Problem Solving through Imitation

Eng-Jon Ong [*] Liam Ellis Richard Bowden

Centre for Vision, Speech and Signal Processing,
School of Electronics & Physical Sciences,
University of Surrey, UK

Abstract

This paper presents an approach to problem solving through imitation. It introduces the Statistical & Temporal Percept Action Coupling (ST-PAC) System which statistically models the dependency between the perceptual state of the world and the resulting actions that this state should illicit. The ST-PAC system stores a sparse set of experiences provided by a teacher. These memories are stored to allow efficient recall and generalisation over novel systems states. Random exploration is also used as a fall-back "brute-force" mechanism should a recalled experience fail to solve a scenario. Statistical models are used to couple groups of percepts with similar actions and incremental learning used to incorporate new experiences into the system. The system is demonstrated within the problem domain of a children's shape sorter puzzle. The ST-PAC system provides an emergent architecture where competence is implicitly encoded within the system. In order to train and evaluate such emergent architectures, the concept of the *Complexity Chain* is proposed. The *Complexity Chain* allows efficient structured learning in a similar fashion to that used in biological system and can also be used as a method for evaluating a cognitive system's performance. Tests demonstrating the *Complexity Chain* in learning are shown in both simulated and live environments. Experimental results show that the proposed methods allowed for good generalisation and concept refinement from an initial set of sparse examples provided by a tutor.

Key words: Cognitive System, Complexity Chain, Learning from Imitation, Problem Solving

[⋆] This work has been supported by EC Grant IST-2003-004176 COSPAL. This paper does not represent the opinion of the European Community, and the European Community is not responsible for any use which may be made of its contents.

[*] Corresponding Author.Centre for Vision,Speech and Signal Processing, School of Electronics & Physical Sciences, University of Surrey, Guildford, Surrey GU2 7XH, UK

Email addresses: e.ong@surrey.ac.uk (Eng-Jon Ong),

1 Introduction

This paper presents an approach to learning to solve problems through imitation. It is demonstrated within the domain of a *Children's Shape Sorter Puzzle* (used as the primary demonstrator within the EU Project COSPAL[6]) but many of the principles are applicable within a wider context. The basic principle is to *record* an incomplete set of experiences gained through tuition, to *remember* these experiences in a form that allows the experience to generalise, to efficiently *recall* appropriate experiences given an unseen scenario and to *randomly explore* possible solutions should a recalled experience fail to solve a scenario. New experiences either leading to failure or success are then added to memory, allowing the systems competence to grow.

To achieve this, the paper presents the ST-PAC system (Statistical & Temporal Percept Action Coupling) which ties together the visual percepts (or appropriate description of the world) with the actions that should be made given that scenario. Percepts and actions are coupled within a statistical domain that allows incremental learning and good generalisation. To facilitate efficient learning, the paper also presents the concept of the *Complexity Chain*, which, is an approach to structured learning, allowing the tutor to bootstrap the learning process through increasing levels of complexity in a similar way to that which is employed with children. The complexity chain is used to provide efficient learning as well as a method to evaluate cognitive systems performance.

The task of developing cognitive architectures capable of associating noisy, high dimensional input spaces to appropriate responses in the agents action space across multiple problem domains is an important task for researches in cognitive sciences, machine learning, AI and computer vision. *Cognitivist* approaches, that rely on hard-coded knowledge or engineered rule based systems have shown limited success, especially when the systems are expected to perform in multiple domains or in domains that stray too far from the idealised world envisaged by the engineer.

For this reason *emergent* architectures, that model the world through co-determination with their environments, have become a popular paradigm [9] [15]. Within the emergent paradigm, this process of co-determination, whereby the system is defined by its environment, should result in the system identifying what is important and meaningful to its task.

A potential downside to the emergent paradigm is the need for continuous exploratory activity that can result in unpredictable or unwanted behaviour. It

`l.ellis@surrey.ac.uk` (Liam Ellis), `r.bowden@surrey.ac.uk` (Richard Bowden).

should also be noted that in cognitivist approaches, the knowledge with which the agent makes its decisions can be easily understood by the engineer whereas in emergent systems it can be difficult to interpret the learnt knowledge and/or mappings as they are grounded in the agents own experiences as opposed to the experiences of the engineer [16].

The ST-PAC system encodes concepts implicitly rather than explicitly (as more often found in traditional AI approaches), making it an emergent architecture. This makes it more flexible to adaptation but, as indicated, this implicit knowledge is difficult to interpret. The complexity chain goes some way to solving this difficulty as each stage of learning can be assessed independently, allowing a better understanding of the extent of knowledge encoded implicitly.

This paper continues by discussing learning through imitation in general terms. The concept of the *Complexity Chain* is then introduced in section 3 followed by a specific instantiation of the *Complexity Chain* for the shape sorter puzzle. Section 5 introduces the ST-PAC architecture and experiments performed on the complexity chain of Section 6 are presented. Finally, discussions are provided in Section 7 before concluding in Section 8.

2 Learning by Imitation

Imitation plays a strong role in the development of cognitive systems. Recent neurophysiological research has shown strong evidence supporting the existence of a mechanism, in both primate and human brains, known in the literature as the "direct-matching hypothesis". The mirror-neuron system essentially provides the system (human/primate brain) with the capability "to recognise actions performed by others by mapping the observed action on his/her own motor representation of the observed action" [1].

The work presented here utilises the idea that an agent is capable of mimicking the actions performed by others and exploits this capability with the aim of generalising simple mimicry to more complex imitation of behaviours and hence towards the development of a computer architecture capable of acting appropriately in a wide range of problem domains.

This work is not concerned with modelling the mirror-neuron system, instead it is assumed that the actions of the "teacher" can be directly recorded by the agent. Alternatively, in the work of Siskind, an attempt is made to analyse from visual data the force dynamics of a sequence and hence deduce the action performed [14]. Furthermore, Fitzpatrick have shown that it is possible for an agent to learn to mimic a human supervisor by first observing simple tasks and then, through experimentation, learning to perform the actions that make up the tasks [7]. Both these approaches deal only with exact mimicry, the work presented here aims to extend these capabilities to more general imitation i.e. incorporating representation of purpose to weight responses generated through approximation to imitation.

The general principle behind coupling percepts to actions is that if an agent is given every possible percept *a priori* along with the appropriate action that should be performed when presented with that percept, then problem solving becomes a simple case of recall. However, even in trivial domains this approach is intractable. Given an agent has obtained, through whatever means e.g. mirror-neuron system, a collection of examples of appropriate behaviour at some task, the problem is how to generalise over these exemplars in order that the system is able to solve the task regardless of small differences in the task set-up. For example, given a number of examples of picking up a cup, how can an agent become capable of picking up the cup regardless of the orientation or location of the cup, i.e. how can the agent identify the important variances and invariance's in the perceptual domain and relate these to the action domain. For this reason, much of the context must be stripped away from experiences to allow then to generalise efficiently.

The exact representation of these exemplars of appropriate behaviour depends to a great extent both the embodiment of the agent as well as the nature of its perceptual system. However, in general an exemplar, E, is a coupled pair of percepts, P, and actions, A, i.e. E={P,A}. The task is then to generalise over a set of examples such that, given some new P, an appropriate response A is ilicited. If P and A are vectors it can be reasonably argued that in general the dimensionality of P will be higher than that of A. This provides a motivation for a key mechanism exploited in this work that is the organisation of experiences through the action domain.

> *Related points in the response domain exhibit a much larger continuity, simplicity and closeness than related points in the input domain. For that reason, the organisation process has to be driven by the response domain signals.* [10]

In the visual domain, there has been limited work done in mapping perceptual spaces to action spaces. Notable examples of which are the work of Young [17], where visual parameters extracted via optical flow are used to map to an action space. In the work of Ellis and Bowden [2], percept-action exemplars are hierarchically clustered to in order to build a generalised mapping from percepts to actions. By hierarchically clustering the exemplars in the action parameter space, Ellis and Bowden show that a hierarchy of meaningful, general-to-specific action models is obtained. Furthermore it is shown that by modelling the variance of the groups of percepts formed through the action clustering important variances and invariances in the perceptual domain are identified. These invariance's can be exploited to help match new situations to known experiences. Although no hierarchical representation is present in this work, the experiences of the system are similarly organised by similarity in the action domain. Instead the proposed approach resembles a Finite State Machine (FSM), where each state represents essentially represents a single action. This allows the modelling of temporal ordering between different actions that take place in a strategy. Relevant percepts are then grouped by each "action-state" with statistical models learnt online by applying the system to increasingly more complex problems.

3 Complexity Chain and Evaluation

When learning occurs in biological cognitive systems, many concepts are learnt holistically through random exploration, however, teaching is not holistic. A child is not given a copy of a dictionary and a thesaurus and expected to master language. Instead learning is staged into layers of complexity where simple nouns are learnt, followed by verbs, then adjectives, and eventually more complex concepts such as simile and metaphor. While this is rather an extreme example, the same is true in most tutored learning scenarios: in mathematics multiplication follows subtraction, follows addition; in elementary art, basic primitives such as lines, triangles and circles are mastered before compound shapes are constructed from the basic primitives; and in children's shape sorting puzzles, the child typically learns to place the circle first due to the rotational invariance before they move to more complex shapes which involve higher degrees of freedom. In all these learning scenarios, the child is forming and evolving key concepts that will be used as the building blocks at the next level of complexity. Therefore the role of the supervisor/teacher is to structure learning to encourage the discovery/understanding of the relevant concepts, as well as establishing the necessary relationships between them for devising solutions. By gradually increasing the complexity and exposing new concepts, the tutor is providing weak supervision to the learning process. In many cases, this supervision may not be essential to learning (in terms of success or failure) but in most cases speeds up the learning process.

