Combining Cognitive Vision, Knowledge-Level Planning with Sensing, and Execution Monitoring for Effective Robot Control

Ronald P. A. Petrick
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
rpetrick@inf.ed.ac.uk

Dirk Kraft **Norbert Krüger**
The Maersk Mc-Kinney Moller Institute
University of Southern Denmark
DK-5230 Odense M, Denmark
{kraft,norbert}@mmmi.sdu.dk

Mark Steedman
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
steedman@inf.ed.ac.uk

Abstract

We describe an approach to robot control in real-world environments that integrates a cognitive vision system with a knowledge-level planner and plan execution monitor. Our approach makes use of a formalism called an Object-Action Complex (OAC) to overcome some of the representational differences that arise between the low-level control mechanisms and high-level reasoning components of the system. We are particularly interested in using OACs as a formalism that enables us to induce certain aspects of the representation, suitable for planning, through the robot's interaction with the world. Although this work is at a preliminary stage, we have implemented our ideas in a framework that supports object discovery, planning with sensing, action execution, and failure recovery, with the long term goal of designing a system that can be transferred to other robot platforms and planners.

Introduction and Motivation

A robot operating in a real-world domain must typically rely on a range of mechanisms that combine both reactive and planned behaviour, and operate at different levels of representational abstraction. Building a system that can effectively perform these tasks requires overcoming a number of theoretical and practical challenges that arise from integrating such diverse components within a single framework.

One of the crucial aspects of the integration task is representation: the requirements of robot controllers differ from those of traditional planning systems, and neither representation is usually sufficient to accommodate the needs of an integrated system. For instance, robot systems often use real-valued representations to model features like 3D spatial coordinates and joint angles, allowing robot behaviours to be specified as continuous transforms of vectors over time (Murray, Li, and Sastry 1994). On the other hand, planning systems tend to use representations based on discrete, symbolic models of objects, properties, and actions, described in languages like STRIPS (Fikes and Nilsson 1971) or PDDL (McDermott 1998). Overcoming these differences is essential for building a system that can act in the real world.

In this paper we describe an approach that combines a *cognitive vision* system with a *knowledge-level planner* and *plan execution monitor*, on a robot platform that can manipulate objects in a restricted, but uncertain, environment. Our system uses a multi-level architecture that mixes a low-level robot/vision controller for object manipulation and scene interpretation, with high-level components for reasoning, planning, and action failure recovery. To overcome the modelling differences between the different system components, we use a representational unit called an *Object-Action Complex (OAC)* (Geib et al. 2006; Krüger et al. 2009), which arises naturally from the robot's interaction with the world. OACs provide an object/situation-oriented notion of affordance in a universal formalism for describing state change.

Although the idea of combining a robot/vision system with an automated planner is not new, the particular components we use each bring their own strengths to this work. For instance, the cognitive vision system (Krüger, Lappe, and Wörgötter 2004; Pugeault 2008) provides a powerful object discovery mechanism that lets us induce certain aspects of the representation, suitable for planning, from the robot's basic "reflex" actions. The high-level planner, PKS (Petrick and Bacchus 2002; 2004), is effective at constructing plans under conditions of incomplete information, with both ordinary physical actions and *sensing* actions. Moreover, OACs occur at all levels of the system and, we believe, provide a novel solution to some of the integration problems that arise in our architecture.

This paper reports on work currently in progress, centred around OACs and their role in object discovery, planning with sensing, action execution, and failure recovery in uncertain domains. This work also forms part of a larger project investigating perception, action, and cognition, and combines multiple robot platforms with symbolic representations and reasoning mechanisms. We have therefore approached this work with a great deal of generality, in order to facilitate the transfer of our ideas to robot platforms and planners with capabilities other than those we describe here.

Hardware Setup and Testing Domain

The hardware setup used in this work (see Figure 1) consists of a six-degree-of-freedom industrial robot arm (Stäubli RX60) with a force/torque (FT) sensor (Schunk FTACL 50-80) and a two-finger-parallel gripper (Schunk PG 70) attached. The FT sensor is mounted between the robot arm and gripper and is used to detect collisions which might occur due to limited knowledge about the objects in the world. In addition, a calibrated stereo camera system is mounted in a fixed position. The AVT Pike cameras have a resolution

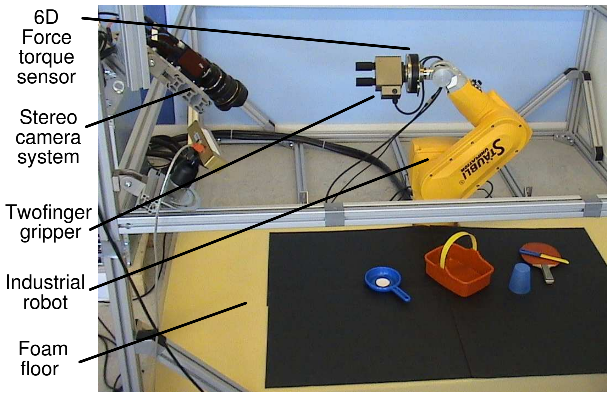


Figure 1: Hardware setup.

of up to 2048x2048 pixels and can produce high-resolution images for particular regions of interest.

