Learning Navigation Teleo-Reactive Programs using Behavioural Cloning

Blanca Vargas and Eduardo F. Morales
National Institute of Astrophysics, Optics and Electronics
Computer Science Department
Luis Enrique Erro 1, 72840 Tonantzintla, México
Email: {blanca,emorales}@ccc.inaoep.mx

***Abstract*—Programming a robot to perform tasks in dynamic environments is a complex process. Teleo-Reactive Programs (TRPs) have proved to be an effective framework to continuously perform a set of actions to achieve particular goals and react in the presence of unexpected events, however, their definition is a difficult and time-consuming process. In this paper, it is shown how a robot can learn TRPs from human guided traces. A user guides a robot to perform a task and the robot learns how to perform such task in similar dynamic environments. Our approach follows three steps: (i) it transforms traces with low-level sensor information into high-level traces based on natural landmarks, (ii) it learns TRPs that express when to perform an action to achieve simple tasks using an Inductive Logic Programming (ILP) system, and (iii) it learns hierarchical TRPs that express how to achieve goals by following particular sequences of actions using a grammar induction algorithm. The learned TRPs were used to solve navigation tasks in different unknown and dynamic environments, both in simulation and in a service robot called Markovito.**

I. Introduction

When people go to a new place, e.g., a conference site, they normally ask for directions of places of interest, like the registration desk or a toilet, and are given general directions, like "at the end of the aisle to your right" or possibly "in room 203". People have to navigate without collisions in an unknown and dynamic environment to a particular destination point through well known natural marks like walls and doors and expected dynamic conditions, like walking people. Imagine you want your robot to learn how to perform a similar skill. You place your robot in an unknown environment and you want it to navigate to a particular point, like a charging station, but the robot has no map, and is only given the general direction of its destination point. The robot has to learn how to perform simple skills, like obstacle avoidance and orientation towards a goal, and use them to safely go to a particular goal in a dynamic and unknown environment. Since we would like the robot to use its learned skills on different environments, e.g., check in at a new hotel, it is desirable to have an expressive representation language where new environment conditions could be represented as instantiations of such representation.

In this paper, a relational approach to learn robot tasks using behavioural cloning is presented. The key idea is to show the robot what to do instead of how to do it (e.g., steering the robot avoiding obstacles), simplifying the programming effort. The examples or traces consist of low-level sensor readings that are transformed into a small set of high-level concepts based on natural landmarks. This transformed data is given to an Inductive Logic Programming (ILP) [1] algorithm to learn a set of reactive control rules known as Teleo-Reactive Programs (TRPs) [2]. Once some basic TRPs are learned, traces of more complex tasks involving previously learned TRPs are given to FOSeq, an algorithm that induces grammars able to reproduce the original human-guided traces.

The contributions of this paper are: (i) transforming a large amount of low-level sensor data into a small set of relational facts suitable for relational learning, (ii) a framework to learn simple TRPs from guided traces, (iii) an algorithm that can learn hierarchies of TRPs expressed as grammars to solve more complex tasks involving sequences of actions to satisfy a goal. We tested the approach in a robotics scenario with both simulated and real environments and show that the robot is able to accomplish several navigation tasks with the learned TRPs in different dynamic and unknown environments.

This paper is organized as follows. Section 2 reviews related work. Section 3 presents an overview of the learning approach. Section 4 presents the landmark identification process. Section 5 describes how to learn basic navigation TRPs. In Section 6 the learning of complex TRPs is explained. Section 7 presents experiments and main results and finally, conclusions and future research directions are given in Section 8.

II. Related Work

In this work we used behavioural cloning to provide examples, and ILP and grammar induction to learn TRPs for mobile robots. We review relevant related work on these areas.

Behavioural cloning [3] is a technique to learn skills from a human operator. Some successful applications include the control of cranes, bicycles and learning to fly a plane. In robotics, D'Este [4] learned to pursuit a target and to avoid obstacles using decision trees. Another application [5] shows how to teach a tracked vehicle to traverse rough terrain also learning decision trees. In both cases, a single task is executed using a propositional framework. In robotics, however, it is common to have multiple goals and it is desirable to transfer the learned skills to different, although similar, domains which is not possible in the previous work.