Moving through a hierarchy of complexity may involve a smooth continuous transition, but there exist key points where moving to a more complex level requires the introduction of a new concept. We define the term *Complexity Chain* as the discrete steps of such a learning process where each discrete level is attached to the introduction of a new, key concept. For example, in the shape sorter scenario, initially a fix board means that no concept of a hole is required. The goal is to move shapes to predefined locations relative to the world coordinate frame. However, once the board can be moved or reconfigured, there is a clear requirement for the concept of a hole. This concept must emerge before the system can refine its concept of the goal. Another example would be to train a system with a puzzle sorter that consists of holes that are basic shapes such as circles and squares. Having learnt the general rules of the game of puzzle sorting, a new puzzle sorter with pieces and holes of different shapes are presented to the system. The system must now adapt existing learnt competence from the previous puzzle sorter for use in solving the new domain.

Evaluation of cognitive systems is also a difficult task. To our knowledge no previous work has addressed this area. For a system with a known goal, performance can be measured naively by either success or failure in reaching that goal. However, this gives little indication as to the benefits or abilities of the system, especially in an emergent architecture where concepts are implicitly embedded within the system. We therefore also propose the complexity chain as a method for evaluation. As each level of complexity is attributed to the discovery of a concept, performance at varying levels within the chain can be used to both assess the systems ability and efficiency at gaining competence. Within the experiments presented here, we use the degree of random exploration at each level of the complexity chain as an indicator of the systems efficiency in concept refinement. We demonstrate how this can be used to evaluate the system at all levels for different parameterisations. The following section now provides an example complexity chain for the *COSPAL Shape Sorter Problem* that will be used for experimentation in later chapters.

4 COSPAL Problem Domain: Shape Sorter Puzzle Complexity Hierarchy

In this section, an example of a complexity hierarchy is presented. This hierarchy will be used specifically for the application of a shape sorter puzzle. Here, various puzzle scenarios with increasing complexity will be presented to the system for training.

The hierarchy consists of 5 main levels (Figure 1), starting with the simplest puzzle scenario at the top and increasing in difficulty as we progress down the hierarchy. We will describe each level of complexity in terms of common physical characteristics present in the puzzle game (e.g. puzzle board is fixed on the floor):

- *Level 1: Fixed holes, Subset of starting configurations*
  At the top level is the puzzle scenario where holes are located at fixed positions (e.g. the puzzle board is nailed to the ground). Additionally, the system is only required to solve the system from a subset of starting configurations for the puzzle pieces. This level of complexity essentially acts

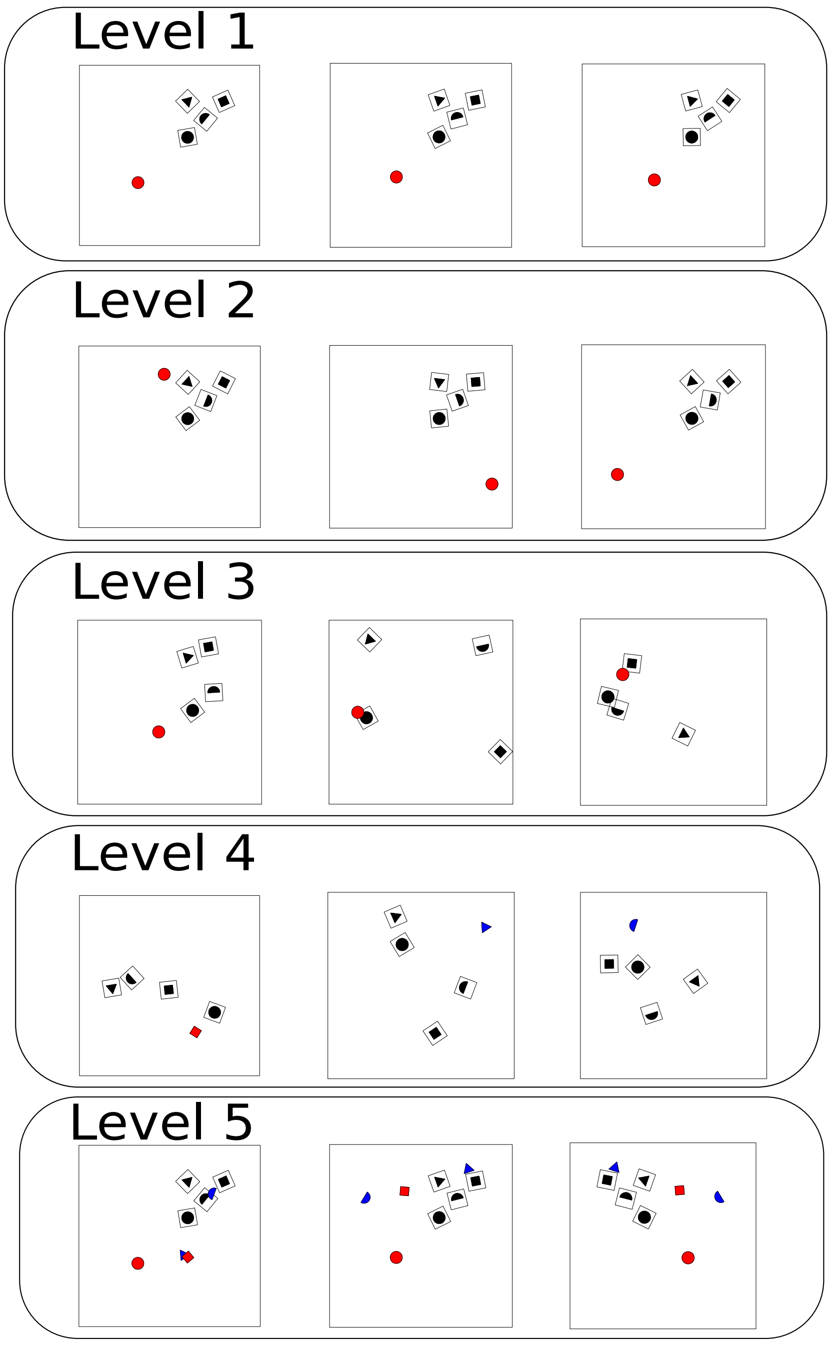


Fig. 1. Complexity hierarchy of the shape sorter puzzle

as a “sanity check” determining if the system can perform pure imitation using memorised knowledge for solving problems with very similar starting configurations to the teacher examples. As such, there is no great need for the system generalise from its initial state in order to solve the puzzles at this level. All the system needs to do is to “copy” the actions of the teacher.

- *Level 2: Fixed holes, All Starting Conditions.* The next level introduces an increased degree of variations present in the starting configurations of the puzzle pieces. Here, any starting configuration is allowed, in particular those that are very different from that presented by the teacher. This level of complexity is used to test the system’s ability to generalise away from the problem configurations that are present in examples provided by the teacher.
- *Level 3: Moveable Holes.* At this level, the puzzle board holes are allowed to move arbitrarily into any positions. As a consequence, each starting configuration of the puzzle may have the holes at different positions and orientations in addition to the puzzle pieces being placed at random locations. An outcome of Level 2 of the chain is that a system may only end up memorising the positions of holes. The system may then solve the shape sorter puzzle by simply moving a particularly shaped piece into a fixed location. In order to force a system away from such a strategy, moving the holes will break this assumption. As a result, it is then necessary for the system to utilise more relevant information to discover the concept of holes and the features that identify them.
- *Level 4: Different Puzzle Pieces, similar rules* In this level, differently shaped and unseen puzzle pieces are presented to the system. The goal is then to be able to “recycle” available knowledge applied to known pieces to help solve the shape sorter puzzle with the new piece. This level of the complexity chain is here to test whether any system that does puzzle solving via learnt imitation can generalise to other objects with different physical characteristics. Here, the rules behind the solution remains the same (e.g. a shaped object is required to be placed into a correctly shaped hole).
- *Level 5: Simultaneous Multiple Puzzle Pieces* In this level, once the system has learnt the ability to solve the shape sorter puzzle for various pieces, a combination of all of these pieces are presented to the system in various randomised starting configurations. This allows one to test the ability of the system to sequentially solve the entire puzzle by sequentially applying the appropriate strategy instantiations that were learnt from the previous four levels.

5 Modelling Imitation using Temporal Statistics of Perception Action Coupling

In this section, a learning framework is proposed for computationally modelling, learning and generating the above described strategies. This will involve mechanisms for committing to memory strategies for solving problems, along with the correct instantiation of using the appropriate strategy to solving a particular problem. The learning framework will consist of two major components. The first is a computational model for problem solving strategies (Section 5.2). Here, the model will take a form that resembles a Finite State Machine (FSM). In this model "states" are represented by an action within the action sequence for a strategy. This allows us to capture the ordered sequence of actions one needs to carry out when using a particular type of problem solving strategy. However, it is not enough to only know which actions need to be performed. It is equally important to carry out actions with the right parameters. This is achieved, using statistical models that can be then used for estimating the parameters for a particular action given some input information.

The next part is an incremental learning method that gradually refines and improves the model continuously through problem solving with the aid of random exploration (Section 5.3). As such, both the learning and application stage are tightly coupled together. A system adopting this framework will be seen to gradually improve itself proportional to the number of times it is used. One important aspect about the method proposed here, is that *both* positive and negative examples obtained from using the model are equally crucial for improvement in accuracy.

5.1 Definition of Strategies and Actions

Since the goal is problem solving via imitation, methods for modelling and using different problem solving strategies are proposed. To start, we define a "strategy" as an ordered sequence of actions to be carried out with some particular parameter, for example an index to a list in an "input scene". In order to formally define the "input scene", suppose that there are N_O entities of interest (e.g. objects in the world), we define the input data to be the set $X = \{x_i\}_{i=1}^{N_O}$, where x_i is a D-dimensional feature vector for the i^{th} entity (see Figure 2).

