SUCCESSIVE RECOGNITION USING LOCAL STATE MODELS

Per-Erik Forssén

Computer Vision Laboratory
Department of Electrical Engineering
Linköping University, SE-581 83 Linköping, Sweden
perfo@isy.liu.se

ABSTRACT

This paper describes how a world model for successive recognition can be learned using associative learning. The learned world model consists of a linear mapping that successively updates a high-dimensional system state using performed actions and observed percepts. The actions of the system are learned by rewarding actions that are good at resolving state ambiguities. As a demonstration, the system is used to resolve the localisation problem in a labyrinth.

1. INTRODUCTION

During the eighties a class of robotic systems known as *reactive robotic systems* became popular. The introduction of system designs such as the *subsumption architecture* [1] caused a small revolution due to their remarkably short response times. Reactive systems are able to act quickly since the actions they perform are computed as a direct function of the sensor readings, or *percepts*, at a given time instant. This design principle works surprisingly well in many situations despite its simplicity. However, a purely reactive design is sensitive to a fundamental problem known as *perceptual aliasing*.

Perceptual aliasing is the situation when the percepts are identical in two situations when the system should perform different actions. There are two main solutions to this problem:

- The first is to add more sensors to the system such that the two situations can be told apart.
- The second is to give the system an internal state. This state is estimated such that it is different in the two situations, and can thus be used to guide the actions.

This paper will deal with the latter solution, which further on will be called *successive state estimation*. We note here that the introduced state can be tailor-made to resolve the perceptual aliasing.

Successive state estimation is called *recursive parameter estimation* in signal processing, and *on-line filtering* in statistics [2]. Successive recognition could potentially be useful to computer vision systems that are to navigate in a known environment using visual input, such as the autonomous helicopter in the WITAS project [3].

2. SYSTEM OUTLINE

Successive state estimation is an important component of an active perception system. The system design to be described is illustrated in figure 1. The state estimation, which is the main topic of this paper, is performed by the *state transition* and *state narrowing* boxes.

The state transition box updates the state using information about which action the system has taken, and the state narrowing box successively resolves ambiguities in the state by only keeping states that are consistent with the observed stimulus.

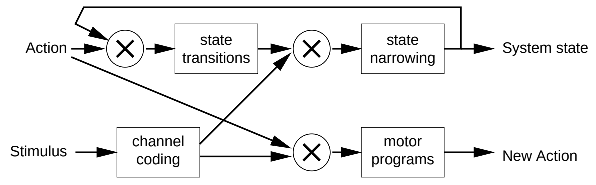

Fig. 1. System outline.

The system uses an information representation called *channel representation* [4, 5]. This implies that information is stored in channel vectors of which most elements are zero. Each channel is monopolar (e.g. either positive, or zero), and its magnitude signifies the relevance of a specific hypothesis (such as a specific system state in our case), and thus a zero value represents “no information”. This information representation has the advantage that it enables very fast associative learning methods to be employed [5], and improves product sum matching [4].

The author wants to acknowledge the financial support of WITAS, the Wallenberg laboratory for Information Technology and Autonomous Systems.

The *channel coding* box in figure 1 converts the percepts into a channel representation. Finally, the *motor program* box is the subsystem that generates the actions of the system. The complexity of this box is at present kept at a minimum.

3. EXAMPLE ENVIRONMENT

To demonstrate the principle of successive state estimation, we will apply it on the problem shown in figure 2. The arrow in the figure symbolises an autonomous agent that is supposed to successively estimate its position and gaze direction by performing actions and observing how the percepts change. This is known as the *robot localisation problem* [2]. The labyrinth is a known environment, but the initial location of the agent is unknown, and thus the problem consists of learning (or designing) a world model that is useful for successive recognition.

