Extending Knowledge-Level Planning with Sensing for Robot Task Planning

Extending Knowledge-Level Planning with Sensing for Robot Task Planning

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

Andre Gaschler
fortiss GmbH
An-Institut Technische Universität München
Munich, Germany
gaschler@fortiss.org

Abstract

We present a set of extensions to the knowledge-level PKS (Planning with Knowledge and Sensing) planner, aimed at improving its ability to generate plans in real-world robotics domains. These extensions include a facility for using externally-defined reasoning processes, an interval-based representation for modelling noisy sensors and effectors, and an application programming interface (API) to facilitate software integration. We demonstrate our techniques in three robot domains, which show their applicability to a broad range of robot planning applications involving incomplete knowledge, real-world geometry, and multiple robots and sensors.

Introduction and Motivation

A robot operating in a real-world domain often needs to do so with *incomplete information* about the state of the world. A robot with the ability to *sense* the world can also gather information to generate plans with *contingencies*, allowing it to reason about the outcome of sensed data at plan time.

In this paper, we explore an application of planning with incomplete information and sensing actions to the problem of task planning in robotics domains. In particular, building models of realistic domains which can be used with general-purpose planning systems often involves working with incomplete (or uncertain) perceptual information arising from real-world sensors. Furthermore, this task may be complicated by the difficulties of bridging the gap between geometric and symbolic representations: robot systems typically reason about joint angles, spatial coordinates, and continuous spaces, while many symbolic planners work with discrete representations in represented in logic-like languages.

Our approach uses the PKS (Planning with Knowledge and Sensing) planner (Petrick and Bacchus 2002; 2004) as the high-level reasoning tool for task planning in robotics domains. PKS is a general-purpose contingent planner that operates at the *knowledge level* (Newell 1982), by reasoning about how its knowledge changes due to action. PKS can represent known and unknown information, and model sensing actions using concise but rich domain descriptions, making it well suited for structured, partially-known environments of the kind that arise in many robot scenarios.

While PKS has previously been successfully used in robot domains (Petrick et al. 2009), it lacks certain features which could improve its applicability to a wider range of robotics tasks. In this paper, we describe a set of extensions designed to improve PKS's ability to generate plans in real-world robot scenarios, by focusing on three tasks: combining high-level symbolic planning with low-level motion planning, reasoning about noisy sensors and effectors, and facilitating planner-level software integration on robot platforms.

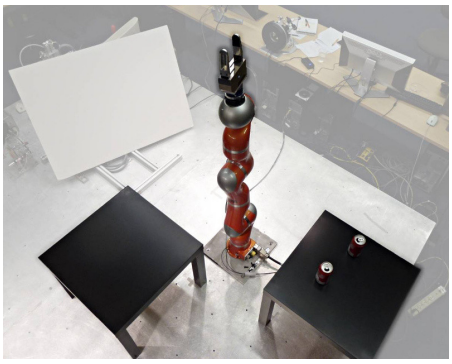

Figure 1: In the FORCE SENSING scenario, a robot manipulator senses if containers are filled by lifting them to determine their weight. Objects must be held upright while moving to prevent spilling, unless they are empty or unopened.

PKS has also been integrated into the existing Knowledge of Volumes framework for robot task Planning (KVP) (Gaschler et al. 2013a), aimed at facilitating the use of planners on robot platforms (see Figures 1 and 5). This framework serves as the basis for the robot examples below.

Planning with Knowledge and Sensing (PKS)

PKS (Planning with Knowledge and Sensing) is a contingent planner that builds plans using incomplete information and sensing actions (Petrick and Bacchus 2002; 2004). PKS works at the *knowledge level* by reasoning about how the planner's knowledge state changes due to action. PKS works with a restricted first-order logical language, and limited inference. Thus, unlike planners that reason with possible worlds or belief states, PKS works directly with formulae representing its knowledge state. This representation supports features like functions and variables; however, as a trade-off, some types of knowledge cannot be modelled.

PKS is based on a generalisation of STRIPS. In PKS, the planner's knowledge state, rather than the world state, is represented by a set of five databases, each of which models a particular type of knowledge. Each database's contents have a formal interpretation in a modal logic of knowledge. Actions can modify the databases, which has the effect of updating the planner's knowledge. To ensure efficient inference, PKS restricts the type of knowledge (especially disjunctions) the databases can represent:

$\mathbf{K_f}$: This database is like a STRIPS database except that both positive and negative facts are stored and the closed world assumption is not applied. K_f can include any ground literal ℓ, where $\ell \in K_f$ means "the planner knows ℓ."

$\mathbf{K_w}$: This database models the plan-time effects of sensing actions with binary outcomes. $\phi \in K_w$ means that at plan time the planner either "knows ϕ or knows $\neg\phi$," and that at execution time this disjunction will be resolved. In such cases we will also say that the planner "knows whether ϕ."

$\mathbf{K_v}$: This database stores function values that will become known at execution, such as the effects of sensing actions that return constants. K_v can contain any unnested function term f, where $f \in K_v$ means that at plan time the planner "knows the value of f." At execution time, the planner will have definite information about f's value. Thus, K_v terms can act as *run-time variables* (Etzioni et al. 1992) in plans.

