow circle, the red cross represents the target position, and the rectangles represent the detected obstacles.

Since the space of weight vectors is continuous, we proposed associating a weight vector for each feature, besides a generic weight vector which ponders all of the behaviour with non-null weights. Only weight vectors associated to features which are on and the generic weight vector are considered when choosing a coordinating action. Such consideration avoids the need of exploring clearly bad actions and allows the agent deciding autonomously which feature is more important.

In RL, positive reinforcements are used to reward desirable situations, while negative reinforcements also can be used to penalise undesirable situations. The robot's primary objective is to get to the target location. When it occurs, the critics module generates a large reward,

defined by $r_{goal}$. It is also desirable that the robot presents a good performance during its actuation, whatever is desired from the robot. Whereas time to get to the target position is clearly desirable, other performance measures, such as energy consumption, may be chosen arbitrary by a programmer.

When a situation change is detected, a reward proportional to the chosen performance measures is calculated, characterising an *intermediate reinforcement*, received before reaching the target. Although such reinforcement can be positive, it must be small when compared to $r_{goal}$, so that a trajectory that reach the target position can always be a winner trajectory.

## 4. Accelerating Goal-driven Reinforcement Learning

Whereas the avoid-collision and go-ahead behaviours can be substituted by different suitable behaviours, the same is not true regarding the move-to-goal behaviour. This behaviour is essential in order to avoid random exploration and an accurate relative position between robot and target position must be obtained, so that the agent can define such a behaviour.

In this section we explore a global state space which may be provided by simultaneous localisation and mapping techniques (SLAM) [19, 20], which can map the environment while exploring it for the target position. Mapping the world is specially important in places where the robot can get stuck. If the target position is always reached, the location of both the robot and the target could be less accurate if the sensory information could guide the robot approach to the target position. In this case, the target position could be represented as an object to be approached and an object-based map could be used [21].

When dealing with continuous state space, it is necessary to discretise the state space into discrete macro states before traditional RL algorithms can be used. The final rules of control that are learned depend on the pattern and granularity of the discretisation. While high-definition discretisation allows near optimal control, the time spent in learning grows exponentially with the number of states. An alternative to deal with this problem is to discretise the space of states non-uniformly. In navigation tasks such high-definition discretisation is mainly required near obstacles, whereas it is unnecessary in free areas [22, 23].

Besides providing a better control near obstacles, it is important to provide an appropriate continuous control. Appropriate robot architectures use two levels of control: one low-resolution discretisation of the state space to control the robot globally (figure 3), and one high-resolution discretisation of the state space to control the robot locally. Since the position of the goal is used mainly to estimate a direction to the goal (move-to-goal behaviour), discretised directions are one interesting choice to model a set of actions to apply in the low-resolution discretisation. The AAREACT or REACT architectures can be used to control continuously the robot in the low-level of the hierarchy. An alternative approach is to define a behaviour that deviates the robot from obstacles.

We propose the CFQ-Learning algorithm (Compulsory Flow Q-Learning) [24] that can do this low level control, instead of using AAREACT at the lowest level of the hierarchy. In the CFQ-Learning algorithm the robots previously learn a local behaviour to follow around obstacles, and this behaviour is named compulsory flow (figure 4). At the highest level of the hierarchy the robot chooses a main direction to go and such action is executed in the lowest level until: 1) an obstacle is found, and the compulsory flow takes control; or 2) a transition among macro-states occurs and a new action is chosen in the highest level.

Another method proposed by us is the Heuristically Accelerated Q-learning (HAQL) [25]. Whereas in its traditional definition the RL algorithm learns optimal control from scratch, HAQL uses heuristics to accelerate the learning process. Such heuristics can be introduced in two different ways: 1) by choosing an appropriate initial $Q$-function; and/or 2) by choosing appropriate actions in the trial-and-error process. Control actions provided by the AAREACT architecture is a good choice to be used as heuristics when no knowledge about the environment

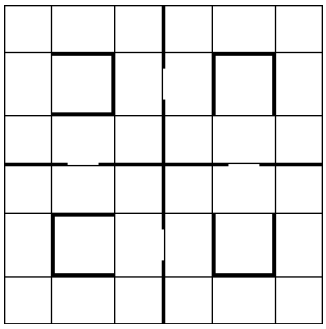

**Figure 3.** An example of discretisation with low definition and non-convex obstacles.

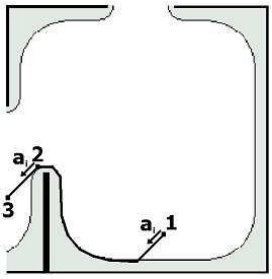


**Figure 4.** An agent's movement, which starts at point 1 and follows the compulsory flow until reaching the point 2, when it is released, reaching point 3.

is known. The distance in a straight line between two states can also be used as an optimistic heuristic function. The performance of the learning process using HAQL can be improved even using very simple heuristic functions. Also, the heuristic function can be modified or adapted online, as learning progresses and new information for enhancement of the heuristic becomes available. Another interesting feature of HAQL is that even if the heuristic is not completely adequate, performance enhancement can take place due to the partial correctness of the heuristic.