The stimulus constitutes a three element binary vector, which tells whether there are walls to the left, in front, or to the right of the agent. For the situation in the figure, this vector will look like this:

$$\mathbf{m} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T$$

This stimulus is converted to percept channels in one of two ways

$$\mathbf{p}_1 = \begin{pmatrix} m_1 & m_2 & m_3 & 1-m_1 & 1-m_2 & 1-m_3 \end{pmatrix}^T$$
$$\mathbf{p}_2 = \begin{pmatrix} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 \end{pmatrix}^T \quad (1)$$

Where $$p_h = \begin{cases} 1 & \text{if} \quad \mathbf{m} = \mathbf{m}^h \\ 0 & \text{otherwise} \end{cases}$$

and $\{\mathbf{m}^h\}_1^8$ is the set of all possible stimuli. This expansion is needed since we want to train an associative network [5] to perform the state transitions, and since the network only has monopolar coefficients, we must have a non-zero input vector whenever we want a response.

The two variants $\mathbf{p}_1$ and $\mathbf{p}_2$ will be called *semi-local*, and *local* percepts respectively. For the semi-local percepts, correlation serves as a similarity measure, or *metric*, but for the local percepts we have no metric—the correlation is either 1 or 0.

The system has three possible actions $\mathbf{a}^1 = \texttt{TURN\_LEFT}$, $\mathbf{a}^2 = \texttt{TURN\_RIGHT}$, and $\mathbf{a}^3 = \texttt{MOVE\_FORWARD}$. These are also represented as a three element binary vector, with only one non-zero element at a time. E.g. TURN_RIGHT is represented like this:

$$\mathbf{a}^2 = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}^T$$

Each action will either turn the agent $90°$ clockwise or anti clockwise, or move it forward to the next grid location (unless there is a wall in the way).

As noted in section 1, the purpose of the system state is to resolve perceptual aliasing. For the current problem

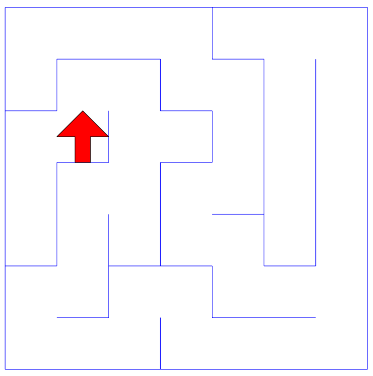

Fig. 2. Illustration of the labyrinth navigation problem.

this means that the system state has to describe both agent location and absolute orientation. This gives us the number of states as:

$$N_s = \text{rows} \times \text{cols} \times \text{orientations} \quad (2)$$

For the labyrinth in figure 2 this means $7 \times 7 \times 4 = 196$ different states.

4. LEARNING SUCCESSIVE RECOGNITION

If the state is in a local representation, that is, each component of the state vector represents a local interval in state space, successive recognition can be obtained by a linear mapping. For the environment described in section 3, we will thus use a state vector with N_s components.

The linear mapping will recursively estimate the state, $\mathbf{s}$, from an earlier state, the performed action, $\mathbf{a}$, and an observed percept $\mathbf{p}$. I.e.

$$\mathbf{s}(t+1) = \mathbf{C}\left[\mathbf{s}(t) \otimes \mathbf{a}(t) \otimes \mathbf{p}(t+1)\right] \quad (3)$$

Where $\otimes$ is the *Kronecker product*, which generates a vector containing all product pairs of the elements in the involved vectors. The sought linear mapping $\mathbf{C}$ is thus of dimension $N_s \times N_s N_a N_p$ where N_a and N_p are the sizes of the action and percept vectors respectively.

In order to learn the mapping we supply examples of $\mathbf{s}$, $\mathbf{a}$, and $\mathbf{p}$ for all possible state transitions. This gives us a total of $N_s N_a$ samples. The coefficients of the mapping $\mathbf{C}$ are found using a least squares optimisation with monopolar constraint:

$$\arg\min_{c_{ij}>0} ||\mathbf{u} - \mathbf{C}\mathbf{f}||^2 \quad \text{where}$$
$$\begin{aligned} \mathbf{u} &= \mathbf{s}(t+1) \\ \mathbf{f} &= \mathbf{s}(t) \otimes \mathbf{a}(t) \otimes \mathbf{p}(t+1) \end{aligned}$$

For details of the actual optimisation see [5].