$\mathbf{K_x}$: This database models the planner's *exclusive-or* knowledge. Entries in K_x have the form $(\ell_1|\ell_2|\ldots|\ell_n)$, where each ℓ_i is a ground literal. Such formulae represent a particular type of disjunctive knowledge that arises in many planning scenarios, namely that "exactly one of the ℓ_i is true."

$\mathbf{LCW}$: This database stores *local closed world* information (Etzioni, Golden, and Weld 1994). The LCW database will not be used in this paper.

PKS databases are inspected using *primitive queries* that ask simple questions about PKS's knowledge. Basic knowledge assertions are tested with a query $K(\phi)$ which asks: "is a formula ϕ true?" A query $K_w(\phi)$ asks whether ϕ is known to be true or known to be false. A query $K_v(t)$ asks "is the value of function t known?" The negation of the above queries can also be used. An inference procedure evaluates queries by checking the contents of the databases, and the interaction between different types of knowledge.

An action in PKS is modelled by its *preconditions* that query the agent's knowledge state, and its *effects* that update the state. Action preconditions are simply a list of primitive queries. Action effects are described by a set of STRIPS-style "add" and "delete" operations that modify individual databases. E.g., *add*(K_f, ϕ) adds ϕ to K_f, and *del*(K_w, ϕ) removes ϕ from K_w. Actions can also have ADL-style conditional effects (Pednault 1989), where the secondary preconditions of an effect are described by lists of primitive queries. Example PKS actions are shown in Figure 3.

PKS constructs plans by forward-chaining: if an action's preconditions are satisfied in a knowledge state, then the action's effects can be applied to produce a new knowledge state. Planning then continues from this state. PKS can also build contingent plans with branches, by considering the possible outcomes of K_w and K_v knowledge. For instance, if $\phi \in K_w$ then PKS can construct two branches in a plan: along one branch (the K^+ branch) ϕ is assumed to be known (ϕ is added to K_f), while along the other branch (the K^- branch), $\neg\phi$ is assumed to be known ($\neg\phi$ is added to K_f). A similar type of multi-way branching plan can be built for restricted K_v information. Planning continues along each branch until the *goal*—a list of primitive queries–is satisfied. A sample PKS plan with branches is shown in Figure 4.

We now consider a set of extensions to PKS which we believe are particularly useful for robot task planning.

Extensions to PKS for Robot Task Planning

In this section we consider three recent extensions to PKS which we believe are useful for robot task planning. First, we describe a mechanism which allows externally-defined procedures (e.g., from support libraries) to be integrated with the internal reasoning mechanisms of the planner. Second, we present an extension of the PKS representation which allows a form of noisy numerical information to be modelled, for instance to represent the effects of error prone sensors. Finally, we describe a software-level application programming interface to PKS, which aids in the engineering task of integrating the planner with a robot system.

Executing Externally-Defined Procedures

The first extension we describe aims to leverage existing reasoning tools by providing a mechanism for PKS to invoke externally-defined procedures (e.g., special purpose libraries) during plan generation. While this idea is not new, and has been applied in other contexts (see the discussion section), this technique is a recent extension to PKS.

In particular, PKS provides an external query mechanism:

$$\texttt{extern}(\mathit{proc}(\vec{x})),$$

where `extern` is a special keyword indicating that control should be transferred to an external procedure with the name *proc*. $\vec{x}$ is a set of parameters that should be passed to *proc*. In general, x can contain any symbols defined in PKS's knowledge state, providing a link between the planner and the externally-defined procedure.

An `extern` call can be used within a PKS action definition, either as a precondition or an effect. The return value of the `extern` call, defined within the external procedure, is passed back to PKS, which interprets it in the context where it occurs in the action. For instance, consider the following definition of a simple pickup action:

```
action pickup(?x : object)
    preconds:
        K(holding = nil) &
        K(onTable(?x)) &
        K(extern(isReachable(?x)))
    effects:
        add(Kf, ¬onTable(?x)),
        add(Kf, holding = ?x),
        add(Kf, weight = extern(objWeight(?x)))
```

In this action encoding the first two preconditions are modelled as standard PKS primitive queries: does the planner know the hand is empty and is object `?o` known to be

on the table? The third precondition combines a K (knows) primitive query together with an `extern` call by invoking an external process (e.g., a robot motion planner) to determine the truth of the `isReachable` property. Here, the parameter `?x` is bound by the planner during plan generation before it is passed to the external reasoner for consideration. The result of the `extern` process is then used by the planner to establish whether the K condition holds or not.