To test our approach, we use a Blocksworld-like object manipulation scenario. This domain consists of a table with a number of objects on it and a "shelf" (a special region of the table). The robot can view the objects in the world but, initially, does not have any knowledge about those objects. Instead, world knowledge must be provided by the vision system, the robot's sensors, and the primitive actions built into the robot controller. The robot is given the task of clearing the objects from the table by placing them onto the shelf. The shelf has limited space so the objects must be stacked in order to successfully complete the task. For simplicity, each object has a radius which provides an estimate of its size. An object A can be stacked into an object B provided the radius of A is less than that of B, and B is "open." Unlike standard Blocksworld, the robot does not have complete information about the state of the world. Instead, we consider scenarios where the robot does not know whether an object is open or not and must perform a test to determine an object's "openness". The robot also has a choice of four different grasping types for manipulating objects in the world. Not all grasp types can be used on every object, and certain grasp types are further restricted by the position of an object relative to other objects in the world. Finally, actions can fail during execution and the robot's sensors may return noisy data.

Basic Representations and OACs

At the robot/vision level, the system has a set Σ of sensors, $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, where each sensor σ_i returns an observation $obs(\sigma_i)$ about some feature of the world, represented as a real-valued vector. The execution of a robot-level action, called a *motor program*, may cause changes to the world which can be observed through subsequent sensing. Each motor program is typically executed with respect to particular *objects* in the world. We assume that initially the robot/vision system does not know about any objects and, therefore, can't execute many motor programs. Instead, the robot has a set of object-independent *basic reflex actions* which it can use in conjunction with the vision system for early exploration and object discovery.

At the planning level, the underlying representation is based on a set of *fluents*, $f_1, f_2, \ldots, f_m$: first-order predicates and functions that denote particular qualities of the world, robot, and objects. Fluents typically represent high-level versions of some of the world-level properties the robot is capable of sensing, where the value of a fluent is a function Γ_i of a set of observations returned by the sensor set, i.e., $f_i = \Gamma_i(\Sigma)$. However, in general, not every sensor need map to some fluent, and we allow for the possibility of fluents with no direct mapping to robot-level sensors.

Fluents may be parametrized and instantiated by high-level counterparts of the objects discovered at the robot level. In particular, for each robot-level object obj^r we denote a corresponding high-level object by obj^p. A *state* is a snapshot of the values of all instantiated fluents at some point during the execution of the system, i.e., $\{f_1, f_2, \ldots, f_m\}$. States represent an intersection between the low-level and high-level representations and are *induced* from the sensor observations (the Γ_i functions) and the object set.

The planning level representation also includes a set of high-level *actions*, $\alpha_1, \alpha_2, \ldots, \alpha_p$, which are viewed as abstract versions of some of the robot's motor programs. Since all actions must ultimately be executed by the robot, each action is decomposable to a fixed set of motor programs $\Pi(\alpha_i)$, where $\Pi(\alpha_i) = \{mp_1, mp_2, \ldots, mp_l\}$, and each mp_j is a motor program. As with fluents, not every robot-level motor program need map to a high-level action.

Although the robot/vision and planning levels use quite different representations (i.e., real-valued vectors versus logical fluents), the notions of "action" and "state change" are common among these components. To capture these similarities, we model our actions and motor programs using a structure called an *Object-Action Complex (OAC)* (Geib et al. 2006; Krüger et al. 2009). Formally, an OAC is a tuple $\langle I, T^S, M \rangle$, where I is an identifier label for the OAC, $T : S \to S$ is a transition function over a state space S, and M is a statistic measure of the accuracy of the transition. OACs provide a universal "container" for encapsulating the relationship between actions (operating over objects) and the changes they make to their state spaces. Each OAC also has an identical set of predefined operations (e.g., composition, update, etc.), providing a common interface to these structures. Since robot systems may have many components, OACs are meant to provide a standard language for describing action-like processes (including continuous processes) within these components, and to simplify the exchange of information between different components.

OACs exist at each level of our system. We encode each motor program on the robot/vision level and each action at the planning level as a separate OAC, with OACs at each level having a different underlying state space. By assigning an accuracy metric to each OAC we also capture the non-deterministic nature of our actions in the real world. Furthermore, since every interaction of the robot with the world provides the robot with an opportunity to observe a small portion of the world's state space (interpreted with respect to the state space of a particular OAC), we can make use of this information to refine or improve the accuracy of the OACs at all levels of our system.

Typically, we consider OACs that are formed from *partial* state descriptions, which may have low reliability. Such descriptions arise since the robot cannot always sense the status of all objects and properties in the world (e.g., occluded or undiscovered objects). Furthermore, the robot's sensors may be noisy and, thus, there is no guarantee that sensor observations are always correct. Certain sensors also have associated resource costs (e.g., time, energy, etc.) which limit their execution. For instance, our robot can perform a test to determine whether an object is open by "poking" the object to check its concavity. Such operations are only initiated on demand at the discretion of the high-level planning system.

Finally, our system includes a middle level component that mediates between the robot and planning levels. This component is responsible for mapping between OACs at different levels of the system (i.e., implementing the Γ_i and Π functions) in order to ensure that observation/state and motor program/action information passing between levels is translated into a form that the destination level understands.

