DELIVERABLE D6.2

EMBODIMENT FOR SOCIAL INTERACTION

Manuel Giuliani (FORTISS), Andre Gaschler (FORTISS), Sören Jentzsch (FORTISS)

**Beneficiaries:** FORTISS (lead)

**Workpackage:** WP6: Social Robotics and Embodiment

**Description:** This deliverable reports on the complete setup of the two robots that were used in the JAMES project. This includes a description of the robot hardware and software, as well as a description of the robots' manipulation skills. The deliverable also contains a summary of the experiments conducted to measure the naturalness of the robot movements. Finally, the report contains an overview for 3 attached papers, which give more details about the work in robot path planning that has been carried out.

**Version:** Final

**Nature:** Report (R)

**Dissemination level:** Public (PU)

**Pages:** 32

**Date:** 2014-01-31
Contents

1 Summary .................................................................................. 3

2 Integrated Evaluation Robot .................................................... 3
   2.1 Hardware ........................................................................ 3
   2.2 Software .......................................................................... 3
   2.3 Manipulation Skills .......................................................... 4

3 Ghost-in-the-Machine Robot ...................................................... 5
   3.1 Hardware ........................................................................ 5
   3.2 Software .......................................................................... 5
   3.3 Manipulation Skills .......................................................... 8

4 Evaluation .................................................................................. 10
   4.1 Experiment Setup ............................................................ 10
   4.2 Questionnaire ................................................................. 10
   4.3 Participants ...................................................................... 10
   4.4 Results ............................................................................ 11
   4.5 Discussion ...................................................................... 11

A Included papers ........................................................................ 15

Robot Task Planning with Contingencies for Run-time Sensing ......... 15
Robot Task and Motion Planning with Sets of Convex Polyhedra ......... 20
KVP: A Knowledge of Volumes Approach to Robot Task Planning ......... 26
1 Summary

In the JAMES project, we developed two different robot systems: we used the first robot, which is shown in Figure 1, for the user studies in which we integrated and tested the software of all project partners; we used the second robot, which is shown in Figure 2, for the Ghost-in-the-machine (GiM) studies. In this report, we describe the technical details for both of these systems, for the integrated evaluation robot in Section 2 and for the GiM robot in Section 3. Both sections contain details about the robots' hardware, software, and manipulation skills. To conclude the report, we summarise in Section 4 the answers that the participants of two integrated evaluations gave to rate the robot movements and how safe they felt during the experiment. Please note, this deliverable also contains three attached publications, which are related to the manipulation skills of the integrated evaluation robot. Section 2.3 gives a summary for these papers.

2 Integrated Evaluation Robot

2.1 Hardware

Figure 1 shows a picture of the robot we used for the integrated evaluations. It consists of two 6-degrees-of-freedom industrial manipulator arms by Mitsubishi with humanoid hands by Meka Robotics, mounted to resemble human arms. Sitting on the main robot torso is an animatronic talking head—a Philips iCat—capable of producing facial expressions, rigid head motion, and synthesised speech. The robot is equipped with two stereo cameras and two Kinect sensors: we use the depth sensor of one of the Kinects to support stereo camera-based vision processing, and the microphone array of the other Kinect for automatic speech recognition. For speech synthesis, the robot is equipped with a stereo loudspeaker system.

![Figure 1: Robot used for integrated evaluations.](image)

2.2 Software

We described the initial version of the software for the integrated evaluation robot in detail in JAMES Deliverable 6.1. Therefore, this section gives details on the extensions that we implemented since the first report.

As described above, the integrated evaluation robot consists of two industrial manipulators and two humanoid hands, which require different real-time connections. Therefore, we implemented two separate software components that communicate over the Internet Communications Engine (Ice). The software components are installed on two separate Linux real-time operating systems: one for controlling the two Mitsubishi robots over TCP at 140 Hz, and one controlling the Meka Robotics hands over Ethercat at 1 kHz. From a high level, all robot actions
are made available through Ice services and can conveniently be called as remote procedure from all other components of the integrated system.

The most important robot action is grasping, which is used to let the robot pick up drinks and place them on arbitrary locations on the bar. To provide these functions, our internal grasp planner works with arbitrary locations at run time, allowing drinks to be served to each customer’s location. To calculate possible grasp configurations at run time, we use a numerical inverse kinematics algorithm that includes task space constraints. This allows us to define a degree-of-freedom around the axis of rotation of a bottle to obtain the full null space of solutions. In this null space, we find the solution with maximum distance from robot joint boundaries in order to achieve a good posture far away from the workspace boundaries. We will make this new feature publicly available as part of the next release of the Robotics Library1.

Once grasp planning finds an optimal configuration for picking or placing an object, we need to generate a fast motion trajectory that takes advantage of the dynamics of the physical robot. For this, we interpolate a quintic polynomial in joint space and optimize the sampling time in order to minimize its duration [16]. This type of trajectory fulfills higher-order dynamics limits and therefore allows very smooth robot motions. After trajectory generation, a real-time collision detection layer verifies that the run-time planned motion is free of collisions. To allow real-time performance, the robot geometry is stored as a small set of convex polyhedra, including a small offset for safety, which is generated by our convex decomposition routines [6]. Using this representation, run-time collision checking only needs to process convex-convex checks, which are of linear complexity in the number of vertices. In our system, all collision checks can be run in less than a millisecond. Because of this, all robot trajectories are verified within the hardware real-time control loop and a safe distance to all static obstacles is maintained, even if the robot is commanded to move to an invalid position.

In response to messages from the output planner, the animatronic robot head also generates synthesized embodied speech together with facial expressions and gestures such as nods. In addition, the robot head is able to look directly at defined positions, and can be turned towards customers while the robot is talking to them; the coordinates are derived from the locations reported by the vision system via the state manager.

2.3 Manipulation Skills

As described above, the main robot manipulation skill that we used in the integrated evaluations was the handing over of drinks to humans. However, we also implemented, in cooperation with Ron Petrick from UEDIN, manipulation skills, which enable the JAMES robot to clear the bar of empty bottles. This section summarises the main results of this collaborative work in robot task planning. Detailed descriptions of this work can be found in two workshop papers [8, 7] and a full conference paper [6], which we attached to this report.

Task planning is a key capability for intelligent service robots. While modern robot technology has made great progress towards reliable robot platforms and sensors, and algorithms for motion planning and computer vision are available as easy-to-use libraries, robot tasks in social settings require complex reasoning about actions and geometry. The underlying idea of our task planning approach is to reason on a similarly high level of abstraction as humans do, as a social robot needs to understand some human concepts of geometry and action in order to successfully interact with them.

Our knowledge of volumes approach to robot task planning, abbreviated KVP, can reason about both symbolic actions and geometric preconditions and effects. On the symbolic reasoning level, we achieve this by knowledge-level task planning, reasoning on the planner’s knowledge state, rather than directly representing the world state. This approach is implemented by the knowledge-based symbolic planner PKS (Planning with Knowledge and Sensing) [14, 15]. On the geometric level, we can evaluate geometric preconditions and effects efficiently using the representation of sets of convex polyhedra.

While our KVP approach is similar to the multi-modal motion planning approach described in Section 3.3 in that both approaches solve tasks involving multiple objects and actions, KVP is fundamentally different in its

1http://sourceforge.net/apps/mediawiki/roblib/
use of a general-purpose symbolic planner. Using the PKS planner, it can solve arbitrary symbolic tasks rather than only robot manipulation, including perception with sensors and interaction management with several human participants. For more details, please refer to Appendices A to C.

3 Ghost-in-the-Machine Robot

3.1 Hardware

Figure 2: Robot used for ghost-in-the-machine evaluations.

Figure 2 shows a picture of the robot we used for the GiM studies. It consists of a 2-degrees-of-freedom torso and a 7-degrees-of-freedom compliant robotic arm that is equipped with a humanoid hand. All of the robotics components were built by Meka Robotics\(^2\). An Apple iPad is mounted on top of the robot torso, to display eye gazes and lip-synchronised speech. As input sensor, the robot is equipped with a Microsoft Kinect, which we are using for person tracking and speech recognition. For speech synthesis, the robot is equipped with a stereo loudspeaker system.

3.2 Software

The low-level software component for robot control runs on a real-time Linux operating system, and interfaces the robot through an Ethercat bus at a control frequency of 1 kHz. For the GiM studies, the humanoid robot only needs to pick up and place bottles from a set of pre-defined locations, and perform simple scripted gestures such as turning its body in the direction of a customer. Because of this, we could keep trajectory generation rather simple and interpolate positions from a list of manually optimized waypoints. These waypoints are chosen to generate smooth and natural-looking motions that keep a safe distance from possible collisions with static objects or with the robot itself.

In contrast to the industrial manipulator used in the integrated evaluations, the Meka robot is driven by series-elastic actuators with torque control, rather than position control. The control scheme in use automatically applies torque levels to compensate for gravity and coriolis forces, and can ideally perform motions to a goal position applying only minimal external forces. As the robot will exert almost no force in case of a collision with a human participant, this torque control scheme is very well suited for safe human-robot interaction experiments. The robot control software component is encapsulated in the Ice middleware service; all higher-level software components can call remote procedures to trigger robot actions to serve different types of drinks or perform gestures, as shown in Figure 5.

For the execution of GiM studies, we implemented a graphical user interface (GUI), which on the one hand shows the data of the robot’s input sensors to experiment participants and on the other hand can be used by the

\(^2\)http://mekabot.com/
participants to control the robot. All text on the GUI is in German, since the experiment participants of the GiM experiment were German native speakers. Figure 3 shows a picture of the GUI.

