Planning Information Processing and Sensing Actions

Mohan Sridharan, Nick Hawes, Jeremy Wyatt, Richard Dearden and Aaron Sloman

School of Computer Science
The University of Birmingham and other partners
Edgbaston
Birmingham B15 2TT
{mzs,nah,jlw,rwd,axs}@cs.bham.ac.uk
http://www.cs.bham.ac.uk

November 17, 2007

Abstract

The goal of the CoSy project is to create cognitive robots to serve as a testbed of theories on how humans work [13], and to identify problems and techniques relevant to producing general-purpose human-like domestic robots. Given the constraints on the resources available at the robot's disposal and the complexity of the tasks that the robot has to execute during cognitive interactions with other agents or humans, it is essential that the robot perform just those tasks that are necessary for it to achieve its goal. In this paper we describe our attempts at creating such a system that enables a mobile robot to plan its information processing and sensing actions. We build on an existing planning framework, which is based on Continual Planning [3]. Continual planning combines planning, plan execution and plan monitoring. Unlike classical planning approaches, here it is not necessary to model all contingencies in advance – the agent acts as soon as it has a feasible plan, in an attempt to gather more information that would help resolve the uncertainty on the rest of the plan. We describe how the system addresses challenges such as state representation, conflict resolution and uncertainty. A few experimental results are provided to highlight both the advantages and disadvantages of the current approach, and to motivate directions of further research. All algorithms are implemented and tested in the playmate scenario.

Contents

1 Introduction

We want to understand how to build robots that are able to perform well on a variety of tasks in a given domain. Such tasks might require the ability to have dialogues with a human about the objects in a scene, to grasp these objects, to identify groups of objects and to understand their spatial relations. Even if we ignore other modalities and just consider the range of *visual* abilities required to perform these tasks it is formidable. To be truly flexible a robot must be able to construct representations of a wide variety of aspects of the visual scene. To do this it might need the ability to find where objects are, what their shape is, to identify them, to categorise them, to identify grasp points, to recognise faces, to track moving objects, and to predict what will happen next in a changing scene. But each of these visual tasks is hard in itself, and creating a system that constructs a model of the world containing information about all these quantities is not only beyond our abilities as of today, but in the opinion of some researchers will remain so permanently. It is not realistic to suggest that we would build a system that would, regardless of its task, extract all these pieces of information from even a small area of the visual scene. In fact there is plenty of evidence that at least some visual processing in humans and other primates is performed selectively. The idea of visual attention, that only certain areas of the scene are processed, is a familiar one. Slightly less familiar is the idea that a machine performing visual processing might also make choices about the precise processing to be done on each area of the scene. An idea with a growing body of evidence from both animals and robots is that vision can be made more effective by tailoring it to the task. It is not yet clear how this process works in humans. In robotics it is usually the case that the human designer hand crafts visual routines for the robots so that it has a task specific, and limited range of visual abilities. In this paper we ask how visual sensing (where to look) and visual processing (what to look for) could be planned jointly. In other words, how, given a task, a robot could infer what visual processing, of which areas of the scene, should be executed. This is what we refer to as an example of joint planning of sensing and sensory processing. The framework we shall outline here is tentative, and frankly at this stage it is also rather rudimentary. Our first foray into this area is to think about how we can pose the problem as a symbolic planning problem. We are not the first to attempt this. Our approach, does however have some novel aspects. First, we look at how, for some fairly simple visual tasks, the planner can reason explicitly about the change that the visual processing will induce upon its knowledge or belief state. Second we try to pose the problem in terms of continual planning as a way of dealing with the fact that the outcome of any visual processing is uncertain.

The rest of the paper is organized as follows: we begin with a specification of the problem in terms of a specific scenario involving a visual scene and various queries about it that a visual system might be asked to run (Section 2). We then constrain this by giving an overview of the cognitive architecture schema that forms the basis of all our work and experimentation (Section 3). This architecture schema significantly constrains and influences how we will pose the planning problem. Section 4 briefly outlines some approaches that could be employed. The background information on the baseline planning approach (continual planning) that we build on is provided in Section 5. We then show how some visual operators with relatively simple epistemic effects can modelled using planning operators (Section 6). After a brief description of related work (Section 7), Section 8 summarizes the work, and discusses what we see as the drawbacks of the proposed approach. At the end we will argue that while posing the problem in this form is a necessary first step, that explicit reasoning about the uncertain outcomes of visual processing (using probabilistic models and decision theoretic planning techniques) may be a worthwhile alternative.

2 The Problem: Planning of Sensing and Sensory Processing

Consider the scene in Figure 1. To reduce the scope of our problem to something conceivable, consider the types of visual operations that the robot would need to perform to answer a variety of questions that a human might ask it about the scene: "is there a blue beaker?", "what colour is the box?", "how many objects are there?". In order to answer these questions, the robot has at its disposal a range of information processing functions and sensing actions. For instance, it may need to run a particular object recognition algorithm to recognize certain types of objects, and it may need to look at scene from different viewpoints in

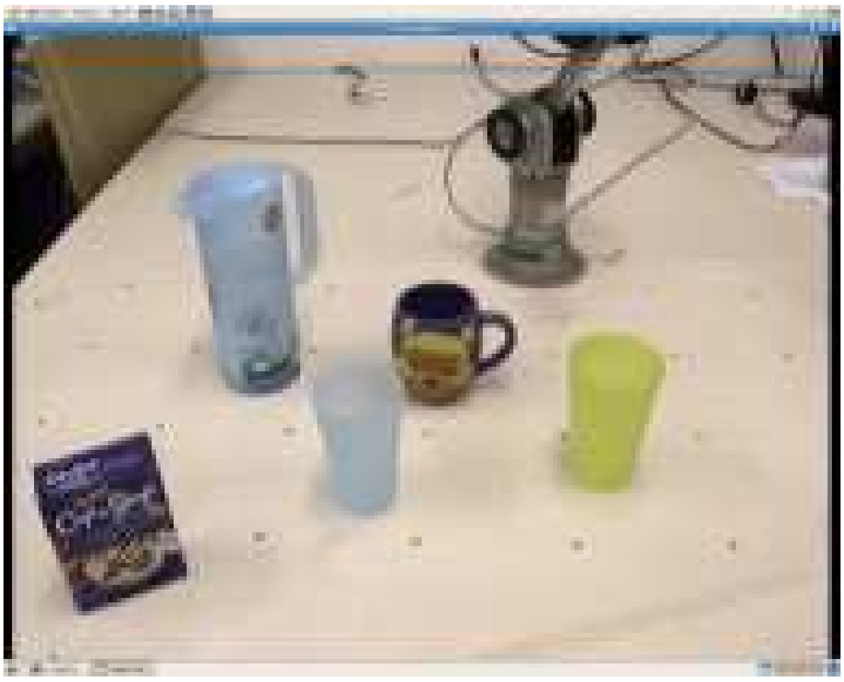

Figure 1: A picture of the typical table top scenario.

order to resolve occlusions and determine all the components of the scene. But, in any reasonably complex scenario (such as the one described above), it is not feasible (and definitely not efficient) for the robot to run all the information processing functions and sensing actions, especially since the cognitive robot system needs to respond to human queries/commands in real-time. For instance, if the robot knows that there is a single blue coloured region in its view, and its goal is to find a blue beaker, it is not necessary to run the beaker-detection algorithm over the entire image. The robot therefore needs some mechanism that enables an autonomous choice of information processing and sensing actions.