In the remainder of this paper we will look at the main components of our system in greater detail, and describe the current (and future) role of OACs in our framework.

Vision-Based Object Discovery

The visual representation used by the lower level of our system is delivered by an early cognitive vision system (Krüger, Lappe, and Wörgötter 2004; Pugeault 2008) which creates sparse 2D and 3D features, so-called *multi-modal primitives*, along image contours from stereo images. 2D features represent a small image patch in terms of position, orientation, phase, colour and optical flow. These are matched across two stereo views, and pairs of corresponding 2D features permit the reconstruction of an equivalent 3D feature. 2D and 3D primitives are then organized into perceptual groups in 2D and 3D. The procedure to create visual representations is illustrated in Figure 2. We note that the resulting representation not only contains appearance information (e.g., colour and phase) but also geometrical information (i.e., 2D and 3D position and orientation).

Initially, the system lacks knowledge of the objects in a scene and so the visual representation is unsegmented: descriptors that belong to one object are not explicitly distinct from the ones that belong to other objects, or the background. To aid in the discovery of new objects, the robot is equipped with a basic reflex action (Aarno et al. 2007) that is elicited by specific visual feature combinations in the unsegmented world representation (e.g., see Figure 3(a)–(c)). The outcome of these reflexes allows the system to gather knowledge about the scene, which is used to segment the visual world into objects and identify basic affordances. We consider a reflex where the robot tries to grasp a planar surface in the scene. Each time the robot executes such a reflex, haptic information allows the system to evaluate the outcome: either the grasp was successful and the gripper is holding something, or it failed and the gripper simply closed.

With physical control, the system visually inspects an object from a variety of viewpoints and builds a 3D representation (Kraft et al. 2008). Features on the object are tracked over multiple frames, between which the object moves with

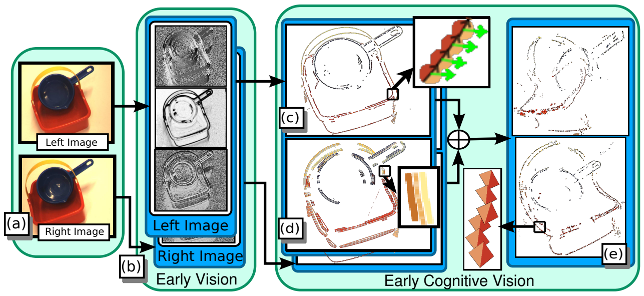


Figure 2: An overview of the visual representation. (a) Stereo image pair, (b) Filter responses, (c) 2D primitives, (d) 2D contours, (e) 3D primitives.

a known motion. If features are constant over a series of frames they become included in the object's representation; otherwise they are assumed to not belong to the object. (See Figure 3(d)–(f) and (Kraft et al. 2008) for a more detailed explanation.) The final description is labelled and recorded as an identifier for a new object class, along with the successful reflex (now a motor program). Using this new knowledge, the system then reconsiders its interpretation of the scene: using a representation-specific pose estimation algorithm (Detry, Pugeault, and Piater 2009) all other instances of the same object class are identified and labelled. By repeating this process, the system constructs a representation of the world objects, as instances of symbolic classes that carry basic affordances, i.e., particular reflex actions that have been successfully applied to objects of this class.[1] This relationship can also be interpreted as a new low-level OAC.

The object-centric nature of the robot's world exploration process has immediate consequences for the high-level representation. First, newly discovered objects are reported to the planning level and added to its representation. At this level, objects are simply labels that act as indices to the object information stored at the robot level. Such a representation means that the planner can avoid reasoning about certain types of real-valued information (e.g., 3D coordinates, orientation vectors, etc.) and instead refer to objects by their labels (e.g., $obj1^p$ may denote a particular red cup on the table). Second, the planner can immediately use such objects during plan generation. Since we assume that object names do not change over time, plans with object references will be understandable to the lower system levels. Finally, the identification of new objects will cause the robot/vision system to start sending regular updates about the state of objects and their properties to the planning level. In particular, low-level observations resulting from subsequent interactions with the world will contain state information about these objects, pro-

[1]We have recently completed the technical implementation of the pose estimation algorithm. Prior to this, a circle detection algorithm was developed (Başeski, Kraft, and Krüger 2009) to recognise cylindrical objects. Four grasp templates were used to define the primitive reflex actions in an object-centric way (where concrete grasps were generated based on the object pose). Although this approach negates the need for the general pose estimation algorithm, the conclusions drawn from experiments in this limited scenario are still easily transferable to the general case.