The GUI is separated in two parts: the upper part shows the robot’s sensor data, the lower part is used for controlling the robot. From top to bottom, the GUI contains the following elements:

- An area that shows the skeleton data from person tracking.
- An area that shows a dot indicating the sound direction of detected speech.
- A list of speech recognition results. This list can be toggled to show either just the top hypothesis or an n-best list of speech recognition results.
- A set of buttons that can be used for drink serving. Participants can choose to let the robot serve either water or a coke on the left, middle or right side of the bar.
- An area that shows a stylised version of the robot that gives visual feedback of the current body orientation and arm position of the robot.
- A set of buttons for controlling the robot’s body orientation and eye gaze. The robot can be turned to the left, right, or middle, and the robot can look to detected humans or to no particular person.
- A set of buttons that let the robot say predefined verbal utterances via speech synthesis. This area also contains a text field in which participants can type free text the robot should say.
We are using the Microsoft Kinect SDK\(^3\) to determine the input for person tracking (upper body skeleton data), sound direction (angle and confidence value), and speech recognition (spoken utterances and confidence values). The GUI can display the data from sound direction and speech recognition in two different settings: when set to setting \textit{certain}, the GUI displays the dot representing sound direction as well as the spoken utterances in black; when set to setting \textit{uncertain}, the GUI shows these inputs in different shades of grey to represent the confidence values by the input modules. The higher the confidence value of sound direction or speech recognition input, the darker the GUI displays these inputs. Additionally, the dot representation for sound direction can be completely switched off. For speech recognition, recognition results can either be displayed with only the top result or as n-best list (with n set to 7).

\begin{verbatim}
module input {
[...]
struct Person {
    int trackingID;
    int enrollmentIndex;
    Joint centerJoint;
    JointList joints;
    ClippedState state;
};
sequence<Person> PersonList;
interface PersonTrackingListener {
    void sendFrame(PersonList persons);
};
struct SpeechHypothesis {
    string speech;
    double speechConfidence;
};
sequence<SpeechHypothesis> SpeechHypothesisList;
interface SpeechRecognitionListener {
    void sendSpeechWithDirection(SpeechHypothesisList speechInputs,
    double angle, double confidence);
[...]
};
}
\end{verbatim}

Figure 4: Extracts of the Internet Communications Engine (Ice) interfaces for person tracking, sound direction, and speech recognition of the ghost-in-the-machine robot.

We used Ice for communication of the GUI with the input and output modules of the robot. Figure 4 shows the parts of the interfaces for the input modules person tracking and speech recognition. Person tracking uses the method \textit{sendFrame()} to submit a list of joint positions for each person that the Kinect sensor can detect in front of the robot. Information for speech recognition and sound direction is submitted by method \textit{sendSpeechWithDirection()}.

Figure 5 shows parts of the Ice interfaces for the output modules of the robot, which control the robot movements and speech synthesis. For robot motion control there are two methods—\textit{give()} and \textit{turnTorso()}—to hand over drinks from the robot on different positions on the bar and to turn the robot's torso, respectively. The GUI uses the methods \textit{getRobotRotation()} and \textit{getHandPosition()} to display the robot feedback to the experiment participants. The speech synthesis has a method \textit{say()}, which is used to submit strings for output generation.

\(^3\)http://www.microsoft.com/en-us/kinectforwindows/
Figure 5: Extracts of the Internet Communications Engine (Ice) interfaces for robot control and speech synthesis of the ghost-in-the-machine robot.

3.3 Manipulation Skills

For intelligent robots to solve real-world tasks, the problem is not only to plan collision-free motion paths, but rather to plan for picking, pushing, sliding, and many other diverse manipulation actions in a complex world of multiple movable objects. Thus, we implemented algorithms, which are able to plan for manipulation and follow the multi-modal nature induced by these actions. We extended basic sampling-based motion planning to integrate Diverse Action Manipulation (DAMA) as first examined by [1], and show that based on the Rapidly-exploring Random Tree (RRT), we can then solve DAMA scenarios of various kinds on various robot platforms.

Major work was focused on solving DAMA problems with the Meka robot used for the GiM studies (Figure 2). In addition to robot motion only, the following diverse action manipulations are available: picking up an object, transferring the rigidly attached object, pushing an object with the interior and pushing an object with the exterior surface of the hand. These manipulation actions induce various hand poses, including grasping and both pushing poses.

Figure 6 illustrates a user defined DAMA scenario, in which the robot has to rearrange three bottles, thereby starting and finishing the work at the position in which the arm is lowered. Considering only translations for objects, the search space for this scenario is 18-dimensional, consisting of important subspaces induced by the constraints of manipulation actions defined for this configuration space. This scenario features multiple support surfaces, and at least all manipulation actions are highly recommended for solving this task, if not required. We then executed this scenario on the GiM robot to illustrate the feasibility of the whole process from modelling to planning, up to execution. Figure 7 shows impressions from this implementation, which are taken from a video\(^4\).

For this, we generated a trajectory out of the solution path and had to adjust several parameters of the robot dynamic, including the payload of the end-effector when grasping an object.

In addition to the flat algorithm that solves DAMA problems in one step, we developed a hierarchical approach, which exploits the multi-modal nature of these DAMA problems, determines an object path and then plans for each action separately. Results on the scenario presented in Figure 6 reveal that in 78% of the cases, a single random run of the flat algorithm finds a solution in under 15 minutes. In contrast, only 46% of the runs of the hierarchical version were able to find a solution given 5 minutes for each action. However, on average on a Dell Latitude E6520 notebook with an Intel Core i5-2520M processor, the hierarchical algorithm finds a solution after 131.8 seconds, compared to 514.7 seconds of the flat version. Nevertheless, computing inverse kinematics and

---

\(^4\)The full video is available at [http://youtu.be/n9ZyAc-tMBY](http://youtu.be/n9ZyAc-tMBY)
Figure 6: A user defined DAMA scenario, illustrating the problem description (a) and a corresponding randomly picked solution path with various robot poses (b). The robot has to rearrange three bottles all on the lower support surface as indicated by the arrows, thereby starting and finishing at the same illustrated position. Edges drawn in red correspond to robot manipulation actions being applied to objects, the blue ones to robot motion only. Vertices of the solution path are drawn as small black dots.

Figure 7: A solution path for the DAMA scenario examined in Figure 6 now executed in the real environment. After picking up and transferring the middle bottle to its goal position, the robot reaches out to push the leftmost bottle towards him. He then picks it up and transfers it to its goal. Finally, the rightmost bottle moves to its goal position by being pushed with the exterior surface of the hand. In the end, the robot transits back to its starting configuration. The overall execution time is about 112 seconds without path smoothing. The full video is available at [http://youtu.be/n9ZyAc-tMBY](http://youtu.be/n9ZyAc-tMBY).

searching for nearest neighbours consumes 92.8% (flat) and 84.3% (hierarchical) of the overall computation time.
4 Evaluation

The robots of the JAMES project were used in several human-robot interaction studies [5, 11, 9, 4] as well as in one of the GiM experiments [12]. In the studies presented in Foster et al. [5] and Giuliani et al. [9] the experiment participants rated their subjective impressions of the integrated evaluation robot. In the following sections, we summarise the results of the participants ratings.

4.1 Experiment Setup

The two studies described in [5] and [9] both used the same scenario: the experimenters instructed the experiment participants to walk up to the integrated evaluation robot and to order a drink. The participants could choose between three different drinks and were asked to use specific drink names when referring to one of the drinks. Participants got no additional constraints to what they could say or do.

The purpose of the first experiment [5] was to test, whether the robot will be able to interact with more than one human. During the interaction, the robot was able to track two humans in front of the bar and it was able to recognise which of the humans was bidding for attention to order a drink of the robot. The second experiment [9] compared the user ratings of participants that interacted with the robot when it showed purely task-based behaviour (i.e. it just took orders and served drinks) and participants that interacted with the robot when it showed social behaviour in addition to task-based actions. In the following sections we will refer to the first and second study as the multi-party interaction experiment and the social interaction experiment, respectively. For more details on the two studies please refer to [5] and [9].

4.2 Questionnaire

In both experiments, participants rated how they perceived the robot's actions after they interacted with it. We measured participants' subjective experiences via a questionnaire based on the GODSPEED questionnaire series [2]. The GODSPEED questionnaire is designed to be a standard user measurement tool for human-robot interaction, and includes items asking the participant to assess the robot on five scales: anthropomorphism (five items), animacy (six items), likeability (six items), perceived intelligence (five items), and perceived safety (three items). Participants gave their responses on semantic differential scales, the multi-party interaction experiment used a five-point scale while the social interaction experiment used a six-point scale. In the social interaction experiment, participants could also choose “no answer” for each question, when they could not or did not want to respond. An example for one of the questionnaire items is given in Figure 8.

![Example for an item of the GODSPEED questionnaire.](image)

Figure 8: Example for an item of the GODSPEED questionnaire.

4.3 Participants

31 participants (22 male) took part in the multi-party experiment. The mean age of the participants was 27.9 (range 21–50), and their mean self-rating of experience with human-robot interaction systems was 2.29 on a scale of 1–5.

40 participants (28 male) took part in the interaction experiment. The mean age of the participants was 27.9 (range 16–50), and their mean self-rating of experience with human-robot interaction systems was 2.50 on a scale of 1–5.
4.4 Results

Table 1 shows the results of the ratings in the questionnaires. We scaled the data from both questionnaires to a scale from -2 to 2, in order to make the results from the different scales comparable. Furthermore, we only compared the values from the questionnaire statements, which clearly refer either to the robot’s movements or to the perceived safety of the participants. The table shows the mean values and standard deviation for the statements in both experiments and the results of a Mann-Whitney U test, which computes the statistical significance of the value comparison.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Multi-party</th>
<th>Social</th>
<th>Mann-Whitney</th>
</tr>
</thead>
<tbody>
<tr>
<td>moving rigidly–moving elegantly</td>
<td>-0.29 (1.01)</td>
<td>-0.88 (1.13)</td>
<td>p = 0.002</td>
</tr>
<tr>
<td>mechanical–organic</td>
<td>-0.48 (1.15)</td>
<td>-0.80 (1.11)</td>
<td>p = 0.165</td>
</tr>
<tr>
<td>anxious–relaxed</td>
<td>0.81 (0.83)</td>
<td>0.54 (1.14)</td>
<td>p = 0.576</td>
</tr>
<tr>
<td>agitated–calm</td>
<td>0.48 (1.23)</td>
<td>0.41 (1.21)</td>
<td>p = 0.836</td>
</tr>
<tr>
<td>surprised–quiescent</td>
<td>0.39 (1.23)</td>
<td>0.24 (1.26)</td>
<td>p = 0.841</td>
</tr>
</tbody>
</table>

Table 1: Results of ratings for relevant questionnaire items in the multi-party and social interaction studies. Columns show the mean values and standard deviations of the ratings, as well as the results of a Mann-Whitney significance test.

