Extending Knowledge-Level Planning with Sensing for Robot Task Planning

Ron Petrick
University of Edinburgh, Edinburgh, Scotland, United Kingdom

Andre Gaschler
fortiss GmbH, An-Institut Technische Universität München, Munich, Germany

UK Planning and Scheduling Special Interest Group (UK PlanSIG)
Darlington, England, United Kingdom
15 December 2014
Robot task planning

• A robot operating in a real-world domain often needs to do so with incomplete or uncertain information about the state of the world.

• A robot with the ability to sense the world can gather information to generate plans with contingencies, allowing it to reason about the outcome of sensed data.

• Contingent planning with incomplete information and sensing provides a promising approach.

• An added complication for robot task planning: bridging the gap between geometric and symbolic representations.
  – Robot level: continuous representations for modelling properties like joint angles and spatial coordinates.
  – Planning level: discrete representations in logic-like languages.

• This work: combine domain independent, high-level symbolic planning with low-level geometric and motion planning, with an application to real-world robotics domains.
Application focus

- Task-oriented domains requiring world and/or agent interaction.
- Support for different robot embodiments.
- Reusable, domain-independent planning based on the PKS planner (Petrick and Bacchus 2002, 2004).

⇒ This talk: extensions to PKS in support of robotic task planning.
PKS: Planning with Knowledge and Sensing

- We use an extended version of the PKS planner (Petrick and Bacchus 2002, 2004), a knowledge-level contingent planner that builds plans based on the planner’s knowledge state.

- PKS uses an extended STRIPS-style representation, based collection of five databases, each of which is restricted to a particular type of knowledge: $K_f$, $K_v$, $K_w$, $K_x$, $LCW$.

- The contents of the databases ($DB$) have a fixed formal translation to formulae in a modal logic of knowledge which formally defines the planner’s knowledge state ($KB$).

- Actions are defined in terms of the changes they make to the planner’s knowledge state (i.e., the databases), rather than the world state.

- Plans are generated by forward search: actions update $DB$ $\Rightarrow$ update $KB$. 
Knowledge in PKS

- $K_f$: knowledge of positive and negative facts (but not closed world!)

\[ p(c) \quad \neg q(b, c) \quad f(a) = c \quad g(b, c) \neq d \]

- $K_w$: knowledge of binary sensing effects

\[ \phi \in K_w : \text{the planner knows whether } \phi \]

- $K_v$: knowledge of function values, multi-valued sensing effects

\[ f \in K_v : \text{the planner knows the value of } f \]

- $K_x$: exclusive-or knowledge

\[ (l_1 | l_2 | \ldots | l_n) \in K_x : \text{exactly one of the } l_i \text{ must be true} \]

- $LCW$: local closed world information (Etzioni et al. 1994)
Actions in PKS

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

- Actions capture the changes they make to the planner’s knowledge state, modelled using a symbolic, logic-like language.

- Sensing actions are encoded by effects that update the planner’s $K_w$ or $K_v$ databases.
Contingent planning and robot tasks

action senseWeight(?o : object)
  preconds:
  !Kw(isSpillable(?o)) &
  K(isGrasped(?o))
  effects:
  add(Kw, isSpillable(?o))

• Contingent plans in PKS are built from $K_w$ and $K_v$ knowledge.

• **Example:** the $K_w$ effect in senseWeight allows the planner to introduce a binary branch in the plan: along one branch isSpillable(?o) is known; along the other $\neg$isSpillable(?o) is known.

• Contingent planning is an old technique in the planning community but is still not widely used in the robotics community.

• Even simple contingent planning can produce some interesting behaviour, e.g., active sensing.
Example 1: Force Sensing domain

- **Task**: remove cans to a special location using weight-dependant grasps.
A contingent planning solution

action senseWeight(?o : object)

action transfer(?o : object)

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

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

action ungrasp(?o : object)
    preconds:
        K(isGrasped(?o)) &
        K(isRemoved(?o))
    effects:
        add(Kf, !isGrasped(?o)),
        add(Kf, emptyGripper)

1. grasp(can1);
2. senseWeight(can1);
3. branch(isSpillable(can1))
4. K+:
5. transferUpright(can1);
6. ungrasp(can1);
7. grasp(can2);
8. senseWeight(can2);
9. branch(isSpillable(can2))
10. K+:
11. transferUpright(can2);
12. ungrasp(can2).
13. K-:
14. transfer(can2);
15. ungrasp(can2).
16. K-:
17. transfer(can1);
18. ungrasp(can1);
19. grasp(can2);
20. senseWeight(can2);
21. branch(isSpillable(can2))
22. K+:
23. transferUpright(can2);
24. ungrasp(can2).
25. K-:
26. transfer(can2);
27. ungrasp(can2).
Example plan execution

A: grasps can1
   senses weight of can1
   branches on isSpillable(can1)

B: transfers can1 to upright

C: ungrasps can1

D: grasps can2
   senses weight of can2
   branches on ¬isSpillable(can1)

E: transfers can2

F: ungrasps can2
Video 1

http://www.youtube.com/watch?v=7l2NP3l9_iY
Planning with external reasoning

action grasp(?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)))

• PKS has been extended to support calls to external procedures, to help evaluate preconditions and effects, allowing optimised or special purpose libraries to be integrated with the planner:

  extern(proc(\bar{x}))

• Similar to semantic attachments and related techniques, e.g.,
  (Dornhege et al. 2009; Erdem et al. 2011; Eiter et al. 2006).
Planning with external reasoning...