5. NOTES ON THE STATE MAPPING

The first thing to note about usage of the mapping, $\mathbf{C}$, is that the state vector obtained by the mapping has to be normalised at each time step, i.e.

$$\begin{cases} \tilde{\mathbf{s}}(t+1) &= \mathbf{C}\left[\mathbf{s}(t) \otimes \mathbf{a}(t) \otimes \mathbf{p}(t+1)\right] \\ \mathbf{s}(t+1) &= \dfrac{\tilde{\mathbf{s}}(t+1)}{\sum_k \tilde{s}_k(t+1)} \end{cases} \tag{4}$$

Another observation is that in the environment described in section 3, we obtain exactly the same behaviour when we use two separate maps:

$$\begin{cases} \mathbf{s}^*(t+1) &= \mathbf{C}_1\left[\mathbf{s}(t) \otimes \mathbf{a}(t)\right] \\ \tilde{\mathbf{s}}(t+1) &= \mathbf{C}_2\left[\mathbf{s}^*(t+1) \otimes \mathbf{p}(t+1)\right] \end{cases} \tag{5}$$

An interesting parallel to *on-line filtering* algorithms in statistics is that $\mathbf{C}_1$ and $\mathbf{C}_2$ actually correspond to the stochastic *transition model* $p(\mathbf{s}(t+1)|\mathbf{s}(t), \mathbf{a}(t))$ and the stochastic *observation model* $p(\mathbf{p}(t)|\mathbf{s}(t))$ respectively (see for instance [2]).

The mappings have sizes $N_s \times N_s N_a$ and $N_s \times N_s N_p$, and this gives us at most $N_s^2(N_a + N_p)$ coefficients compared to $N_s^2 N_a N_p$ in the single mapping case. Thus the split into two maps is advantageous, provided that the behaviour is not affected.

Aside from the gain in number of coefficients, the split into two maps will also simplify the optimisation of the mappings considerably. If we during the optimisation supply samples of $\mathbf{s}^*(t+1)$ that are identical to $\mathbf{s}(t+1)$ we end up with a mapping, $\mathbf{C}_2$, that simply weights the state vector with the correlations between the observed percept and those corresponding to each state during optimisation. In other words, equation 5 is equivalent to:

$$\tilde{\mathbf{s}}(t+1) = \mathrm{diag}(\mathbf{P}\mathbf{p}(t+1))\mathbf{C}_1\left[\mathbf{s}(t) \otimes \mathbf{a}(t)\right] \tag{6}$$

Where $\mathbf{P}$ is a matrix with row n containing the percept observed at state n during the training, and $\mathrm{diag}()$ generates a matrix with the argument vector in the diagonal.

6. EXPLORATORY BEHAVIOUR

How quickly the system is able to recognise it's location is of course critically dependent on which actions it takes. A good exploratory behaviour should strive to observe new percepts as often as possible, but how can the system know that shifting its attention to something new when it does not yet know where it is?

In this system the actions are chosen using a *policy*, where the probabilities for each action are conditional on the previous action $\mathbf{a}(t-1)$ and the observed percept $\mathbf{p}(t)$. I.e. the action probabilities can be calculated as:

$$p(\mathbf{a}(t) = \mathbf{a}^h) = \mathbf{c}^h\left[\mathbf{a}(t-1) \otimes \mathbf{p}_2(t)\right] \tag{7}$$

Where $\{\mathbf{a}^h\}_1^3$ are the three possible actions (see section 3). The coefficients in the mappings $\{\mathbf{c}^h\}_1^3$ should be defined such that $\sum_h p(\mathbf{a}(t) = \mathbf{a}^h) = 1$.