The average ratings for the two statements moving rigidly–moving elegantly and mechanical–organic, which rate the robot movements, are in the negative area. The average ratings for all statements of the perceived safety category are in the positive area. The standard deviations for the two statements agitated–calm and surprised–quiescent are quite high. We measured a statistically significant difference for the comparison of ratings between experiment only for statement moving rigidly–moving elegantly.

4.5 Discussion

The experiment participants ratings show that they perceived the movements of the integrated evaluation robot as machine-like and not organic. This does not have to be interpreted as a negative result. Saygin et al. [17] conducted a functional magnetic resonance imaging (fMRI) study in which they showed body movements of humans, robots, and androids to participants. The results of this study were that the human action perception system responds differently to the stimuli, when between appearance and body movements match (e.g., human appearance and human movements) or when there is a mismatch between appearance and body movements (e.g., human appearance and robotic movements). The authors argue that this could explain the uncanny valley [13] phenomenon, but on the other hand also suggests that robots with a robotic appearance, as for example the JAMES integrated evaluation robot, should move machine-like.

In our view, the positive ratings for the statements in the perceived safety category are a result of our implementation of smooth robot motions when handing over drinks, as described in Section 2.2. Huber et al. [10] showed\(^5\) that humans who watch the movements of a robot, feel safer when the robot uses a human-like minimum jerk trajectory for motion generation. Huber et al. concluded that the predictability of the motions leads to an increased feeling of safety for the humans. This aspect of legibility of robot motions has also been the focus of recent work by Dragan and Srinivasa [3]. We did not use a minimum jerk profile for motion generation in our implementation, however, the usage of quintic polynomials in joint space and subsequent optimisation of sampling time leads to a smooth robot motion, which might have the same effect as the one reported by Huber et al.

The significant difference in the ratings for statement moving rigidly–moving elegantly is surprising. The robot movements that we used in the two experiments were similar and differed in only two ways:

\(^5\)In this study, the authors used the same robot that we are using in JAMES. The experiment was conducted in a previous European project called Joint Action Science and Technology (JAST)
• **Head movements.** In the multi-party interaction experiment, the robot looked straight to the front when handing over a drink; in the social interaction experiment, the robot looked to the drink first, picked it up, looked up to the human and then handed over the drink.

• **Placing location.** In the multi-party interaction experiment, the robot always placed bottles at a fixed location on the bar; in the social interaction experiment, the robot was able to place bottles in front of the human who ordered the drink.

This means, the ratings for statement *moving rigidly–moving elegantly* should tend towards *moving elegantly* in the case of the social interaction experiment, at least more than in the multi-party interaction experiment. However, the results suggest the contrary. An explanation for this could be that the ratings by the users were influenced not only by the robot movements, but by other factors, which is reasonable to assume when considering the differences in experiment design.
References


A Included papers

The following papers are included in this deliverable:


Abstract: In this work, we present a general approach to task planning based on contingent planning and runtime sensing, which forms part of a robot task planning framework called KVP. Using the general-purpose PKS planner, we model information-gathering actions at plan time that have multiple possible outcomes at run time. As a result, perception and sensing arise as necessary preconditions for manipulation, rather than being hard-coded as a task itself. We demonstrate the effectiveness of our approach on two simple scenarios covering visual and force sensing, and discuss its applicability to more general tasks in automation and mobile manipulation, involving arbitrary numbers of sensors and manipulators.


Abstract: Geometric volumes can be used as an intermediate representation for bridging the gap between task planning, with its symbolic preconditions and effects, and motion planning, with its continuous-space geometry. In this work, we use sets of convex polyhedra to represent the boundaries of objects, robot manipulators, and swept volumes of robot motions. We apply efficient algorithms for convex decomposition, conservative swept volume approximation and collision detection, and integrate these methods into our existing “knowledge of volumes” approach to robot task planning called KVP. We demonstrate and evaluate our approach in several task planning scenarios, including a bimanual robot platform.


Abstract: In this work, we present a general approach to task planning based on contingent planning and runtime sensing, which forms part of a robot task planning framework called KVP. Using the general-purpose PKS planner, we model information-gathering actions at plan time that have multiple possible outcomes at run time. As a result, perception and sensing arise as necessary preconditions for manipulation, rather than being hard-coded as a task itself. We demonstrate the effectiveness of our approach on two simple scenarios covering visual and force sensing, and discuss its applicability to more general tasks in automation and mobile manipulation, involving arbitrary numbers of sensors and manipulators.
Robot Task Planning with Contingencies for Run-time Sensing

Andre Gaschler, Ronald P. A. Petrick, Torsten Kröger, Alois Knoll and Oussama Khatib

Abstract—In this work, we present a general approach to task planning based on contingent planning and run-time sensing, which forms part of a robot task planning framework called KVP. Using the general-purpose PKS planner, we model information-gathering actions at plan time that have multiple possible outcomes at run time. As a result, perception and sensing arise as necessary preconditions for manipulation, rather than being hard-coded as a task itself. We demonstrate the effectiveness of our approach on two simple scenarios covering visual and force sensing, and discuss its applicability to more general tasks in automation and mobile manipulation, involving arbitrary numbers of sensors and manipulators.

I. INTRODUCTION

In order to model realistic environments for robot task planning, symbolic task planners need to reason about incomplete knowledge and perceptual information as provided by sensors. In order to facilitate this task and apply general purpose planning to the robotics domain, we developed the Knowledge of Volumes framework for robot task Planning (KVP), which was initially presented in [1].

KVP is guided by three principles, which make it useful for a broad range of robot task planning applications that require incomplete knowledge, real-world geometry, and multiple robots and sensors: (1) KVP uses PKS (Planning with Knowledge and Sensing) [2], [3], a general-purpose symbolic planner that operates at the knowledge level. PKS can represent known and unknown information, and model sensing actions using clear and concise domain descriptions, making it well suited for reasoning in structured, partially known environments of the kind that arise in many robot scenarios. (2) KVP is one of the first approaches to treat 3D geometric volumes as an intermediary representation between continuously-valued robot motions and discrete symbolic actions, tackling the general problem in robot task planning of bridging the gap between geometric and symbolic representations. (3) By using the intermediate representation of volumes, KVP can model continuous geometry, in contrast to arbitrary discretization, as discussed in [1].

The central contribution of this work is to apply general-purpose, contingent planning techniques to the robotics domain and demonstrate the effectiveness of this approach in two scenarios, namely a FORCE SENSING (Fig. 1) and a VISUAL SENSING (Fig. 3) scenario.

In the following, we first compare our approach with existing solutions. We then discuss our framework in Sec. II and demonstrate its effectiveness in two scenarios in Sec. III. Finally, we discuss future work and conclude in Sec. IV.

A. Related Work

Early work on robot task planning dates back to systems like Shakey [4] and Handey [5]. Since that time, the field has made significant developments, and the general problem of robot task planning has been approached from diverse areas of research, including probabilistic techniques from artificial intelligence [6], closed-world symbolic planning [7], [8], [9], formal synthesis [10], [11], and manipulation planning [12].

A recent and notable contribution is the belief space planner by Kaelbling and Lozano-Pérez [6], which models a belief space of probability distributions over states, making it robust against uncertainty and change. In contrast to belief states, our work instead relies on discrete knowledge and is designed for structured environments with incomplete information and sensing. Furthermore, while Kaelbling and Lozano-Pérez use octrees to represent swept volumes of robot motion, we use sets of convex shapes, allowing efficient collision detection in the deterministic case [1]. In both cases, sensing actions are formulated as preconditions for manipulation, rather than being hard-coded as tasks themselves.
A number of approaches also address the problem of integrating symbolic planning and motion planning. For instance, our work is in part inspired by Kaelbling and Lozano-Pérez’s earlier work on hierarchical task and motion planning in [13], borrowing the continuous geometry of swept volumes. However, while the geometric preconditions may be similar, their underlying aggressively hierarchical planning strategy differs from the knowledge-based planner we use here, which has also been used in prior work to connect robot vision and grasping with symbolic action [14]. Further approaches that integrate symbolic and geometric reasoning are presented by Cambon, Alami and Gravot [7], handling geometric preconditions and effects; Dornhege et al. [9]; and, more recently, Plaku and Hager [8], 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 either true or false. On the contrary, our approach represents knowledge in an open-world manner, which allow us to model incomplete knowledge and information-gathering actions. We elaborate on the advantages of this representation in Sec. II-B.

II. APPROACH

In our work, sensing in robot task planning is seen as a necessary precondition for manipulation and, as such, requires an integrated approach with solutions from distinct fields ranging from motion planning to formal methods. In particular, our KVP framework combines several of these techniques. Symbolic task planning, which includes information-gathering actions and contingencies, is performed by the PKS planner [2], [3], details of which are given below. Motion planning and collision detection rely heavily on the Robotics Library (RL)\(^1\) [15], with several crucial additions to swept volume computation with sets of convex bodies [1]. In order to efficiently generate these sets of convex bodies, Mamou and Ghorbel’s approximate convex decomposition algorithm [16] is applied. An overview of KVP’s component architecture is shown in Fig. 2 and its implementation is discussed in depth in [1].

A. Planning with Knowledge and Sensing

Symbolic planning in KVP relies on the general-purpose PKS planner, which constructs plans in the presence of incomplete information and sensing actions. PKS works at the knowledge-level by reasoning about how the planner’s knowledge state, rather than the world state, changes due to action. PKS works with a restricted subset of a first-order language, allowing it to support a rich representation with features such as functions and run-time variables. This approach differs from planners that work with possible worlds models or sets of worlds forming belief states.