Similarly, the final action effect for `pickup` also contains an `extern` reference. Here, an external process establishes the value of `objWeight(?x)` (i.e., the weight of object `?x`), which is returned and assigned to the PKS property `weight` before being stored in the K_f database. In general, additional tests or reasoning can be performed on any value returned by an `extern` call, which can be assigned to a PKS domain property and included in the knowledge state.[1]

While no restrictions are placed on when `extern` can be used in a planning domain, in practice such calls are most beneficial if used for complex reasoning that cannot easily be modelled in the planner's restricted representation, or where more efficient reasoning engines already exist. In order to provide some control over the nature of an external reasoning process, a variant of the standard `extern` query can be specified as:

$$\texttt{extern*}(\mathit{proc}(\vec{x})).$$

In contrast to an ordinary `extern` directive which simply invokes the external process $\mathit{proc}(\vec{x})$, `extern*` directs the planner to save the results of the call for the given input parameters $\vec{x}$. Subsequent calls to `extern*` using the same set of parameters are therefore guaranteed to produce the same output, and the planner is directed to simply used the previously cached result. For instance,

```
add(K_f, outcome = extern(flipCoin(c1)))
```

could be used to model the effects of an action that flips a coin `c1` and assigns the results of the coin flip to the property `outcome`. Subsequent calls may produce different outcomes, depending on the external implementation of `flipCoin`. On the other hand,

```
add(K_f, outcome = extern*(flipCoin(c1)))
```

directs the planner to save the outcome of `c1`'s first coin flip. Subsequent applications of the same rule will therefore always return the same result as previously saved. Such a facility is particularly useful in cases where the external process may be computationally expensive; however, it requires a strict functional interpretation of the `extern` call, where a particular set of inputs is associated with exactly one output.

The `extern` mechanism provides a simple, yet powerful tool in robotics domains by augmenting PKS's core reasoning capabilities with the addition of motion planning, collision detection, and other special purpose robotics libraries. E.g., geometric predicates and continuous motions can be evaluated with `extern`, and reasoned about at the symbolic level, enabling us to solve problems which may be difficult to model directly at either the motion planning or symbolic planning level alone. Examples of this are given below.

[1] Quite complex expressions can be formed by nesting PKS function terms and `extern` calls. For instance,
`add(`K_f`,holding=extern(closestObj(extern(robotLoc))))`
could be used to model an action effect where a robot picks up the closest object to the robot's current location, using nested `extern` calls to perform the necessary spatial reasoning.

Reasoning with Interval-Valued Fluents

One type of sensed information that arises in many real-world robotics contexts is *numerical* information, which is often necessary for modelling state properties (the robot is 10 metres from the wall), limited resources (ensure the robot has enough fuel), constraints (only grasp an object if its radius is less than 10 cm), or arithmetic operations (advancing the robot one step reduces its distance to the wall by 1 metre). Reasoning with numerical information is often problematic, however, especially when planners represent incompletely known state properties by sets of states, each of which denotes a possible configuration of the world state. E.g., if a fluent f could map to any natural number between 1 and 100, then we require 100 states to capture f's possible mappings. The state explosion resulting from large sets of mappings can be computationally difficult for planners that must reason directly with individual states.

In PKS, we build on a previous planning approach which uses *interval-valued fluents (IVFs)* (Funge 1998; Petrick 2011) to avoid some of the computational problems involved with uncertain numerical information. The idea is simple: instead of representing each possible mapping by a separate state, a single interval mapping is used, where the endpoints of the interval indicate the fluent's range of possible values. Thus, a fluent f that could map to values between 1 and 100 can be denoted in an interval-valued form by $f = \langle 1, 100\rangle$.

In general, PKS treats each IVF as a function whose denotation is an *interval* of the form $\langle u, v\rangle$. The *endpoints* of the interval, u and v, indicate the bounds on the range of possible mappings for the fluent. Since we are interested in planning with incomplete information, a mapping $f = \langle u, v\rangle$ will mean that the value of f is known to be in the interval $\langle u, v\rangle$. If a fluent maps to a *point interval* of the form $\langle u, u\rangle$, then the mapping is certain and known to be equal to u.

PKS's knowledge of IVFs are stored in its K_x database, as a generalisation of its exclusive-or information. In addition to basic intervals, *disjunctive intervals* (i.e., sets of disjoint intervals) are also permitted. For instance, if a fluent f could possibly map to any value between 5 and 10 or, alternatively, map to values between 15 and 18, we can represent such information by the K_x formula $(f = \langle 5, 10\rangle \,|\, f = \langle 15, 18\rangle)$.

Certain types of IVFs can also be represented in the K_v and K_w databases. For instance, a fluent of the form $f : \langle x - c, x + c\rangle$ in K_v means that the value of the fluent f is known, and f is in the range $x \pm c$, for some numeric constant c and unknown fluent value x. This mechanism can be used to model the results of noisy sensors. In K_w, we also permit numeric relations of the form f **op** c, where **op** $\in \{=, \neq, >, <, \geq, \leq\}$ and c is a numeric constant. Thus, $f > 5 \in K_w$ can be used to model a sensing action that determines whether f is greater than 5 or not. Since K_w is used to build contingent branches into a plan, this extension also enables PKS to build branches based on IVFs.

A Planning Application Programming Interface

The task of integrating planners on robot platforms often centres around the problem of representation, and how to abstract the capabilities of a robot and its working environment into a suitable form for use by the planner. Integration also typically requires the ability to communicate information between system components and, thus, requires a consideration of certain engineering-level concerns, to ensure proper interoperability with components that aren't traditionally considered in theoretical planning settings.