In [6] it is described how to learn to control an aircraft using behavioural cloning and reinforcement learning. It was shown

that with the learned policy it is possible to fly the aircraft on different missions. In [7], navigation plans are learned using behavioural cloning. From sequences of places traversed by the robot while carrying out its task, a relational decision tree is learned. This tree can be used to guide the robot on the same and on similar tasks re-using plans to reach a goal in unknown environments. Both approaches demonstrate the advantage of using a relational representation, however, they are unable to deal with dynamic elements in the environment and their actions have to be discretized.

Klingspor et al. [8] learn relational concepts such as *in_front_of_door*, *along_door*, *move_closer_to_wall*, *line, concave*, and *convex* using five levels of abstraction. These concepts are used to control a robot, however, they are unable to solve a navigation task and all the hierarchies and required information need to be defined in advance.

Teleo-reactive programs (TRPs) are sets of reactive rules that sense the environment continuously and apply an *action* whose continuous execution eventually satisfies a goal condition. In this paper, basic TRPs only include low-level actions (e.g. *turn_right*, *go_fwd)*, whereas hierarchical TRPs can include as action another TRP (e.g., *orient* and *wander*) in order to accomplish a more difficult task (e.g., *goto*). Even though TRPs have been proved to be an effective framework for robotics, they have not been exploited broadly. In [9], the authors describe a control architecture for mobile robots with a visual development environment for TRPs. The limitation is that the TRPs are manually constructed, which can be a difficult and time-consuming process, even with a visual tool. Broda and Hogger [10] present an algorithm based on the construction of an automata to learn TRPs; their approach however is propositional and is difficult to extend to real domains or use it in other instances of the problem. TRAIL [2] was able to learn basic TRPs from traces in three simulated domains: a construction world, an office delivery task, and a flight simulator. However, TRAIL was not able to construct complex TRPs and it was tested only on simulated environments. Konik and Laird [11] recently proposed a system to learn teleoreactive logic programs with behavioural cloning. Their approach, however, requires a high-level symbolic representation as input, along with a predefined hierarchy of TRPs. In our work, raw data of robot's sensors obtained from human guided traces are used to learn simple and complex TRPs that are able to guide a robot in dynamic and unknown environments.

III. Overview

This section describes general concepts and the overall TRP learning framework.

> **Definition**: A TRP is a set of clauses that continuously perform an action in the current state while their conditions are satisfied. The first clause has as condition a predicate that indicates whether a particular task has been satisfied (goal predicate).

In this paper TRPs have the following format:

$$
\begin{aligned}
&\text{trp(State,Action)} \leftarrow \text{goal\_condition(State).}\\
&\text{trp(State,Action)} \leftarrow \text{pred1(State,...),}\\
&\qquad\qquad\qquad\quad \text{pred2(State, ...), } \ldots\\
&\text{trp(State, TRP)} \leftarrow \text{pred1(State,...),}\\
&\qquad\qquad\qquad\quad \text{pred3(State, ...), } \ldots\\
&\ldots
\end{aligned}
$$

where *trp/2* is the name of the TRP, and the number of arguments of the predicate, goal_condition is a predicate that indicates when a goal has been satisfied and *preds* are conjunctions of predicates that indicate the conditions under which the action in the head of the clause should be executed.

A graphical representation can be seen in Figure 1 where the nodes represent actions: low-level actions, basic TRPs or hierarchical TRPs. The labels on arcs show the conditions for taking the actions. For instance, when a robot has to reach a place, it has to be oriented towards it. If the area between the goal and the robot is not cleared, the robot has to find a safe zone to get oriented. A simple strategy is to wander until it reaches a safe zone and then, turn toward its destination point. The graph shows a TRP hierarchy: *wander* is a basic TRP to move a robot safely through an environment. Its action set includes the low-level actions *turn-left*, *turn-right* and *go_fwd*. The hierarchical TRP *orient* turns the robot towards a target point only if the robot is located in a zone without obstacles. Otherwise, *orient* uses *wander* as an action until it is located in a safe zone.