The problem of jointly choosing the appropriate sensing actions and information processing functions could possibly be addressed by two different approaches: learning and planning [1]. In the learning approach, the aim would be to have the robot *learn* the combination of information processing and sensing actions that is essential to respond to a particular query. In order to do so, the robot would attempt to respond to queries using different possible combinations of the information processing and sensing actions, and these combinations would be scored based on the level of success the robot achieves in responding to the query (possibly even with human feedback). Over several repeated trials, the robot may then converge on a possible sequence of actions to execute in response to a particular query. Methods such as Reinforcement Learning [24] would be used to formulate and solve the problem. The main disadvantage of posing the problem entirely as a learning task is that performing repeated trials on a physical robot is a 'expensive' operation, which would involve significant human effort and considerable time, and may not be a feasible endeavour.

In this work we explore the other alternative, i.e. that of enabling the robot to jointly *plan* its information processing and sensing actions under different scenarios. The end result is still the same as before: the robot should be able to choose the best combination of information processing and sensing actions to apply in response to a human query/command. But instead of learning over repeated trials, we encode the knowledge of the domain (and the available actions) in the approach and use it to make the choice. For instance, if the goal is to find the colour of a 'box', the colours of only those image regions need to be examined, which have been classified as being a box. Formulating the choice of information processing and sensing actions as

[1]The ability to plan may itself have to be learned but we do not consider that alternative here.

a planning problem presents a different set of challenges, in terms of the issues that need to be addressed: an appropriate state representation, conflict resolution (when multiple actions are feasible/suggested) and planning under uncertainty (when the results of actions are not deterministic). The drawback of this approach is of course that it may require significant human effort to encode all the domain knowledge relevant to the task at hand. As we argue towards the end of this paper (Section 8), for a reasonably complex scenario, this approach may also pose a problem when a newly added information processing or sensing ability interacts with existing abilities. But planning for information processing and sensing actions is still an open problem, and we present our approach, based on continual planning [3], in Section 5. Before that, we briefly describe the cognitive architecture that the planning approach is going to be based on.

3 The System Context: CAS Architecture

In this section, we provide a brief overview of the general architecture of the system that forms the basis of all our work. Based on the analysis of several possible scenarios, we ended up with three essential design requirements for the cognitive architecture underlying the robot system: support for concurrent modular processing, structured management of knowledge, and dynamic control of processing. These are met by the CoSy Architecture Schema (CAS) [11, 13]. As described in [13], in terms of the terminology commonly followed while describing cognitive architectures, scenarios are analyzed to arrive at a set of *requirements*, which in turn lead to *design principles* for architectures, expressed in terms of *architectural schemata*. Schemata define a design space that contains several specific architectural designs, while a *schema* provides the constraints on this design space.

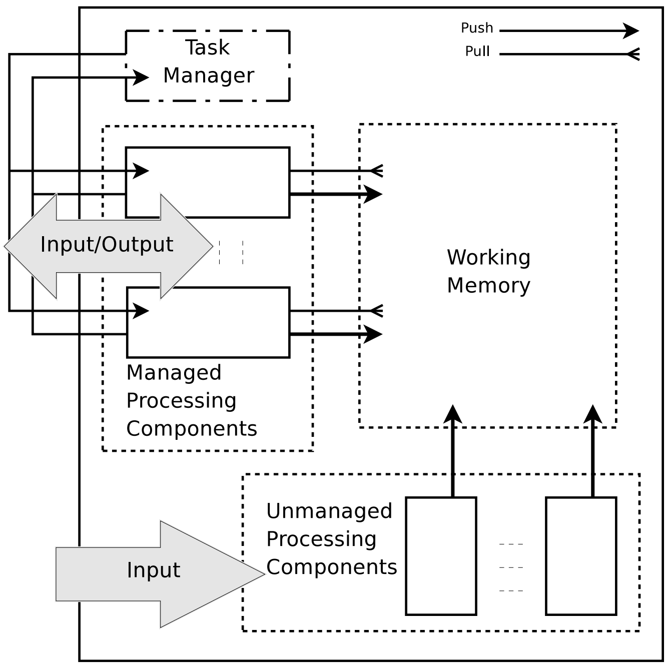
Push
Pull
Task Manager
Input/Output
Working Memory
Managed Processing Components
Unmanaged Processing Components
Input

Figure 2: The CoSy Architecture Schema Subarchitecture design schema.

The CAS provides a collection of loosely coupled subarchitectures (SAs), where each SA consists of a number of processing components (managed and unmanaged) which share information through a working memory (WM), and a task manager. Figure 2 shows a pictorial representation of the CAS SA design schema, with the various components. The unmanaged processing components perform simple data processing and run continuously to push data into the WM, while the managed processing components, which are computationally expensive and hence run only when necessary, monitor the contents of the WM and suggest possible processing tasks using the data in the WM. The task manager decides which of the suggested tasks should be executed, based on the current requirements of the entire system and the computational resources available. The unmanaged components can therefore be considered as the pre-attentive processes while the managed components are the post-attentive processes. *The goal of this work is to have the robot autonomously choose the sequence of post-attentive processes to run, such that the desired task is accomplished efficiently.*

Each SA's WM can be accessed (i.e. read) by the any component of any of the other SAs in the system, but it can be written to only by the processes within the same SA, and by a small number of other privileged SAs. The privileged SAs components can push instructions into any of the other SAs, resulting in top-down goal creation. In the event of multiple goals within a SA, the task manager mediates between them. This mixture of top-down and data-driven processing allows the system to readily handle goals that require coordination within a single SA or across several SAs.

Several other popular cognitive architectures exist, for instance ACT-R [1], SOAR [16] and ICARUS [17], and are used extensively in information processing tasks, but they are not readily applicable to domains involving robotic systems. The features that distinguish the CoSy Architecture Schema from the other cognitive architectures are as follows:

- Most cognitive architectures use a unified representation for all information, while CAS allows specialized representations within each subarchitecture.
- Cognitive architectures typically implement processing units as logical rule-based systems while processing units can be arbitrary processing modules in CAS.
- Most cognitive architectures have mechanisms for learning, while no specific learning mechanism exists at the schema level in CAS, though learning is very much possible in specific instantiations.
- A key distinguishing feature of CAS is that components run in parallel providing concurrent control and access to information. Most other cognitive architectures have their components arranged in a sequential fashion (other than a few exceptions such as [20]), and hence do not allow concurrent access of information. Even in robotic architectures, parallelism is usually only present at the lower levels of processing [5].
- Cognitive architectures typically scale up to additional data sources but do not provide an easy means of adding new components. CAS on the other hand scales out, i.e. new components are easily merged with existing ones and new tasks are readily tackled.

