A Learning Mobile Robot: Theory, Simulation and Practice

Nuno Chalmique Chagas and John Hallam*

* *University of Edinburgh, Department of Artificial Intelligence,* {nuno,john}@dai.ed.ac.uk

Abstract.
This paper presents an implementation of the SINS multi-strategy learning controller for mobile robot navigation. This controller uses low-level reactive control that is modulated on-line by a learning system based on case-based reasoning and reinforcement learning. The case-based reasoning part captures regularities in the environment. The reinforcement learning part gradually improves the acquired knowledge. Evaluation of the controller is presented in a real and in a simulated mobile robot.

1 Introduction

How to specify behaviour in a robot has come a long way since the low-level languages of assembly robotics (Lozano-Perez, 1982). The *classical AI* approach to control[1] proved too slow and too fragile for the real world but showed that representations of the environment, however difficult to maintain, produce interesting behaviour. In *nouvelle AI*, *e.g.* (Brooks, 1985; Brooks, 1991a; Brooks, 1991b), agents merely react to the current environmental situation posed, limited to the scope of the designer's built-in behaviours which are very hardwired. However, low-level behaviours are very good at being fast, like reflexes.

Learning in mobile robot controllers is advantageous because it lessens the burden of exhaustive specificity. Learning can help make continually updated representations of the environment, as well as optimise low-level behaviours. Robot controllers that learn are more flexible and more prone to succeed at whatever task they are meant to do.

The solution that is the object of discussion in this paper is the Self Improving Navigation System — SINS — learning architecture (Ram and Santamaria, 1993b; Ram and Santamaria, 1993a; Moorman and Ram, 1992). This is a case-based learning system on top of a reactive system that stands out as a blend of reactive/learning strategies. What appeals most is that the designer knowledge that is actually put into the system — the reactive behaviours — is plastic and controlled on-line by a learning scheme allowing quite complex behaviour to be learned with no bias from the designer and relatively low effort to program. In this paper we apply it for the first time to a real robot as the original work was developed as a computer simulation only.

SINS is a highly competent controller which has proven in simulation to be a good hybrid solution with good results incorporating advantages from the different techniques it uses: reactive control, reinforcement learning and case-based reasoning. It is also capable of temporal learning because it encodes acquired knowledge as structured sequences through time.

In (Brooks and Mataric, 1993), learning is divided into four types. SINS achieves the first three of these:

- *Learning of Numerical Functions for Calibration or Parameter Adjustment.*
  Operational parameters of an existing behavioural structure are optimised during the life of the robot.

- *Learning about the World.*
  Internal functional representations of the world are constructed and altered.

[1] See (Pfeifer and Scheier, 1994) for a discussion.

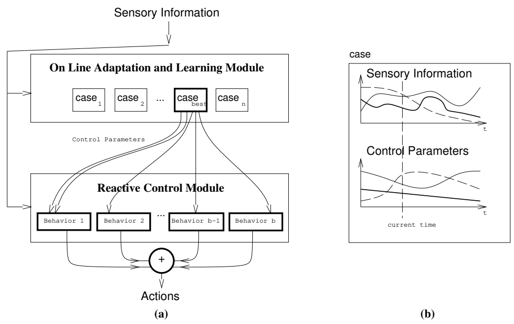

Figure 1: SINS Architecture

- *Learning to Coordinate Behaviours.*
  Existing behavioural structures and their effects in the world are used to change the conditions and sequences in which they are triggered under control of internal learned functions.

- *Learning New Behaviours.*
  SINS does not learn new behavioural structures, as such. However, the behaviours are configurable by parameters, which adds a degree of plasticity.

The next section is a description of the SINS architecture. Section 3 describes the testing methodology used, its purpose and the results obtained. Finally section 4 ponders on the results obtained.

2 The SINS architecture

The SINS — Self-Improving Navigation System — (Ram and Santamaria, 1993b; Ram et al., 1992) is a learning controller architecture that makes use of reactive control, case-based reasoning and reinforcement-learning. It was devised, developed and tested as a computer simulation.