Now, suppose we have a strategy $(S(P))$ that consists of a sequence of $|S|$ number of actions, and the corresponding parameters for the actions are given in set $P = \{p_t\}_{t=1}^{|S|}$. In this paper, all the parameters are positive integers:

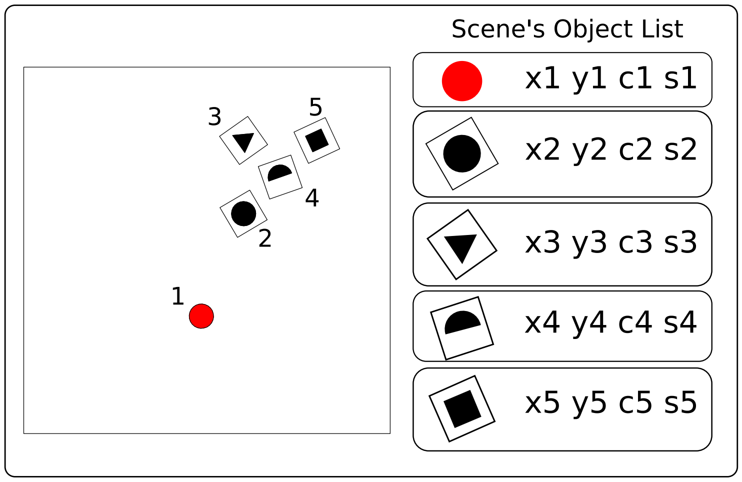


Fig. 2. An illustration of an input scene with 5 objects of interest. Each object is represented by a D-dimensional vector (here 5). The entire scene is then the list of the 5 vectors. In this example, each object is reprsented by a 5 dimensional vector. The first 2 dimensions are the position of the object, the third is the colour ID, and the last is a unique shape ID.

$p_t \in \{+Z\}$. It must be noted that there may exist some actions that require no parameters, consequently, in these cases, the parameter value can be arbitrary and ignored. An action in this sequence can be defined as: $A_t(p_t) \in \{-1, +1\}$. If an action was successfully carried out A_t would return the value of $+1$ and -1 otherwise. An example of an action would be a move-to command, where the parameter is the object to approach. An illustration of a action sequence is given in Figure 3. The ordered action set, $S(P)$, can then be defined as the set:

$$S(P) = \{A_t(p_t)\}_{t=1}^{|S|} \tag{1}$$

5.2 Modelling Problem-Solving Strategies

This section will provide the definition and description of the computational model proposed for representing problem solving strategies. An applied example of this model is given in Section 6.

In order to define the model for problem solving, we follow from the definition of a strategy given in Eq. 1, where $|S|$ denotes the number of actions within a particular strategy. Our proposed model will be a set of different components.

The first set contains an ordered set of actions ($A = \{A_t(p_t)\}_{t=1}^{|S|}$) as defined in the previous section. The second set contains a set of binary elements to allow us to model actions with (value of 1) and without (value of 0) parameters: $B_t = \{b_t\}_{t=1}^{|S|}, b_t \in \{0, 1\}$. The third is a set of feature vector weights

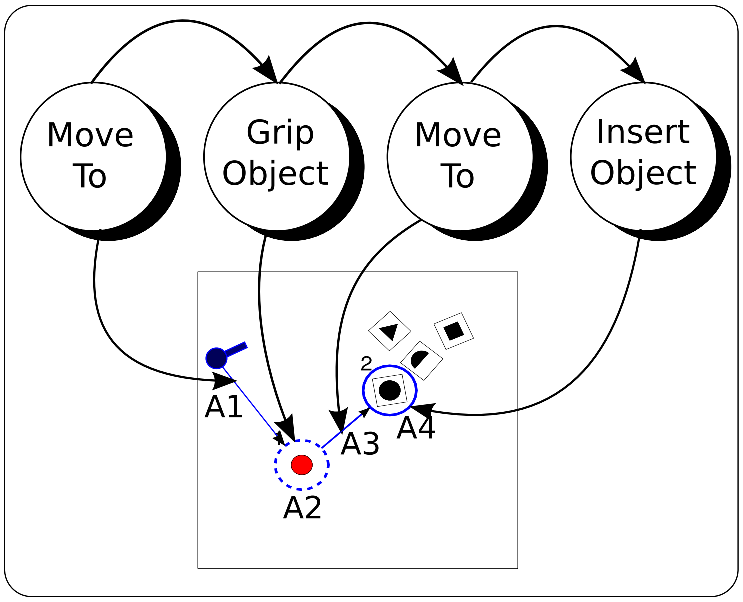


Fig. 3. An example illustrating a sequence of actions that make up the strategy of filling a hole with an appropriately shaped object in the shape sorter puzzle game. Here, the system has a "gripper" with three capabilities: $MoveTo(objID)$, $Grip()$, $Insert()$. A strategy to fill the hole can be defined as the sequence of four actions: $S(P) = \{MoveTo(objID_1), Grip(), MoveTo(objID_2), Insert()\}$. In relation to Equation 1, we have $A_1 = MoveTo$ and so on. The grip and release commands are illustrated as circles with broken and complete lines respectively. Lines are used to show $MoveTo$ commands.

($C = \{c_d\}_{d=1}^{D}$), determining how important a particular feature is in helping us find solutions to a particular class of problems. When the system is initialised, all the feature weights c_t are set to 1, giving the system equal access to the information from each feature. Section 5.3.4 proposes a feature weighting method for determining the values of c_t based on the outcome of the system in solving a problem.

An important component is the mechanism that will be used to retrieve the correct parameters for each action in the strategy. This is achieved by modelling the statistics of plausible sets of parameters for the actions (i.e. $\{p_t\}_{t=1}^{|S|}\}$). However, it is first important to note that a given strategy can be often applied to a range of similar problems. For example, the strategy for filling a hole in a shape sorter puzzle can be "recycled" for various different shapes. To account for this, we define a set of action "parameter-set" statistical models ($R = \{R_i\}_{i=1}^{|R|}$) that capture correct instantiations of a strategy. The number of correct instantiations is defined as $|R|$. More specifically, R_i is made up of essentially a set of Parzen windowing models, one for each action parameter.

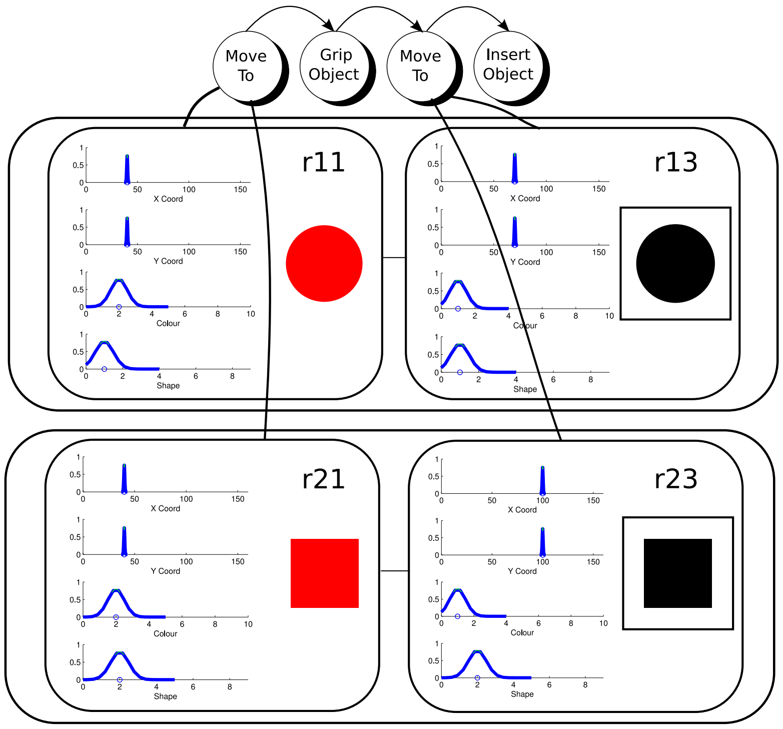


Fig. 4. An example of two Action Parameter Models (APM), each modelling the instantiations for filling a circular hole and square hole with their respective objects. Similar to Figure 3, the system has a gripper with 3 capabilities:$moveTo(objectID)$, $gripObject()$ and $insertObject()$. The strategy used to solve the shape sorter puzzle is modelled as a sequence of four actions: $moveTo(objectID)$, $gripObject()$, $moveTo(objectID)$ and $insertObject()$. Here, there are two possible correct instantiations for this strategy, one for filling the square hole with a cube and another for filling the round hole with the cylinder. This results in $|R|$ being 2. Suppose that R_1 and R_2 are the action parameter-set for filling the square and circular hole respectively. r_{11} would then be the parameter action model with peak responses at the feature subspace associated with cylindrical objects. Similarly, the responses of r_{13} peaks at the feature subspace for circular holes.