We have defined several specific systems using this schema, for instance [13], where there are separate SAs for each of the main modules such as visual information processing, speech processing, navigation, spatial representation and planning. In addition to a small number of coordinating SAs, there are several components that operate in parallel to process the information and refine the common representations. To facilitate experimentation, the CAS has been instantiated in the form of a software toolkit (CAST) [12] that allows components written in different languages to be combined into CAS instantiations. Functionally different systems can be readily generated without recompilation or changes to components.

With different SAs interpreting data from a range of sensors, each SA may generate a representation of the surrounding environment that is distinct from those produced by the other SAs. For instance the same object may result in different representations in the spatial, visual and communication SAs. In order to have the information from different SAs to be fused, the different representations need to be merged and bound. This *binding* of representations from different SAs is accomplished using a binding subarchitecture.

A binding SA has a WM, which mirrors the contents of all SAs that contain information about concepts that need to be shared across SAs (for instance an 'object'). The binding SA also has a *binder* that maintains the contents of the binding WM by merging the information from different SAs. Based on the task at hand, the individual candidates are filtered and abstracted away by the *binding monitor*. When there are any updates to the contents of the individual WMs, the binder is triggered, causing the binding candidates from the individual WMs to be filtered and bound by the binding SA's binding monitor, resulting in *instance bindings*. The monitor typically attempts to perform intra-modular binding so as to remove domain dependence. A shared language, based on *features* and *relations*, has been created to allow other SAs to communicate with the binding SA. Once an instance binding has been created, it can be used by any component of the architecture to access the candidates bound by the binding. The binding SA's WM provides proxies between all the SAs of the system because the binding candidates may refer back to detailed representations in the original SA. For more details and an implementation example (with experimental results) of binding, please refer to [13].

In our work, we have been looking at a particular application scenario for testing our algorithms and architectural designs: the playmate scenario. The scenario consists of a robot arm mounted on a table-top setting, where the robot's task is to observe/manipulate objects that are arranged on the table. The inputs to the system are through a colour camera mounted on the arm, an external stereo camera that is looking at the scene (both the arm and the table-top are visible), and through a speech/text-based interaction system (for interaction with a human observer). The robot can take voice/text commands from the human observer and either learn representations based on the input, or perform actions in response to (or in order to be able to respond to) queries. There are therefore several SAs operating in parallel, for instance: visual SA, spatial SA, manipulation SA, coordination SA, communication SA, planning SA and binding SA. Figure 3 depicts some of the SAs in our current system. For more details on the specific instantiations, please refer to one of our recent technical reports [11, 12].

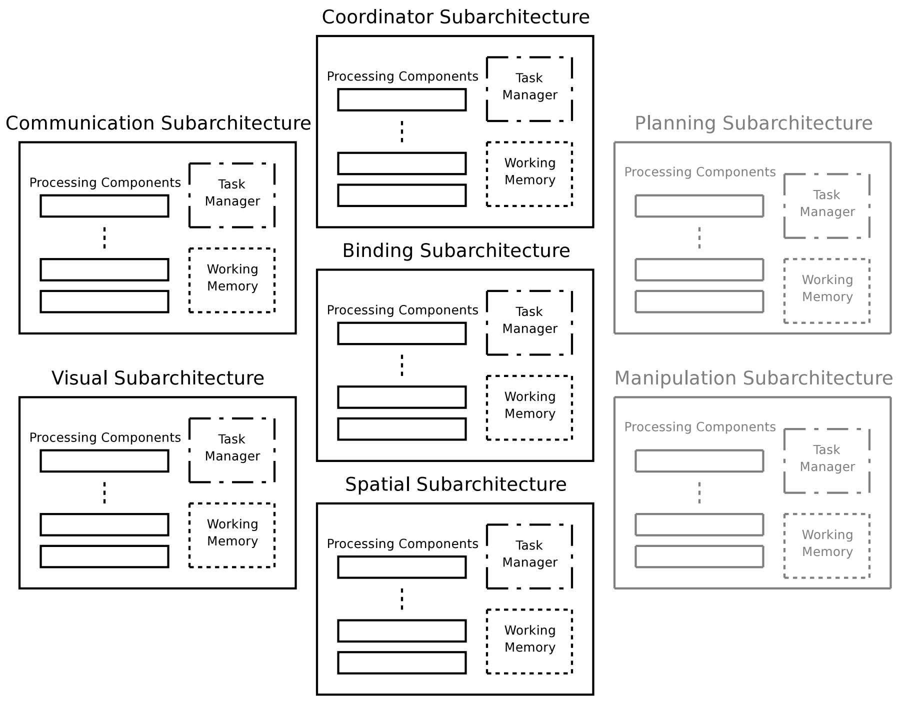

Figure 3: The CoSy Architecture Schema Subarchitecture Instantiations.

The parallelism of component execution is a desirable feature of the architecture. But the extensive parallelism within the system, and the constraints on computational resources motivate the need for an efficient choice of information processing and sensing actions. There are several different means available for planning information processing tasks, depending on the type of architecture used, the manner in which states and actions are represented, and the scheme used for handling contingencies in the plan. Some of the more popular methods are Behaviour-based Planning [5], Dynamic Planning [9] and Continual Planning [3]. We briefly discuss the planning alternatives in the next section.

4 Possible Planning Approaches

One approach to addressing the problem of planning for information processing and sensing actions would be to use a behaviour-based architecture. The *subsumption architecture* [5] is an example of such a system. The motivating principle here is that the representation for information depends on the intended use of the information. For instance, predicting the behaviour of every part of the system has entirely different modeling requirements than those required for planning actions on the system given the sensor and actuator constraints. More specifically, the idea is to decompose complicated behaviours into simple behaviour modules that are in turn organized into layers. A layer implements a particular behaviour goal of the agent, and the abstraction increases as we proceed to the higher levels, i.e. higher-level goals *subsume* those of the lower layers. For instance, the lowest layer could encode the goal "avoid obstacles" or "follow the wall", followed by a higher layer that encodes "explore the environment", while the topmost layer could have the goal "create a map of the environment". The main features can be summarized as follows:

- Behaviour is distributed rather then being centralized, an advantage since it promotes modularity. But decomposition based on behaviour tasks can quickly lead to systems that are domain-dependent, which do not generalize across tasks/domains.
- Behaviours and agents are organized in a bottom-up manner, with the lowest layers encoding simple behaviours and the higher layers encoding complex tasks.
- The response is mostly reflexive, and the representation of sensing and motor actions (and their effects) is built by performing those actions. There is very little cognitive reasoning involved in the perception-action decision making. The representations used in behaviour-based systems are not suitable for robot systems that perform cognitive tasks, especially those that require a common representation across different modalities.
- It is not suitable for planning in dynamic environments, especially in environments with non-trivial complexity. Specifically, if planning requires a search through a space of possible actions, and if the search cannot be done by executing those physical actions (for instance an intelligent wheelchair trying to determine the best location to cross a busy road), then the behaviour-based architecture would not be a valid option.