Figure 2 shows the process for learning basic and hierarchical TRPs. A human provides traces of tasks for the robot to learn. These traces are transformed into a sequence of facts, with a high-level representation, and an ILP system is used to learn TRPs for such tasks. For more complex tasks, involving several basic TRPs, the original trace is first transformed into a high-level representation where the applicable TRPs are identified. This transformed trace with TRPs is given to an algorithm that induces a grammar, representing a hierarchical TRP, capable of recognizing the traces of more complex tasks. This process is detailed in the following sections.

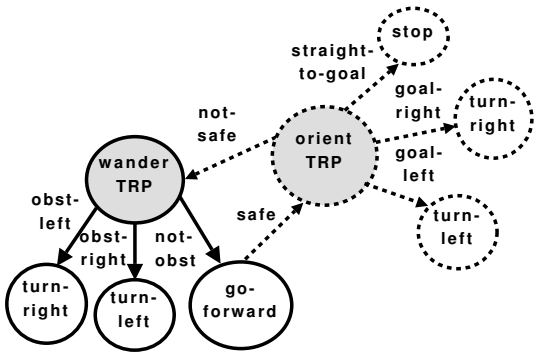


Fig. 1. Hierarchical Action Tree. *Orient* is a hierarchical TRP that turns the robot towards a goal if it is located in a safe zone. Otherwise, the *wander* TRP is used as an action and returns the control to *orient* until the action *go_fwd* applies.

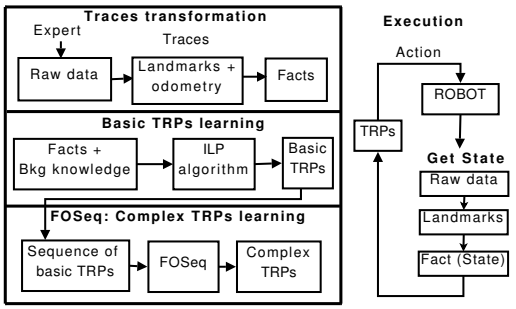


Fig. 2. An overview of how to learn basic and complex TRPs

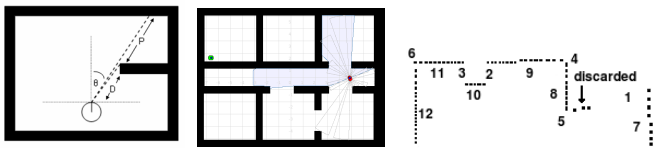
IV. Landmark Representation

Given a trace of low-level raw data from the sensors of a robot[1], a natural landmark identification process [12] is used to produce a smaller set of more meaningful information consisting of: (1) discontinuities, (2) corners and (3) walls. A discontinuity is defined as an abrupt variation in the measured distance of two consecutive readings of the laser (0.20cm in this paper), as shown in Fig. 3(a).

A natural landmark is represented by four attributes: D_L and θ_L are the distance from the landmark to the robot and the orientation of the landmark relative to the robot respectively, T is the type of the landmark; l for left discontinuity, r for right discontinuity, c for corner and w for walls, and A is a distinctive attribute and its value depends on the type of the landmark; depth for discontinuities and length for walls. For example, a subsample of the robot's sensors (in this case the laser readings) in a particular state, shown in Figure 3(b), is: 2.127, 2.036, . . ., 1.001, 1.192, 5.170, 5.164, ...

Here the system recognizes several discontinuities, in particular, the sudden change in values between 1.192 and 5.17 in two consecutive laser readings, that produces the following landmark: $1.19, -26.65, 3.98, r$ representing, respectively, the relative distance to the robot, its angle, the depth (difference between the readings), and its type, r, which means that this discontinuity is to the right of the robot. Fig. 3(c) and (d) show an example of sensor readings and natural landmarks identified by this process.