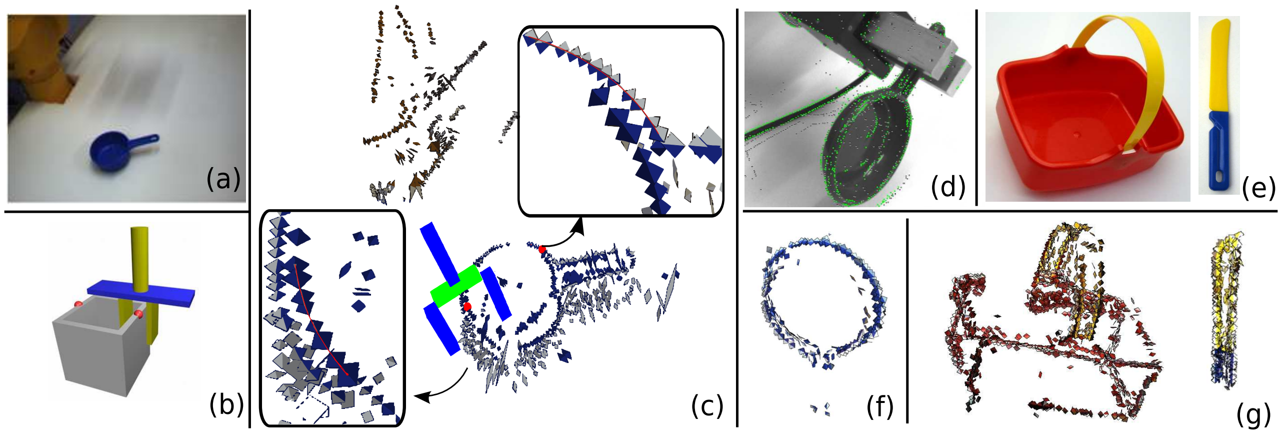


Figure 3: (a)–(c) Initial grasping behaviour: (a) A Scene, (b) Definition of a possible grasp based on two contours, (c) Representation of the scene with contours generating a grasp. (d)–(f) Accumulation process ("birth of the object"): (d) One step in the process. The dots on the image show the predicted structures. Both spurious primitives, parts of the background that are not confirmed by the image, and the confirmed predictions are shown, (e) Images of objects, (f),(g) Extracted models.

vided they can be sensed by the robot.

Knowledge-Level Planning with Sensing

The high-level planner constructs plans that direct the behaviour of the robot to achieve a set of goals. Plans are built using PKS ("Planning with Knowledge and Sensing") (Petrick and Bacchus 2002; 2004), a conditional planner that can operate with incomplete information and sensing actions. Like other symbolic planners, PKS requires a goal, a description of the initial state, and a list of the available actions. Unlike classical planners, PKS operates at the *knowledge level* by explicitly modelling what the planner knows and does not know about the state of the world. PKS can reason efficiently about certain restricted types of knowledge, and make effective use of features like functions, which often arise in real-world scenarios.

PKS is based on a generalization of STRIPS (Fikes and Nilsson 1971). In STRIPS, a single database represents the world state; actions update this database in a way that corresponds to their effects on the world. In PKS, the planner's knowledge state is represented by five databases, each of which stores a particular type of knowledge. Actions describe the changes they make to the database set and, thus, to the underlying knowledge state. PKS also supports ADL-style conditional action effects (Pednault 1989), numerical reasoning, and a set of program-like control structures.

Table 1 shows an example of some of the PKS actions available in the testing domain. As in standard planning representations, like PDDL, actions in PKS are described by their preconditions and effects. Actions may be parametrized (e.g., *graspA*(x)), with an action's parameters replaced with references to specific world objects when an action is instantiated in a plan. As we described above, objects at the planning level are labels to actual objects identified by the robot/vision system.

Preconditions and effects are specified in terms of a set of high-level predicates and functions, i.e., fluents that model particular qualities of the world, robot, and objects. For instance, the actions in Table 1 include references to fluents:

- *open*(x): object x is open,
- *gripperEmpty*: the robot's gripper is empty,
- *onTable*(x): object x is on the table,
- *isIn*(x, y): object x is stacked in object y,
- *radius*$(x) = y$: the radius of object x is y, and
- *reachableX*(x): object x is reachable using grasp type X,

among others. While most high-level properties abstract the information returned by the robot-level sensors (e.g., *onTable* requires data from a set of visual sensors concerning object positions), some properties correspond more closely to individual sensors (e.g., *gripperEmpty* closely models a low-level sensor that detects whether the robot's gripper can be closed without contact).

One significant difference between PKS and other planners is that all actions in PKS are modelled at the knowledge level: preconditions denote conditions that must be true of the planner's knowledge state while effects describe changes to what the planner knows. For instance, precondition expressions of the form $K(\phi)$ denote a knowledge-level query that asks "does the planner know ϕ to be true?" while an expression like $K_w(\phi)$ asks "does the planner know whether ϕ is true or not?" Effect expressions of the form *add*(D, ϕ) assert that ϕ should be added to database D, while *del*(D, ϕ) means that ϕ should be removed from database D. In Table 1, K_f refers to a database that models the planner's definite knowledge of facts, while K_w is a specialized database that stores the results of sensing actions that return binary information.