PKS is based on a generalization of STRIPS [17]. Unlike STRIPS, which uses a single database to model the world state, PKS’s knowledge state is represented by five databases, each of which models a particular type of knowledge. Actions can modify any of these databases, which has the effect of updating the planner’s knowledge state. To ensure efficient inference, PKS restricts the type of knowledge (especially disjunctions) that it can represent in each database:

- \(K_f\): This database is like a standard STRIPS database except that both positive and negative facts are permitted and the closed-world assumption is not applied. \(K_f\) is used to model action effects that change the world. \(K_f\) can include any ground literal \(\ell\), where \(\ell \in K_f\) means “the planner knows \(\ell\).” \(K_f\) can also contain known function (in)equality mappings.
- \(K_v\): This database models the plan-time effects of sensing actions that return binary values. A formula \(\phi \in K_v\) means that at plan time, the planner knows whether \(\phi\) or \(\neg\phi\) holds, and that at run time this disjunction will be resolved. The use of \(K_v\) for robot sensing is described in detail below.
- \(K_o\): This database stores information about function values that will become known at execution-time. In particular, \(K_o\) can model the plan-time effects of sensing actions that return constants. \(K_o\) can contain any unnested function term \(f\), where \(f \in K_v\) means the planner “knows the value of \(f\)”.
- PKS also includes databases for modelling a restricted type of disjunctive information (\(K_x\)) and “local closed-world” information (\(LCW\)), and a mechanism for denoting interval-bound values which is useful for reasoning about noisy sensors. These features are not used in this paper.

PKS knowledge states can be queried in three different ways. First, simple knowledge assertions can be tested with a query \(K(\phi)\) which asks: “is a formula \(\phi\) true?” Second, a query \(K_v(\phi)\) asks whether \(\phi\) is known to be true or known to be false (i.e., does the planner “know whether \(\phi\)”?). Finally, \(K_o(t)\) asks “is the value of function \(t\) known?” The negation of the above queries can also be used.

Using this representation, symbolic actions are defined in PKS by describing their (typed) parameters, preconditions, and effects. Preconditions contain a list of queries that must evaluate as true before an action can be applied. Effects are described by a list of add and del operations, similar to STRIPS. Example PKS actions are shown below in Table I.

B. Contingency Planning for Robot Sensing

One aspect of PKS that is particularly important for robot task planning is its ability to model sensing actions that return information about the state of the world. In particular, PKS offers two databases, \(K_v\) and \(K_o\), that represent unknown information (binary or function values, respectively) that will become known at run time after the sensing actions.

\(^1\)http://roblib.sf.net/ (accessed Mar. 17, 2013)
TABLE I
FORCE SENSING SCENARIO: Example actions and goal definition

<table>
<thead>
<tr>
<th>action</th>
<th>preconds:</th>
<th>effects:</th>
<th>goal:</th>
</tr>
</thead>
<tbody>
<tr>
<td>senseWeight</td>
<td>¬K_w(isSpillable(?o)) &amp; K(isGrasped(?o))</td>
<td>add(K_w, isSpillable(?o))</td>
<td>for all K(?o:object)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(K(isRemoved(?o))</td>
</tr>
<tr>
<td>transferUpright</td>
<td>K(isSpillable(?o)) &amp; K(isGrasped(?o)) &amp; K(!isRemoved(?o))</td>
<td>add(K_f, isRemoved(?o))</td>
<td></td>
</tr>
</tbody>
</table>

TABLE II
FORCE SENSING scenario: Example solution

grasp(can1)
senseWeight(can1)
branch isSpillable(can1)
transferUpright(can1)
ungrasp(can1)
grasp(can2)
senseWeight(can2)
branch isSpillable(can2)
transferUpright(can2)
ungrasp(can2)
branch !isSpillable(can2)
transfer(can2)
ungrasp(can2)
branch !isSpillable(can1)
transfer(can1)

are actually executed in the world. Using these databases, PKS can reason about the possible outcomes of sensing actions during plan construction, by generating plans with contingencies (conditional branches).

For instance, in general, PKS builds plans by reasoning about actions in a forward-chaining manner: if the preconditions of an action are satisfied by the planner’s knowledge state, then the effects of that action can be applied to produce a new knowledge state. If a formula $\phi$ is in the $K_w$ database, denoting the fact that $\phi$ or $\neg\phi$ will become known at run time, then a pair of conditional branches can be added to a plan, with one branch assuming $\phi$ is true and the other branch assuming $\neg\phi$ is true. (The construction of contingent plans using $K_w$ is similar.) Planning then continues along each branch until the goal conditions (a set of queries) are satisfied. A sample plan with branches is shown in Table II, and described in greater detail below.

III. EVALUATION

We now demonstrate and evaluate our approach in two simple scenarios. In the FORCE SENSING scenario (Fig. 1), a compliant robot manipulator has the ability to grasp, lift, and transfer beverage containers which are located on a table. When a container is lifted, the robot can sense its weight and, from this, reason whether the drink must be held upright in order to prevent spilling. The goal is to transfer all containers to a second table, and the robot may hold its gripper upright during these motions, if required. In order to keep this scenario simple, the location of all objects are known and no sensing except force sensing is available.

The second scenario is a demonstration of VISUAL SENSING (Fig. 3) using a bimanual robot 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. The goal is to “clean up” all empty bottles and remove them to a certain “dishwasher” location. To do this, the robot must move objects that are only accessible by its left arm to a location that its right arm can reach, a behavior which arises purely from symbolic planning.

In the following sections, we discuss the symbolic domain definitions of both scenarios. We provide an example solution for the first scenario, using conditional branches, and discuss aspects of the plan-based solution for the second scenario.

A. Force Sensing Scenario

Table I shows two PKS actions in the FORCE SENSING scenario, which includes an action senseWeight which senses a beverage container $?o$. To perform this action, the robot must first be grasping the object and must not already know whether it is spillable (which acts as an efficiency condition to ensure only new knowledge is gained from this action). When this action is performed, knowledge of whether $?o$ is spillable or not is added to PKS’s $K_w$ database.

An example manipulation action, transferUpright, is also shown in Table I. Using this action, objects that are grasped and not yet “removed” to the second table can be transferred. Besides transferUpright, which only handles objects that can be spilled, the complete domain definition also contains a similar action transfer, handling all other objects, as well as the necessary grasp and ungrasp actions that precede and follow the transfer actions.

Table II shown an example contingent plan generated at plan time. The KVP architecture (Fig. 2) executes the actions in this plan and chooses appropriate branches to follow by assessing the results of sensed information. This scenario was physically evaluated on a joint-impedance controlled lightweight 7-DoF robot with a force-controlled parallel gripper. Force was measured by internal torque sensing.

B. Visual Sensing scenario

In contrast to the previous scenario, visual information in this domain (defined in Table III) can be gathered independently from manipulation actions. In this case, the sensing action senseIfEmpty has no precondition other than the requirement that the knowledge it gathers must be new. An example manipulation action, pickUp, is also shown. Since this scenario contains two robot manipulators, and not all locations can be reached by both hands, the preconditions define an external call isReachable to the motion planning component to check reachability for a specific manipulator.
As future work, we plan to generalize our symbolic approach to task space constraints for object manipulation, and explore more efficient heuristic search strategies at the symbolic planning level, including building plans with loops.

V. ACKNOWLEDGMENTS

This research was supported in part by the European Union’s Seventh Framework Programme through the JAMES project (Joint Action for Multimodal Embodied Social Systems) under grant agreement no. 270435.

REFERENCES


2http://www.james-project.eu/ (accessed Mar. 17, 2013)
Robot Task and Motion Planning with Sets of Convex Polyhedra

Andre Gaschler∗, Ronald P. A. Petrick†, Torsten Kröger‡, Oussama Khatib‡, Alois Knoll∗

∗fortiss An-Institut der TU München, Munich, Germany, Email: gaschler@fortiss.org
†School of Informatics, University of Edinburgh, Edinburgh, United Kingdom, Email: rpetrick@inf.ed.ac.uk
‡Artificial Intelligence Laboratory, Stanford University, Stanford, USA

Abstract—Geometric volumes can be used as an intermediate representation for bridging the gap between task planning, with its symbolic preconditions and effects, and motion planning, with its continuous-space geometry. In this work, we use sets of convex polyhedra to represent the boundaries of objects, robot manipulators, and swept volumes of robot motions. We apply efficient algorithms for convex decomposition, conservative swept volume approximation and collision detection, and integrate these methods into our existing “knowledge of volumes” approach to robot task planning called KVP. We demonstrate and evaluate our approach in several task planning scenarios, including a bimanual robot platform.

I. INTRODUCTION

The problem of combining task planning and motion planning in a robot system presents significant representational difficulties that must be overcome: high-level task planners typically rely on symbolic representations of objects and actions, while motion planning systems need to reason about physical bodies and robot motion in a continuous space. Integrating geometric and symbolic reasoning in a common framework is therefore a challenging and important task.

In this paper, we describe an approach that represents the boundaries of objects, robots, and swept volumes of robot motions as geometric volumes, which serves as an intermediate representation between continuous and discrete reasoning. In particular, this work focuses on algorithms for convex decomposition, conservative swept volume generation, and collision detection, implemented in an existing “knowledge of volumes” approach to robot task planning called KVP [9, 10].

In addition to using volumes as a representation of robot motions and objects, KVP is further characterized by symbolic planning at the knowledge-level, with direct function calls to the robotics component for geometric and kinematic queries. As a symbolic AI planning system, we use the general purpose PKS planner (Planning with Knowledge and Sensing [18, 19]), which allows contingent planning with incomplete information and sensing actions. In particular, this approach enables us to use planned sensing actions as preconditions for manipulation, rather than being hard-coded as tasks themselves [10].

The remainder of the paper is organised as follows. In Section II, we situate our work with respect to related approaches. In Sections III-A and III-B, we analyse the efficiency of our approach, show the quadratic convergence of our swept volume approximation in joint space, and describe a complexity class for collision checking. Details of the symbolic planner are given in Section III-C. Finally, we evaluate the efficiency of the KVP approach in Section IV, in three example domains.

II. RELATED WORK

Although the problem of robot task planning has been investigated since the days of early robotic systems like Shakey [17], more recently the field has gained substantial attention, both from the planning and robotics communities. Modern approaches to robot task planning include a diverse range of techniques from probabilistic models in artificial intelligence [14], closed-world symbolic planning [2, 21], formal synthesis [15, 3], and multimodal motion planning [12].