[1]In this paper we use information from a laser Sick-LMS200 sensor and from a ring of sonars, which is a common setting to many mobile robot's platforms.



(a) (b) (c)

No	D	θ	A	T
1	2.18	-73.0	1.41	l
2	1.10	-6.0	0.40	l
3	1.11	10.0	0.41	r
4	2.17	-47.0	0	c
5	1.71	-68.0	0	c
6	1.84	36.0	0	c
7	3.46	-89.50	1.06	w
8	1.59	-89.75	0.85	w
9	1.49	0.0	1.36	w
10	1.10	0.0	0.39	w
11	1.50	0.0	0.80	w
12	1.10	89.75	1.43	w

(d)

Fig. 3. (a) A discontinuity to the right (r). Attributes: *angle*(θ) with respect to the front of the robot, *distance*(D) between the robot and the discontinuity, *depth*(P) between two contiguous readings. (b) Example of a particular state of the robot showing its laser readings. (c) Laser readings. Points are discarded if they are below a threshold value. (d) Set of identified natural landmarks

To summarize, this process receives as input a set of low-level sensor readings from a mobile robot and produces as output a set of identified natural landmarks with information about walls, corners and discontinuities. This is important because the huge amount of low level and noisy data produced by many sensors in robotics has limited the use of machine learning techniques with expressive representation languages. Our process is able to automatically transform in real-time low-level readings into a suitable representation for a relational learning algorithm and opens the possibility of using other machine learning algorithms in this area.

V. Learning basic navigation TRPs

The objective of this phase is to learn simple navigation TRPs. A trace is created with information of all the natural landmarks identified by the robot sensors, with the robot position relative to the map of the environment provided by a localization process [12], and information if there is an obstacle at the rear of the robot using the ring of sonars. This information represents the current State of the robot. The *Action* is also identified on each State from the robot's odometry: *go_fwd*/*go_backward* (a displacement speed of at least 0.3 m/s), *turn_right*/*turn_left* (a 5 deg/s turnspeed) and stop. For instance, the position of the robot in Figure 3(b) produces the following state:

[robot(4.85,-0.04,248.60), rear(n), goal(0,0),
landmark(1.19,-26.65,3.98,r), landmark(2.17,30.67,0.78,l),
landmark(0.78,-29.66,0.00,c), landmark(0.79,-30.67,1.85,w), . . .]

where: (i) robot(4.85,-0.04,248.60) indicates the robot's pose: robot(X, Y, Theta), (ii) rear(n) informs if there are obstacles in the rear of the robot. It can get the values "y" or "n". (iii) goal(0,0) is the X and Y position of the point to reach, and (iv) landmark(1.19,-26.65,3.98,r), landmark(2.17,30.67,0.78,l),... correspond to the set of landmarks obtained from the laser readings. The user steers the robot to produce certain desirable behavior. The information gathered from the sensors during the process are transformed into traces of target predicates with two arguments: the current State, as previously described, and the *Action*. The goal is to produce a set of predicates that return a particular action, capturing the intended behaviour, given information from the current state.

To navigate through an office/house environment some useful skills are: (i) moving through the environment without collisions (*wander*), (ii) getting oriented towards its target point, and (iii) leaving traps. A trap is a narrow space where the robot can enter but has difficulty in leaving. Each skill can be learned as a single TRPs. Algorithm 1 summarizes the steps to learn basic TRPs. To illustrate the process, we show how to learn the *wander* TRP (see Fig 1).

Algorithm 1 Learning basic TRPs

Let Pos = High-level traces produced by guiding the robot to execute the task
for each Pos example **do**
  Replace the action with a different action from the action set
  Add the example to Neg
end for
if there are $Concepts$ to learn **then**
  Learn $Concepts$ with ALEPH
end if
Let $Bknowledge$ = $Concepts \cup$ additional knowledge (if available)
Given Pos, Neg and $Bknowledge$
Learn TRPs with ALEPH

Training examples. A set of traces where the user navigates in an unknown environment avoiding possibly dynamic obstacles was given to the system. The low-level information from the sensors and odometry of the robot are transformed into high-level state-action pairs with 1,617 instances: 525 for the action *turn_right*, 544 for the action *turn_left*, and 548 for the action *go_fwd*. Negative examples are automatically generated from the traces by interchanging actions, in this case *go_fwd* with *turn_left* or *turn_right* actions.