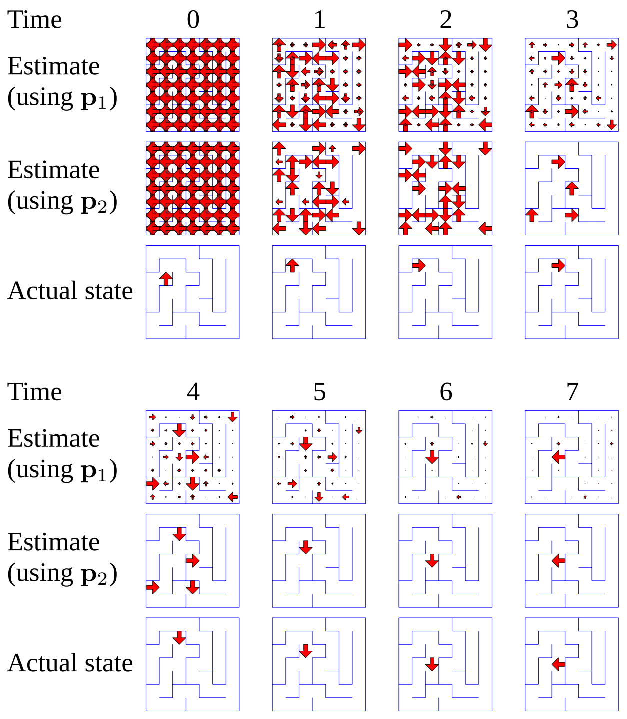
Time
0
1
2
3
Estimate (using $\mathbf{p}_1$)
Estimate (using $\mathbf{p}_2$)
Actual state
Time
4
5
6
7
Estimate (using $\mathbf{p}_1$)
Estimate (using $\mathbf{p}_2$)
Actual state

Fig. 3. Illustration of state narrowing.

A random run of a system with a fixed policy $\{\mathbf{c}^h\}_1^3$ is demonstrated in figure 3. The two different kinds of percepts $\mathbf{p}_1$ and $\mathbf{p}_2$ are those defined in equation 1.

7. EVALUATING NARROWING PERFORMANCE

The performance of the localisation process may be evaluated by observing how the estimated state $\mathbf{s}(t)$ changes over time. As a measure of how *narrow* a specific state vector is we will use:

$$n(t) = \frac{\sum_k s_k(t)}{\max_k\{s_k(t)\}} \tag{8}$$

If all state channels are activated to the same degree, as is the case for $t = 0$, we will get $n(t) = N_s$, and if just one state channel is activated we will get $n(t) = 1$. Thus $n(t)$ can be seen as a measure of how many possible states are still remaining.

Figure 4 (top) shows a comparison of systems using local and semi-local percepts for 50 runs of the network. For each run the true initial state is selected at random, and $\mathbf{s}(0)$ is set to $\mathbf{1}/N_s$.

Since the only thing that differs between the two upper plots in figure 4 is the percepts, the difference in convergence has to occur in step 2 of equation 5. We can further demonstrate what influence the feature correlation has on the convergence by modifying the correlation step in equation 6 as follows:

$$\tilde{\mathbf{s}}(t+1) = \mathrm{diag}(\mathrm{f}(\mathbf{P}\mathbf{p}(t+1)))\mathbf{C}_1\left[\mathbf{s}(t) \otimes \mathbf{a}(t)\right] \tag{9}$$

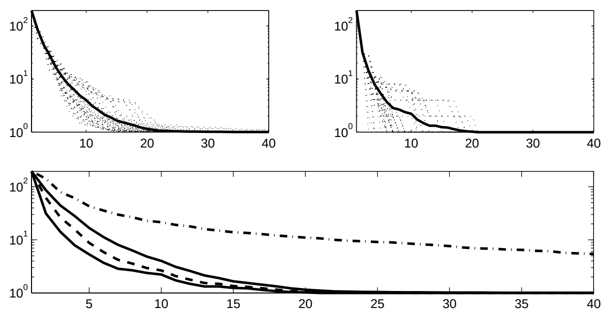

Fig. 4. Narrowing performance.
Top left: $n(t)$ *for a system using* $\mathbf{p}_1$. *Top right:* $n(t)$ *for a system using* $\mathbf{p}_2$. *Each graph shows* 50 *runs (dotted). The solid curves are averages. Bottom: Solid:* $\mathrm{n}(t)$ *for* $\mathbf{p}_1$ *and* $\mathbf{p}_2$. *Dashed:* $\mathbf{p}_1$ *using* $\mathrm{f}_1()$. *Dash-dotted:* $\mathbf{p}_1$ *using* $\mathrm{f}_2()$. *Each curve is an average over* 50 *runs.*

We will try the following two choices of f() on correlations of the semi-local percepts:

$$\mathrm{f}_1(c) = \sqrt{c} \quad \text{and} \quad \mathrm{f}_2(c) = \begin{cases} 1 & \text{if } c > 0 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

All four kinds of systems are compared in the lower graph of figure 4. As can be seen, the narrowing behaviour is greatly improved by a sharp decay of the percept correlation function. However, for continuous environments there will most likely be a trade off between sharp correlation functions and state interpolation and the number of samples required during training.