To facilitate the task of providing software-level planning services to robot systems, we created an application programming interface (API) which abstracts many common planning operations into a set of functions which provide direct, programme-level access to these services. For instance, the API includes methods for manipulating domain representations and controlling certain aspects of the plan generation process (e.g., selecting goals, generation strategies, or planner-specific settings). Plans can also be manipulated as first-class entities, e.g., for replanning purposes. A fragment of the API is given in Figure 2.

Overall, the API is designed to be generic and is not tied to a particular planner. For instance, the configuration methods are meant to provide a way to set properties of the underlying planner, and provide access to features needed for debugging. The domain configuration functions provide the means for defining planning domain models, either from traditional domain/problem files, or via string-based descriptions. A key idea behind the API is that it offers the possibility of specifying domains to the planner incrementally, using function calls alone, rather than specifying a single monolithic domain file. This means that an initial domain could be specified and then later revised, for instance using information discovered during execution (e.g., new objects, revised action models, additional properties, etc.). Finally, the plan generation and iteration functions provide ways of controlling certain aspects of plan generation, and provide a way for external processes to control monitoring and replanning activities, including goal change.

The PKS API is implemented as a C++ library, but also supports a network-based interface using the Internet Communications Engine (ICE)[2]. An interface for the Robot Operating System (ROS)[3] is also currently under development.

```
// Configuration and debugging
string  getPlannerProperty(string);
bool    setPlannerProperty(string, string);

// Domain configuration
bool    defineDomain(string);
bool    defineSymbols(string);
bool    defineActions(string);
bool    defineProblems(string);
bool    definePlanState(string);
bool    defineObservedState(string);

// Plan generation and iteration
bool    buildPlan();
bool    isPlanDefined();
string  getCurrentPlan();
Action  getNextAction();
bool    isNextActionEndOfPlan();
bool    setProblem(string);
bool    setProblemGoal(string);
```

Figure 2: A fragment of the PKS API.

Example Domains

To demonstrate our approach, we describe three robotics scenarios that make use of knowledge-level planning: the FORCE SENSING and the BIMANUAL robot scenarios, based on domains first described in (Gaschler et al. 2013c) and tested on real robots, and the ROBOT LOCALISATION scenario, tested in simulation. In all scenarios, the robot uses sensing actions to obtain knowledge of some domain property which is necessary for achieving the goal. In the first scenario, basic PKS is used. In the second scenario, PKS's external procedure mechanism is used to link a motion planning library to PKS's internal reasoning mechanisms. In the final scenario, we use IVFs in a simple localisation task.

Force Sensing Scenario

In the FORCE SENSING scenario, a robot manipulator must transfer beverage containers from one table to another, as shown in Figure 1. Using torque sensors, it can sense the external force of a grasped container, and decide whether or not that drink could be spilled. The robot should hold drinks exactly upright to prevent spilling, unless a drink is known to be completely empty, in which case a faster arbitrary motion may be performed. For simplicity, the location of all objects are known and no sensing except force sensing is available.

Figure 3 shows some of the PKS actions in this scenario, which include a sensing action, `senseWeight`, which senses the weight of a beverage container `?o`. To perform this action, the robot must first be grasping object `?o`. To ensure only new knowledge is gained from this action, and to increase planning efficiency, we include a precondition that the robot must not yet know whether `?o` is spillable. When this action is performed, knowledge of whether `?o` is spillable or not is added to PKS's K_w database.

This scenario also includes a number of actions for manipulating domain objects, including `transferUpright`, `transfer`, `grasp`, and `ungrasp`. For example, in `transferUpright`, the robot can move a grasped container from one table to another, while keeping the orientation of its parallel gripper fixed. Only objects that are grasped and not yet removed can be transferred.

An example plan for the FORCE SENSING scenario is shown in Figure 4 for the case of two objects in the domain. In particular, a sensing action is performed on each object (`can1` and `can2`) and the objects are individually manipulated depending on whether their contents are spillable or not. The resulting plan therefore considers four contingent situations which could arise during plan execution. This scenario was tested on a joint-impedance controlled light-

[2] `http://www.zeroc.com/ice.html`

[3] `http://www.ros.org/`

```
action senseWeight(?o:object)
    preconds:
        ¬K_w(isSpillable(?o)) &
        K(isGrasped(?o))
    effects:
        add(K_w, isSpillable(?o))

action transfer(?o:object)
    preconds:
        K(¬isSpillable(?o)) &
        K(isGrasped(?o)) &
        K(¬isRemoved(?o))
    effects:
        add(K_f, isRemoved(?o))

action transferUpright(?o:object)
    preconds:
        K(isSpillable(?o)) &
        K(isGrasped(?o)) &
        K(¬isRemoved(?o))
    effects:
        add(K_f, isRemoved(?o))

action grasp(?o:object)
    preconds:
        K(emptyGripper) &
        K(¬isRemoved(?o))
    effects:
        add(K_f, isGrasped(?o)),
        add(K_f, ¬emptyGripper)

action ungrasp(?o:object)
    preconds:
        K(isGrasped(?o)) &
        K(isRemoved(?o))
    effects:
        add(K_f, ¬isGrasped(?o)),
        add(K_f, emptyGripper)
```