Background knowledge. The user can provide simple background knowledge to learn the TRPs. Table I shows the possible background knowledge and actions for *wander*: (i) the front-zone(State,obstacle) predicate is true if there is an obstacle in front of the robot, (ii) the predicate closest-obst(State,Lmk,Distance,Ang) extracts from State, the closest landmark (Lmk) and its distance and angle from the robot, and (iii) the relations less than (lteq) and greater than (gteq) can be used to learn the value range for Distance and Angle.

The high-level traces of state-action pairs and the background knowledge are given to an Inductive Logic Programming (ILP) system called ALEPH [13] to learn TRPs. ALEPH selects an instance from the trace and builds the most specific clause (*bottom clause*) by selecting a set of facts from the background knowledge that are true for this instance. ALEPH searches for some subset of the literals in the bottom clause that is more general and selects the clauses that cover many positives examples but few negative examples.

Concept Learning. Depending on the teacher's guidance, the distance criteria to avoid obstacles can be different. The user may show a "cautious" or a "daring" behavior. Distance and angular ranges capture the differences between guidances styles. This information can be learned from the traces. Although the learning process is similar to the basic TRP learning algorithm, the main differences are: (i) the traces are snapshots of the environment where the robot is located. The user provides the class value that describes the zone (e.g. obstacle-free), and (ii) negative examples are generated by obtaining snapshots of different environment zones.

The learned concept front-zone receives the State as input argument. Its possible values are: obstacle if the distance from the closest landmark is less than 0.49m and obstacle-free otherwise. This concept was learned and included as background knowledge.

front-zone(State,obstacle) ←
  closest-obst(State,_,Distance,_),
  lteq(Distance,0.49).
front-zone(State,obstacle-free).

Learned TRPs: *wander*. Given positive and negative examples and background knowledge to ALEPH [13], the *wander* TRP was learned. It can take the values: *go_fwd*, if there is no obstacle in front of the robot, *turn-left* if there is an obstacle to the right (i.e. orientation from the robot is less than -1.51rads), and *turn-right*, otherwise.

wander(State,*go_fwd*) ←
  front-zone(State,obstacle-free).
wander(State,*turn-left*) ←
  front-zone(State,obstacle),
  closest-obst(State,Lmk,Dist,Ang),
  lteq(Ang,-1.51)).
wander(State,*turn-right*) ←
  front-zone(State,obstacle).

In addition to *wander*, the following TRPs have been learned:

- ***Orient.*** Given a target point, the robot has to orient toward it. If there is a wall between the goal and the robot, there is no point to get oriented because the robot will never go through the wall. Therefore, the robot has to wander until it reaches a safe orientation zone.
- ***Leave-a-trap.*** If the robot enters in a narrow place where there is not enough space to turn, it has to go backward and turn according to the available space and leave the trap.
- ***Follow a mobile object.*** The robot has to follow a mobile object keeping a safe distance from it.

To summarize this process, the user guides the robot to perform a particular skill, like obstacle avoidance. These traces are transformed into state-action pairs that are used to learn a set of relational rules. The rules are then used to control a robot to perform that skill in similar dynamic environments. This learning strategy provides the robot with some basic skills. The next section describes how to learn more complex behaviours involving different sequences of actions and combining previously learned basic skills.