8. LEARNING A NARROWING POLICY

The conditional probabilities in the policy defined in section 6 can be learned using *reinforcement learning* [6]. A good exploratory behaviour is found by giving rewards to conditional actions $\{\mathbf{a}(t)|\mathbf{p}(t), \mathbf{a}(t-1)\}$ that reduce the narrowing measure in equation 8, and by having the action probabilities $p(\mathbf{a}(t) = \mathbf{a}^h|\mathbf{p}(t), \mathbf{a}(t-1))$ gradually move towards the conditional action with the highest average reward. This is called a *pursuit method* [6].

In order for the rewards not to die out, the system state is regularly reset to all ones, for instance when t mod $30 = 0$. The first attempt is to define the reward as a plain difference of the narrowing measure in equation 8. I.e.

$$r_1(t) = n(t-1) - n(t) \tag{11}$$

With this reward, the agent easily gets stuck into sub-optimal policies, such as constantly trying to move into a wall. Better behaviour is obtained by also looking at the narrowing difference one step into the future. i.e.

$$r_2(t) = r_1(t) + r_1(t+1) = n(t-1) - n(t+1) \tag{12}$$

The behaviours learned using equations 11 and 12 are compared with a random walk in figure 5.

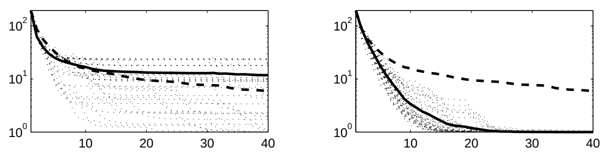

Fig. 5. Narrowing performance.
Left: $n(t)$ *for a policy learned using* $r_1(t)$. *Right:* $n(t)$ *for a policy learned using* $r_2(t)$. *Each graph shows* 50 *runs (dotted). The thick curves are averages. Dashed curves show average narrowing for a completely random walk.*

9. CONCLUSIONS

The aim of this paper has not been to describe a useful application, but instead to show how the principle of successive recognition can be used. Compared to a real robot navigation task, the environment used is way too simple to serve as a model world. Further experiments will extend the model to continuous environments, with noisy percepts and actions.

10. REFERENCES

[1] R. Brooks, “A robust layered control system for a mobile robot”, *IEEE Trans. on Robotics and Automation*, , no. 2, pp. 14–23, 1986.

[2] N. Vlassis, B. Terwijn, and B. Kröse, “Auxiliary particle filter robot localization from high-dimensional sensor observations”, Tech. Rep. IAS-UVA-01-05, Computer Science Institute, University of Amsterdam, 2001.

[3] Gösta Granlund, Klas Nordberg, Johan Wiklund, Patrick Doherty, Erik Skarman, and Erik Sandewall, “WITAS: An Intelligent Autonomous Aircraft Using Active Vision”, in *Proceedings of the UAV 2000 International Technical Conference and Exhibition*, Paris, France, June 2000, Euro UVS.

[4] Per-Erik Forssén, “Sparse Representations for Medium Level Vision”, Lic. Thesis LiU-Tek-Lic-2001:06, Dept. EE, Linköping University, February 2001, Thesis No. 869, ISBN 91-7219-951-2.

[5] Gösta Granlund, Per-Erik Forssén, and Björn Johansson, “HiperLearn: A High Performance Learning Architecture”, Tech. Rep. LiTH-ISY-R-2409, Dept. EE, Linköping University, January 2002.

[6] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning, An Introduction*, MIT Press, Cambridge, Massachusetts, 1998, ISBN 0-262-19398-1.