In our robot scenario, high-level actions represent counterparts to some of the motor programs available at the robot level. For instance, the planner has access to actions like:

- *graspA*(x): grasp x from the table using grasp type A,
- *graspD*(x): grasp x from the table using grasp type D,
- *putInto*(x, y): put x into y on the table,

Action	Preconditions	Effects
$graspA(x)$	$K(reachableA(x))$	$add(K_f, inGripper(x))$
	$K(gripperEmpty)$	$add(K_f, \neg gripperEmpty)$
	$K(onTable(x))$	$add(K_f, \neg onTable(x))$
	$K(clear(x))$	
	$K(radius(x) \geq minA)$	
	$K(radius(x) \leq maxA)$	
$graspD(x)$	$K(reachableD(x))$	$add(K_f, inGripper(x))$
	$K(gripperEmpty)$	$add(K_f, \neg gripperEmpty)$
	$K(onTable(x))$	$add(K_f, \neg onTable(x))$
	$K(radius(x) \leq maxD)$	
$putInto(x, y)$	$K(x \neq y)$	$add(K_f, gripperEmpty)$
	$K(inGripper(x))$	$add(K_f, isIn(x, y))$
	$K(open(y))$	$add(K_f, clear(y))$
	$K(clear(y))$	$add(K_f, \neg inGripper(x))$
	$K(onTable(y))$	
	$K(radius(y) > radius(x))$	
$putAway(x)$	$K(inGripper(x))$	$add(K_f, onShelf(x))$
	$K(shelfSpace > 0)$	$add(K_f, gripperEmpty)$
		$add(K_f, \neg inGripper(x))$
		$add(K_f, shelfSpace -= 1)$
$findout\text{-}open(x)$	$\neg K_w(open(x))$	$add(K_w, open(x))$
	$K(onTable(x))$	

Table 1: PKS actions in the testing domain.

- $putAway(x)$: put x away onto a shelf space, and
- $findout\text{-}open(x)$: determine whether x is open or not,

among others. Some actions like "grasp" are divided into multiple actions (e.g., *graspA*, *graspD*, plus actions for grasp types B and C). The object-centric nature of these actions means they do not require 3D coordinates, joint angles, or similar real values but, instead, include parameters that can be instantiated with specific objects. Actions like *putInto* and *putAway* account for different object/location configurations, although the motor programs that implement these actions do not necessarily make such distinctions. (The complete action list has a larger set of such actions.) The *findout-open* action is an example of a high-level *sensing action* that directs the robot to gather information about the world state that is not normally provided as part of its regular sensing cycle. From the planner's point of view, an action's sensory effects are assumed to only change the planner's knowledge state, while leaving the world state unchanged.

Each planning level action is treated as an individual OAC with its own identifier and transition function corresponding to the action's preconditions and effects. All planning level OACs share a common state space consisting of the high-level predicates and functions. Each OAC also maintains a measure, M, of its reliability, which is updated by the plan execution monitor (see below). Currently, PKS does not use this information (or any probabilistic measures) during plan generation, but instead relies on its ability to reason about incomplete information and replan from action failure.

As an example, consider the situation in the testing domain where two unstacked and open objects $obj1^p$ and $obj2^p$ are on a table, the planner can construct the following plan for clearing all *open* objects from the table:

$$\begin{array}{l} graspD(obj2^p), \\ putInto(obj2^p, obj1^p), \\ graspD(obj1^p), \\ putAway(obj1^p). \end{array}$$

In this plan, $obj2^p$ is grasped from the table using grasp type D (an overhand grasp) and put into $obj1^p$, before the stacked objects are grasped and removed to the shelf.

The planner can also build more complex plans using sensing actions. For instance, if the planner is given the goal of removing the *open* objects from the table in the example scenario, but does not know whether object $obj3^p$ is open or not, then it might construct the conditional plan:

$$\begin{array}{l} findout\text{-}open(obj3^p), \\ branch(open(obj3^p)) \\ K^+ : \\ \quad graspA(obj3^p), \\ \quad putAway(obj3^p) \\ K^- : \\ \quad nil. \end{array}$$

This plan senses the truth value of the predicate $open(obj3^p)$ using *findout-open* and reasons about the possible outcome of this action. As a result, two branches are included in the plan denoting potential execution paths: if $open(obj3^p)$ is true (the K^+ branch) then $obj3^p$ is grasped and put away; if $open(obj3^p)$ is false (the K^- branch) then no action is taken.

State Generation and OAC Interaction

From an integration point of view, the robot/vision system is linked to the planning level through a component which mediates between the state spaces and OACs used by the two levels of the system. Since the planner is not able to handle raw sensor data as a state description, or directly control the robot, the low-level observations generated by the robot/vision system must be abstracted into a language the planner understands, and planned actions must be converted into appropriate robot-level motor programs.

For state space information, sensor data is "wrapped" and reported to the planner in the form of a fluent-based symbolic state representation that includes predicates and functions. Currently, the mappings between certain sensor combinations and the corresponding high-level fluents (i.e., the Γ_i functions) are simply hardcoded. For example:

- *inGripper*, *gripperEmpty*: Initially the gripper is empty and the predicate *gripperEmpty* is formed. As soon as the robot grasps an object ($objX^r$), and confirms that the grasp is successful by means of the gripper not closing up to mechanical limits, the system knows that it has the object in its hand and can form a predicate $inGripper(objX^p)$. Releasing the object returns the gripper to an empty state.
- *reachableX*: Based on the position of a circle forming the top of a cylindrical object in the scene we can compute possible grasp positions (for the different grasp types) for each object. Using standard robotics path planning methods we then compute whether or not there is a collision-free path between the start position and the gripper pose needed to reach the object for a particular grasp.