Our KVP approach is in part inspired by Kaelbling and Lozano-Pérez’s work on hierarchical task and motion planning [13], borrowing the continuous geometry of swept volumes. However, while the geometric preconditions may be similar, their underlying aggressively hierarchical planning strategy differs from our knowledge-based planning approach, which has previously been used to connect robot vision and grasping with symbolic action [20]. In more recent work, Kaelbling and Lozano-Pérez [14] present a belief space planner which models probability distributions over states, making it robust against uncertainties. In contrast, our choice of symbolic planner is more geared towards structured environments with incomplete and discrete information. In both approaches, sensing actions may be formulated as preconditions for manipulation, and need not be hard-coded as tasks themselves.
**TABLE I**

**REMOVE n OBJECTS: SYMBOLIC DOMAIN DEFINITION**

<table>
<thead>
<tr>
<th>symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>types: object;</td>
</tr>
<tr>
<td>predicates: isRemoved/1;</td>
</tr>
<tr>
<td>constants: object bottle1, bottle2 ...</td>
</tr>
</tbody>
</table>

**action** remove(?o : object)

**preconds:**
- K(!isRemoved(?o)) & forallK (?p : object) (K(isRemoved(?p)) | !extern(graspMotionCollides(?o, ?p)) )

**effects:**
- add(Kf, isRemoved(?o))
- forallK (?p : object) K(!isRemoved(?p))
- goal: forallK (?o : object) (K(isRemoved(?o)))

**III. APPROACH**

We will demonstrate our approach with several examples, among them the very simple **REMOVE n OBJECTS** scenario defined in Table I, which involves clearing objects from a table. In order to pick up and remove an object, collisions with other objects need to be avoided. A successful plan must therefore include **remove** actions in an appropriate order (see Figure 4).

To represent these manipulation actions, we automatically decompose robot and object geometry models into sets of convex polyhedra, which can be prepared off-line. During planning, the evaluation of kinematic and geometric queries (e.g., **graspMotionCollides** in the **remove** action) requires the calculation of motion paths, swept volumes, and collision checking. Symbolic planning is performed by the PKS planner, which invokes the geometric reasoner as needed, to evaluate preconditions and effects. We discuss the algorithms and components behind this process in greater detail below.

**A. Convex Decomposition of Volumes**

In our KVP approach, it is crucial to efficiently represent arbitrary objects and robot manipulators as sets of convex polyhedra. In the general case, however, decomposing a non-convex polyhedron into a small (or even minimal) set of convex polyhedra is a challenging problem. Although the problem of minimal exact decomposition is known to be NP-hard, Mamou and Ghorbel [16] recently proposed an approximate algorithm that is sufficiently efficient and precise for practical instances. This approach shows better approximation results than existing algorithms, both with respect to approximation errors as well as the number of decomposed convex polyhedra.

In principle, their algorithm hierarchically segments the non-convex polyhedron on its dual graph by half-edge decimation. The segmentation is guided by a weighted cost function, trying to minimize concavity and an aspect ratio measure they define as the squared perimeter divided by the area of a given mesh, adjusted by a constant factor to yield one in the case of a disk. The cost function is weighed such that the aspect ratio guides the first few iterations of the algorithm, quickly simplifying the mesh. After that, the simplification is mostly lead by the concavity measure, which they define as the maximum distance of mesh points projected onto the convex hull of that mesh, measured in surface normal direction.

As indicated in Figure 2, Mamou and Ghorbel’s algorithm produces a concise set of convex bodies, which are well suited for efficient collision detection. For instance, a typical six-axes robot manipulator can be approximated by 6 to 10 convex polyhedra, totalling no more than 100 vertices. The swept volume of a typical motion of such a robot will simplify to no more than 20–100 convex polyhedra.

**B. Efficient Swept Volume Computation**

The representation of a robot’s volume as a small set of convex polyhedra allows us to efficiently compute the swept volume of a robot motion. In the static case, the volume of the robot geometry \( R \) is given by the union of \( n \) convex hulls of vertices \( R = \bigcup_n \text{conv}(V_n) \). More generally, we can also efficiently approximate the swept volume SV of a robot geometry \( R \) along a configuration space path \( Q \). In order to show the quadratic convergence of the swept volume approximation, we first consider the motion of a single convex polyhedron, which is given by the vertices \( V \). Let \( r_{\text{max}} \) be the maximum distance of a vertex to the first axis, summing up all link lengths and the distance to the last axis. It is a well-known fact that a point on a single link of length \( r \) rotating by an angle \( q < \pi/2 \) will deviate from the chord (straight line from start to end) by \( \epsilon = r \cos(q/2) \) [23]. For general serial kinematics of multiple revolute joints, Baginski [1] describes an upper bound for the deviation of the path of a point from the chord of that path:

\[
\epsilon \leq r_{\text{max}} \left(1 - \cos \left( \frac{\sum_{i=1}^{n} |q_i|}{2} \right) \right).
\] (1)

Intuitively, this bound is tight when all link lengths but the last one tend to zero and all joints rotate in the same direction.
around the same axis. ([22] describes a similar bound by roughly approximating path segments as screw motion.) In our implementation, we calculate the bound for each link and obtain tighter bounds especially for the first few joints.

Using the second order series expansion of the cosine

$$
\cos(q) \geq 1 - \left( \frac{q^2}{2} \right)
$$

we can approximate Eq. 1 as a quadratic function of the angular step size $\Delta q$ for our swept volume approximation:

$$
\epsilon \leq r_{\text{max}} \left( \sum_i |\Delta q_i| \right)^2 / 8.
$$

Choosing an angular step size $\Delta q = \sqrt{8\epsilon/r_{\text{max}}}$, we can therefore generate swept volumes at a desired precision $\epsilon$, and at a quadratic convergence rate. To construct the swept volume, we compute the convex hull $\text{conv}$ for each convex polyhedron sampling the path $Q$ as $q(i)$ at angular distances $\Delta q$ and applying the forward kinematic transformation $\text{FK}$, as implicitly done in [22, 23]:

$$
SV(R, Q) = \bigcup_i \text{conv} (\text{FK}(q(i+1))V_a \cup \text{FK}(q(i))V_a).
$$

Denoting $|Q|$ as the length of the path in joint space, it follows that $O(n \cdot |Q|/\sqrt{\epsilon})$ convex polyhedra are needed to represent the swept volume of a robot motion at a precision of $\epsilon$, with each one having at most $2|V|$ vertices. This result implies that doubling the number of sampling points of the path $Q$ will quadruple the precision of the approximation. In order to construct a conservative (superset) swept volume, $\epsilon$-enlarged models of $R$ may be computed off-line and—provided $R$ is a conservative approximation of the real robot geometry—all computed swept volumes are conservative and, most importantly, false negatives in collision checking are avoided.

For collision checking, we use the Bullet Physics Library implementation of the Gilbert-Johnson-Keerthi (GJK) algorithm [11], which has been shown to detect collisions between two convex polyhedra at a computational complexity linear in the total number of involved vertices. For an environment of $m$ convex polyhedra, a collision check with a swept robot volume can be performed in $O(nm|Q|/\sqrt{\epsilon})$ time. However, for many practical problems, the computation can be leveraged by fast broad-phase algorithms and may be considerably quicker.

C. Planning with Knowledge and Sensing (PKS)

Symbolic planning in KVP relies on the general-purpose PKS planner [18, 19], which constructs plans in the presence of incomplete information and sensing actions. PKS works at the knowledge level by reasoning about how the planner’s knowledge state, rather than the world state, changes due to action. PKS works with a restricted subset of a first-order language, allowing it to support a rich representation with features such as functions and run-time variables. This approach differs from planners that work with possible worlds models or representations based on belief states.

PKS is based on a generalization of STRIPS [7]. Unlike STRIPS, which uses a single database to model the world state, PKS’s knowledge state is represented by five databases, each of which models a particular type of knowledge and has a fixed, formal interpretation in a modal logic of knowledge. Actions can modify any of these databases, which has the effect of updating the planner’s knowledge state. To ensure efficient inference, PKS restricts the type of knowledge (especially disjunctions) that it can represent in each of its database: $K_f$: This database is like a standard STRIPS database except that both positive and negative facts are permitted and the closed-world assumption is not applied. $K_f$ is used to model action effects that change the world. $K_f$ can include any ground literal $\ell$, where $\ell \in K_f$ means “the planner knows $\ell$.” $K_f$ can also contain known function (in)equality mappings.

$K_a$: This database models the plan-time effects of sensing actions that return binary values. A formula $\phi \in K_a$ means that at plan time, the planner knows whether $\phi$ or $\neg \phi$ holds, and that at run time this disjunction will be resolved. The use of $K_a$ for robot sensing is described in detail below.

$K_v$: This database stores information about function values that will become known at execution-time. In particular, $K_v$ can model the plan-time effects of sensing actions that return constants. $K_v$ can contain any unnested function term $f$ where $f \in K_v$ means the planner “knows the value of $f$.”

( PKS also includes two additional databases, $K_e$ and $LCW$, which aren’t used in this paper.)

PKS knowledge states can be queried in three different ways. First, simple knowledge assertions can be tested with a query of the form $K(\phi)$, which asks “is a formula $\phi$ true?” Second, a query $K_v(\phi)$ asks whether $\phi$ is known to be true or known to be false (i.e., does the planner “know whether $\phi$”). Finally, $K_v(t)$ asks “is the value of function $t$ known?” The negation of the above queries can also be used.

Using this database representation and query mechanism, symbolic actions are defined in PKS by describing their (typed) parameters, preconditions, and effects. Preconditions contain a list of queries that must evaluate as true before an action can be applied. Effects are described by a list of add and del operations, similar to STRIPS. For instance, Table I shows the definition of a PKS action $\text{remove(?o)}$, which has the effect of clearing an object ?o from a table.