Figure 3: Actions in the FORCE SENSING domain.

```
 1.  grasp(can1) ;
 2.  senseWeight(can1) ;
 3.  branch(isSpillable(can1))
 4.  K+: transferUpright(can1) ;
 5.      ungrasp(can1) ;
 6.      grasp(can2) ;
 7.      senseWeight(can2) ;
 8.      branch(isSpillable(can2))
 9.      K+: transferUpright(can2) ;
10.          ungrasp(can2).
11.      K-: transfer(can2) ;
12.          ungrasp(can2).
13.  K-: transfer(can1) ;
14.      ungrasp(can1) ;
15.      grasp(can2) ;
16.      senseWeight(can2) ;
17.      branch(isSpillable(can2))
18.      K+: transferUpright(can2) ;
19.          ungrasp(can2).
20.      K-: transfer(can2) ;
21.          ungrasp(can2).
```

Figure 4: A plan for removing 2 objects from a table in the FORCE SENSING domain.

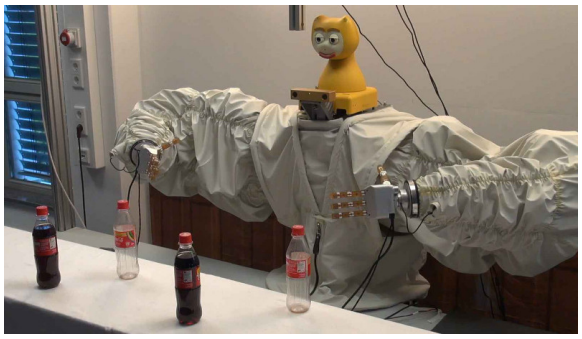

Figure 5: In the BIMANUAL scenario, a camera is used to recognise empty bottles which a bimanual robot should remove from to a "dishwasher" location on the left side, behind the table (Gaschler et al. 2013a; Giuliani et al. 2013).

weight 7-DoF robot with a force-controlled parallel gripper. Forces were measured by internal torque sensing.

Bimanual Robot Scenario

The second scenario is a demonstration of a BIMANUAL robot (Figure 5) whose hands can reach different areas of a table. In this case, the robot can sense if bottles on the table are empty or full using a top-down camera. Its goal is to clean up all empty bottles by removing them to a certain "dishwasher" location. In order to achieve this goal, the robot must move objects that are only accessible by its left arm to a location that its right arm can reach, a behaviour which arises purely from symbolic planning. In contrast to the previous FORCE SENSING scenario, the BIMANUAL robot scenario relies on visual information, which can be gathered without requiring manipulation.

Figure 6 shows the PKS actions in the BIMANUAL scenario. Two robot arms are tasked with removing all empty bottles that are visible on a table to the dishwasher, which can only be reached by the right robot arm. The domain includes one sensing action, `senseIfEmpty`, which has no precondition other than the requirement that the knowledge it gathers must be new. For manipulation, both robot arms can perform the `pickUp` and `putDown` actions. Not all locations can be reached by both hands, so the preconditions of these actions include an **extern** call to `isReachable`, which is defined in a motion planning library and which checks reachability for a specific hand and location.

Figure 7 shows a plan for the case of 4 objects. The plan senses each object to detect whether it is empty and then includes conditional branches to remove empty objects to the dishwasher. The resulting plan therefore considers 16 possible configurations of empty/non-empty bottles which could arise at execution. (The actions for the case where bottles `b0` and `b2` are empty are shown.) We note that this simple scenario already gives rise to interesting behaviour: since the right arm cannot directly reach all objects that need to be transferred to the dishwasher, the left arm must pass those objects to a location reachable by both hands. This

```
action senseIfEmpty(?o:object)
    preconds:
        ¬K_w(isEmptyBottle(?o))
    effects:
        add(K_w, isEmptyBottle(?o))

action pickUp(?r:robot, ?o:object, ?l:location)
    preconds:
        K(?l = getObjectLocation(?o)) &
        K(handEmpty(?r)) &
        K(extern(isReachable(?l, ?r)))
    effects:
        del(K_f, ?l = getObjectLocation(?o)),
        del(K_f, handEmpty(?r)),
        add(K_f, inHand(?o, ?r))

action putDown(?r:robot, ?o:object, ?l:location)
    preconds:
        K(inHand(?o, ?r)) &
        K(extern(isReachable(?l, ?r)))
    effects:
        del(K_f, inHand(?o, ?r)),
        add(K_f, ?l = getObjectLocation(?o)),
        add(K_f, handEmpty(?r))
```

Figure 6: Actions in the BIMANUAL domain.

behaviour is not pre-programmed, but arises purely from combined symbolic and geometric planning. This domain was tested on a two 6-DoF industrial manipulator setup with Meka Robotics H2 hands, as in (Foster et al. 2012).