We are currently investigating such alternatives. First, by applying the CFQ-Learning algorithm in the Pioneer robot, using both the compulsory flow control coded by hand and the compulsory flow control learned directly from interactions with the environment. We are also implementing a higher level of control in the AAREACT, in the same way as CFQ-Learning, so that the compulsory flow control can be replaced by the AAREACT control. In both architectures we will also experiment RL algorithms based on heuristics so that the target position is reached more quickly.

Whereas robot navigation presents randomness in control dynamics and robot perception, if the constructed partial map is accurate and we use the assumption that connected neighbour position can be reached by the AAREACT architecture, planning algorithms can be used in order to avoid the robot getting stuck. However, RL algorithms can couple better with randomness in the control dynamics.

Since we are mainly interested in the robots reaching fast the target position in its first trial with the environment, we plan working with algorithms which keep track of previous experiences and constantly update the $Q$-function where and when it is necessary [26, 27]. In this case, the definition of the learning rate $\alpha$ should be non-stationary providing an appropriate use of initial experiences.

Prioritised Sweeping is an algorithm which keeps information about visited states-action-states triads $(s, a, s')$ and its dependence on the $Q$-values of next states $s'$. If after an experience at time $t$ the difference between the updated max value $\max_{a' \in \mathcal{A}} Q_t(s', a')$ and the previous max value $\max_{a' \in \mathcal{A}} Q_{t-1}(s', a')$ is beyond a threshold, the value $Q(s, a)$ is updated simulating the previous experiences related to the triad $(s, a, s')$. The order in which simulation occurs is prioritised on the difference regarding current update and the dependence of a triad on such difference.

## 5. Relational RL as a generic solution

Given the time needed to learn an optimal control, some works in the area of RL were devoted to the possibility of reusing learned information [28, 29]. This reuse can be done among different environments or among different states in the same task [30]. The first case is usually obtained through the use of locally controlled sub-tasks, whereas the second case is based on some state similarity measure. A global reuse of information can only occur when the representation of states is the same among environments.

Factored states are commonly used as a technique to allow reuse of information [31]. A factored state is represented by a vector of $n$ factors. Similarity among states can be measured

using a sub-set of factors. Reuse can be done in two ways: $Q$-function and actions. Sometimes the similarity is taken as equality of $Q$-function values, meaning that the states are at the same distances to the goal. In this case the value attributed to one state can be attributed to similar states. A more common approach is to consider that we can simulate a transition between two states as occurring in other similar states or that the optimal actions to be performed are the same for some set of similar states.

Whereas in the AAREACT architecture a locally factored state is considered in order to provide generalisation, the set of factors is chosen *a priori* and it is incomplete. If generalisation is possible because of such incompleteness, it is also the source of problems when such *a priori* factors are not enough. A relative new set of methods, Relational Reinforcement Learning (RRL) proposes starting with a complete set of factors and finding an adequate subset of them [32].

The RRL framework considers an MDP where the states and actions are structured into first-order propositions. First-order propositions are based on an object domain set $\mathcal{D}$ and a set of predicates $\mathbb{P}$. A ground state is a conjunction of predicates $p \in \mathbb{P}$ applied to objects $o \in \mathcal{D}$. An abstract state considers variables instead of objects when applying predicates, then representing a set of ground states. Generalisation is obtained choosing actions to abstract states.

RRL algorithms have been applied to block world problems, showing generalisation among tasks involving different numbers of blocks in the world. However, it strongly depends on the formulation of the factored states. The application of these methods to navigation problems have been less fruitful [33, 34]. Some works have taken advantage of structured buildings (rooms, halls and lifts) to infer general control rules like: entering room first needs to access a hall, accessing a hall first needs to access the lift, accessing the lift first needs to access any hall, etc. But, if buildings present connections among rooms, this kind of generalisation fails (the robot may found a target room navigating only among rooms). In this case a recover strategy is used based on a list of actions previously experienced. However, such solution does not couple with stochastic transitions in the world.