Formally, we can define it as $R_i = \{r_{it}\}_{t=1}^{|S|}$, where r_{it}:

$$r_{it}(x) = B_i \sum_{k=1}^{|r_{it}|} K(x, \mu_{itk}) u_{itk} \tag{2}$$

where $r_{it}(x)$ is 0 when no kernel centres are available (i.e. $B_i = 0$). This is used when an action is associated with no parameters. When B_i is 1, the above equation is essentially a modified Parzen window equation, where $|r_{it}|$ is the number of kernels in r_{it}, μ_{itk} are the kernel centres in the t^{th} action parameter model, $u_{itk} \in \{-1, +1\}$ is the sign of the respective kernel. Here, a kernel sign u of $+1$ and -1 is used to record whether the respective kernel represents a successful or failed example respectively. The sign parameter is used to allow us to incorporate information arising from wrong usage of this strategy (Section 5.3). $K(x, \mu)$ represents the kernel chosen. In this paper, a weighted sum of Gaussians function is used:

$$K(x, \mu) = \sum_{l=1}^{D} \frac{c_l}{\sigma_l \sqrt{2\pi}} e^{-(x-\mu_l)^2 / 2\sigma_l^2} \tag{3}$$

where x is the example feature vector, $(\mu = (\mu_l)_{l=1}^{D})$ is the kernel centre, c_l are the feature vector weights defined in Section 5.2, and the "bandwidth" parameter, $\{\sigma_l\}_{l=1}^{D}$, are the widths of the Gaussian functions. An analysis of how different values of σ_l affects the system performance using the complexity hierarchy will be given in Section 6. From here, we shall refer to Eq. 2 as an *Action Parameter Model* (APM). An illustration of APMs is given in Figure 4.

5.3 Strategy Refinement: Incremental Learning from Failure and Success

Given the model defined in the previous section, we will now see how its parameters are learnt and refined as the system is gradually exposed to various problem solving configurations and is detailed in the following four subsections:

(1) Firstly, Section 5.3.1 describes how the strategy model can be initialised when provided with an example from a "teacher" (i.e. memorising the teacher's example for imitation).
(2) Following this, Section 5.3.2 then describes how, when given input data, the usage of a particular strategy can be through the estimation of action parameters.
(3) We then see how incremental learning examples can be collected by executing a chosen strategy with the aid of random exploration as a backup strategy in Section 5.3.3.
(4) With the availability of learning examples, Section 5.3.4 describes how the parameters' statistical models can be updated to improve the accuracy and efficiency of the ST-PAC model.

5.3.1 Memorising the Teacher Example

A teacher example represents a correctly executed strategy for solving one instance of a particular problem. In order to provide a more formal definition, we firstly note that the superscript T is used to denote variables associated with teacher examples. A single teacher example is then defined as the tuple: (A^T, X^T, P^T). The sequence of $|A^T|$ number of actions that forms the strategy to be memorised is defined as $A^T = \{A_t^T\}_{t=1}^{|A^T|}$. The set of parameters for the actions are defined as $P^T = \{p_t^T\}_{t=1}^{|A^T|}$. Additionally, the input list observed when each action in the sequence was executed is also provided, and defined as $X^T = \{X_t^T\}_{t=1}^{|A^T|}$. From this, it is possible to build a set of underlying feature vectors for the action parameters: $F^T = \{f_t\}_{t=1}^{|A^T|}$, where:

$$f_t = X_{p_t^T}^T \tag{4}$$

With the above definition, the memorisation of the teacher example is straightforward. The action sequence of the model A is then A^T. At present, there is only a single correct instantiation of this strategy, namely that shown by the teacher. When the underlying feature vectors in F^T exists (i.e. $f_t \neq \emptyset$), they are used to initialise the action parameter models: their kernel centres are set as the feature vectors themselves, $\mu_{1,t,1} = f_t^T$ and since this is a correct instantiation, we only have one kernel at present, so $|r_{1t}| = 1$, and the respective kernel weights are all one (i.e. this is a successful example), $u_{1,t,1} = 1$. An illustration of the memorisation process can be seen in Figure 5.

5.3.2 Planning the Strategy: Action Parameter Estimation

Given that the strategy model exists (e.g. initialised by memorising a teacher example), it is now possible to plan the usage of a strategy given some input list of entities in the world $X = \{x_i\}_{i=1}^{N_O}$ (as defined in Section 5). This involves estimating the parameter indices for each action in the strategy. To achieve this, we firstly introduce the method for obtaining the best parameter for an action (A_t) using the action parameter model (r_{it}) of a particular strategy instantiation (R_i):

$$P_{it}^X = \arg max_j r_{it}(x_j) \tag{5}$$

where the superscript X is used to denote that the index of an action obtained using the input list X. Now, the final set of parameters for the actions can be obtained as follows:

$$P^X = \arg max_i \sum_{t=1}^{|A|} r_{it}(X_{P_{it}^X}) \tag{6}$$

where $P^X = \{p_t^X\}_{t=1}^{|A|}$ is defined as the obtained parameter set given the input list X. An illustration of this process is shown in Figure 6.

5.3.3 Obtaining Learning Feedback

With the obtained parameter index set, it is now possible to execute the strategy and obtain the relevant feedback as to whether this estimation was correct or wrong. One straightforward method to achieve this goal is to simply carry out the actions in the model using the estimated parameters above (P^X), and return a failed feedback if any of the actions failed to be executed (i.e. $A_t(p_t^X) = -1$). To improve on this, random exploration will be used when failure occurs in executing an action. More specifically, suppose that we have

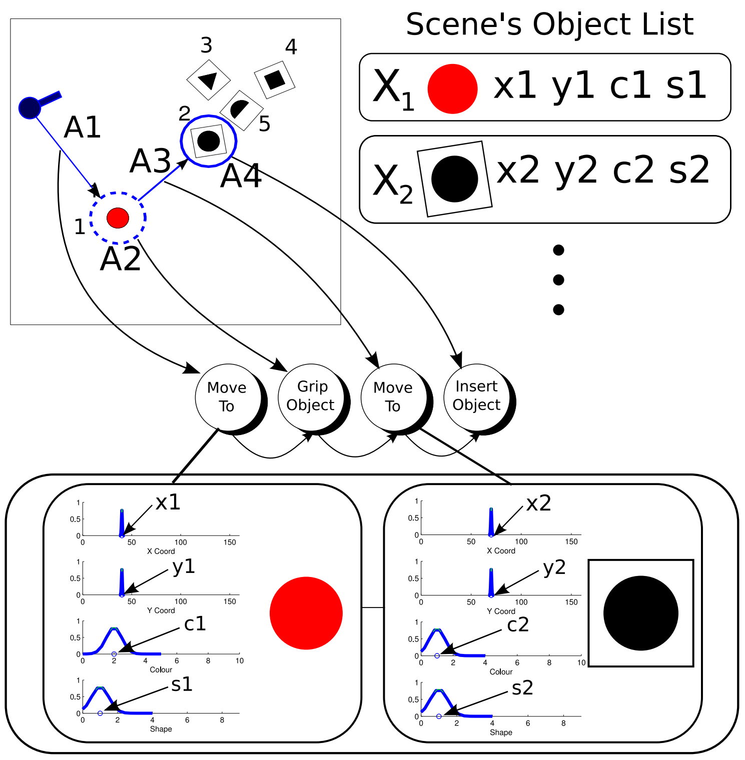


Fig. 5. An illustration of how a teacher example is memorised. The teacher example is given as a sequence of four actions $(A_1, ..., A_4)$. Each action has a parameter. Suppose the parameter for A_1 is 1 (circle), and for A_3 is 2 (circular hole). The underlying feature vectors for these two actions can be directly obtained from the input scene list as $X_1 = (x_1, y_1, c_1, s_1)$ and $X_2 = (x_2, y_2, c_2, s_2)$ respectively. A single APM can the be constructed, where the kernel centres for its two kernel models with non-zero B_i values, r_{11} and r_{13}, is set to X_1 and X_2 respectively.

successfully executed $t-1$ actions, and the current action to be carried out is now $A_t(p_t^X)$. Should A_t be unsuccessful, it is possible to then "randomly try" all other possible parameters (i.e. indices of other entities in the input list X).

In order to incorporate random exploration into the strategy execution process, the estimated action parameter values P^X are combined with random exploration parameters in the form of an *action parameter list*. Formally, the action parameter list for A_t is defined as:

$$\hat{P}_t = \{p_t^X, Q_t\} \tag{7}$$

where Q_t is the random permutation of the set $\{1, ..., N_O\}$. The algorithm for executing a planned strategy with the aid of random exploration is as follows:

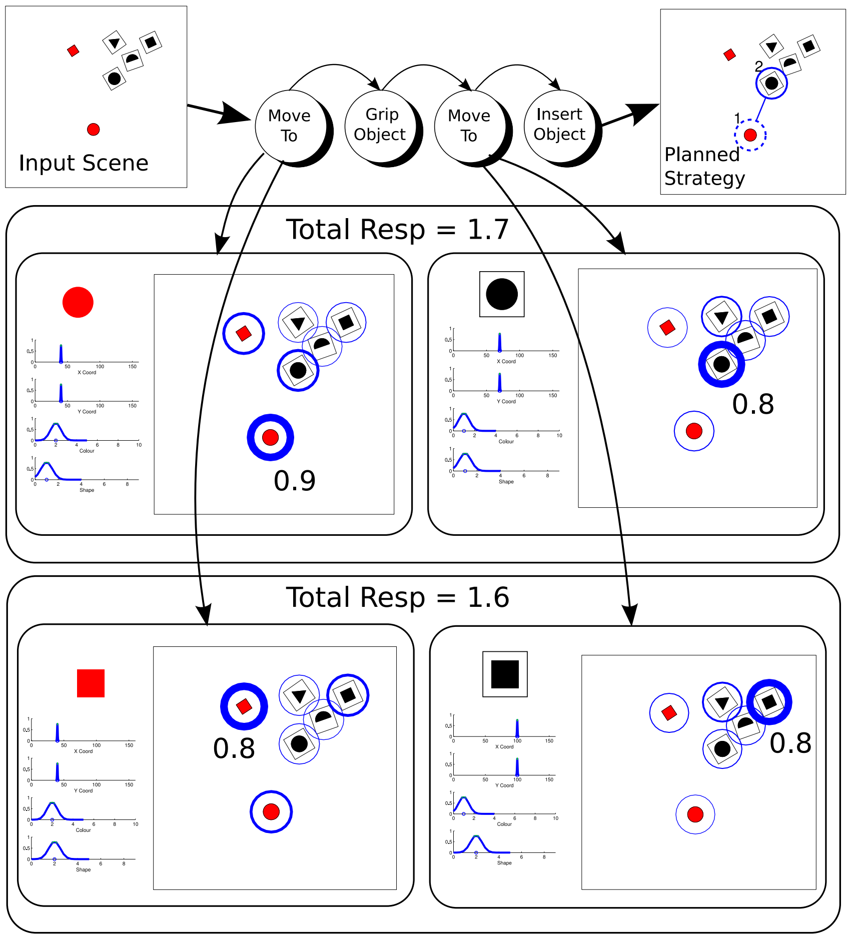


Fig. 6. An illustration of using the APMs to estimate the parameters of actions for a given strategy. Suppose we are given an input scene as show on the top left of the figure. It is then possible to obtain the responses of each APM's kernel models for each input object. The magnitude of the kernel model's response ($r_{it}(x_j)$ from Eq. 5) is shown in proportion to the thickness of the circle around each object. The value of the largest response is also shown next to the corresponding object. The total response for each APM on the input scene (Eq. 6) is then shown. In this case, the APM for inserting a circular object has the largest total response and will be the system's next set of actions to be carried out, as shown on the upper right of the figure.

```
$t = 1$ {Initialise to start at the first action}
$successF = []$ {Initialise list of feature vectors resulting in success}
$successP = []$ {Initialise the correct action parameter index set}
$randomExFlag = 0$
while $t \leq |A|$ do
   Get new updated input list $X$
   $curP_t = \hat{P}_{t1}$
   if $curP_t \neq -1$ then
      $\hat{P}_t = \{\hat{P}_{t2}, ..., \hat{P}_{|A|}\}$ {Remove first element from set $\hat{P}_t$}
   end if
```

```
        if $A_t(curP_t) == 1$ then
            $successF[t] = X_{curP_t}$
            $successP[t] = curP_t$
            $t = t + 1$ {Go to the next action}
        else
            randomExFlag = 1 {Note that random exploration was used}
            if $|P_t| = 0$ then
                break from while loop
            end if
        end if
    end while
    if $t > |A|$ then
        return $(1, randomExFlag, successF, succesP)$
    else
        return $(0, randomExFlag, P^X, [])$
    end if
```

The algorithm above returns a triplet: (S_f, R_e, F_S, P_S). The success flag ($S_f \in \{0, 1\}$) will indicate if the system was ultimately successful in solving the problem using the strategy containing the sequence of actions A. When a successful instantiation of the strategy used was made ($S_f = 1$), we can then determine the exact method used with the random exploration flag ($R_e \in \{0, 1\}$). Here, $R_e = 0$ indicates the the initial parameters estimated using the action parameter models (Section 5.3.2) were successfully used. If $R_e = 1$, this indicates that random exploration was used to find the correct action parameters for solving the current problem configuration. In the framework proposed here, should the system fail to solve the puzzle despite using random exploration ($S_f = 0$), it will be reported to the user that A represents an unsuitable strategy for solving the current problem.

5.3.4 Updating the Action Parameter Models

Using the algorithm described in the previous section, it is now possible to obtain learning information for updating the action parameter models. We note the update to the action parameter models will only take place if the system was ultimately successful in solving the problem. However, there are two different methods for updating the relevant APMs.

The first case is when random exploration was not used ($R_e = 0$). This implies that the initial estimated parameters in Section 5.3.2 were correct. Suppose that the i^{th} strategy instantiation (R_i) was used to obtain the correct initial action parameters. The updating of its relevant APMs is simply an addition of a new kernel centre to it. This kernel centre will be the feature vectors given in F_S.

The second case is when random exploration was used ($R_e = 1$). In such cases, there can be two possible causes: the first is due to the feature weights being suboptimal, implying the system is not utilising available information as best as it should; the second is that the existing strategy instantiations are simply inadequate for dealing with the problem configuration presented.

In order to decide between these two, we start by updating the feature weights ($C = \{c_d\}_{d=1}^{D}$) in such a way as to reduce the estimation of the wrong parameters given the previously presented problem configuration. To achieve this, suppose that the original, incorrect, estimated action parameter set is P^X, to be used with the input list X (see Section 5.3.2). We will denote the corresponding "wrong" feature vectors set as $\tilde{F}^X = \{\tilde{f}_t^X = X_{p_t^X}\}_{t=1}^{|A|}$, and the correct set, as given by random exploration, $F_S = \{f_t\}_{t=1}^{|A|}$. The absolute difference between $\tilde{F}^X$ and F_S is: $\delta F = \{\delta f_t = |\tilde{f}^X - f_t|\}_{t=1}^{|A|}$. An updated feature vector weight can now be computed as follows:

$$c_d^{new} = \max(\delta f_t)\alpha_d, \forall d \in \{1, ..., D\} \tag{8}$$

where α_d is the increment factor for updating the feature weights. How different settings of α_d affect the system performance with regards to the complexity chain will be discussed and analysed in Section 6.

Having updated the feature vector weights, the original input list X can then be presented to the system again, and obtain a new estimated action parameter set (i.e. P^X) using the updated feature vector weights. In this case, if P^X is correct, it will match the "correct" action parameter set originally obtained by random exploration, P_S, that is:

$$\prod_{i=1}^{|A|} B_i(p_i^X - p_{S,i}) = 0 \tag{9}$$

When this happens, the strategy instantiation that was responsible for producing this parameter set is updated with the correct feature vectors F_S.

However, when the re-estimated P^X does not completely match P_S, this implies that existing strategy instantiations were inadequate for solving the problem encountered. As a result, a new strategy instantiation is added to the strategy model, where the parameters of the APM is that given by random selection. The details of how this is done is exactly like those in initialising the system with a single strategy instantiation at the start (Section 5.3.1).

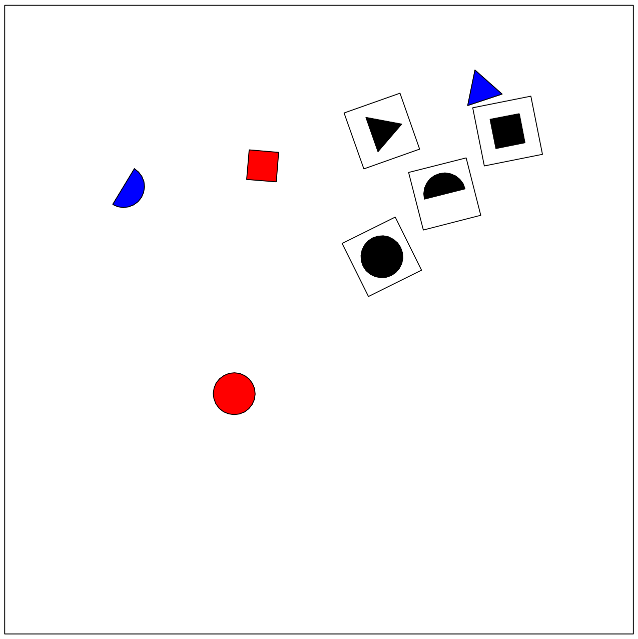


Fig. 7. An example of the COSPAL shape sorter puzzle. There are 4 puzzle pieces along with corresponding holes (shown with square borders).

6 Experiments

The main aims of the experiments performed in this section are to test whether the proposed methods are capable of successfully learning a teacher example and imitating it to solve similar examples. Additionally, we also wanted to determine if the act of "imitation" can be further generalised to examples where starting configurations are different to that given by the teacher example. Finally, it was also necessary to see if these learnt examples can be extended to different unseen objects. Initial experiments are performed on a simulator for the COSPAL environment. Two sets of experiments were performed. The first set of experiments is carried out by progressively applying the system through the different levels of the complexity chain described in Section 4. This is repeated for various shape sorter pieces. All the pieces are presented simultaneously to the system. Additionally, we also show example scenarios of the system applied to the live COSPAL demonstrator system.

6.1 Experimental Setup

For evaluation, a simulated virtual shape sorter puzzle will be used. This consists of a "board" with a number of different shaped holes in it. These holes are different to those in more real-world shape sorter puzzles in that they can be arbitrarily positioned and oriented. This flexibility is used for the last level in the complexity chain. Similar corresponding shaped pieces are also

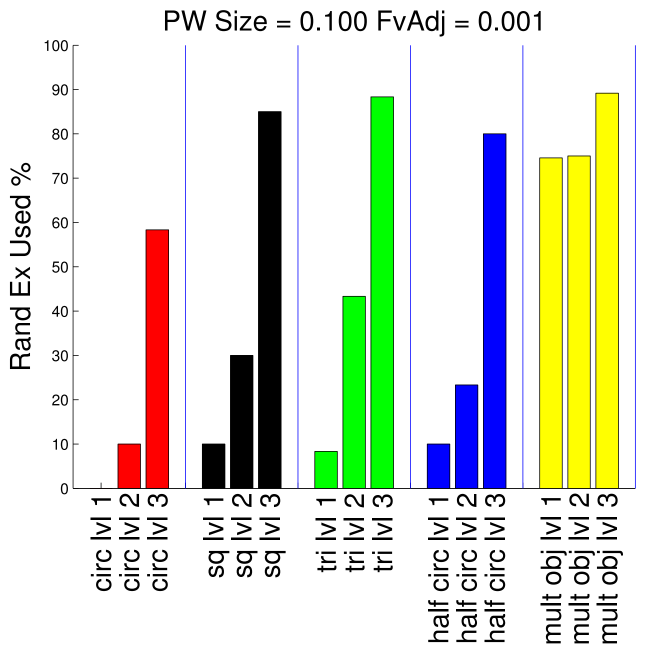


Fig. 8. The complexity chain performance graph for a single parameter setting of the system.

available. The objective of the game, is to place the pieces into similarly shaped holes. For the experiments, the puzzle will have 4 different holes and pieces with the following shapes: circles, half circles, squares and triangles (Figure 7). The holes and pieces will be represented using fixed dimensional feature vectors of the form: $(x, y, shapeid, r, g, b)$, where (x, y) is the 2D position of the piece, $shapeid \in \{1, 2, 3, 4, 5\}$ is a unique identifier for each of the five different shapes, and (r, g, b) are the mean red, green and blue colour intensities for the object respectively.