Robot Localisation Scenario

In the final example, we consider a robot whose location, represented by the IVF `robotLoc`, is measured by the robot's distance to a wall. The robot has two physical actions available to it: `moveForward`, which moves the robot either 1 or 2 units towards the wall; and `moveBackward`, which moves the robot 1 unit away from the wall. The robot also has a sensing action, `atTarget`, which senses whether the robot is at a target location, specified by the function `targetLoc`. Additionally, the robot also has a second sensing action, `withinTarget`, that determines whether or not the robot is within the target distance `targetLoc`.

The definitions of the PKS actions for this scenario are given in Figure 8 (all action preconditions are assumed to be true). The robot's initial location is specified by the interval mapping $\texttt{robotLoc} = \langle 3, 4\rangle$ stored in K_x. The goal is to move the robot to the target location, i.e., $K(\texttt{robotLoc} = \texttt{targetLoc})$, where $\texttt{targetLoc} = 2$ is stored in K_f.

One solution generated by PKS is the conditional plan in Figure 9. Since forward movements may change the robot's position by either 1 unit or 2 units, `noisyForward` in step 1 results in an even less certain position for the robot, namely $\texttt{robotLoc} = \langle 1, 3\rangle \in K_x$. However, the sensing action in step 2, together with the branch point in step 3, lets us split this interval into two parts. In step 4, we assume that $\texttt{robotLoc} \leq 2$ and consider the case where $\texttt{robotLoc} = \langle 1, 2\rangle$. `atTarget`, together with the branch in step 6, lets us divide this interval even further: in step 7,

```
 1.  senseIfEmpty(b0) ;
 2.  senseIfEmpty(b1) ;
 3.  senseIfEmpty(b2) ;
 4.  senseIfEmpty(b3) ;
 5.  branch(isEmptyBottle(b0))
 6.  K+: branch(isEmptyBottle(b1))
 7.      K+: ...
 8.      K-: branch(isEmptyBottle(b2))
 9.          K+: branch(isEmptyBottle(b3))
10.              K+: ...
11.              K-: pickUp(left,b0,l0) ;
12.                  putDown(left,b0,l5) ;
13.                  pickUp(right,b2,l2) ;
14.                  putDown(right,b2,dishwasher) ;
15.                  pickUp(right,b0,l5) ;
16.                  putDown(right,b0,dishwasher).
17.          K-: ...
18.  K-: ...
```

Figure 7: A plan for 4 objects in the BIMANUAL domain.

```
action moveForward
   effects:
       add(K_f, robotLoc := robotLoc - <1,2>)

action moveBackward
   effects:
       add(K_f, robotLoc := robotLoc + 1)

action atTarget
   effects:
       add(K_w, robotLoc = targetLoc)

action withinTarget
   effects:
       add(K_w, robotLoc <= targetLoc)
```

Figure 8: Actions in the LOCALISATION domain.

$\texttt{robotLoc} = 2$ and the goal is satisfied, while in step 8, $\texttt{robotLoc} = 1$ and a `moveBackward` action achieves the goal. In step 10 we consider the other sub-interval of the first branch, i.e., $\texttt{robotLoc} = 3 \in K_f$. In this case we have definite knowledge, however, a subsequent `noisyForward` results in $\texttt{robotLoc} = \langle 1, 2\rangle$. The rest of the plan in steps 12–16 is the same as in steps 5–9: the robot conditionally moves backwards in the case that `robotLoc` is determined to be 1, while the plan trivially achieves the goal if $\texttt{robotLoc} = 2$.

While we have only tested this domain in simulation, experimentation on a real robot is the focus of current work.

Related Work and Discussion

Applications of automated planning to robotics have a long history, going back to robots like Shakey (Nilsson 1984) and Handey (Lozano-Pérez et al. 1989). Recently, the field has made substantial progress, with many approaches to robot task planning proposed, including probabilistic AI techniques (Kaelbling and Lozano-Pérez 2013), closed-world symbolic planning (Cambon, Alami, and Gravot 2009; Plaku and Hager 2010; Dornhege et al. 2009b), formal synthesis (Kress-Gazit and Pappas 2008; Cheng

```
                                          robotLoc
 0.                                       ⟨3, 4⟩
 1.  noisyForward ;                       ⟨1, 3⟩
 2.  withinTarget ;
 3.  branch(robotLoc ≤ targetLoc)
 4.  K+:                                  ⟨1, 2⟩
 5.      atTarget ;
 6.      branch(robotLoc = targetLoc)
 7.      K+: nop.                           2
 8.      K-:                                1
 9.          moveBackward.                  2
10.  K-:                                    3
11.      noisyForward ;                   ⟨1, 2⟩
12.      atTarget ;
13.      branch(robotLoc = targetLoc)
14.      K+: nop.                           2
15.      K-:                                1
16.          moveBackward.                  2
```

Figure 9: A plan in the LOCALISATION domain.

et al. 2012), and sampling-based manipulation planning (Zacharias, Borst, and Hirzinger 2006; Barry 2013).