- *open*: Objects are not assumed to be "open." Unlike the above properties which are determined directly from ordinary sensor data, the robot must perform an explicit test to determine an object's openness. In this case, the robot attempts to use its gripper to "poke" inside the potential opening of an object. If the robot encounters a collision (determined by the FT sensor), the object is assumed to be closed. Otherwise, we assume the object is open.

To compute these predicates, the mediator interacts with the robot/vision system to maintain a snapshot of the current world state which, besides the state information necessary for the planner, also contains information needed for consistency and action computations. In particular, object positions are represented here. To cope with sensor noise (especially the vision-based information about the number and location of circles) a simple mechanism to avoid spurious object disappearance and appearance is employed.

From the planner's point of view, it begins operation without any information about the state of the world. After an initial exploration of the environment, the robot/vision system begins to gather observations and generate (partial) state reports about the current set of objects it believes to be in the world, along with the properties it senses for those objects. This observation set (converted into a fluent-based representation) is then sent to the planner and used as its initial (incomplete) knowledge state: the predicate and function instances are treated as *known* state information, with all other state information considered to be unknown. Subsequent state reports are interpreted by the plan monitor (see below) and used to update the reliability of high-level OACs.

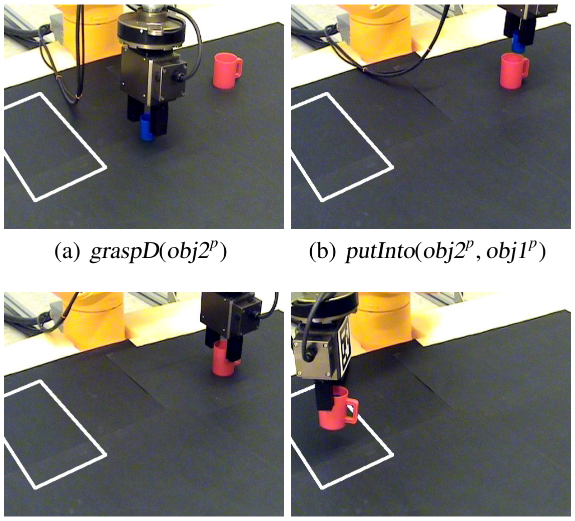
High-level planning actions, in the form of OACs, must also be mapped to their appropriate low-level counterparts, for execution by the robot system in the real world. We currently assume that the set of action schema is supplied to the planner as part of its input, as are the mappings from planning actions to robot motor programs (the Π function).

For instance, the high-level OAC *graspD* is realised on the lowest level as a mapping to an object-independent OAC, $graspD^r$.[2] This low-level OAC requires the object position (retrieved using the object label as an index) as an input to computing suitable grasping positions. The preconditions of this OAC require that there be a grasping position on the brim of the object for which a collision free path from the current position to the grasp position exists. The motor program associated with this OAC is a motion sequence that first completely opens the gripper's fingers, followed by a movement of the arm along the joint trajectory and, lastly, closes the fingers and lifts the arm. After the motor program has been executed the expected outcome state expresses that the fingers should no longer be totally open nor totally closed. In this case, closed fingers indicate that the action failed and no object has been grasped.

Plan Execution and Failure Recovery

Once a plan is generated, the planning level interacts with the robot/vision level (through the mid-level mediator) to execute the plan. Actions are sent to the robot one at a time, where they are converted into motor programs and executed in the world. A stream of observations is also generated, arising from the executed motor programs, and processed into high-level state information. Upon action completion the robot/vision level returns this information to the higher reasoning levels, along with an indication of the success or failure of the action which are used to update the reliability measure *M* of the high-level OACs. The execution cycle then continues. For instance, Figure 4 shows the execution of the four step plan described above for clearing a table.

[2]In general, a high-level OAC may be realised by multiple robot-level OACs.


(a) $graspD(obj2^p)$ (b) $putInto(obj2^p, obj1^p)$

(c) $graspD(obj1^p)$ (d) $putAway(obj1^p)$

Figure 4: Executing a high-level plan to clear a table.

An essential component in this process is the *plan execution monitor*, which assesses action failure and unexpected state information resulting from feedback provided to the planner from the execution of planned actions at the robot level. The execution monitor operates in conjunction with the planner and mid-level mediator, and is responsible for controlling replanning and resensing activities in the system. In particular, the difference between predicted and observed states are used to decide between (i) continuing the execution of an existing plan, (ii) asking the vision system to resense a portion of a scene at a higher resolution in the hope of producing a more detailed state report, and (iii) replanning from an unexpected state using the current state report as a new initial planning state. The plan execution monitor also has the important task of managing the execution of plans with conditional branches, resulting from the inclusion of high-level sensing actions. In each case, the decision of the monitor depends on the type of action being processed and the state information returned by the robot.