In order to interact with the environment (e.g. shape pieces), a gripper object is provided. The gripper has the following three different capabilities (with the required arguments): $moveto(listID), align(), gripping()$ and $insertion()$. Both the grip and insertion have no arguments and work on objects that are directly below the system's gripper.

Prior to carrying out the experiments, a teacher example demonstrating one instance of a single shape placed into its respective hole is given. Specifically, a single example of a circle being placed into a circular hole is given: $moveto(circleobjlistID)$, $align()$, $grip()$, $move_to(circleholelistID)$, $align()$, $insertion()$. This example is then presented to the system to be memorised. Following this, the next section will describe the set of experiments that use the complexity chain to evaluate the performance of the system when presented with problems that are increasingly complex.

The aim of the experiments will be to use the complexity chain to evaluate the performance of the system with different parameter settings. The results will then allow us to analyse the importance of the different components of the system with regard generalisation (i.e. to solve problems of increasing complexity through imitation). Firstly, we define the *performance* of the system at each level as the percentage of times random exploration was used to solve the puzzle. In other words, it is a measure of how many times the proposed strategy model within the system failed to provide a correct solution to some test example.

The system is then presented with examples from the entire complexity chain. We start with the first three complexity levels for the circular object, since it was the teacher example presented. Upon completion, we move on to the fourth complexity level. This involves presenting examples from the first three complexity levels for the square object to the system. Following this, the triangle and half circle object are also presented. For the fifth and final complexity level, scenes with all objects present are presented. Again, these scenes are of configurations with objects at positions increasingly distant to those in the first levels of their respective complexity level. For a single complexity level, 12 example scenes are consecutively presented to the system. The performance of the system in terms of the percentages out of these 12 examples where the system required random exploration to complete the puzzle are then obtained. These series of tests are repeated 10 times and the average performance of the system at each complexity level is shown in a *complexity level performance graph* (Figure 8).

This shows the system not generalising very well with the shown parameter values ($\sigma = 0.1$ and $\alpha = 0.001$). This can be seen from the increasing amounts of random explorations as the system is presented examples that are increasingly further away from the teacher example. The failure of generalisation is finally confirmed at the final level where multiple objects are present. Here, 70-90% of the time, random exploration is required. The other examples are solvable by the system simply because they are close to starting configurations memorised by the system from previous test examples.

It is important to note that Figure 8 only shows the system for one particular parameter setting. To gain a deeper understanding and assess the overall performance of the system, the above complexity chain experiment is run over a range of parameter settings. In particular, we vary the feature weight adjustment value (α) from 0.001 to 1 at increments of 0.25. The parzen window bandwidth (σ) is also varied from 0.1 to 2.1 at increments of 0.5. This results in a grid of 25 different complexity chain performance graphs which will be

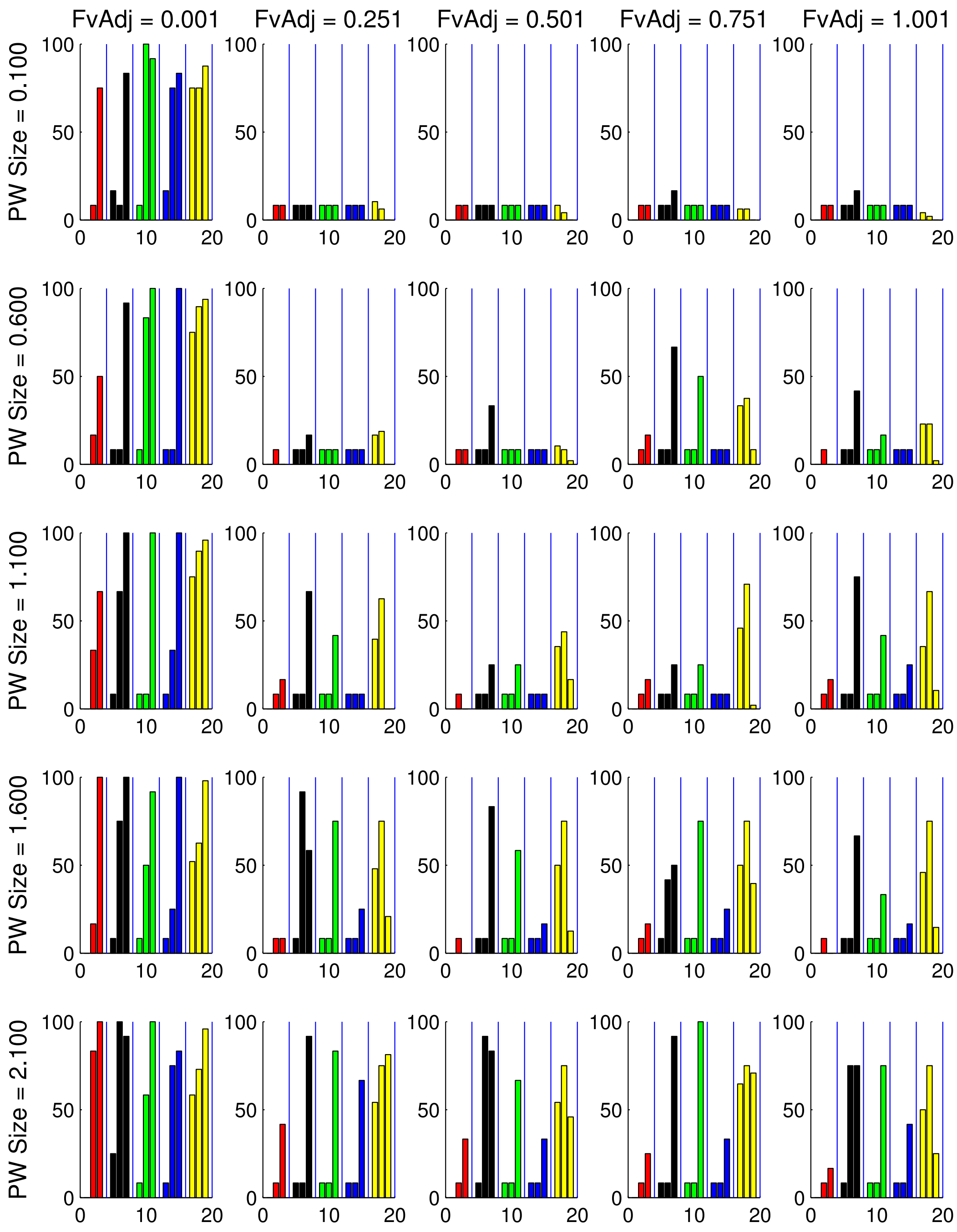


Fig. 9. The results of the system's performance across the complexity hierarchy over various settings for the parzen window size and the feature vector weight update value.

discussed in the next section.