As part of this work, we use the existing Knowledge of Volumes framework for robot task Planning (KVP) (Gaschler et al. 2013a; 2013b; 2013c). KVP uses PKS an its underlying symbolic planner, and combines it with the idea of treating 3D geometric volumes as an intermediary representation between continuously-valued robot motions and discrete symbolic actions, to bridge the gap between geometric and symbolic planning representations. In our case, Motion planning and collision detection rely heavily on the Robotics Library (RL) (Rickert 2011),[4] extended with additions to swept volume computations with sets of convex bodies (Gaschler et al. 2013a). The integration of PKS within KVP is achieved using the API described in this paper.

A number of other approaches also address the problem of integrating symbolic planning and motion planning, notably Kaelbling and Lozano-Pérez's work on hierarchical task and motion planning (Kaelbling and Lozano-Pérez 2011). However, while their geometric preconditions may be similar, their underlying aggressively hierarchical planning strategy differs from our knowledge-level planning approach. Further approaches that integrate symbolic and geometric reasoning are presented by Cambon, Alami and Gravot (2009), handling geometric preconditions and effects; Dornhege et al. (2009b); and, more recently, Plaku and Hager (2010), which additionally allow differential motion constraints in a sampling-based motion and action planner. We note that the latter three approaches assume a closed world, where all symbols must be true or false. We use open-world knowledge, which lets us model incomplete information and high-level sensing. Prior work has also used PKS to connect robot vision and grasping with planning (Petrick et al. 2009).

In terms of our PKS extensions, the ability to link external libraries to internal reasoning processes is key to our approach. While this idea is not new, and has been previously applied (Eiter et al. 2006; Dornhege et al. 2009a; Erdem et al. 2011), the introduction of this technique to PKS is a recent addition. Current work is focused on extending this interface, to allow external procedures partial access to internal PKS states, for more efficient external execution during plan generation. One drawback with this facility, is that there is currently no control over how long an external procedure may take to execute, or whether it will terminate. As a result, we are extending our `extern` implementation to include a timeout facility that will force external procedure calls to terminate if a specified cutoff time is reached.

Interval-valued numeric models have been previously investigated in planning contexts, e.g., for modelling time as a resource (Edelkamp 2002; Frank and Jónsson 2003; Laborie 2003). A similar representation to ours for bounding noisy numeric properties has also been proposed by Poggioni, Milani, and Baioletti (2003). This idea also has parallels to work on register models (van Eijck 2013). The importance of numerical reasoning in planning has been recognised with the inclusion of numeric state variables in PDDL, and in planners like MetricFF. We believe representations such as IVF offer a useful middle ground between discrete and fully probabilistic models of uncertainty. While PKS has provided a successful prototype for testing this representation, we are also currently adapting this technique for use in other planners. Current work is focused on encoding a representation of IVFs in a temporal-numeric planner, such as POPF2 (Coles et al. 2010).

Finally, we note that our planning API can be thought of as a set of abstract planning services which are implemented by an underlying "black box" planner. As in other complex software systems, such an interface removes the need for the programmer to know how such services are actually implemented, but instead allows the designer to build more complex components that simply use these services. This API has been deployed on four different robot systems to enable access to PKS, and is being integrated on a fifth system to provide onboard task planning services for a set of robots in an industrial warehouse setting (Crosby and Petrick 2014).

Conclusions

We described a set of extensions to PKS, aimed at improving its applicability to problems in robot task planning. We demonstrated these capabilities in solving typical robot tasks at the knowledge level, including the combination of high-level planning with low-level motion planning, with experiments on real and simulated robots. simulation. As part of ongoing work, we are continuing to refine and apply our extensions in more complex scenarios, to gather empirical data and better understand the limits of our techniques. We believe our approach is useful for a broad range of robot planning applications that require incomplete knowledge, real-world geometry, and multiple robots and sensors.

Acknowledgements

This research was supported in part by the European Commission's Seventh Framework Programme under grant no. 270435 (JAMES, `james-project.eu`), grant no. 270273 (XPERIENCE, `xperience.org`), and grant no. 610917 (STAMINA, `stamina-robot.eu`).

[4] `http://www.roboticslibrary.org/`

References

Barry, J. L. 2013. *Manipulation with Diverse Actions*. Ph.D. Dissertation, Massachusetts Institute of Technology.

Cambon, S.; Alami, R.; and Gravot, F. 2009. A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research* 28(1):104–126.

Cheng, C.; Geisinger, M.; Ruess, H.; Buckl, C.; and Knoll, A. 2012. Game solving for industrial automation and control. In *IEEE Int. Conf. on Robotics and Automation*, 4367–4372.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 42–49.

Crosby, M., and Petrick, R. P. A. 2014. Temporal multiagent planning with concurrent action constraints. In *Proceedings of the ICAPS 2014 Workshop on Distributed and Multi-Agent Planning (DMAP)*, 16–24.

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009a. Semantic attachments for domain-independent planning systems. In *Proc. of the Int. Conference on Automated Planning and Scheduling (ICAPS)*, 114–121.