It is composed of two modules (figure 1a):

- the Reactive Control Module
- the On-line Adaptation and Learning Module

Both modules have free access to the sensory information of the robot and run concurrently. The first module is a collection of parameterisable reactive behaviours which account for the immediate responsiveness of the system. The second one is a case-based and reinforcement learning system that learns sets of numerical functions of time — cases (outlined in Figure 1b) — that output control parameters for the low-level behaviours of the first module. Each function in a case attempts to represent either sensory information or control parameters. Each control parameter function in a case controls one parameter of a behaviour. Each sensory information function in a case represents a history of sensory *indicator* values. An indicator is a function of the sensors of the robot, *e.g.* an obstacle-density indicator can reflect if the robot has obstacles around it, based on proximity sensor information. They are predefined by the designer and fixed.

All of the behaviours run in parallel. They are given independent access to the vehicle's sensors and compute a vector pointing in the direction to pursue. All the vectors, one per behaviour, are simply summed up to produce the final vector which commands the robot. Each behaviour has at least one parameter that regulates its operation — that is the magnitude or gain of its output vector. Each behaviour may have other parameters that can be modulated, pertaining to its function. All of the behaviours' parameters are dynamically set by the on-line adaptation and learning module, while the robot is in operation.

The On-line Learning and Adaptation Module consists of a loop of the following four phases:

- *Perceive* — Monitors the sensors of vehicle and computes current indicator values;

- *Retrieve* — Detects and discriminates between different environmental configurations by searching exhaustively the collection of learned cases for the case that best matches the current environmental situation. A *goodness of match* measure is computed for every learned case to reflect how similar this, or part of this case, is to the current environmental situation;

- *Adapt* — Sets the behaviours' parameters in a dynamic on-line manner based on the case picked. This is based on the assumption that a similar situation is being encountered and that the case has led to positive results in the past;

- *Learn* — Simultaneously updates the internal representation of the case according to current experience.

The learning phase involves three different methods of incorporating new knowledge. How relevant a particular learned case is to the particular environmental situation is determined directly by the goodness of match measure — the better the match, the higher the *relevance*. According to relevance a case can either be:

- *Created* — Creation of a new case based on the current situation is performed if the relevance of all existing cases is minimal;
- *Modified* — Modification of the associations made in a case, if relevance is acceptable. This actually changes the functions of time stored in a case according to a reinforcement learning formula that will improve the case. This can take two different forms. Depending on whether an experience is successful or not, either associations contained in a case are reinforced or alternative associations are explored randomly by an amount proportional to the failure of the experience;
- *Extended* — Extension of the size of a case to yield associations over larger time windows is performed if relevance is very high.

The goal of SINS is to navigate efficiently to a target position through a cluttered environment. It has a reactive control module with three behaviours:

- *Move-To-Goal* — produces a vector in the direction of the goal;
- *Avoid-Obstacles* — produces a vector away from obstacles;
- *Noise* — produces a new random vector from time to time, making the robot describe a jagged-line trajectory. It has NOISE-PERSISTENCE as a parameter which controls how often a new random vector is generated.

There are thus four parameters that the on-line learning and adaptation module controls. These are NOISE-PERSISTENCE (just described) and the gains to each of the behaviours — MOVE-TO-GOAL-GAIN, AVOID-OBSTACLES-GAIN and NOISE-GAIN. The parameters take analogue values in a bounded range. The set of these four parameters determines at each moment the composite behaviour the robot actually assumes, by allowing each behaviour's influence on the total behaviour of the robot to take any value from a maximal value through to zero.

There are four indicators in this implementation of SINS:

- *Obstacle-density* reflects how much free space there is in the immediate vicinity of the robot;
- *Motion-to-goal* reflects whether the robot is progressing towards or away from the goal;
- *Relative-motion* reflects how much the robot is moving, any motor activity;
- *Absolute-motion* reflects how efficient the motion of the robot is, *i.e.* is low if the robot is moving back-and-forth.

2.1 SINS: In the Khepera world

SINS as described so far (Ram and Santamaria, 1993b) required some modifications in order to be able to run on the Khepera mobile robot and its simulator (Michel, 1996) instead of a Cartesian computer simulation. These changes were imperative when the architecture required global world information. Such is the case of Cartesian movement, relative motion and above all absolute motion.

A list of the changes made followed by a brief description of each is given below.