The simplicity of the action decomposition schemes makes it infeasible to perform one of the fundamental tasks in cognitive systems with different sensory and actuation modalities: the binding of information across different modules that refer to the same 'object'. The guiding principle in CAS is that knowledge representation is essential for achieving cognitive abilities. *The behaviour-based architecture and the associated task decomposition is therefore not a popular alternative for cognitive robot systems.* It can still be part of a larger architecture that runs in different modes at different times, delegating simple control tasks to a behaviour-based subsystem (for instance, wall following behaviour).

In most modern planning methods, especially those that involve cognitive interactions, knowledge/state representation is an essential feature. Actions not only act on the objects in the scenario but also act on the representations themselves. For instance, if a red mug is moved from location A to location B, the 'move' action, in addition to causing the mug to be moved to location B, also changes the robot's knowledge so that it *knows* that the red mug is now in location B, and that location A is now vacant. Actions can therefore change the knowledge representations by adding/removing certain facts about the world. The challenge is to represent knowledge in a suitable manner. We would like states to be *epistemic* in that they refer to a piece of knowledge, or refer to the conditions appropriate for obtaining/modifying knowledge. Specific examples of the state/knowledge representation depend on the scenario, and a few examples are provided in Section 5.

Another popular approach is dynamic planning [9]. The good feature of this planning scheme is that it can cope readily with highly dynamic and unpredictable environments. In addition, unlike classical planning, dynamic planners only plan a few steps in advance, based on the current knowledge of the state – in simple/familiar situations or in cases where the *costs* of the available actions are known, only the next step/action is computed. Such a system automatically requires an appropriate state representation, and a plan essentially involves composing a sequence of states that would lead the agent to the goal. This plan could be represented in different ways, for instance simple condition-action rules, finite state machines (FSMs), and connectionist approaches.

- Condition-action or if-then rules, popularly known as *productions*, state that if the condition holds, the corresponding action is to be performed. The conditions are typically boolean while the actions can involve external (movement, manipulation etc) or internal (state representation) modification. Such rules can also be organized hierarchically, or they may be organized as decision trees, with a particular path being traversed depending on the start state and the desired goal state. Conflict resolution between proposed actions can be solved by different mechanisms, such as preference assignment (SOAR [16]) and relative utility learning (ACT-R [1]).
- Finite State Machines (FSMs) model the transitions between system states as productions. Just one state of the FSM is active at any given instant, and all possible transitions from this state are evaluated to pick the valid transition. Such transitions can result in external actions or modifications of internal state representations. Note that it is possible to represent FSMs using condition-action rules, i.e. FSMs can also be thought of as imposing a layer of abstraction on the production systems.
- Connectionist approaches such as neural networks have basic units whose input links feed in an abstract entity, and output links propagate the entity to other links. The units are connected in a layered structure. Such networks provide smoother behaviour than production rules, allow for adaptation and inhibition (for conflict resolution). But the development of such networks requires a lot of work on the part of the designer. In addition the connectionist approaches, especially ones that adapt over time, can only encode relatively simple behaviour.

For a competent cognitive (robot) agent, for scenarios such as the one described above (Section 2), we need a planning mechanism that allows for human-understandable state representations, modular task decompositions, and smooth conflict resolutions. None of the methods outlined above satisfy all these requirements. In the next section, we describe our approach, based on continual planning, for planning information processing and sensing actions, and discuss how the framework attempts to address these requirements.

5 Continual Planning

Continual planning [3] is based on the understanding that in dynamic, partially observable environments, it is typically not possible to model all possible contingencies and their probabilities, especially if there are other agents in the environment. In such situations, "classical" AI planning schemes are not applicable since they unfortunately require that the planner has complete knowledge of states, and that all changes in the state of the world are as a result of the actions executed by the planner. Planning in such cases may still be posed as a contingent, conformant or probabilistic planning task, which though able to deal with incomplete information and uncertainty, still requires that the plans be guaranteed to succeed under all circumstances. Continual Planning (CP) is a solution to such problems – it interleaves planning, plan execution and plan monitoring, and does not make the limiting assumptions inherent in the other methods mentioned above. Instead of considering all contingencies in advance, an agent in CP executes part of the plan in an attempt to gather more information, so as to reduce the uncertainty for the rest of the plan. In multiagent domains, CP is extended to provide Collaborative Continual Planning (CCP). Since we are primarily focusing on the activities of a single agent (robot) we do not provide further details on the multiagent framework, which is more significant for other challenges such as speech processing and communication. The underlying principle is that the agent starts acting as soon as it has a feasible plan, and planning the conditional subplan is delayed until the agent has enough information to plan the details of the subplan. The main features of CP are:

- Active knowledge gathering: agents' beliefs and sensing capabilities are modeled as part of the formal planning domain.

- Agents can update their beliefs (their own as well as those of other agents) in three possible ways: individual sensing, communication, and joint sensing. These three schemes also allow for making inferences and possibly forming new theories.
- Know-If (KIF) variables can be appended to the state variable description in standard planning frameworks: KIF variables describe whether the state of a variable is known or not.
- A Multiagent Planning Language (MAPL) is introduced, with the planning for actions being performed, possibly concurrently, by several agents. The MAPL plans are able to interleave action, sensing and communication, and we use this framework to plan the actions of a single agent. The syntax and semantics of MAPL are very similar to PDDL [19]. Such a planning system is very much applicable to robots that have different parts/subsystems that can do different things in parallel, either independently or collaboratively.
- In order to be able to represent ignorance about facts in addition to propositional truth, multi-valued state variables (MVSVs) are used instead of simple propositions. One alternative, though we do not consider it here, would be to have a meta-language, in which it would be possible to state what is and is not known.
- Actions are modeled as logical rules, they can be executed when the preconditions are satisfied, leading to the specified effects. Effects include actual physical actions and modifications to the current state representation.
- In CP, the delayed planning of a conditional subplan, until enough information is available to resolve contingency, is accomplished through *assertions*. If the preconditions of an assertion are satisfied they are expanded and a new planning phase is triggered. The available information is used to replace the assertion with concrete actions.
- In addition to actions, sensor models are also specified as actions, with a set of preconditions that specify the circumstances under which the agent can perceive the value of a specific (or set of) state variable(s).

Though we do not give the complete formal description of continual planning, as mentioned in [3], we do provide the following definitions. Though all definitions refer to multiagent planning (because the same formulation is used for work on language/communication), we only have one agent in our experiments.

Definition 1 *A* ***multiagent planning domain*** *is a tuple $\mathcal{D} = (\mathcal{A}, \mathcal{V}, \mathcal{C}, \mathcal{E})$ consisting of agents $\mathcal{A}$, multi-valued state variables $\mathcal{V}$, constants $\mathcal{C}$, and events $\mathcal{E}$. For each state variable $v \in \mathcal{V}$, there is a finite domain $dom_v \subset \mathcal{C}$.*
Events $e \in \mathcal{E}$ have the form $e = (a, pre, eff)$ where $a \in \mathcal{A}$ is the controlling agent, and pre and eff are partial variable assignments over $\mathcal{V}_{\mathcal{A}}$, the set of state variables associated with with agent, called the ***preconditions*** *and* ***effects*** *of e.*

The first definition defines a *multiagent planning domain* in terms of the agents, state variables, the constants corresponding to the values that the state variables can take, and the events i.e. feasible conditional plan-steps. These events are different from other events that occur in the environment, without an agent, and which are not represented in terms of preconditions and effects, though they may influence the preconditions. Preconditions and effects are modeled in the form of predicates that represent the state/value of the variables – the system is able to deal with partial variable assignments.