TABLE I
WANDER: BACKGROUND KNOWLEDGE AND ACTION SET

Background knowledge	Action set
front-zone(State,obst-free)	*go_fwd* (goal)
rear-zone(State,obst)	*turn-left*
rear-zone(State,obst-free)	*turn-right*
closest-obst(State,Lmk,Distance,Ang)	
lteq(Ang,Angval1)	
gteq(Ang,Angval2)	

VI. FOSeq: Learning Complex TRPs

The objective of FOSeq is to learn TRPs that are able to produce particular sequences of actions satisfying a particular goal. In this case, the TRPs can be expressed in terms of other TRPs. The user controls the robot to achieve a particular goal, e.g., go to a particular area. The low-level information from the sensors is transformed into a high-level trace of state-action pairs from which applicable TRPs, previously learned, are identified and replaced in the sequence. FOSeq is then used to learn how to produce similar sequences by learning Definite Clause Grammars (DCGs).

The general algorithm can be stated as follows: from a set of sequences, (i) learn a grammar for each sequence, (ii) parse all the sequences with each induced grammar, evaluate how well each grammar parses all the traces, and return the b best evaluated grammars, and (iii) apply a generalization process to the best grammar trying to cover most of the sequences.

Given a trace of predicates and a minimum initial support value ($t = 2$), the algorithm generates a list of *n-grams*, that is terms that are repeated in the sequence at least t times (support value). As in Apriori [14], the candidate *n-grams* are incrementally searched by their length, in our case, the support value is updated in order to keep only the maximum, since we are interested only in the n-grams that are more frequently repeated. n-grams candidates which support value cannot be higher than the current maximum support value are discarded. The *n-gram* with maximum support value is selected, generating a new grammar rule and replacing, in the sequence, all occurrences of the *n-gram* with a new non-terminal symbol. For example, from the sequence:

orient(State,*turn_left*),*wander*(State,*go_fwd*),
wander(State,*turn_left*), *wander*(State,*go_fwd*),
orient(State,*turn_right*),*wander*(State,*go_fwd*),
...
wander(State,*turn_right*),*wander*(State,*go_fwd*),
in-goal(State,nil)

FOSeq learned the following grammar:

R1 → *orient*(State,*turn-left*),*wander*(State,*go_fwd*)
R2 → *orient*(State,*turn-right*),*wander*(State,*go_fwd*)
S → R1,*wander*(State,*turn-left*),*wander*(State,*go_fwd*),R2
wander(State,*turn-right*),*wander*(State,*go_fwd*),
in-goal(State,*nil*)

where R1 and R2 are the rules, corresponding to the *n-grams* with more repetitions and S is the new sequence obtained by replacing the *n-grams* with the rules. This process iterates until the sequence has no longer repeated *n-grams*. Every learned grammar is used to parse all the sequences in the set of traces provided by the user. The best evaluated grammar is selected. The measure of how well the grammar parses is calculated using the following function:

$$eval(G_i) = \sum_{j=1}^{n} \frac{c_j}{c_j + f_j} \quad (1)$$

where, c_j and f_j are the number of items that the grammar is able or unable to parse, respectively. The process continues until the end of the sequence. R1 and R2 are replaced by the user's defined task name. For example, *goto*(State,*Action*) producing:

goto(State,*Action*) → *orient*(State,*turn_left*),*wander*(State,*go_fwd*)
goto(State,*Action*) → *orient*(State,*turn_right*),*wander*(State,*go_fwd*)

After learning a grammar for each sequence, FOSeq selects the grammar that best parses the set of sequences. During the evaluation process, FOSeq registers the differences between each grammar and the other sequences. FOSeq selects the grammars that have more dissimilar rules, and the highest difference in the evaluation list. The idea is to try to produce a more general grammar using information from what are considered as very different grammars. The key idea of the generalization process is to take the difference between pairs of grammars and apply adaptation rules to modify the best grammar. A set of adaptation rules decides if: (i) a rule has to be modified, or (ii) a new rule has to be added. The output is a new grammar that cover the two sequences that were used in the induction of the grammars.