Continuing a plan's execution During plan execution, actions are delivered to the lower control levels for execution on the robot. After the execution of each action, a state report representing the *observed* state of the world is returned to the plan monitor and compared against the planner's *predicted* state as constructed during planning, to determine if plan execution should continue or resensing/replanning

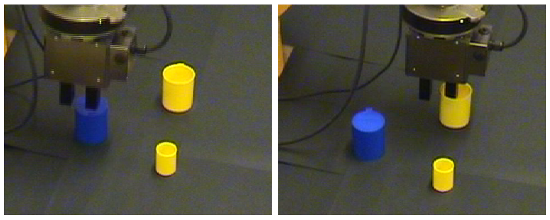
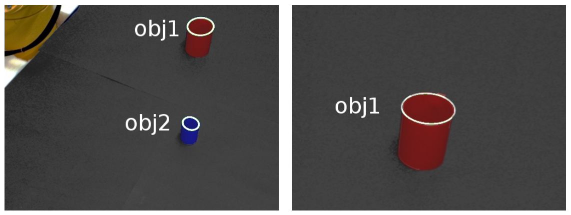

(a) Object is not open (b) Object is open

Figure 5: Testing the openness of an object.

should be activated. Since states in our testing domain tend to be partial, we currently use a limited horizon lookahead method, that attempts to verify that the preconditions for the next n actions in the plan are satisfied in the current (partial) state, and the states that follow when the predicted effects of those actions are applied. (In our testing domain $n = 1$ is often sufficient to ensure good performance.) This means that it is possible for an action to only achieve some of its effects and for the plan to continue, provided the action did not report that it outright failed, and the state is sufficiently correct to ensure the execution of the next action in the plan. (Thus, we defer possible replanning over plan continuation if possible.) If a state match is successful, the monitor then proceeds with the current plan. Otherwise, resensing is considered as a secondary test before replanning (see below).

Sensing actions and conditional plan execution The plan execution monitor also has the added task of managing the execution of plans with sensing actions and associated conditional plan branches. When a high-level sensing action is encountered in a plan it is sent to the robot/vision level like any other action and executed on the robot (as determined by the Π mappings). The actual execution of a sensing action is left to the lower control level which can make more informed decisions about motor program execution. For instance, the *findout-open* action in our example domain is executed at the robot level as a combination of "physical" action (e.g., "poking" an object to determine its openness) and "observational" action (i.e., observing the result); as far as the planner is concerned, the action is executed under the assumption that it is *knowledge producing* and will return an expected piece of information. (Figure 5 shows the execution of *findout-open* by the robot in the case where (a) an object is not open and (b) an object is open.) The sensing result will subsequently be observed by the robot system and returned to the planner as part of the state update cycle.

Plans may also have conditional branch points resulting from sensing actions. When faced with a branch in a plan, the plan execution monitor makes a decision as to the correct plan branch it should execute, based on its current knowledge state. If only partial state information is available, but the required information needed for branch determination is missing (e.g., due to a failure at the robot/vision level), resensing or replanning is triggered. For instance, the example conditional plan given above includes the branch point $branch(open(obj3^p))$, i.e., branch on the truth of the fluent $open(obj3^p)$. If $open(obj3^p)$ is true according to the planner's knowledge state then the "positive" (K^+) branch of the plan is followed and the next action is considered; if $\neg open(obj3^p)$ is true then the "negative" (K^-) branch is followed. If the planner has no information about $open(obj3^p)$ then replanning or resensing is activated. It is important to note that the robot/vision system will never be aware of the conditional nature of a plan, and will never receive a "branch" action. From the point of view of the robot, it will only receive a sequential stream of actions.



(a) (b)

Figure 6: Resensing the scene using the region of interest capabilities of the high resolution cameras.

Resensing at the monitoring level Sensing also plays a role during plan monitoring as a strategy for improving the monitor's accuracy. When the monitor has determined an action's predicted effects do not match the observed state, resensing is considered. At this point, the accuracy of the action's predictions are checked by comparing the M component of the high-level OAC, weighted together with the M components of the OACs of the underlying motor programs which implement this action (the Π mapping), against a threshold value. If the accuracy measure falls below the threshold (i.e., the predictions are considered too spurious), then replanning is activated; otherwise, resensing is performed.

When resensing is required, the plan monitor provides the vision system with a list of the objects considered relevant to the execution of the action that is reported to have failed, based on the parameters in the high-level action description. This information lets the vision system use its high resolution camera to target particular regions of interest in the scene with greater resolution, to reevaluate the sensors that provide information about these objects. New state information returned by this operation may help the monitor decide between continuing a plan's execution and replanning.

For instance, Figure 6(a) shows the state of the world before the $graspD(obj2^p)$ action in our example plan for clearing a table is executed and $obj2^p$ is grasped; both objects in the scene are correctly detected and identified. After executing $graspD(obj2^p)$, however, it is possible that $obj1^p$ may no longer be detected, leading the monitor to resense both $obj1^p$ and $obj2^p$ since the next action in the plan, $putInto(obj2^p, obj1^p)$, depends on these two objects. In Figure 6(b), the old position of $obj1^p$ is resensed, leading to a rediscovery of the object. The old position of $obj2^p$ is also resensed to confirm that it is no longer on the table. In this case, the conditions in the state are sufficient for the monitor to decide that the next action in the plan can be executed.

Replanning When the monitor determines that an action has failed based on the available (resensed) state information, a new plan is constructed for the given goal using the current state as the planner's new initial knowledge state. We use rapid replanning techniques, rather than plan repair, due to the success of planners like FF-Replan (Yoon, Fern, and Givan 2007). This technique also provides a way of overcoming PKS's inability to work with probabilistic representations: if a plan fails we direct PKS to construct an alternate plan for achieving the goal. So far this technique has proven to be effective during testing in our example domain.