If generalisation cannot always be accurate, we believe that it can be a good source of heuristic so that we can have better guesses when exploring an unknown environment. For instance, when looking for a target point, if the robot is not far away from the target point, the AAREACT control should be a good generalisation, whereas when the robot is far away from the target point, the control rule "get access to a hall" should be a good generalisation. However, when the robot is in the hall, it can learn rules stating the disposal of visiting rooms in the opposite direction to the target and exploring rooms that are in the same direction and that had not yet been visited.

Our current research aims at formalising partial generalisation. If generalisation cannot be accomplish fully, partial generalisation may be a strong advice of how pondering appropriately the rate of exploration of each action. RRL algorithms results in the indication of an action for abstract states, i.e., for a set of ground states. However, some states may not agree about the optimal action, even if the full set of predicates is considered; in this case it should be considered the proportion of each winner action as an indication of exploration rates.

In order to apply RRL algorithms to the navigation towards a goal position, the world must be modelled regarding objects and predicates. We consider a mapping relating walls, doors and convex obstacles. Convex obstacles near the robot can immediately be observed and treated appropriately by the architecture AAREACT. Walls and doors must be identified in order to generate abstract states and abstract actions.

We use the following predicates in order to describe an abstract state: `hasDoor(X)`, `hasFreeSpace(X)`, `isDirection(X,Y)`, `targetDirection(Y)` and `isTargetVisible()`. The predicate `hasDoor(X)` indicates that a door `X` can be seen, whereas the predicate `hasFreeSpace(X)` indicates there is a free space `X`, i.e., without walls. The predicate `isDirection(X,Y)` indicates the relative direction `Y` (east, north, etc.) of a door or an open

space `X` regarding the current robot position, the predicate `targetDirection(Y)` indicates the direction `Y` of the target and `isTargetVisible()` indicates if the robot can reach the target position.

We consider three abstract actions: `moveToDoor(X)`, `moveDirection(Y)` and `moveToTarget()`. All of this actions is executed by the AAREACT architecture, where in the case of the action `moveDirection(X)` the move-to-goal behaviour keeps constant the direction to the goal.

A second and more complex direction to take consists in generalising not only among optimal actions, but also generalising among alternative actions as long as they present similar performance. Although the $Q$-value can be used to analyse the quality of an action in one step, being an indicator of possibility of improvement, when analysing worse actions it may not perform well as an indicator of performance. Policy gradient analysis may help when evaluating the true value of changing one of the known optimal actions [35].

## 6. Conclusion

In this paper we present our advances in the task of navigating towards a goal position. We divide them in two parts: reactive solutions and learned solutions.

Despite being quite simple, reactive solutions are essential in many aspects. First, they serve as an interface between the continuous analog world and the discrete digital world. Whereas intelligent solutions backed by one of these two worlds are not satisfactory, the union between them can bring the best of both worlds. Second, hand-coded motor schemas are easy ways of providing safe control to the robot. Using motor schemas and appropriate sensors, it is very easy to do implement collision avoidance, falling edge avoidance, among other risky behaviours. Third, even with these hand-coded behaviours, arbitrary performances can be learned to adapt the motor schemas to a particular environment

Our approaches to accelerate Reinforcement Learning have also been promising. Although the approaches have been developed separately from our reactive approaches, we believe that they will unite naturally. The use of Relational RL seems to be very promising, and despite some discouraging examples in navigation tasks we believe our designed approach will be fruitful.

Although not explored in this paper, SLAM has been another focus of our research, mainly regarding semantic knowledge augmenting the map with label to objects and situations. Besides fitting well with our current necessities, our mapping solution is very useful to more complex tasks involving objects. Although the task of navigating to a target appears to be largely independent from objects, navigation is a basic task in a top layer of hierarchical architectures and most often is motivated by some objects in the environment. Navigating toward a target position often comes along with interactions with objects, so that the basic navigation task would be more like: "go to the red table". Then our map solution will be very helpful in such cases.

## Acknowledgments

This work was conducted under project LogProb (FAPESP proc. 2008/03995-5). Valdinei F. Silva thanks FAPESP (proc. 09/14650-1) and Anna H. R. Costa thanks CNPq (proc. 305512/2008-0).