Dornhege, C.; Gissler, M.; Teschner, M.; and Nebel, B. 2009b. Integrating symbolic and geometric planning for mobile manipulation. In *IEEE International Workshop on Safety, Security & Rescue Robotics*, 1–6.

Edelkamp, S. 2002. Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research* 20:195–238.

Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2006. Effective integration of declarative rules with external evaluations for semantic-web reasoning. In *The Semantic Web: Research and Applications*, 273–287.

Erdem, E.; Haspalamutgil, K.; Palaz, C.; Patoglu, V.; and Uras, T. 2011. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Int. Conference on Robotics and Automation*, 4575–4581.

Etzioni, O.; Hanks, S.; Weld, D.; Draper, D.; Lesh, N.; and Williamson, M. 1992. An approach to planning with incomplete information. In *Proceedings of the International Conference on Knowledge Representation and Reasoning (KR)*, 115–125.

Etzioni, O.; Golden, K.; and Weld, D. 1994. Tractable closed world reasoning with updates. In *Proc. of the International Conference on Knowledge Representation and Reasoning (KR)*, 178–189.

Foster, M. E.; Gaschler, A.; Giuliani, M.; Isard, A.; Pateraki, M.; and Petrick, R. 2012. Two people walk into a bar: Dynamic multi-party social interaction with a robot agent. In *Proceedings of the ACM International Conference on Multimodal Interaction (ICMI)*.

Frank, J., and Jónsson, A. 2003. Constraint-based attribute and interval planning. *Journal of Constraints, Special Issue on Constraints and Planning* 8:339–364.

Funge, J. 1998. Interval-valued epistemic fluents. In *AAAI Fall Symposium on Cognitive Robotics*, 23–25.

Gaschler, A.; Petrick, R.; Giuliani, M.; Rickert, M.; and Knoll, A. 2013a. KVP: A Knowledge of Volumes Approach to Robot Task Planning. In *IEEE/RSJ Intl Conf on Intelligent Robots and Systems (IROS)*, 202–208.

Gaschler, A.; Petrick, R.; Kröger, T.; Khatib, O.; and Knoll, A. 2013b. Robot task and motion planning with sets of convex polyhedra. In *RSS Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*.

Gaschler, A.; Petrick, R.; Kröger, T.; Knoll, A.; and Khatib, O. 2013c. Robot task planning with contingencies for run-time sensing. In *ICRA Workshop on Combining Task and Motion Planning*.

Giuliani, M.; Petrick, R.; Foster, M. E.; Gaschler, A.; Isard, A.; Pateraki, M.; and Sigalas, M. 2013. Comparing task-based and socially intelligent behaviour in a robot bartender. In *Proceedings of the International Conference on Multimodal Interaction (ICMI)*.

Kaelbling, L. P., and Lozano-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *IEEE International Conference on Robotics and Automation (ICRA)*, 1470–1477.

Kaelbling, L. P., and Lozano-Pérez, T. 2013. Integrated task and motion planning in belief space. *International Journal of Robotics Research* 32(9–10):1194–1227.

Kress-Gazit, H., and Pappas, G. 2008. Automatically synthesizing a planning and control subsystem for the darpa urban challenge. In *Automation Science and Engineering, 2008. CASE 2008. IEEE International Conference on*, 766–771.

Laborie, P. 2003. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence* 143:151–188.

Lozano-Pérez, T.; Jones, J.; Mazer, E.; and O'Donnell, P. 1989. Task-level planning of pick-and-place robot motions. *Computer* 22(3):21–29.

Newell, A. 1982. The Knowledge Level. *Artificial Intelligence* 18(1):87–127.

Nilsson, N. 1984. Shakey the robot. Technical Report 323, AI Center, SRI International.

Pednault, E. P. D. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 324–332.

Petrick, R., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, 212–221.

Petrick, R., and Bacchus, F. 2004. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2–11.

Petrick, R.; Kraft, D.; Krüger, N.; and Steedman, M. 2009. Combining cognitive vision, knowledge-level planning with sensing, and execution monitoring for effective robot control. In *ICAPS Workshop on Planning and Plan Execution for Real-World Systems*, 58–65.

Petrick, R. 2011. An extension of knowledge-level planning to interval-valued functions. In *AAAI 2011 Workshop on Generalized Planning*.

Plaku, E., and Hager, G. 2010. Sampling-based motion planning with symbolic, geometric, and differential constraints. In *IEEE Int. Conference on Robotics and Automation*, 5002–5008.

Poggioni, V.; Milani, A.; and Baioletti, M. 2003. Managing interval resources in automated planning. *Journal of Information Theories and Applications* 10:211–218.

Rickert, M. 2011. *Efficient Motion Planning for Intuitive Task Execution in Modular Manipulation Systems*. Dissertation, Technische Universität München.

van Eijck, J. 2013. Elements of epistemic crypto logic. Slides from a talk at the LogiCIC Workshop, Amsterdam.

Zacharias, F.; Borst, C.; and Hirzinger, G. 2006. Bridging the gap between task planning and path planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 4490–4495.