Definition 2 *A* ***multiagent planning task*** *for a multiagent planning domain $\mathcal{D}$ is a pair $\mathcal{T} = (\mathcal{I}, \mathcal{G})$ consisting of an initial state I and a goal state G both possibly incomplete and defined over $\mathcal{V}_{\mathcal{A}}$.*
An asynchronous plan $\mathcal{P}$ is a ***solution*** *to a planning task $\mathcal{T} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$ if $\mathcal{G} \subseteq res(\mathcal{I}, \mathcal{P})$*

Definition 2 defines the planning task in terms of the initial and goal states, though the set of available actions and agents is also an important part of the task since altering that set while keeping the initial and goal states fixed would define an entirely different planning task. A *plan* is a solution to this task if the goal state ($\mathcal{G}$) is a subset of the state reached as a result of applying the plan in the initial state ($\mathcal{I}$), i.e. ($\mathcal{G} \subseteq res(\mathcal{I}, \mathcal{P})$).

Definition 3 *An* ***assertion*** *is an action* a *with a distinguished, non-empty set of preconditions $repl(a) \subseteq pre(a)$, called the replanning conditions. Preconditions $p \notin repl(a)$ are called the ordinary preconditions of* a. *If $repl(a) = \phi$ then* a *is an ordinary action. The set of assertions among actions A is denoted by $A^r \subseteq A$.*

Definition 3 states that the assertion is just a special case of an action operator – the replanning conditions ($repl(a)$) are similar to the preconditions in Definition 1, i.e. they are predicates on the state variables. But instead of being executed when the preconditions are satisfied, it is *expanded*, i.e. it results in re-planning based on the current state of the world. If there are no replanning conditions ($repl(a) = \phi$), it is a normal action.

Definition 4 *An asynchronous plan $\mathcal{P}$ is an* ***assertional solution*** *for a planning task $\mathcal{T} = (\mathcal{I}, \mathcal{G})$ if (1) $\mathcal{P}$ is a solution to $\mathcal{T}$ as in Definition 2, and (2) $repl(a) \nsubseteq \mathcal{I}$ for all assertions $a \in P$.*

The final definition refines Definition 2 to state that in order for a plan to be an assertional solution for a planning task, the preconditions of all the assertions in the plan should not be satisfied in the initial state (condition (2) in the definition). We provide one example each of assertions, actions and sensor models – the syntax is almost identical to PDDL. Though not applicable to our application domain, these examples are used in other domains within the CoSy framework. For simplicity these examples all refer to the motion of a agent in a gridworld – each cell is either empty or occupied. The examples corresponding to our application scenario of sensory processing are provided in the next section (Section 6).

An action that allows an agent to move from its current position in a grid to another empty grid cell:

```
(:action move
 :agent (?a - agent)
 :parameters (?c - gridcell)
 :variables (?ca - gridcell)
 :precondition (and
      (occupant ?ca : ?a)
      (occupant ?c : empty) )
 :effect (and
      (occupant ?c : ?a)
      (occupant ?ca : empty) )
)
```

Variable names have the prefix '?' attached to them, and each variable name is followed by the variable type. For instance *?c* is a variable of type *gridcell.* The value assignments happen through a colon(':') – for instance *(occupant ?c : empty)* assigns the value 'empty' to gridcell 'c'. In the absence of a colon, the entire predicate is evaluated to the boolean 'true'. Both the preconditions and effects of the action consist of one or more predicates that evaluate to true or false. In addition, all actions in MAPL have an 'agent' performing them (in this case a particular robot). Next, we define a sensor model that states that the occupant of every grid cell in sensing distance of the agent, will be perceived by the agent.

```
(:sensor sense-gridcell
 :agent (?a - agent)
 :parameters (?c - gridcell)
 :variables (?ca - gridcell)
 :precondition (and
        (occupant ?ca : ?a)
        (in-sensing-distance ?ca ?c) )
 :sense (occupant ?c)
 )
```

The effect of the sensor model/action is slightly different from that of an action. Instead of changing the state of the world, it modifies the agent's belief in terms of its sensing capabilities. Finally, an example of a move action with an assertion: it just states that an agent will be able to find a plan to reach a grid cell once it knows whether the grid cell is empty – note the *knowIf* variable used in the assertion.

```
(:action move_assert
 :agent (?_pa - planning_agent)
 :parameters (?a - agent ?c - gridcell)
 :variables (?ca - gridcell)
 :precondition (and
        (occupant ?ca : ?a) )
 :replan
        (KIF ?_pa (occupant ?c) )
 :effect (and
        (occupant ?c : ?a)
        (occupant ?ca : empty) )
 )
```

The KIF predicate distinguishes this action from the move action defined above. The KIF predicate is used by the agents to question its belief state. If the agents *knows* that the particular cell it is attempting to move to is already occupied (in this case by another agent), the current plan, which is using this move action as one of its steps, cannot lead to the goal, and hence the agent needs to replan. The replanning is the effect that the assertion is used for.

The continual planning algorithm can be briefly described as follows: first a check is made to determine whether the current state (or the satisfaction of a replanning condition, i.e. assertion) requires a new plan. If yes, in order to exploit the power of classical planners, all expandable assertions are removed and a plan is generated using a classical planner (Axioms-FF [14]). Irrespective of whether a new plan is created, after executing one step of the plan, the expected perceptions are compared with the observed perceptions (sensor models are used in this step) to detect failure in action execution if any. Then the old knowledge is fused with new perceptions and expected changes. The planning is continued until either the goal is reached or until no valid plan can be generated.

6 Operators and Examples

In this section, we provide an example of planning in the visual space, both for gathering information and for answering user-defined queries. In order to do so, we first describe some sample queries that we would like the robot to be able to answer in the playmate scenario, which we group into categories. We then list a set of operators (actions) which the robot can use, and define each of the operators in terms of (pre)conditions under which they can be applied, and the effects they produce when applied. After that we pose some sample goals for the planning system and describe the steps the planning algorithm proceeds through in order to achieve those goals (using these operators).

6.1 Sample Queries

Before we describe the operators, we first present the scenario and possible queries that we would like the robot to be able to answer in this scenario. The operating environment is the same table-top playmate scenario described in Section 3. A robot arm is mounted on a table and is set the task of observing/manipulating objects (non-transparent solid objects such as boxes, mugs etc) that are arranged on the table. The inputs to the system are through a colour camera at the tip of the arm, an external stereo camera, and through a speech/text-based interaction system (for interaction with a human observer). The robot can take voice/text commands from the human observer and either learn representations based on the input, or perform actions in response to queries.