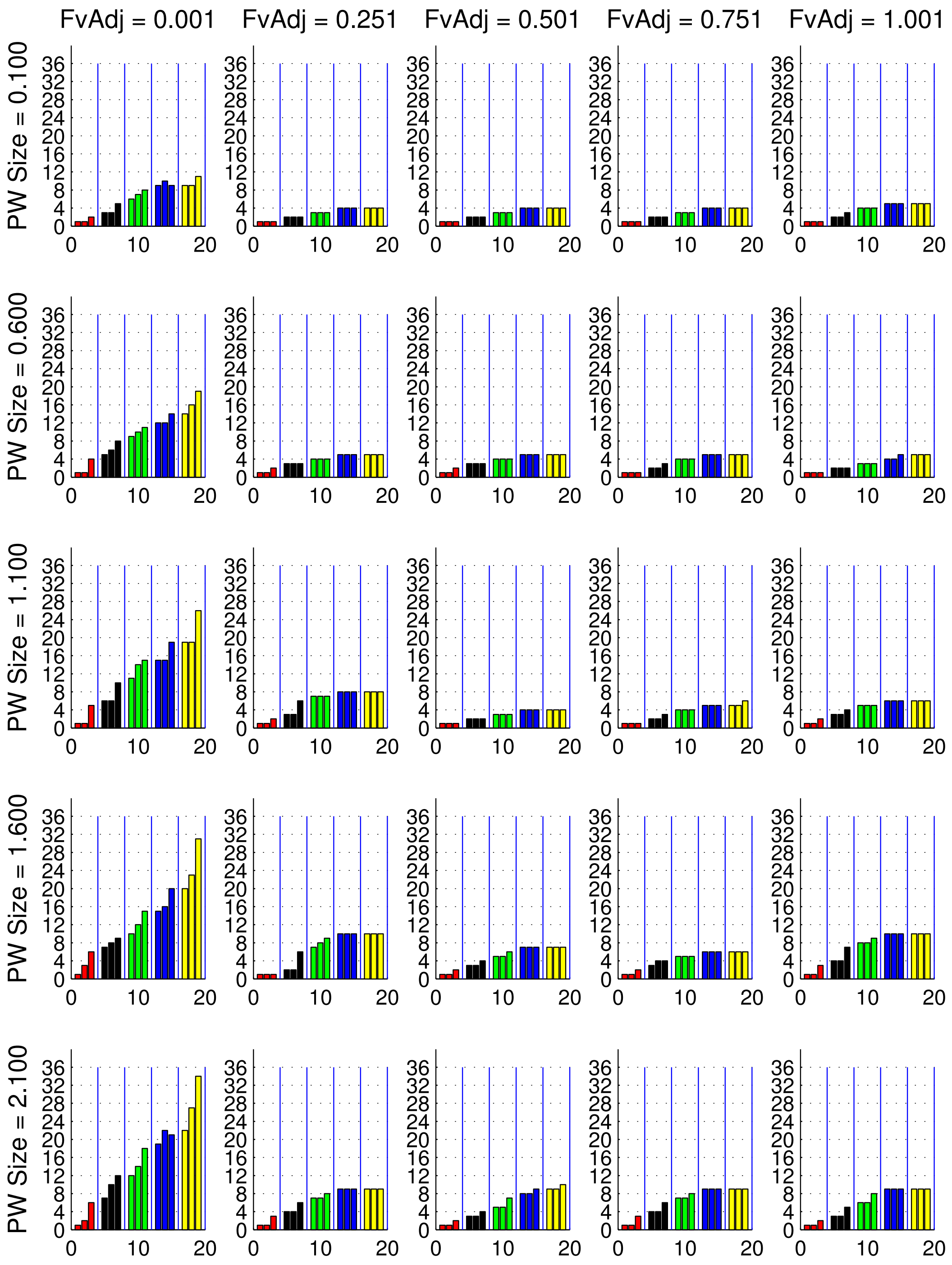


Fig. 10. This figure shows for each system parameter setting, how many strategy instantiations (APMs) were created at the end of each complexity level (y-axis of each graph). The ideal APM number should be 4, since there are only four puzzle shapes available.

The grid of complexity chain performance graphs for the experiments performed are shown in Figure 9. Here, 25 different parameter configurations were tested. The first thing one notices is that when the feature weight adjustment value is very small (0.001), regardless of the setting for the parzen window size, the performance of the system is on the whole very bad. As the complexity level goes up for each of the objects, the amount of random exploration increases as well. This is apparent for all objects, as well as the multiple object level. This is because important features are not emphasised soon enough. This results in too many APMs being created, as can be seen in Figure 10. Ideally, since there are only four different types of puzzle pieces, there should only be four APMs. If too many are created, it stops any one APM's kernel model from being updated frequently enough to provide a good generalisation capability.

However, when the feature adjustment level is appropriate, from 0.251 onwards, the performance of the system then becomes dependant on the size of the parzen windows. The performance of the system deteriorates as the parzen window size increases, as is reflected in the performance graphs with parzen window values of 1.1 and above.

An increase in parzen window size effectively increases the radius of the Gaussians in the kernel models. This will cause the feature subspaces modelled by different APMs to overlap. As a consequence, there in an increase in the ambiguity of responses for memories of different strategy instantiations (modelled by APMs). This is not apparent in results where only one puzzle piece is present. However, with multiple puzzle pieces, there is an increase in the use of random exploration. This is because the system may, for example, apply the strategy instantiation of filling square objects into square holes to a circle objects, in effect putting circular objects into square holes.

6.4 Application to the Live COSPAL Demonstrator System

The proposed system was finally applied to the live COSPAL demonstrator system aimed at solving the shape sorter puzzle [3]. Again, the system was presented with various configurations of the shape sorter puzzle, as structured by the complexity chain.

The live COSPAL system is configured very closely to the simulated environment described in the previous subsections. There is a robot manipulator that provides the four capabilities: $moveto(listID), align(), gripping()$ and $insertion()$. The mechanisms for learning and providing the motor controls required for these four capabilities are described in [12,11,13]. Additionally, numerous object detection and shape classification methods were used to provide the input list representation is described in [4,5,8].

The planned strategies provided by the system at the end of the complexity chain can be seen in Figure 11 and 12. In these figures, a grip command is visualised as a hashed-line circle, whilst a release command is shown as a full line circle. The line between the two circles illustrates a move-to command. The object is always gripped first, then moved to somewhere before released. The thickness of the line shows the system's confidence in the planned strategy, which is proportional to the response of the memories that were used.

In Figure 11, we see the situation where all puzzle pieces and their respective holes can be seen clearly. In (a) the detections of all objects (puzzle pieces and holes) are highlighted by circles. The planned strategies for each puzzle piece are separately shown for clarity in (b) - (e). Here, (b) shows the strategy of inserting square objects into square holes. (c) shows bridge-shaped pieces inserted into their respective holes. (d) shows for triangular objects, (e) for half cylinders and finally (f) for cylinders.

In Figure 12, we see how the system reacts when a hole for a puzzle piece is occluded. Additionally, the camera viewpoint has changed which has the effect of "moving" the board. Despite these changes, it can be seen that the system still manages to perform well in proposing possible strategies for solving the puzzle. The puzzle pieces and board holes can be seen in (a), with their detections shown in (b), highlighted by circles. The planned strategies for solving the puzzle is visualised in (c) - (e). In (c), we see the strategy planned by the system for square objects. Similarly, in (d) and (e), we see the strategy of filling the holes of bridges and triangles respectively. It can be seen that (c) - (e) all show correct moves. In (f), we see that the system has decided to place half cylinder holes into a circular hole. This was because the half cylinder hole itself was occluded by the triangular object. However, the confidence in this strategy is very low, since filling circular holes require circular objects, whereas in this case, half cylinders were used instead.

7 Discussions

In the previous section, various experiments were described on both the simulated and live *shape sorter* systems. In both cases, the complexity chain was used to structure the scenarios presented to the system. However, there is one fundamental difference between the simulated system and the live system, that of speed of operation. The live COSPAL demonstrator system consists of multiple sub-systems, some responsible for motor control, some responsible for