Discussion and Conclusions

We believe OACs provide a useful tool for overcoming some of the challenges surrounding the representation of affordances, actions, and state change in real-world robot systems: OACs facilitate the description of different system components in terms of a common representation and common set of interfaces. Although we have grounded many of our system components in terms of the OAC concept, and can describe processes like object discovery and action execution in terms of OACs, our work is preliminary and we have not used this representation to its full potential.

For instance, while our OACs maintain a measure of reliability (i.e., the M measure), this property is not significantly used in our system. We are currently exploring how to improve the reliability of lower-level OACs based on state observations, which could in turn "refine" related higher-level OACs. Closely related to OAC update is the idea of learning completely new OACs. To this end, we are investigating how high-level action schema (i.e., planning level OACs) can be learned directly from (partial) state snapshots provided by the robot level (Mourão, Petrick, and Steedman 2008). Furthermore, we would also like to automatically induce the mapping between OACs at different levels. Thus, the OACs in this paper are not as fully featured as those of (Krüger et al. 2009) and implementing the full set of OAC properties remains a future goal of this work.

The robot/vision components of our system are also being improved. After a recent significant increase in the frequency at which the robot/vision level can provide state updates, we are exploring a more sophisticated mechanism to cope with the sensor noise using multiple consecutive updates. In the future we will also investigate whether a probabilistic framework can increase the reliability of the information provided to the planning level. More work is also needed to properly compare our approach to other existing architectures in the literature.

Although this work is preliminary, we have implemented a framework with all the control mechanisms described here. This has enabled us to test our system in a domain similar to the one described in the paper, but with more actions, more objects, and more complex plans. While the results of our initial experiments look promising, we are also in the process of transferring some of our ideas to a humanoid robot that can operate in a real-world kitchen with real-world objects and appliances. This will provide us with a challenging environment to test the scalability of our system and, in particular, our approach to planning and plan execution.

Acknowledgements

This work was partly funded by the European Commission through the PACO-PLUS project (FP6-2004-IST-4-27657).

References

Aarno, D.; Sommerfeld, J.; Kragic, D.; Pugeault, N.; Kalkan, S.; Wörgötter, F.; Kraft, D.; and Krüger, N. 2007. Early reactive grasping with second order 3D feature relations. In *The IEEE International Conference on Advanced Robotics*.

Başeski, E.; Kraft, D.; and Krüger, N. 2009. A hierarchical 3d circle detection algorithm applied in a grasping scenario. In *Proc. of VISAPP-09*, 496–502.

Detry, R.; Pugeault, N.; and Piater, J. 2009. A probabilistic framework for 3D visual object representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. To appear.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Geib, C.; Mourão, K.; Petrick, R.; Pugeault, N.; Steedman, M.; Krueger, N.; and Wörgötter, F. 2006. Object action complexes as an interface for planning and robot control. In *IEEE-RAS Humanoids-06 Workshop: Towards Cognitive Humanoid Robots*.

Kraft, D.; Pugeault, N.; Başeski, E.; Popović, M.; Kragic, D.; Kalkan, S.; Wörgötter, F.; and Krüger, N. 2008. Birth of the Object: Detection of Objectness and Extraction of Object Shape through Object Action Complexes. *Special Issue on "Cognitive Humanoid Robots" of the International Journal of Humanoid Robotics* 5:247–265.

Krüger, N.; Piater, J.; Wörgötter, F.; Geib, C.; Petrick, R.; Steedman, M.; Ude, A.; Asfour, T.; Kraft, D.; Omrčen, D.; Hommel, B.; Agostini, A.; Kragic, D.; Eklundh, J.-O.; Krüger, V.; Torras, C.; and Dillmann, R. 2009. A formal definition of object-action complexes and examples at different levels of the processing hierarchy. PACO-PLUS Technical Report, available from `http://www.paco-plus.org/`.

Krüger, N.; Lappe, M.; and Wörgötter, F. 2004. Biologically Motivated Multi-modal Processing of Visual Primitives. *The Interdisciplinary Journal of Artificial Intelligence and the Simulation of Behaviour* 1(5):417–428.

McDermott, D. 1998. PDDL – The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Mourão, K.; Petrick, R. P. A.; and Steedman, M. 2008. Using kernel perceptrons to learn action effects for planning. In *Proc. of CogSys 2008*, 45–50.

Murray, R.; Li, Z.; and Sastry, S. 1994. *A mathematical introduction to Robotic Manipulation*. CRC Press.

Pednault, E. P. D. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. of KR-89*, 324–332. Morgan Kaufmann.

Petrick, R. P. A., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. of AIPS-2002*, 212–221.

Petrick, R. P. A., and Bacchus, F. 2004. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proc. of ICAPS-04*, 2–11.

Pugeault, N. 2008. *Early Cognitive Vision: Feedback Mechanisms for the Disambiguation of Early Visual Representation*. Ph.D. Dissertation, Informatics Institute, University of Göttingen.

Yoon, S.; Fern, A.; and Givan, R. 2007. FF-Replan: A baseline for probabilistic planning. In *Proc. of ICAPS-07*, 352–359.