## References

[1] Arkin R C 1998 *Behavior-Based Robotics* (Cambridge, MA: The MIT Press)
[2] Russel S and Norvig P 2003 *Artificial Intelligence: A Modern Approach* 2nd ed (Upper Saddle River, New Jersey: Prentice Hall) ISBN 0-13-790395-2
[3] Nolfi S 2002 *Neurocomputing* **42** 119 – 145 ISSN 0925-2312
[4] Woolley B and Peterson G 2009 *Journal of Intelligent & Robotic Systems* **55**(2) 155–176 ISSN 0921-0296
[5] Hester T and Stone P 2009 *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems* (Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems) pp 717–724 ISBN 978-0-9817381-7-8

[6] Szita I and Szepesvári C 2010 *ICML* ed Fürnkranz J and Joachims T (Omnipress) pp 1031–1038
[7] Macal C M and North M J 2010 *Journal of Simulation* vol 4 (Operational Research Society) pp 151 – 162
[8] Kwiatkowska M, Norman G and Parker D 2007 *Formal Methods for Performance Evaluation* (*Lecture Notes in Computer Science* vol 4486) ed Bernardo M and Hillston J (Springer Berlin / Heidelberg) pp 220–270
[9] Bonet B and Geffner H 2001 *Applied Intelligence* **14** 237–252
[10] Haddawy P and Hanks S 1998 *Computational Intelligence* **14** 392–429
[11] Keeney R L and Raiffa H 1976 *Decisions with Multiple Objectives: Preferences and Value Tradeoffs* (New York: Wiley)
[12] Sutton R and Barto A 1998 *Reinforcement Learning: An Introduction* (The MIT press, Cambridge, MA)
[13] Kaelbling L P, Littman M L and Moore A 1996 *Journal of Artificial Intelligence Research* **4** 237–285
[14] Pacheco R N and Costa A H R 2002 *Workshop de Computação – WORKCOMP'2002* ed Sakude M T S and de A Castro Cesar C (ITA) pp 125–130
[15] Selvatici A H P and Costa A H R 2007 *Mobile Robots: The Evolutionary Approach* (The Netherlands: Springer-Verlag) chap 11, pp 161–184
[16] Selvatici A H P and Costa A H R 2007 *Revista Controle e Automação* **18** 173–186
[17] Sutton R S and Barto A G 1998 *Reinforcement Learning: An Introduction* (Massachussets, MA: MIT Press)
[18] Kalmár Z, Szepesvári C and Lörincz A 1998 *Machine Learning* **31** 55–85
[19] Bailey T and Durrant-Whyte H 2006 *Robotics and Automation Magazine* **13** 1–10
[20] Durrant-Whyte H and Bailey T 2006 *Robotics and Automation Magazine* **13** 1–9
[21] Selvatici A H P, Costa A H R and Dellaert F 2008 *IV Workshop de Visão Computacional* (Bauru, Brazil)
[22] Munos R and Moore A 2002 *Machine Learning* **49** 291–323
[23] Reynolds S I 2000 *Proceedings of the 17th International Conference on Machine Learning* pp 783–790
[24] Silva V F d and Costa A H R 2009 *Journal of the Brazilian Computer Society* **15** 65 – 75 ISSN 0104-6500
[25] Bianchi R, Ribeiro C and Costa A 2008 *Journal of Heuristics* **14**(2) 135–168 ISSN 1381-1231 10.1007/s10732-007-9031-5 URL `http://dx.doi.org/10.1007/s10732-007-9031-5`
[26] Moore A W and Atkeson C G 1993 *Machine Learning* **13** 237–285
[27] Sutton R S 1988 *Machine Learning* **3** 9–44
[28] Taylor M E, Kuhlmann G and Stone P 2008 *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems* (Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems) pp 283–290 ISBN 978-0-9817381-0-9
[29] Taylor M E and Stone P 2009 *J. Mach. Learn. Res.* **10** 1633–1685 ISSN 1532-4435
[30] Ribeiro C H C, Pegoraro R and Costa A H R 2002 *International Joint Conference on Autonomous Agents and Multi-Agent Systems AAMAS´2002* ed Castelfranchi C and Johnson W L pp 1239–1245
[31] Boutilier C, Dearden R and Goldszmidt M 2000 *Artificial Intelligence* **121** 49–107
[32] van Otterlo M 2009 *The Logic of Adaptive Behavior - Knowledge Representation and Algorithms for Adaptive Sequential Decision Making under Uncertainty in First-Order and Relational Domains* (*Frontiers in Artificial Intelligence and Applications* vol 192) (IOS Press) ISBN 978-1-58603-969-1
[33] Cocora A, Kersting K, Plagemann C, Burgard W and De Raedt L 2006 *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Beijing, China)
[34] Kersting K, Plagemann C, Cocora A, Burgard W and De Raedt L 2007 *Advanced Robotics. Special Issue on Imitative Robots* **21**
[35] Peters J and Schaal S 2008 *Neural Netw.* **21** 682–697 ISSN 0893-6080