Learning the *goto* task. FOSeq learned the *goto* TRP from 10 different traces from several environments, using predicates such as *orient* and *wander*, that were learned using ALEPH. The goal predicate (in_goal(State)), given by the user, is true when the robot has reached the intended destination. The possible values for goto are: (i) the *Action* generated by the *wander* TRP if the robot is oriented towards the goal, (ii) the *Action* generated by the *orient* TRP if the robot does not have obstacles surrounding it, and (iii) the *Action* produced by the *wander* TRP, otherwise.

goto(State,*nil*) ←
in_goal(State).
goto(State,*Action*) ←
orient(State,*nil*),
wander(State,*Action*).

goto(State,*Action*) ←
wander(State,*go_fwd*),
orient(State,*Action*).
goto(State,*Action*) ←
wander(State,*Action*).

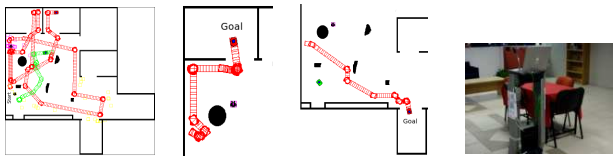
The *goto/2* TRP is able to go to a particular place in an unknown environment avoiding possibly dynamic objects.

Learning from human-guided traces to perform basic skills, like obstacle avoidance, and more complex tasks, like going to a particular destination point, is an important step towards facilitating the programming effort of skills in service robots and extends their applicability to achieve goals according to the user's needs. To summarize, FOSeq learns from human traces a generalized grammar to build hierarchical TRPs. This process allows the robot to perform complex tasks by reusing previously learned skills.

VII. Experiments and Results

The learned TRPs were tested (i) in simulation and evaluated by the percentage of tasks successfully completed, and the number of operator interventions (e.g., if the robot enters in a loop), and (ii) in a service mobile robot called Markovito [15]. The environment maps used to test the TRPs were different from those used to train the robot.

Simulation. We evaluate the performance of the learned TRPs in different scenarios, with different obstacles' sizes and shapes, and with static and dynamic obstacles to test the



(a) (b) (c) (d)

Fig. 4. Wander: fixed (a) and dynamic (b) obstacles; Goto: (c) and (d).

robot's capability to react to unexpected events without losing its goal. The robot's initial pose and the goal point (when applicable) were randomly generated. In the experiments, an operator intervention is required when the robot: (i) enters in a loop, or (ii) gets trapped and it has to be released. In each experiment two operator interventions are allowed, otherwise, the experiment fails.

- *Wander*. We ran 18 experiments in four different environments to test only the *wander* TRP. The experiment succeeds if the robot does not collide. The robot completed successfully 88.9% and 5 operator interventions were required (see Figure 4(a)).
- *Orient*. We performed 20 experiments for the TRP, which succeeded if the robot oriented correctly towards the goal. It succeeded in 90% of the given tasks.
- *Leave-a-trap*. The robot is located into a trap and it has to leave it successfully. We performed 25 experiments and the robot succeeded in 92% of the tasks.
- *Goto*. In these experiments, the hierarchical TRP *goto*, learned by FOSeq is evaluated. We performed 120 experiments in 10 different environments. The experiment succeeds if the robot reaches the goal. From the 120 given tasks, the robot succeeded in 87.5%, and 10 operator interventions were required. Two examples are shown in Figure 4.
- *Follow a mobile object*. The robot has to follow another mobile robot for 10 meters. If the robot doesn't know what to do, the task fails. From 20 experiments the robot succeded in 95% of the tasks.

The next step was to test the learned TRPs in a real robot. The TRPs were integrated as the navigation module of a PeopleBot ActivMedia robot that used a sonar ring and a Laser SICK LMS200. The tasks to accomplish were: (i) following a human under user commands, (ii) navigating to several places in the environment designated semantically (see Fig. 4(d)), (iii) finding one of a set of different objects in a house and (iv) delivering messages and/or objects between different people. The first three tasks are part of the RoboCup@Home challenge. In the *follow-a-human* task, the robot was trained using the laser sensor to follow another robot. To make a robust approach, a visual landmark process replaced the landmark identification process. The relational approach used to learn TRPs facilitates their integration in different tasks and the replacement of the sensor in a straightforward manner. Navigation and *follow-a-human* videos can be seen at: http://www.youtube.com/user/anon9899