- *The relative and absolute motion indicators* are based on dead reckoning from left and right encoders, and not from global X and Y coordinates. This may produce an effect that never happened in Cartesian simulation, since the X and Y coordinates of the robot were not prone to error as dead-reckoning is.
- *The random behaviour* produces a jagged line trajectory by left and right motor commands and not by Cartesian coordinates.
- *The goal* is represented by a light. The distance to the goal can only be measured by the robot's light sensors.
- *The Move-To-Goal behaviour* is somewhat different from the original, in that a constant gradient of light is not always present, so, often, the robot doesn't know which way to go. In this case, the behaviour now tells it to go forward, even though it may be getting further away from the light source.
- *The bump sensor* was programmed to be triggered if any of the distance sensors saturates. This is because there is no physical bump sensor on the Khepera.
- *The reinforcement function* was changed from a function that penalised bumps and low absolute-motion, to a function that also rewards motion-to-goal.

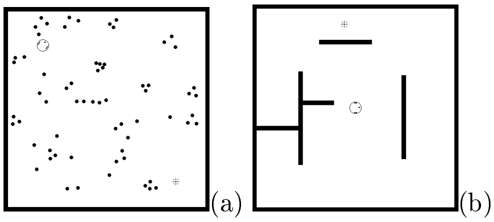

Figure 2: (a) Cluttered environment; (b) Maze environment

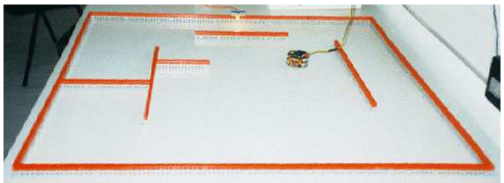

Figure 3: Practical Setup - LEGO maze environment, Khepera robot and light (goal)

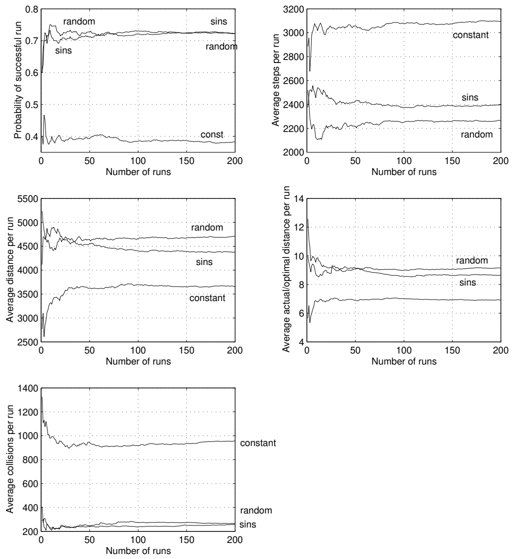

Figure 4: Khepera simulator results for the cluttered environment with "de-fogged" light.

3 Experiments

Three sets of experiments were made, each addressing a specific problem:

1. *New implementation vs. original implementation* — by experimenting with the Khepera simulator in a cluttered environment and comparing with the results in (Ram and Santamaria, 1993b);
2. *Effect of Maze environment on navigation* as opposed to cluttered environment navigation (see figure 2);
3. *Effect of running on a real robot* — by comparing the Khepera simulator results for the maze environment (figure 2b), with the real Khepera performance in an equivalent, real maze environment (figure 3).

For the comparison of results, five estimators were used to evaluate the performance:

1. *Percentage of runs solved*
2. *Average steps per world*
3. *Average distance travelled per world*
4. *Average* $\frac{travelled}{optimal}$ *distance per world*
5. *Average collisions per world*

These estimators were also used in (Ram and Santamaria, 1993b) and provide a direct way of comparing results.

Three types of controllers were used to test the efficiency of the on-line learning and adaptation module:

- *Sins* — the case-based reasoning and reinforcement learning on-line learning and adaptation module;
- *Constant* — which yields constant behaviour parameters. All of the behaviours' parameters, *i.e.* MOVE-TO-GOAL-GAIN, NOISE-GAIN, NOISE-PERSISTENCE and AVOID-OBSTACLES-GAIN are set to $\frac{1}{2}$. The range for these parameters is $[0, 1]$.
- *Random* — which continually sets behavioural parameters randomly.

The performance evaluation is based on the concept of *run*. A run is a trajectory from a random starting point, which is always different for different runs, to the light (goal). A limit of 4000 steps per run was set which, if reached, means that the run was not solved. The optimal distance is the distance measured in a straight line from the starting point of the run to the goal.

The functions in the graphs are running means.