The types of queries/commands we would like the system to be able to handle may be grouped as follows. Note that some of the later categories may in turn require the robot to issue queries/commands that correspond to the earlier categories.

- *Location:* Questions on the locations of a particular object, refered to by a label based on the object or its property. Examples include: *where is the red beaker?*, *where is the mug?*
- *Type/Property-analysis:* Questions/commands that are based on a particular property of an object, or which compare two objects based on a particular property. Examples include: *What is the blue thing on the table?*, *What is the colour of the mug?*, *Is the blue beaker bigger/smaller than the red mug?*, *Is there an orange design on the white jug?*
- *Action-related:* Questions/commands that aim to find out if a particular action can be performed on a given object. This includes questions of the form: *Can the red beaker be grasped?*, *Can the blue jug be moved to location (x,y)?*
- *Spatial relationship:* Questions/commands that aim to determine/alter the relationships between two or more objects, both in terms of their location and the values of particular properties. Examples are: *Is the red mug above/below the blue block?*, *Is the green jug behind/in front of the orange ball?*, *Move all red blocks to the right of the black mug.*
- *Scene analysis:* At the highest level, we would like to be able to pose queries such as: *What/how many objects are in view?*, *Has the collection of objects changed since the last view?*, *Describe the objects on the table.*

In the playmate scenario with a colour camera and coloured objects in the scene, we next consider the list the actions (i.e. operators) that the agent (i.e. robot) can use while planning its information processing for sensing actions. Note that this list is not exhaustive: it is a small set of operators made available as part of the current version of the CoSy code-base.

- Detecting saliencies/salient points in the image.
- Colour segmenting the image.
- Region property detector – for instance determining the size, colour, shape of an image region.
- Detecting regions-of-interest.
- SIFT [18] operator for object recognition.

- Camera zoom operator.
- Camera motion to a particular location – for instance to get a better view of a scene.

6.2 Sample Action Operators

Once the possible actions have been determined, we then need to write MAPL action operators for them, i.e. we need to clearly specify the (pre)conditions under which they can be applied, and the effects produced on applying them. For example, one common condition for almost all operators is that they should not be applied on a scene (or image) if they have already been applied, unless the scene (image) has changed. Another common feature in almost all operators is that if the result of applying that operator is already true, it triggers a re-planning operation.

1. The saliency detector can be applied if there has been a change in the scene since the last time the operator was applied, and if the operator has not been applied already. The result of application of the operator is that the salient points have been detected in the image. Note that we distinguish between 'image' and 'scene' because several images are captured continuously of the same scene. As expected, if the result of the operation is already true (in this case the salient points in the image have already been detected), it is worth re-planning.

```
(:action saliencyDetector
 :agent (?a - robot)
 :parameters (?img - image)
 :precondition (and
        (changeDetected true)
        (applied-saliencyDetector false) )
 :replan
        (exists-salientPoints ?img true)
 :effect (and
        (applied-saliencyDetector true)
        (exists-salientPoints ?img true) )
 )
```

2. The colour segmentation operator, which is also defined in a similar fashion.

```
(:action colourSegmentor
 :agent (?a - robot)
 :parameters (?img - image)
 :precondition (and
        (changeDetected true)
        (applied-colourSegmentor false) )
 :replan
        (exists-segmentedRegions ?img true)
 :effect (and
        (applied-colourSegmentor true)
        (exists-segmentedRegions ?img true) )
 )
```

3. In both the above-mentioned actions, we use the *changeDetected* flag. The change detection routine is one of the very few routines that runs continuously on the input images (i.e. on each image). As soon as a change is detected in the image, based on background subtraction and image histograms, the corresponding flags are set/reset appropriately. For example, for the above two operators, it would involve reset the two flags:

   $(applied - saliencyDetector false)$

   $(applied - colourSegmentor false)$

4. A SIFT operator for detecting objects from visual regions obtained from colour-segmenting input images. Note that this is a domain-specific representation in that the SIFT operator can be applied on the entire image as well; we use colour regions because the current vision implementation in the CoSy system uses colour as one of the main features. This operator works on visual regions and assumes that a SIFT representation for this object type has been trained in advance.

   ```
   (:action siftDetector
    :agent (?a - robot)
    :parameters (?vr - visRegion ?objT - objectType)
    :precondition (and
          (trained-siftDetector ?objT true)
          (not (applied-siftDetector ?vr ?objT) ) )
    :replan
          (containsObject ?vr ?objT)
    :effect (and
          (applied-siftDetector ?vr ?objT)
          (containsObject ?vr ?objT) )
    )
   ```

5. A shape detector which detects shapes from visual regions. As with the SIFT operator, the operator works on visual regions, and requires that a representation for the shape be trained in advance.

   ```
   (:action shapeDetector
    :agent (?a - robot)
    :parameters (?vr - visRegion ?shapeP - shapeProp )
    :precondition (and
          (trained-shapeDetector ?shapeP true)
          (not (applied-shapeDetector ?vr ?shapeP) ) )
    :replan
          (containsShape ?vr ?shapeP)
    :effect (and
          (applied-shapeDetector ?vr ?shapeP)
          (containsShape ?vr ?shapeP) )
    )
   ```

6. Operators for detecting other properties of regions are also defined similarly, for instance operators for colour, region size etc.

7. Zoom operator. It can be applied to any image region that produces an ambiguous response to a recognition process. The preconditions and effects are simple – in actual implementation the preconditions

would also indicate the dependency on the object-recognition and/or shape-recognition routines.

```
(:action cameraZoom
 :agent (?a - robot)
 :parameters (?vr - visRegion)
 :precondition (and
       (applied-cameraZoom ?vr false) )
 :effect (and
       (applied-cameraZoom ?vr true) )
)
```

Next we show an example of applying some of these operators on a task.

6.3 Sample Planning Output

This section presents the results planning information processing and sensing actions in the visual space. The tasks were run on a simulator that simulates the planning and the effects of running an operator. For ease of understanding we assume that visual regions, obtained as a result of applying the colour segmentor, are already available. We then define the planning task, starting with the domain:

```
(:types
      ;; Types for general variables...
      agent boolean - object
      human robot planning_agent - agent
      ;; Define visual regions and their properties...
      visRegion colourProp objectType shapeProp - object
)
```

The constants of the domain are defined as follows:

```
;;; Constants involved in the domain...
(:constants
      ;;; Possible colour labels...
      red blue green - colourProp
      ;;; Possible object types detectable by SIFT...
      beaker mug jug - objectType
      ;;; Possible object types detectable by shape detector...
      circle square rect - shapeProp
      ;;; Boolean values that variables can take...
      true false - boolean
      ;;; Dummy agents to use for the operators...
      mohan - robot
)
```

We then specify the initial conditions by defining a few ($N = 3$) visual regions (expected output of the vision module after applying the colour segmentor):