PKS also has the ability to model sensing actions that return information about the state of the world, an idea that is important for robot task planning [10]. In particular, PKS offers two databases, $K_u$ and $K_v$, that represent unknown information (binary or function values, respectively) that will become known at run time after the sensing actions are executed in the world. Using these databases, PKS can reason about the possible outcomes of such actions during plan construction, by generating plans with contingencies. For instance, the $\text{senseWeight(?o)}$ action in Table III is an action of a sensing action that tests whether an object ?o is spillable or not.
TABLE II
BIMANUAL SCENARIO: EXAMPLE ACTIONS

<table>
<thead>
<tr>
<th>action</th>
<th>pickUp((?r:robot, ?o:object, ?l:location))</th>
</tr>
</thead>
<tbody>
<tr>
<td>preconds:</td>
<td>(K(?l = \text{getObjectLocation}(?o)) \land K(\text{handEmpty}(?r)) \land K(\text{extern}(\text{isReachable}(?l, ?r))))</td>
</tr>
<tr>
<td>effects:</td>
<td>(\text{del}(K_f, ?l = \text{getObjectLocation}(?o)), \text{del}(K_f, \text{handEmpty}(?r)), \text{add}(K_f, \text{inHand}(?o, ?r)))</td>
</tr>
</tbody>
</table>

In general, PKS builds plans by reasoning about actions in a forward-chaining manner: if the preconditions of an action are satisfied by the planner’s knowledge state, then the action’s effects are applied to produce a new knowledge state. Planning then continues from this new state. PKS can also build plans with contingencies by considering its \(K_w\) and \(K_v\) knowledge. For instance, if \(\phi\) is in \(K_w\) then PKS can introduce two branches into a plan: along one branch \(\phi\) is assumed to be true, while along the other branch \(\neg\phi\) is assumed to be true. Planning then continues along each branch until the goal conditions (a set of queries) are satisfied.

A second feature of PKS which is key to the KVP approach is its ability to integrate externally-defined procedures (e.g., from support libraries) with its internal reasoning mechanisms. While the idea of transferring reasoning to external processes is not a new idea [5, 6, 4], the introduction of such techniques in PKS is a more recent extension. A special keyword, \texttt{extern} provides an interface to this facility, where an expression \texttt{extern(proc(\vec{x}))} means that the parameters \(\vec{x}\) should be passed to an external procedure \texttt{proc} for execution. \(\vec{x}\) can contain symbols defined in PKS’s knowledge state, providing a link between the planner and the externally-defined procedure. The return value of an \texttt{extern} call is passed back to PKS, which can perform additional tests on this value, or include it in its knowledge base. For instance, the \texttt{remove} action in Table I uses an \texttt{extern} call to invoke path planning and collision checking in the evaluation of \texttt{graspMotionCollides}. An \texttt{extern} call can also be directed to cache its return value for efficiency. As a result, PKS’s core reasoning capabilities can be augmented by the addition of motion planning, collision detection, and other special purpose robotics libraries.

IV. EVALUATION

We now demonstrate and evaluate the efficiency of our approach with three scenarios. In the \texttt{REMOVE n OBJECTS} scenario, a single manipulator is used to remove \(n\) objects from a table while avoiding collisions. This scenario can readily be generalised to multiple manipulators, as shown in the \texttt{BIMANUAL} scenario. As a third example, we give the \texttt{FORCE SENSING} scenario, which uses a sensing action that can be modelled by the knowledge-level planner.

For the \texttt{REMOVE n OBJECTS} scenario defined in Table I, we evaluate the performance of the planner with respect to the number of objects \(n\), shown in Table IV. For this simple scenario, the planning time seems almost linear in the number of objects. In general, planning time is lower than or within the same order of magnitude as typical execution on a robot would take. Even though the number of collision tests may be higher than quadratic in the worst case, it must be noted that collisions of convex polyhedra can be checked very efficiently and collision checking only amounts to a negligible fraction of the total planning time for the numbers observed. Swept volume generation time mostly depends on the number of objects grasped, and slightly on the location of those grasps. In the worst case, only one of the objects can be picked up at a time. Even though this will increase the number of collision checks, influence on the total planning time is minor. As a demonstration, a sample task with \(n = 3\) bottles was executed on a real robot, as depicted in Figure 4.

The \texttt{BIMANUAL} scenario shows that pick-and-place actions can be planned for arbitrary numbers of manipulators, denoting the respective robot \(?r\) as an argument for each manipulation action in the symbolic domain, as shown in Table II. It is worth noting that multiple manipulators may lead to non-
TABLE IV
EVALUATION OF THE REMOVE n OBJECTS SCENARIO

<table>
<thead>
<tr>
<th>Number of Objects</th>
<th>Total Planning Time [s]</th>
<th>Inverse Kinematics [s]</th>
<th>Path Planning [s]</th>
<th>Swept Volume Generation [s]</th>
<th>Number of Triangles Generated</th>
<th>Number of Convex Bodies Generated</th>
<th>Number of Collision Tests</th>
<th>Collision Testing [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.77 0.35 0.43 1.99</td>
<td>1936 121 1  .00006</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5.49 0.73 0.40 4.36</td>
<td>3168 198 4  .00021</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8.42 1.03 1.06 6.33</td>
<td>4400 275 6  .00010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>13.40 1.72 1.55 10.13</td>
<td>8976 561 22  .00049</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>14.60 1.77 2.69 10.14</td>
<td>8976 561 18  .00030</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>18.52 1.99 4.35 12.18</td>
<td>10912 682 22  .00035</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>19.15 2.32 2.71 14.12</td>
<td>12672 792 35  .00050</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>22.21 2.77 3.04 16.40</td>
<td>13376 836 70  .00088</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>24.33 2.99 3.10 18.23</td>
<td>15312 957 94  .00135</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>40.32 4.91 5.63 29.78</td>
<td>25872 1617 314 .00306</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>56.70 6.60 10.16 39.93</td>
<td>34848 2178 530 .00601</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

“Worst case”: Objects in a line, only 1 of n can be picked up

<table>
<thead>
<tr>
<th>Number of Objects</th>
<th>Total Planning Time [s]</th>
<th>Inverse Kinematics [s]</th>
<th>Path Planning [s]</th>
<th>Swept Volume Generation [s]</th>
<th>Number of Triangles Generated</th>
<th>Number of Convex Bodies Generated</th>
<th>Number of Collision Tests</th>
<th>Collision Testing [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.66 0.34 0.35 1.97</td>
<td>1936 121 1  .00004</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5.56 0.69 0.78 4.09</td>
<td>4224 264 3  .00009</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8.30 0.99 1.13 6.18</td>
<td>6160 385 6  .00012</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10.93 1.33 1.45 8.15</td>
<td>8096 506 10  .00018</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>13.33 1.63 1.78 9.92</td>
<td>9680 605 15  .00029</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>18.97 2.25 2.89 13.83</td>
<td>13552 847 36  .00059</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>21.81 2.57 3.50 15.74</td>
<td>15488 968 44  .00071</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>25.21 2.97 4.54 17.70</td>
<td>17424 1089 112 .00140</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>27.85 3.23 5.12 19.50</td>
<td>19360 1210 125 .00170</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>42.01 4.89 7.73 29.36</td>
<td>28336 1771 544 .00581</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>56.81 6.72 8.75 41.32</td>
<td>36256 2266 1257 .01227</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

trivial behaviour: in order to move bottles from the right side of the bar to the goal location (Figure 1), one arm needs to move them to a location that the second arm can reach—an intermediate location to “pass it on” to a different manipulator. A previous implementation of this scenario, demonstrated on a two-manipulator robot setup, is described in [9, 8].

The FORCE SENSING scenario (Figure 3) illustrates the use of sensing actions. (A more detailed account of this domain is given in [10].) Table III shows the definitions of the sensing action SenseWeight and the manipulation action transferUpright. When grasping a beverage container ?o, the robot can sense its weight and will come to know whether it can be spilled or not, adding this knowledge to PKS’s Kω database. The planner will then generate a contingent plan with binary branches, each of which account for one possible outcome of this knowledge at execution time.

V. CONCLUSION AND FUTURE WORK

In this paper, we extend our “knowledge of volumes” approach to robot task planning (KVP) and its representation of volumes as sets of convex polyhedra. In particular, we present a conservative ϵ-precise algorithm for swept volume computation of sets of convex polyhedra with quadratic convergence and discuss the efficiency of this representation in collision detection and, more generally, in pick-and-place tasks. We further demonstrate the effectiveness of KVP in several scenarios, including those involving multiple manipulators.

As future work, we plan to generalize our approach to mobile manipulation and investigate tighter bounds on ∆q with respect to the kinematics, possibly allowing even more efficient swept volume generation.

ACKNOWLEDGEMENTS

The authors would like to thank Quirin Fischer for his help with the implementation and evaluation. This research was supported in part by the European Union’s 7th Framework Programme through the JAMES project (Joint Action for Multimodal Embodied Social Systems, grant no. 270435).2

REFERENCES


2http://www.james-project.eu/ (accessed May 20, 2013)
Fig. 4. **REMOVE n BOTTLES** scenario: The solution of the task in Table I was implemented and tested on a Meka Robotics H2. In this scenario, the robot can remove only the rightmost bottle in order to avoid collisions.


KVP: A Knowledge of Volumes Approach to Robot Task Planning

Andre Gaschler, Ronald P. A. Petrick, Manuel Giuliani, Markus Rickert and Alois Knoll

Abstract—Robot task planning is an inherently challenging problem, as it covers both continuous-space geometric reasoning about robot motion and perception, as well as purely symbolic knowledge about actions and objects. This paper presents a novel “knowledge of volumes” framework for solving generic robot tasks in partially known environments. In particular, this approach (abbreviated, KVP) combines the power of symbolic, knowledge-level AI planning with the efficient computation of volumes, which serve as an intermediate representation for both robot action and perception. While we demonstrate the effectiveness of our framework in a bimanual robot bartender scenario, our approach is also more generally applicable to tasks in automation and mobile manipulation, involving arbitrary numbers of manipulators.

I. INTRODUCTION

In recent years, symbolic task planners have made substantial progress in their ability to reason about incomplete knowledge and perceptual information [1], [2], [3], [4], laying the foundation for modeling realistic robot environments. One promising technique is the knowledge-based planning approach [5], [6], which enables a robot to reason about unknown state information and the effects of perceptual actions, thereby providing the tools needed for robust planning when a robot may only have incomplete information about its environment. However, integrating the geometric properties of the robot and the environment into a purely symbolic task planner poses significant challenges: robot systems must reason about joint angles, spatial coordinates and physical bodies in continuous spaces; while high-level task planners typically rely on discrete, symbolic representations of features, values, and actions described in logical languages [2]. As a result, very few approaches have successfully bridged the gap between geometric and symbolic representations, allowing logical task planners to work effectively with geometric constraints [7], [8].