VIII. Conclusions and Future Work

This paper introduced a two phase process to learn TRPs for mobile robots in office environments. In particular, it is shown how to transform low-level sensor readings from the robot into high-level predicates useful for relational learning. This is a relevant step as it allows to use a wide variety of algorithms directly from robot traces. The paper also showed how to learn relational TRPs that can be used to navigate a real robot on different dynamic and unknown environments. TRPs naturally incorporate continuous actions, suitable for a robotic domain, and are able to deal with unexpected events, like moving obstacles. The paper showed how to learn hierarchical TRPs using a simple grammar induction algorithm. This opens the possibility of incremental learning of more complex tasks, given some previously learned tasks. The learned TRPs were used for navigation tasks on different environments with dynamic objects on simulation and a real robot with very promising results. As part of our future work, we are working on learning more concepts, and more basic and complex TRPs to solve other tasks, like going to several goals. We are also working on a better merging algorithm for grammars.

References

[1] S. Muggleton, *Proceedings of the International Workshop of Inductive Logic Programming*, Viana de Castelo, Portugal, 1991.

[2] S. Benson and N. J. Nilsson, "Reacting, planning, and learning in an autonomous agent," *Machine Intelligence*, vol. 14, pp. 29–62, 1995.

[3] D. Michie and C. Sammut, "Behavioral clones and cognitive skill models," *Machine Intelligence*, vol. 14, pp. 395–404, 1995.

[4] C. D'Este, M. O'Sullivan, and N. Hannah, "Behavioural cloning and robot control," in *Proceedings of International Conference on Robotics and Applications, International*, Salzburg, Austria, June 2003.

[5] M. W. Kadous, C. Sammut, and R. K.-M. Sheh, "Autonomous traversal of rough terrain using behavioural cloning," in *Proceedings International Conference on Autonomous Robots and Automation*, Palmerston North, New Zealand, 2006.

[6] E. F. Morales and C. Sammut, "Learning to fly by combining reinforcement learning with behavioural cloning," in *Proceedings of the twenty-first international conference on Machine learning table of contents*, Banff, Alberta, Canada, 2004. [Online]. Available: citeseer.ist.psu.edu/morales04learning.html

[7] A. Cocora, K. Kersting, C. Plagemann, W. Burgard, and L. De Raedt, "Learning relational navigation policies," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Beijing, China, 2006.

[8] V. Klingspor, K. Morik, and A. D. Rieger, "Learning concepts from sensor data of a mobile robot," *Machine Learning*, vol. 23, no. 2-3, pp. 305–332, 1996. [Online]. Available: citeseer.ist.psu.edu/klingspor96learning.html

[9] S. Zelek and M. D. Levine, "Spott: A mobile robot control architecture for unknown or partially known environments," *AAAI Spring Symposium on Planning with Incomplete Information for Robot Problems*, 1996.

[10] K. Broda and C. J.Hogger, "Designing and simulating individual teleo-reactive agents," *Poster Proceedings, 27th German Conference on Artificial Intelligence, Ulm*, 2004.

[11] T. Konik and J. E. Laird, "Learning goal hierarchies from structured observations and expert annotations," *Machine Intelligence*, vol. 64, pp. 263–287, 2006.

[12] S. Hernández and E. Morales, "Global localization of mobile robots for indoor environments using natural landmarks," *IEEE International Conference on Robotics, Automation and Mechatronics (RAM)*, 2006.

[13] A. Srinivasan, "The aleph 5 manual."

[14] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. of the 20th Int'l Conference on Very Large Databases*, Santiago, Chile, 1994.

[15] H. Avilés, E. Corona, A. Ramírez, B. Vargas, J. Sánchez, L. Sucar, and E. Morales, "A service robot named markovito," *IEEE Latin American Robotic Symposium / IX Congreso Mexicano de Robotica (4th IEEE LARS 07 / IX COMRob 07)*, 2007.