$(newDeclaration \text{ “}vr0''\text{, “}visRegion'')$

$(newDeclaration \text{ “}vr1''\text{, “}visRegion'')$

$(newDeclaration \text{ “}vr2''\text{, “}visRegion'')$

Let us assume that the known object types are *beaker, mug, jug*, and the known shapes are *circle, square, rectangle* (though the likelihood of finding a circular beaker in the real world is low!). Let us also assume that the goal is to find a circular mug, and let us assume that visual region *vr1* is the one representing the circular mug. The goal state is defined as:

$goal - state \equiv (and(exists(?vr - visRegion)(and(containsObject?vrmug)(containsShape?vrcircle))))$

The predicates used in the operators also need to be defined:

```
;;; Definition of the predicates involved in the process...
(:predicates
        ;;; Colour property value of a particular visual region...
        ( colourProp-val ?vr - visRegion ?c - colourProp )
        ;;; Object property value, i.e. beaker, mug or jug...
        ( objectType-val ?vr - visRegion ?objT - objectType )
        ;;; Shape property value, i.e. circle, square or diamond...
        ( shapeProp-val ?vr - visRegion ?shapeP - shapeProp )
        ;;; Visual region's properties to reflect the Sift's application...
        ( applied-siftDetector ?vr - visRegion ?objT - objectType )
        ;;; Visual region's properties to reflect shape detector's application...
        ( applied-shapeDetector ?vr - visRegion ?shapeP - shapeProp )
        ;;; Predicates related to the effect of the operators...
        ( containsObject ?vr - visRegion ?objT - objectType )
        ( containsShape ?vr - visRegion ?shapeP - shapeProp )
)
```

Next, we define a few facts of the world, which form the initial state of the agent's environment:

(trained-siftDetector Beaker true),(trained-siftDetector Mug true),(trained-siftDetector Jug true)

(trained-shapeDetector Circle true),(trained-shapeDetector Square true),(trained-shapeDetector Rect true)

The information is translated into the following internal representation of the current state:

Current state: [trained-siftDetector Beaker true, trained-siftDetector Mug true, trained-siftDetector Jug true, trained-shapeDetector Circle true, trained-shapeDetector Square true, trained-shapeDetector Rect true]

The planning system then works backwards from the goal state and tries to achieve each of the two parts of the goal state, i.e. *(containsObject ?vr mug)* and *(containsShape ?vr circle).* The steps in the planning cycle are as follows. Note that the changes in state at each step are indicated in **bold-face**, and are shown *emphasized* in subsequent steps.

1. The first valid step in the plan is to have a 'robot' (named after one of the authors) perform a valid action that leads towards the goal:
   taking step: siftDetector mohan vr0 mug
2. Now when the state is observed (using the simulation that mimics the actual visual processing) and the expected and observed information is fused, we end up with the state:
   Current state: [**applied-siftDetector vr0 mug**, trained-siftDetector Beaker true, trained-siftDetector Mug true, trained-siftDetector Jug true, trained-shapeDetector Circle true, trained-shapeDetector Square true, trained-shapeDetector Rect true]
3. The next valid step in the plan is:
   taking step: siftDetector mohan vr1 mug

4. At this step the visual feedback (simulated) succeeds in finding the 'mug' leading to the state:
   Current state: [*applied-siftDetector vr0 mug*, **applied-siftDetector vr1 mug**, trained-siftDetector Beaker true, trained-siftDetector Mug true, trained-siftDetector Jug true, trained-shapeDetector Circle true, trained-shapeDetector Square true, trained-shapeDetector Rect true, **containsObject vr1 mug**]

5. Then, the plan moves to the next step where the target is to match the second part of the goal, i.e. *(containsShape ?vr circle)*, and the valid step of the plan is:
   taking step: shapeDetector mohan vr1 circle

6. Note that since partial goal has been achieved, the plan is suitably focused on the appropriate visual region. Now, the state of the system is:
   Current state: [*applied-siftDetector vr0 mug*, **applied-siftDetector vr1 mug**, **applied-shapeDetector vr1 circle**, trained-siftDetector Beaker true, trained-siftDetector Mug true, trained-siftDetector Jug true, trained-shapeDetector Circle true, trained-shapeDetector Square true, trained-shapeDetector Rect true, **containsObject vr1 mug**, **containsShape vr1 circle**]

7. At this point, both subgoals and hence the overall goal is satisfied and the plan terminates.

That provides one complete cycle of the plan. As an additional test, if we were to insert the fact obtained in step 4, i.e. *(containsObject vr1 mug)* in the initial state, the system directly plans for finding the circular shape (and finishes in one step!).

7 Related Work

In this section we look at a few representative samples of work done on planning asequence of operators for visual processing. There is a significant body of work in the image processing community on planning sequences of visual processing operations [6, 10, 22, 8, 7, 26]. This technology is also known as software reuse, or program supervision. Often the goal is to enable scientists who are not image processing specialists to use libraries of image processing algorithms. The high level goal is specified by the user. This is handed to a classical AI planner which constructs a pipeline of image processing operations. The planners use deterministic models of the effects of information processing: handling the pre-conditions and the effects of the operators using propositions that are required to be true a priori, or are made true by the application of the operator. Uncertainty is handled by evaluating the output images using hand-crafted evaluation rules [8, 22, 26]. If the results are unsatisfactory execution monitoring detects this and the plan is repaired. This either involves re-planning the type and sequence of operators or modification of the parameters used in the operators [22, 26].

One problem which significant efforts have been made to address is the development of the representation used for the action-models of the image processing operators. One approach is to use a modeling language similar to STRIPS operators, where the the preconditions and effects of operators [21] are specified. This approach differs from our proposed work in that the rules mainly decide on the legal sequences of image processing operators, and whether the application of an operator should establish a certain proposition about the image. There is typically little or no attempt to predict any detailed (non-determined) characteristics of the output image, primarily because it is extremely difficult to estimate the likely output of an operator in any more detail [8]. Another challenge is the need to create a domain specific set of rules for evaluating the output image from each operator. The formation of these rules requires expert knowledge. Though our current approach does require domain knowledge during its formulation, our aim is to model the high level probabilistic effects of operators on the beliefs of the agent, and use the resulting probability distributions as the basis for a more generic evaluation method.

Planning and program supervision have been used for the automation of image processing for astronomy [6, 22], and for bio-medical applications [8]. There has also been some work on perception for autonomous object detection and avoidance in vehicles [25, 23] and on interpretation of 3D objects' structure [15], but extending program supervision beyond image processing to general computer vision has proven difficult.

8 Conclusions

We have presented a framework for planning information processing and sensing actions for a cognitive robot system. We have built our work on the existing continual planning framework, and modeled information processing and sensing actions as action operators that can be provided as input to this planning scheme. We have shown that the combined system is able to plan the execution of its operators such that the goal state is achieved in an efficient manner. Though we have only described a simple example, the same scheme can be extended to address several of the queries we would like the robot to be able to answer (Section 6.1).