In this paper, we describe a knowledge of volumes approach to robot task planning (abbreviated, KVP), which treats volumes as an intermediary representation between continuous-valued robot motions and discrete symbolic actions. An off-the-shelf, general purpose, symbolic AI planner called PKS (Planning with Knowledge and Sensing; [6], [9]) is employed as a backend reasoning engine for efficiently computing knowledge-level task plans, utilising the volume-based representation in the underlying description of the robot’s operational domain. By combining these two techniques, the robot can reason with incomplete information about objects, motions, and actuation actions, and perform information-gathering sensing actions when necessary. This gives rise to a novel approach to robot task planning which has certain inherent advantages:

1) 3D geometric volumes are a natural intermediate representation, making it suitable for bridging the gap between motion planning and high-level task planning.
2) Continuous geometry is preferable over indiscriminate discretisation [8]: novel geometric simplification techniques [10] allow computationally efficient collision detection of volumes [11], enabling continuous geometry planning to be applied in real systems.
3) Knowledge-based planning naturally allows reasoning about acting and sensing in a structured, partially known environment, which is reflected by concise and clear domain descriptions, keeping the overall approach generic for a broad set of robotics applications.

As a result, this work makes a number of strong contributions to the problem of combining continuous-space geometric reasoning with high-level symbolic planning. Our approach is among the first to use 3D geometric volumes as the underlying representation for symbolic planning and motion planning, and the first to combine this idea with an off-the-shelf, general-purpose AI planner supporting deterministic planning with incomplete information and sensing. In contrast to more specialised approaches, which limit the scope of application, our use of a general-purpose planner provides us with a tool that can be applied in contexts beyond that of the demonstrated bimanual robot, including tasks in automation and mobile manipulation, involving arbitrary numbers of manipulators. Moreover, by building on standard planning representations, we can take advantage of new planning engines that become available from the planning...
community, which may be better optimised for our task. Finally, our work is grounded in real-world application, and all of our experiments are tested on actual robot platforms, rather than solely in simulated environments.

The rest of this paper is structured as follows. In Section II, we compare our work to related approaches, and then describe our framework in greater detail in Section III. In Section IV, we evaluate our approach in a real robot domain, namely a two-handed robot bartender scenario. Finally, in Section V, we discuss future work and conclude.

II. RELATED WORK

Some of the earliest work on robot task planning dates back to systems like Shakey [12] and Handey [13]. Since those early approaches, the field has made significant developments, with interest in the problem of real-world task planning gaining momentum in recent years. In particular, the problem has been explored from several diverse areas of research, including probabilistic models from AI [14], closed-world symbolic planning [7], [15], [16], formal synthesis [17], [18], [19], and manipulation planning [20].

A very recent contribution, most closely related to our approach, is the belief space planner by Kaelbling and Lozano-Pérez [14], which operates on a belief space of probability distributions over states, providing robustness against uncertainty and unexpected change. Perception (and estimation) arises as a necessary precondition for manipulation, rather than being hard-coded as a task itself. Using a simulation of a PR2 mobile manipulator, Kaelbling and Lozano-Pérez demonstrate the effectiveness of their approach through a series of experiments in initially unknown environments. In contrast to their notion of belief, our knowledge-based planning approach relies on discrete logic and abstracted state descriptions, which we believe is a viable alternative in structured environments with certain, but incomplete, information about the world. Moreover, we use an existing planner which has proven successful in previous robot deployments [2], rather than designing a new planner for the task. In terms of volumes, Kaelbling and Lozano-Pérez work with octrees, while we use sets of convex shapes, allowing efficient collision detection in the deterministic case.

More traditionally, high-level task planning is often seen as a computational layer on top of motion planning, and several approaches try to integrate both layers. Our work is in part inspired by Kaelbling and Lozano-Pérez’ earlier work on hierarchical task and motion planning in [8], borrowing the continuous geometry of swept volumes. But, whereas the geometric preconditions may be similar, their aggressively hierarchical planning strategy differs from our knowledge-based approach which make use of abstract (but non-hierarchical) structures. Cambon, Alami and Gravot [7], [21] also propose a task planning algorithm that can internally handle geometric preconditions and effects. Their approach systematically integrates symbolic PDDL-(Planning Domain Definition Language) [22] based states and actions together with geometric motion planning. Dornhege et al. [16] also present a mobile manipulation planner, which is similar to the way the planner we use invokes and evaluates geometric functions. More recently, Plaku and Hager [15] develop a similar, sampling-based motion and action planning approach, additionally allowing differential motion constraints.

It is important to note that many of the above approaches assume a closed world, where all symbols are either true or false, which is a significant limitation for many robotic scenarios. In contrast to this closed world assumption, our task planning approach is based on an open-world knowledge representation which naturally models incomplete information and sensing actions—advantages which we elaborate on in Section III-B. It should further be noted that in the related works described above, either the evaluation of the combined task and motion planner is confined to simulation [8], [7], [16], [15], or a general domain-independent planning scheme is not the focus of the work [20] when a real robot system is used to evaluate the approach.

III. APPROACH

Integrated robot task planning requires a multidisciplinary approach, adapting solutions from different fields ranging from motion planning to formal methods. KVP combines several of these techniques, which we outline below. We begin by discussing the geometric models we use (Section III-A), followed by a description of the knowledge-level PKS planner (Section III-B), and the planning domain we have defined (Section III-C). Finally, we describe the overall system architecture of our framework (Section III-D). Our evaluation scenario is a simple robot bartender that clears away empty bottles from a table (see Figure 1), requiring both sensing and manipulation actions.

A. Convex Decomposition of Volumes

We use volumes as an intermediary representation of a geometric shape, modelled in a high-level symbolic form. In KVP, volumes are used to represent the physical boundaries of both static objects and dynamic motions—as well as the view cone of a sensor, representing an area of perception. It is therefore important to process these volumes using a computationally-efficient data type, for which we use sets of convex bodies. A convex body is defined by a set of 3D points, and the body is the set of all convex combinations of these points. To ease computations, the set of outer triangles (triples of these point indices) is usually saved within the same data object. Even though a single convex body may be too conservative an approximation of the boundaries of most volumes in our domain, a small set of convex bodies usually suffices to accurately approximate volumes as complex as the swept volumes of a robot, as shown in Figure 2. Using sets of convex bodies also leads to a computationally-efficient collision detection process, which is a significant advantage of this approach in real-world applications.

Although the above approach provides a computationally-efficient method that can be used online at run time, the offline task of decomposing arbitrary geometric models into sets of convex shapes remains a challenging problem. For our KVP framework, we chose an implementation of the
Fig. 2. Convex decomposition allows an efficient approximation of swept volumes. A typical robot mesh has $10^6$ vertices and is non-convex (left). Convex decomposition [10] simplifies this to 6 convex bodies with 10 vertices each (centre), allowing a typical swept volume description with only 40 convex bodies, totalling 400 vertices (right) [23].

algorithm in [10] by Mamou and Ghorbel. In particular, their approach performs a hierarchical segmentation on the dual graph of triangles, which is lead by a cost function based on concavity and an aspect ratio measure. As a concavity measure for a given 3D mesh, they choose the maximum distance of all mesh surface points in surface normal direction to their projection onto the convex hull of that mesh. For the aspect ratio, they define the squared perimeter divided by the area of a given 3D mesh, adjusted by a constant factor to yield one in the case of a disk. In the iterative surface simplification, both cost function components are weighted such that aspect ratio dominates the early stage of the algorithm, which quickly compacts the surface, followed by the concavity measure determining the final simplifications. At each iteration, a half-edge collapse decimation is performed on the dual graph.

Mamou and Ghorbel show that their approach is superior to existing solutions, both in terms of the approximation quality and number of convex shapes, as well as speed. As indicated in Figure 2, their algorithm produces a concise set of convex bodies, which are ideal for efficient collision detection. A typical six-axes robot manipulator can be approximated by 6 to 10 convex bodies, totalling no more than 100 points. A swept volume of a typical motion for such a robot will simplify to no more than 20–100 convex bodies, and can be efficiently computed by merging subsequent samples of the robot model along the motion path [23]. In practice, sets of convex bodies provide a sufficiently efficient geometric representation so that collision detection often accounts for only a small fraction of total computation time.

B. Planning with Knowledge and Sensing (PKS)

High-level planning in KVP is provided by the off-the-shelf PKS (Planning with Knowledge and Sensing) planner [6], [9], which is able to construct plans in the presence of incomplete information and sensing actions. PKS works at the knowledge-level [5] by reasoning about how the planner’s knowledge, rather than the world state, changes due to action. PKS is a symbolic planner that is built on a subset of a first-order logical language with restricted inference, allowing it to efficiently support a wide range of optimised features that result from limiting its representation language. This approach differs from planners based on possible worlds or belief states, which often trade tractability for more comprehensive representations and reasoning capabilities.

PKS is based on a generalisation of STRIPS [24] and uses a database mechanism as its underlying state representation. In particular, knowledge states in PKS are represented by a set of five databases, each of which models a particular type of knowledge. Actions can modify any of the databases, which has the effect of updating the planner’s knowledge state. To ensure efficient inference, PKS restricts the type of knowledge (especially disjunctions) that it can represent. In this work, we mainly focus on three of PKS’s databases:

- **$K_f$:** This database is like a STRIPS database that stores the values of regular fluents that the planner knows. $K_f$ is primarily used for modelling the effects of actions that change the world. Unlike standard STRIPS, it works with an open world model that can explicitly represent both positive and negative facts. In particular, $K_f$ can include any ground literal $\ell$, where $\ell \in K_f$ means “the planner knows $\ell$.”

- **$K_w$:** This database stores information about the effects of sensing actions that return one of two possible outcomes, providing support for information-gathering actions that observe the world but do not necessarily change it. A formula $\phi \in K_w$ means that at plan time the planner either “knows $\phi$ or knows $\neg\phi$.” However, this disjunction will not be resolved until run time when the action is actually executed.