Controller	% runs solved	steps per run	distance per run	$\frac{travelled}{optimal}$ distance per run	collisions per run
SINS	72.25%	2395	4377	7.92	257.8
SINS*	100.00%	141.9	107.2	2.63	10.7
SINS ratio	0.72	16.9	40.8	3.0	24.1
CONSTANT	38.28%	3096	3655	6.57	957.5
STATICH*	82%	315.5	126.2	3.10	48.1
CONSTANT ratio	0.47	9.81	29.0	2.1	19.9
RANDOM	72.17%	2268	4710	8.46	266.1
RANDOM*	100.00%	141.0	113.7	2.79	13.7
RANDOM ratio	0.72	16.1	41.4	3.0	19.4

Table 1: Khepera simulator results for the cluttered environment with a "de-fogged" light. *results from (Ram and Santamaria, 1993b).

3.1 New implementation vs. original implementation

The experiments reported in (Ram and Santamaria, 1993a) were replicated in the Khepera simulator. A cluttered world (figure 2a) was used and the light was "de-fogged" so that it could be seen from a long distance. This allows a gradient of light to be felt from any part of the arena as in the original implementation the goal could always be seen. This was made so that the comparative study is more faithful.

Table 1 shows the performance of the three different controllers in the new and in the the original implementation. As the two implementations of estimators differ, a ratio is presented for each pair of comparable performance estimators,

From the ratios, the implementation of the SINS and RANDOM controllers, conformed to the expectations, agreeing to an acceptable extent. The CONSTANT controller behaves quite differently from the STATICH constant heuristic controller, which parameters are unknown.

Figure 4 shows the evolution of the results through the runs. It can be concluded that SINS and RANDOM are not significantly different and that CONSTANT performs badly.

3.2 Effect of maze environment on navigation

After this, and with the purpose of having simulated results that could be more faithfully compared with real-robot results, the light was made more "real" in that it could only begin to be seen in a local vicinity which is circa $\frac{1}{9}$ of the area of the environment. The maze environment shown in figure 2b was used. This particular maze configuration creates a local minimum at the location near the light, with the wall between the robot and the light which it can see. This is a special pitfall for any controller since it can only escape from that particular zone by means of a surge of the random behaviour.

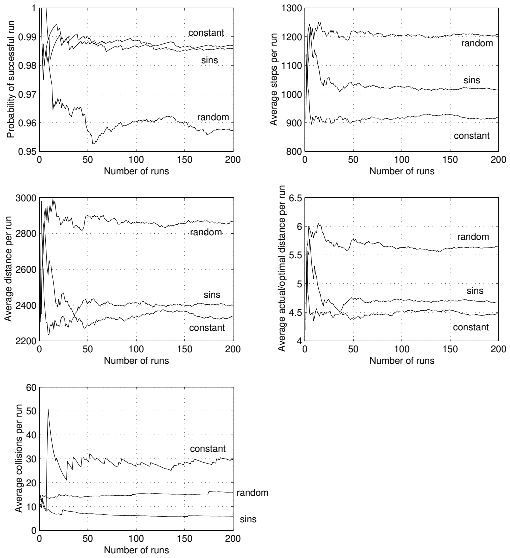

Figure 5: Khepera simulator results for the maze environment.

SINS converges fast after about 50 runs (figure 5). After this it manages to hold its performance. SINS learns very efficiently not to collide, surpassing noticeably all other controllers on this indicator. This is at some expense of the steps per run measure,

Controller	% runs solved	steps per run	distance per run	$\frac{travelled}{optimal}$ distance per run	collisions per run
SINS	98.57%	1017	2402	4.69	5.9
CONSTANT	98.70%	916	2332	4.50	29.6
RANDOM	95.73%	1207	2864	5.62	16.0

Table 2: Khepera simulator results in the maze environment.

Controller	solved runs/ total runs	steps per run	distance per run	collisions per run
SINS	40/40	1056	2436	15
SINS^{**}_{10}	10/10	879	2046	10
CONSTANT	19/20	1041	2529	18
RANDOM	20/20	1206	2693	16

Table 3: Khepera robot results in the real maze environment. **Last 10 runs of SINS (runs 31 to 40).

which is not as low as for the CONSTANT controller. On a trajectory analysis, SINS gets into tighter spots than other controllers, and manages to come out of them, proving to be a more thorough explorer.