The current planning framework enables us to build a clear model of what would be possible to accomplish in the absence of noise, where the inputs (and outputs) are not probabilistic. But that fact that it does not allow a probabilistic formulation of the actions and their effects is a significant drawback of the system. Since the events are discrete, it is difficult to base the preconditions of an action on the probabilistic occurrence of events. The results of an action too suffer from the same problem. But, in realistic settings, actions and their outcomes are far from deterministic, and it is not always feasible to discretize the values of continuous-valued state variables. The problem is more pronounced while dealing with physical systems such as robots that operate with noisy sensors and end-effectors. One future direction of research would be to modify the underlying planning system so that it is based on a probabilistic formulation; we could for instance use decision-theoretic planning [2, 4]. The decomposition of the available actions in terms of its preconditions and effects is still valid – we would only have to change the manner in which these conditions are modeled. Including such a probabilistic framework would make the system more robust and possibly allow us to answer complex queries, for instance those that involve a highly cluttered scene.

Another challenge in the current formulation, which is true of a large number of planning approaches, is that all the operators/actions have to be defined in advance, and this involves significant human effort in decomposing the actions into meaningful preconditions and effects. In addition, if the new operators/capabilities added to the system can influence the performance of existing operators, then all the inter-dependencies may have to be manually resolved.

As mentioned in Section 2, the other alternative for choosing the best information processing and/or sensing action (or sequence of actions) for achieving a goal, is to use learning. But as mentioned there, for a reasonably complex scenario, it does not currently seem practical to attempt to learn the entire mapping from the desired goals to the appropriate actions, without encoding a significant amount of domain knowledge into the system. A reasonable approach would be introduce learning within the planning framework. We could start with a small set of (essential) of domain knowledge encoded in the system, in the form of suitable action operators with known preconditions and effects. But we could then let the robot learn from experiences and modify/fine-tune these operators. Having a probabilistic framework would be a advantage here because we could then encode confidences on the events and their effects, and modify the confidences with experience. The robot/agent can use its experience (over trials) to modify these operators and/or add to the existing operators to produce rules that adapt as the operating environment changes. It may also be possible for the agent to selectively 'ask' the human for feedback, when it is unable to resolve conflicts (some work is being done in the language-based communication module in this direction). Ultimately we would like to provide the robot system with cognitive competences that enable it to indulge in meaningful 'conversations' with a human observer, readily learning from inputs and responding to human queries and commands.

Acknowledgements

This work was supported by the EU FP6 IST Cognitive Systems Integrated Project (CoSy) FP6-004250-IP, and the Leverhulme Trust Research Fellowship Award Leverhulme RF/2006/0235.

References

[1] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin. An Integrated Theory of the Mind. *Psychological Review*, 111(4):1036–1060, 2004.

[2] C. Boutillier, T.L. Dean, and S.Hanks. Decision theoretic planning: structural assumptions and computational leverage. *Journal of AI Research*, 11:1–94, 1999. sample article on decision theoretic planning.

[3] M. Brenner and B. Nebel. Continual Planning and Acting in Dynamic Multiagent Environments. In *The International Symposium of Practical Cognitive Agents and Robots*, 2006.

[4] J. Bresina, R. Dearden, N. Meuleau, S. Ramkrishnan, D. Smith, and R. Washington. Planning under continuous time and resource uncertainty: A challenge for ai. In *Proceedings of 18th Conference in Uncertainty in Artificial Intelligence*, pages 77–84. Morgan Kaufmann, 2002.

[5] Rodney A. Brooks. A Robust Layered Control System for a Mobile Robot. *Robotics and Automation*, 2:14–23, 1986.

[6] S. Chien, F.Fisher, and T. Estlin. Automated software module reconfiguration through the use of artificial intelligence planning techniques. *IEE Proc. Software*, 147(5):186–192, 2000.

[7] S. Chien and H. Mortensen. Automating image processing for scientific data analysis of a large image database. *IEEE Trans. on Pattern Analysis*, 18(8):854–859, 1997.

[8] R. Clouard, A. Elmoataz, C. Porquet, and M. Revenu. Borg: A knowledge-based system for automatic generation of image processing programs. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 21(2):128–144, 1999.

[9] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice.* Morgan Kaufmann, San Francisco, CA 94111, 2004. Very good (and standard) book on planning, both from the general AI perspective and wrt Robotics...

[10] L. Gong. *Composition of Image Analysis Processes Through Object-Centred Hierarchical Planning.* PhD thesis, Rutgers University, 1992.

[11] N. Hawes, J. Wyatt, and A. Sloman. An Architecture Schema for Embodied Cognitive Systems, csr 06-12. Technical report, School of Computer Science, University of Birmingham, December 2006.

[12] N. Hawes, M. Zillich, and J. Wyatt. BALT and CAST: Middleware for Cognitive Robotics, CSR 07-1. Technical report, School of Computer Science, University of Birmingham, 2006.

[13] Nick Hawes, Aaron Sloman, Jeremy Wyatt, Michael Zillich, Henrik Jacobsson, Geert-Jan M. Kruiff, Michael Brenner, Gregor Berginc, and Danijel Skocaj. Towards an Integrated Robot with Multiple Cognitive Functions. In *The Twenty-second National Conference on Artificial Intelligence (AAAI)*, 2007.

[14] Jorg Hoffmann and Bernhard Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[15] X. Jiang and H. Bunke. Vision planner for an intelligent multisensory vision system. Technical report, University of Bern, Bern, Switzerland, 1994.

[16] J. E. Laird, A. Newell, and P.S. Rosenbloom. SOAR: An Architecture for General Intelligence. *Artificial Intelligence*, 33(3):1–64, 1987.

[17] Patrick Langley and D. Choi. An Unified Cognitive Architecture for Physical Agents. In *The Twenty-first National Conference on Artificial Intelligence (AAAI)*, 2006.

[18] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110, 2004.

[19] D. McDermott. PDDL: The Planning Domain Definition Language, Technical Report TR-98-003/DCS TR-1165. Technical report, Yale Center for Computational Vision and Control, 1998.

[20] M. L. Minsky. *The Society of Mind.* William Heinemann Ltd., London, 1987. Marvin Minsky's famous article – interesting read...

[21] S. Moisan. Program supervision: Yakl and pegase+ reference and user manual. Rapport de Recherche 5066, INRIA, Sophia Antipolis, France, December 2003.

[22] S. Moisan, R. Vincent, and M. Thonnat. Program supervision: from knowledge modelling to dedicated engines. Rapport de Recherche 3324, INRIA, Sophia Antipolis, France, December 1997 1997.

[23] C. Shekhar, S. Moisan, and M. Thonnat. Use of a real-time perception program supervisor in a driving scenario. In *Intelligent Vehicle Symposium '94*, Paris, France, Oct 1994.

[24] R. L. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, USA, 1998. Sutton's THE standard book on RL...

[25] M. Thonnat, V. Clement, and J. van den Elst. Supervision of perception tasks for autonomous systems: the ocapi approach. Rapport de Recherche 2000, INRIA, Sophia Antipolis, France, June 1993.

[26] M. Thonnat and S.Moisan. What can program supervision do for program reuse? *IEE Proc. Software*, 147(5):179–185, 2000.