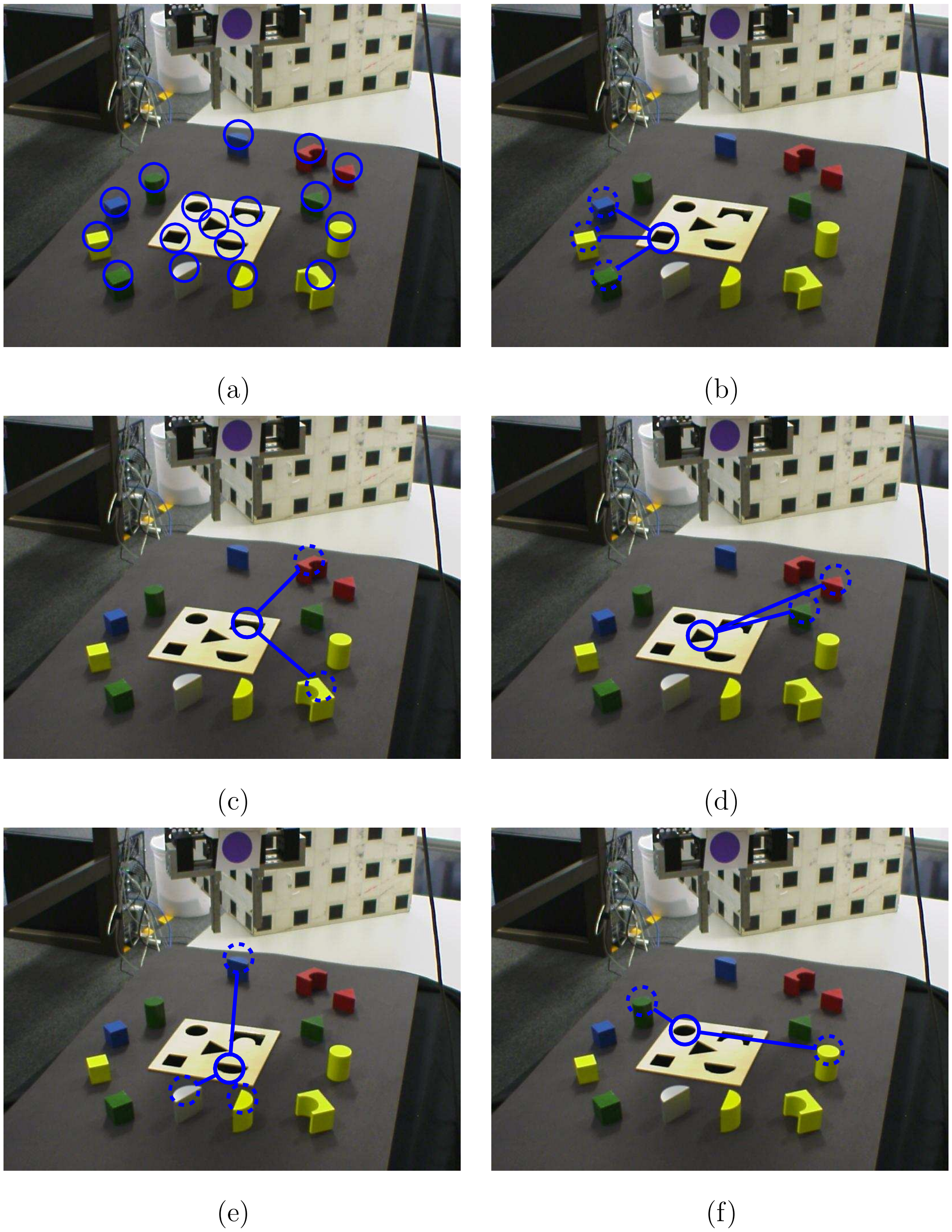
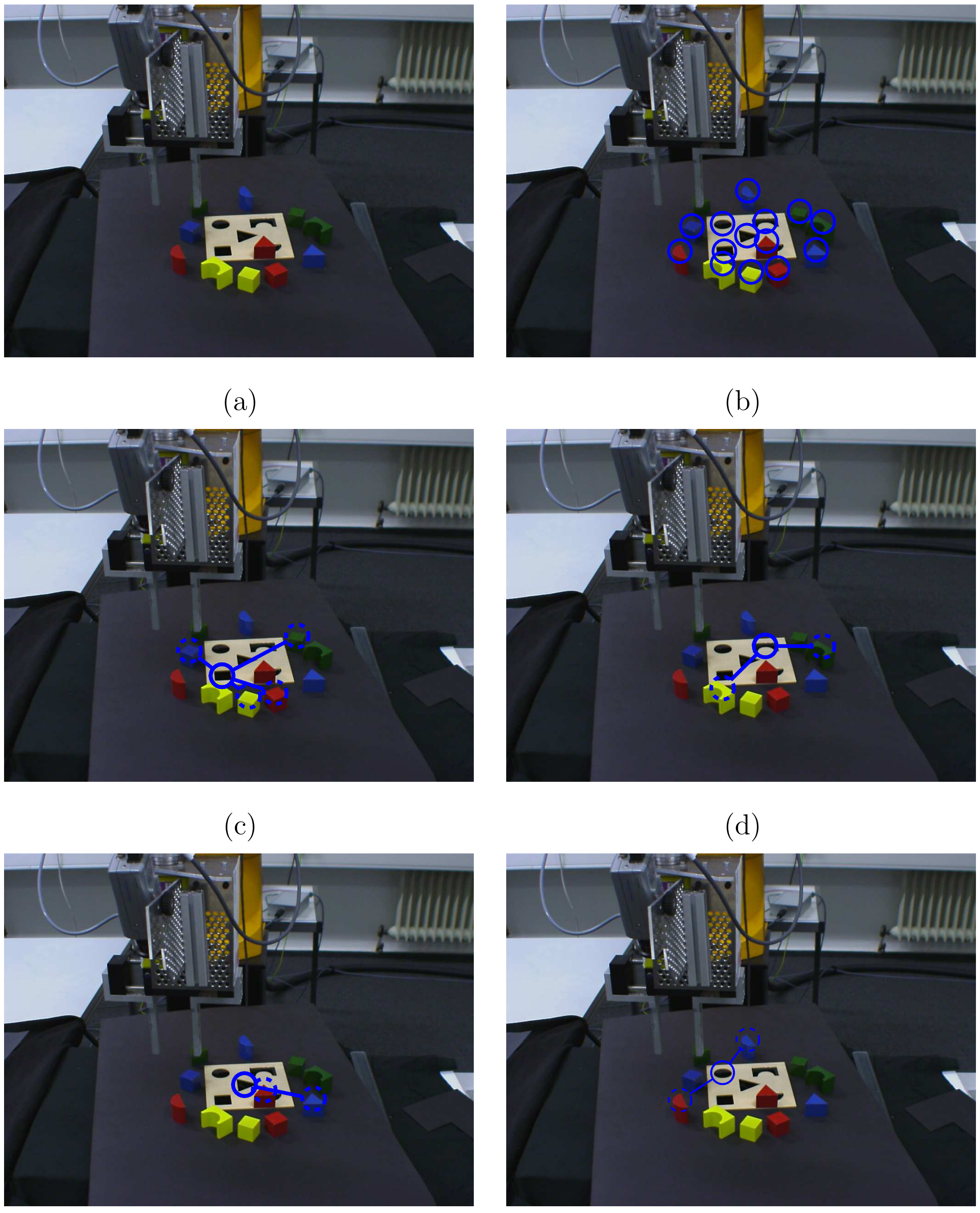

(a) (b) (c) (d) (e) (f)

Fig. 11. This figure shows the proposed system applied to the real COSPAL shape sorter puzzle. Detailed explanations can be found in Section 6.4.

visual object detection. In the simulated environment, a virtual robot can be made to move instantaneously, a real robotic manipulator will requires time, with additional constraints for safety. Additionally, the visual subsystems will have to deal with image noise. All these factors eventually result in the live demonstrator system being significantly slower than that of a simulated sys-



(a) (b) (c) (d) (e) (f)

Fig. 12. This figure shows the proposed system applied to another configuration of the real COSPAL shape sorter puzzle. Detailed explanations can be found in Section 6.4.

tem.

The requirement of accuracy in the live system is also of paramount importance. In the simulated system, a perfect *grab* or *moveTo* command can be guaranteed by relaxing the tolerances between the alignment of objects, holes and manipulator. This is far more difficult to achieve in a live system that visually guides a robot manipulator in real time. In a live instantiation, slight alignment issues can cause complete failure of the system. As a result, even if the strategy is correct, if the robot releases a piece slightly offset (say 0.01 mm) from the centre of its respective hole, the puzzle piece can catch the edge of the hole does not pass through the hole successfully. This can result in failures, not because of incorrect strategies from the proposed system, but because of the small errors in the visual and motor capabilities of the system. However, it is important to note that ultimately, this is not a problem. In such cases, random exploration is used and the proposed strategy will eventually be executed correctly. Once done, the outcome of the random exploration will match that initially proposed by the system, and this will only serve to strengthen and reinforce the correct memory models in the system as described in Section 5.3.4. However, this process of failures due to finite alignment issues, serves to further slow down the operation of the live system.

Due to these factors, it is not practical with the current implementation to carry exhaustive experimentation in the live system. The amount of time it would have taken to complete a similar number of experiments to those performed in simulation is simply not feasible. However, it is a benefit of the system architecture that learning at this high symbolic level is consistent across domains and lessons learnt in the simulation can be directly applied to the live system.

8 Conclusions

To conclude, this paper proposed a system to solve problems through imitation. This was then applied the domain of a shape sorter puzzle within the COSPAL system. To this end, an incomplete set of experiences provided by a teacher was memorised with the aim of being used to generalise and efficiently recall appropriate experiences given novel scenarios of similar problems. Additionally, random exploration was also used as a fall-back “brute-force” mechanism should a recalled experience fail to solve a scenario.

To this end, the ST-PAC system was proposed which ties together the input percepts (i.e. a description of the world) with the actions that should be performed given that scenario. Statistical models were used to couple groups of percepts with similar actions. An approach to incremental learning that provided good generalisation was also proposed.

Furthermore the concept of the *Complexity Chain* was proposed as a way of structuring learning and a method for evaluating a cognitive system’s performance. The system was tested in two types of experiments, a simulated environment and a live demonstrator system. For both environments, it was found that the system provided a platform that allowed both generalisations over experiences from a sparse set of memorised examples provided by a tutor and the capability to refine learning in light of new experiences.

References

[1] G. Buccino, F. Binkofski, L. Riggio, The mirror neuron system and action recognition., Brain Lang 89 (2) (2004) 370–376.

[2] L. Ellis, R. Bowden, Learning responses to visual stimuli: A generic approach, in: Proc. of the 5th International Conference on Computer Vision Systems, 2007.

[3] M. Felsberg, P.-E. Forssén, A. Moe, G. Granlund, A cospal subsystem: Solving a shape-sorter puzzle, in: AAAI Fall Symposium: From Reactive to Anticipatory Cognitive Embedded Systems, 2005.

[4] M. Felsberg, P.-E. Forssén, H. Scharr, Channel smoothing: Efficient robust smoothing of low-level signal features, IEEE Transactions of Pattern Analysis and Machine Intelligence 28 (2) (2006) 209–222.

[5] M. Felsberg, J. Hedbord, Real-time visual recognition of objects and scenes using p-chennel matching, in: Proc. of 15th Scandinavian Conference on Image Analysis, 2007.

[6] M. Felsberg, J. Wiklund, E. Jonsson, A. Moe, G. Granlund, Exploratory learning structure in artificial cognitive systems, in: International Cognitive Vision Workshop, 2007.

[7] P. Fitzpatrick, From first contact to close encounters: A developmentally deep perceptual system for a humanoid robot, PhD thesis, MIT.

[8] P.-E. Forssén, A. Moe, Autonomous learning of object appearances using colour contour frames, in: 3rd Canadian Conference on Computer and Robot Vision, 2006.

[9] T. Gelder, R. Port, It's about time: an overview of the dynamical approach to cognition (1995) 1–43.

[10] G. H. Granlund, The complexity of vision, Signal Processing 74 (1) (1999) 101–126.

[11] F. Hoppe, Local learning for visual robotic systems, Ph.D. thesis, Christian-Albrechts-Universität zu Kiel, Institut für Informatik (2007).

[12] F. Hoppe, G. Sommer, Fusion algorithm for locally arranged linear models, in: Proc. of 18th Int. Conf on Pattern Recognition (ICPR), 2006.

[13] F. Larsson, E. Jonsson, M. Felsberg, Visual servoing for floppy robots using lwpr, in: RoboMat, 2007.

[14] J. Siskind, Reconstructing force-dynamic models from video sequences, Artificial Intelligence 151 (1-2) (2003) 91–154.

[15] E. Thelen, L. Smith, A Dynamic Systems Approach to the Development of Perception and Action, MIT Press, 1994.

[16] D. Vernon, G. Metta, G. Sandini, A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents, IEEE Transactions on Evolutionary Computation, Special Issue on Autonomous Mental Development.

[17] D. Young, First-order optic flow and the control of action, in: Proc. of European Conference on Visual Perception, Groningen, 2000.