SINS leans to navigate at lower velocities than the other controllers, and so, takes more time steps to reach the goal. This is because it learns to care about obstacles and navigates at sensible speeds which allow it to minimise collisions.

Overall, SINS converges to be just about as good as CONSTANT in every measure except in that it travels 12% slower, but bumping only $\frac{1}{5}$ of the times CONSTANT does.

3.3 Effect of running on a real robot

From the Khepera simulator, experiments were then made with a real Khepera in a LEGO environment. The practical setup can be seen in figure 3 and has the same configuration as the simulated environment in figure 2b. The walls have small round holes which let the light through. This is in accordance with the simulator too, in which obstacles and walls are "transparent" to light.

For this experiment, only a limited number of runs was made and fewer performance estimators were drawn, due to the time-consuming nature of real robot experiments.

The practical results attained are shown in table 3. They can be compared with those of table 2. The results, albeit the reduced number of runs, indicate that SINS holds up in the Khepera robot and even manages to outperform the CONSTANT and RANDOM controllers in the last 10 runs. Not only did SINS do better in average after 40 runs in two of the three estimators, but also manages The robot learns and the steps/distance all the indicators improve. The CONSTANT controller failed to reach the light once because it was unable to negotiate the wall by the light.

Also, it is is worth mentioning that the disadvantage SINS suffers in computing time against the RANDOM and CONSTANT controllers goes by unnoticed in the real robot implementation because of the time the robot needs to physically move.

4 Conclusion

A model of a controller developed in simulation capable of efficient adaptive navigation for a mobile robot was presented and its efficiency re-evaluated.

The controller was re-implemented in order to work on a real robot simulator and on a real robot. The improbable toy worlds of simulation were replaced by a real Khepera world.

Considerations that extend to the simulations done are that SINS managed only to equal, but not better, the CONSTANT controller in a maze environment and the RANDOM controller in a cluttered environment.

The real robot results, however statistically few, showed that in the real world SINS does better than other non-learning controllers after only 30 learning runs in a maze environment.

By the time of the workshop this ongoing study will have a more extensive evaluation of the SINS controller.

Acknowledgements

Nuno Chagas is supported by PRAXIS XXI grant BD/2948/94. Thanks to Sandra Gadanho for comments. Facilities were provided by the University of Edinburgh.

References

Brooks, R. and Mataric, M. (1993). Real robots, real learning problems. In Connell, J. and Mahadevan, S., editors, *Robot Learning*, chapter 8, pages 193–213. Kluwer Academic Publishers.

Brooks, R. A. (1985). A robust layered control system for a mobile robot. AI Memo 864, MIT.

Brooks, R. A. (1991a). Intelligence without reason. AI Memo 1293, MIT.

Brooks, R. A. (1991b). Intelligence without representation. *Artificial Intelligence*, pages 139–159.

Lozano-Perez, T. (1982). Robot programming. AI Memo 698, Massachusetts Institute of Technology - Artificial Intelligence Laboratory.

Michel, O. (1996). Khepera simulator package version 2.0: Freeware mobile robot simulator. Technical report.

Moorman, K. and Ram, A. (1992). A case-based approach to reactive control for autonomous robots. In *AAAI Fall Symposium on "AI for Real-World Autonomous Mobile Robots"*, Cambridge, MA.

Pfeifer, R. and Scheier, C. (1994). From perception to action: The right direction? In Gaussier, P. and Nicoud, J.-D., editors, *From Perception to Action Conference*, pages 1–11. IEEE Computer Society Press.

Ram, A., Arkin, R., Moorman, K., and Clark, R. (1992). Case-based reactive navigation: A case-based method for on-line selection and adaptation of reactive control parameters in autonomous robotic systems. Technical Report GIT-CC-92/57, Georgia Institute of Technology - College of Computing.

Ram, A. and Santamaria, J. C. (1993a). A multistrategy case-based and reinforcement learning approach to self-improving reactive control systems for autonomous robotic navigation. In *Proceedings of the Second International Workshop on Multistrategy Learning*, WV. Harpers Ferry.

Ram, A. and Santamaria, J. C. (1993b). Multistrategy learning in reactive control systems for autonomous robotic navigation. *Informatica*, 4(17):347–369.

Steels, L. (1995). Intelligence - dynamics and representations. In Steels, L., editor, *The Biology and Technology of Intelligent Autonomous Agents*, Berlin. Springer-Verlag.