• An optimised form of `extern` allows the returned values to be cached for fast retrieval on subsequent calls:

\[
\text{extern}^*(\text{proc}(\vec{x}))
\]

• Example:

\[
K(\text{extern}(\text{isReachable}(?x)))
\]

versus

\[
\text{add}(Kf, \text{weight} = \text{extern}^*(\text{objWeight}(?x)))
\]

• Advantage: don’t need to model certain types of complex reasoning in the planning representation language.

• Disadvantage: current implementation is integrated within the plan generation process and is blocking.

• Application focus: integration with low-level robot motion planning.
Example 2: Bimanual domain

- **Task**: recognise and remove empty bottles from the table to a dishwasher location.
A solution with external geometric reasoning

preconds:
  K(?l = getobjectLocation(?o)) &
  K(handEmpty(?r)) &
  K(extern(isReachable(?l, ?r)))
effects:
  del(Kf, ?l = getobjectLocation(?o)),
  del(Kf, handEmpty(?r)),
  add(Kf, inHand(?o, ?r))

action senseIfEmpty(?o:obj)
preconds:
  !Kw(isEmptyBottle(?o))
effects:
  add(Kw, isEmptyBottle(?o))

preconds:
  K(inHand(?o, ?r)) &
  K(extern(isReachable(?l, ?r)))
effects:
  del(Kf, inHand(?o, ?r)),
  add(Kf, ?l = getobjectLoc(?o)),
  add(Kf, handEmpty(?r))

1. senseIfEmpty(bottle0);
2. senseIfEmpty(bottle1);
3. senseIfEmpty(bottle2);
4. senseIfEmpty(bottle3);
5. branch(isEmptyBottle(bottle0))
6. K+
7. branch(isEmptyBottle(bottle1))
8. K+:
9. K-:
10. branch(isEmptyBottle(bottle2))
11. K+
12. branch(isEmptyBottle(bottle3))
13. K+:
14. K-:
15. pickUp(left, bottle0, 10);
16. putDown(left, bottle0, 15);
17. pickUp(right, bottle2, 12);
18. putDown(right, bottle2, dishwasher);
19. pickUp(right, bottle0, 15);
20. putDown(right, bottle0, dishwasher).
21. K-:
22. K-:
Robotics side: 3D volume intermediate representation

• **3D geometric volumes** act as a natural intermediate representation for bridging motion planning and high-level task planning.

• We use an efficient swept volume computation with sets of convex bodies (Gaschler et al. 2013a,b,c), building on the approach of (Mamou and Ghorbel 2009).
Example plan execution

A senseIfEmpty(bottle0)
... senseIfEmpty(bottle3)

B pickUp(robotleft, bottle0, loc0)

C putDown(robotleft, bottle0, loc5)

D pickUp(robotright, bottle2, loc2)

E putDown(robotright, bottle2, dishwasher)

F pickUp(robotright, bottle0, loc5)

G putDown(robotright, bottle0, dishwasher)
Video 2

http://www.youtube.com/watch?v=yMmZkhHr8ss
Implementation: a generic planning API

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

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

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

• Integration on real robot platforms requires the planner to be part of a larger system of interacting components.

• To facilitate integration across different platforms, we created an abstract software-level interface to typical planning services.

• Interfaces: C++, Internet Communications Engine (ICE), ROS (planned).
• The extended version of PKS has also been integrated within a larger framework for robot task planning called KVP (Gaschler et al. 2013a,b,c).
• Robot task planning in a multiagent warehouse environment (Crosby and Petrick 2014).
Conclusions

• Knowledge-level planning with sensing continues to be a promising approach for the robot task planning domains we consider.

• Current work: scalability experiments to determine the cost/limits of our approach; applications to new robotics domains.

• External reasoning is a simple, yet powerful, mechanism for enhancing the basic operation of the planner.
  
  **Future work:** timeouts, limited state queries within extern calls.

• We are also working on improving the software side of our tools to facilitate integration on diverse robot platforms.

• Also in the paper: Interval-Valued Fluents.

• For more information on the project visit the JAMES website at http://james-project.eu/ and the STAMINA website at http://stamina-robot.eu/.
References