- **$K_v$:** This database stores information about function values that will become known at execution time. In particular, $K_v$ can model the effects of sensing actions that return one of many possible values. $K_v$ can contain any function term $f$, where $f \in K_v$ means the planner “knows the value of $f$.”

PKS also includes databases for modelling a restricted type of disjunctive information ($K_c$), and local closed world information ($LCW$) [25], which are not used in this paper.

Reasoning in PKS is done through a limited set of primitive queries that ask simple questions about the planner’s knowledge state: (i) is a fact $\phi$ known to be true (i.e., $K(\phi)$) or known to be false (i.e., $K(\neg\phi)$)? (ii) does the planner know whether a property $\phi$ is true or not (i.e., $K_v(\phi)$)? (iii) is the value of a function $t$ known (i.e., $K_c(t)$)? (iv) or the negation of the above queries. An efficient inference algorithm evaluates primitive queries by checking the contents of the databases and the relationship between the databases. Details of this procedure can be found in the original PKS papers by Petrick and Bacchus [6], [9].

An action in PKS is described by its parameters, preconditions, and effects. An action’s parameters are a list of typed variables which may be used throughout the action definition. When evaluated, the planner replaces these parameters with objects of the appropriate type, chosen by the planner, which

---

1 PKS is available from http://homepages.inf.ed.ac.uk/rpетrick/software/pks/.
are bound to each occurrence of the parameter. An action’s preconditions are a list of primitive queries which must evaluate as true before an action can be applied. Action effects are described by a collection of STRIPS-style “add” and “delete” operations that allow information to be inserted and removed from individual databases. For instance, \( \text{add}(K_w, \phi) \) adds \( \phi \) to \( K_w \), and \( \text{del}(K_w, \phi) \) removes \( \phi \) from \( K_w \). Two PKS actions from our evaluation domain are shown in Table I, which we will discuss in greater detail in Section III-C.

An important class of actions that PKS can model are sensing actions that return information about the state of the world. In PKS, sensing actions are specified by effects that update the \( K_w \) or \( K_u \) databases (i.e., databases that track properties with multiple potential outcomes). Given such information, the planner can build contingencies into a plan by introducing a conditional plan branch for each possible outcome of the sensing action. (E.g., the \text{senseIfEmpty}(?o) action in Table I is an example of a sensing action in our evaluation domain that provides information about the state of \text{isEmptyBottle}(?o)\), which can be true or false.)

In general, PKS builds plans by reasoning about actions in a forward-chaining manner: if the preconditions of an action are satisfied by the planner’s knowledge state, then the action’s effects are applied to produce a new knowledge state. Planning then continues from this new state. The choice of which action to apply next in a given state is achieved by using either a depth-first search or breadth-first search strategy, optimised by a set of heuristics. (Depth-first search typically produces plans more quickly than breadth-first search, however, breadth-first search generally results in plans with fewer steps.) PKS can also build plans with contingencies by considering the potential outcomes of a sensing action (i.e., its \( K_u \) and \( K_w \) knowledge). For instance, if \( \phi \) is in \( K_u \) then PKS can introduce two branches into a plan: one branch \( \phi \) is assumed to be true, while along the other branch \( \neg \phi \) is assumed to be true. Planning continues along each branch until the goal conditions (a set of primitive queries) are satisfied in each case.

One important feature of PKS, central to the KVP framework, is its ability to integrate externally-defined procedures (e.g., from support libraries) with its internal reasoning mechanisms [26]. A special keyword, \text{extern}, supports this facility, where an expression of the form \text{extern}(\text{proc}(\vec{x}))\) means that the parameters \( \vec{x} \) should be passed to an external procedure \text{proc} for execution. \( \vec{x} \) can contain symbols defined in the planning domain, providing a link between the planner and the externally-defined procedure. The return value of an \text{extern} call is passed back to PKS, which can then perform additional tests on this value, or include it in its knowledge base. (E.g., the \text{pickUp} action in Table I uses an \text{extern} call to invoke a path planner.) An \text{extern} call can also be directed to cache its return value for efficiency. PKS’s ability to invoke external procedures is key to our KVP approach, enabling us to augment PKS’s reasoning capabilities through motion planning, collision detection, and other special purpose robotics libraries.

### Table I: Example PKS actions and goals for the bartender scenario

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{senseIfEmpty}(?o:object)</td>
<td>( \neg K_w(\text{isEmptyBottle}(?o)) )</td>
<td>( \text{add}(K_w, \text{isEmptyBottle}(?o)) )</td>
</tr>
<tr>
<td>\text{pickUp}(?r:robot, ?o:object, ?l:location)</td>
<td>( K(?l = \text{getObjectLocation}(?o)) \land K(\text{handEmpty}(?r)) )</td>
<td>( \text{del}(K_f, ?l = \text{getObjectLocation}(?o)) ), ( \text{del}(K_f, \text{handEmpty}(?r)) ), ( \text{add}(K_f, \text{inHand}(?o, ?r)) )</td>
</tr>
</tbody>
</table>

### Fig. 3: Overview of the implemented KVP software architecture.

#### C. Planning Domain Definition

The experimental application domain for this work is a robot bartender scenario, where the robot can manipulate objects (e.g., bottles) in the world. In order to use PKS in this scenario we must first supply a symbolic domain description, characterising its properties and actions, and the robot’s goal.

Properties in PKS are similar to logical predicates and functions, and may have typed arguments. For our experimental domain we define three types: \text{robot}, \text{object}, and \text{location}. Using these types, we define predicates that characterise the domain’s state space: \text{isEmptyBottle}(?o)\, (object \( ?o \) is an empty bottle), \text{handEmpty}(?r)\, (robot hand \( ?r \) is empty), \text{inHand}(?o, ?r)\, (object \( ?o \) is in robot hand \( ?r \)), and \text{isReachable}(?l, ?r)\, (location \( ?l \) is reachable by robot hand \( ?r \)). We also include a function, \( \text{getObjectLocation}(?o) = ?l \)\, (object \( ?o \) is in location \( ?l \)).

Using these symbols, we can formulate PKS actions, as described above in Section III-B. Table I shows an example of two PKS actions in the bartender scenario. Here, \text{senseIfEmpty}(?o)\, is a sensing action that determines whether or not a bottle \( ?o \) is empty. This information-gathering action is modelled by an effect that adds the \text{isEmptyBottle}(?o)\, predicate to the \( K_w \) database, provided this information isn’t already known to the planner.
The inclusion of `senseIfEmpty(?o)` allows the planner to build contingent branches in its plan, where each branch considers one of the possible outcomes of `isEmptyBottle(?o)` (i.e., one contingency if `isEmptyBottle(?o)` is true, and another if `isEmptyBottle(?o)` is false). However, at run time, when the true value of `isEmptyBottle(?o)` is known, the appropriate branch of the plan will be executed on the robot. A more detailed description of sensing actions and branched plans in this domain is given in [27].

Table I also includes the action `pickUp(?r,?o,?l)`, a physical action that uses robot hand `?r` to pick up object `?o` from location `?l`. This action is modelled by a set of preconditions that verify the planner knows the location of `?o` is `?l`, that hand `?r` is empty, and that location `?l` is reachable with `?r`. When the action is applied, the planner comes to know that `?o` is no longer at `?l`, hand `?r` is no longer empty, and that the robot is holding `?o` in hand `?r`.

For a typical robot action, motion planning often needs to be performed. To specify such behaviour as part of the `pickUp` action, we include an external call to the procedure `isReachable(?l,?r)` (indicated by the `extern` directive), which directs PKS to invoke the path planner for a particular location `?l` and robot hand `?r`, and report if a path could be generated. This allows us to define actions to pick and place objects. In contrast to the sensing action `senseIfEmpty`, whose outcome is evaluated at run time, the motion planning procedure `isReachable` is evaluated at planning time.

The complete PKS domain description also includes an action `putDown(?r,?o,?l)`, which allows the robot to place an object `?o` at a location `?l` using hand `?r`. This action is defined in a similar manner to the `pickUp` action in Table I.

The final component of our domain definition is a specification of the robot’s goal. In our bartender scenario, the robot is given the task of moving all empty bottles to a location named `dishwasher`, specified by the goal expression in Table I. The goal is achieved if each object is either in the dishwasher, or isn’t an empty bottle. With the slight addition of swept volume collision checking to avoid collisions between robot hands (which we omit for clarity of presentation), we obtain the domain definition used in the evaluation in Section IV below. We note that even though we use the bartender scenario as an example of our approach, this domain description may easily be generalised to other robot tasks as it involves common object transfers and collision-avoiding robot motions.

D. System Architecture

The components described above—volume simplification, the PKS planner, and the planning domain description—account for only part of the KVP platform. To test the effectiveness of our approach, we have implemented and evaluated our framework both in simulation and on a real robot system. The complete system architecture is shown in Figure 3. Although this architecture contains a number of support modules which aid the components described above, the only computationally expensive task in this framework is volume simplification, which can mostly operate offline: the static scene, all object types, and the robot limbs need to be simplified only once. Swept volumes need to be computed as part of the planning process, and can be simplified by using the previously calculated robot limb volumes.

During planning, PKS can generate motion plans and check for collisions by directly calling functions from the motion planning and collision detection components. Both of these components rely heavily on the Robotics Library (RL)\(^2\) by Rickert [28]. RL includes several efficient motion planning algorithms and a manually optimised algebraic solution of the inverse kinematics for a broad range of six degrees-of-freedom robots, including the industrial manipulator used in the evaluation. For path planning, we apply a simple interpolation in configuration space between locations in the defined domain. We also include a simple grasp planner that can evaluate a set of grasping poses of the end effector.

In the evaluated bartender scenario (see below), the grasping poses are generated from one defined end effector pose sampled around a bottle’s axis of rotation in 30 degree steps. During path planning, paths are only checked for collisions with the static environment. All other collision checks are listed in the action preconditions and dispatched by PKS, calling the collision checker when necessary. The actual collision detection is based on the Bullet Physics Library,\(^3\) which is geared towards efficiency and can handle large numbers of convex bodies using fast broad-phase detection.

Action execution at run time is mediated by a simple plan execution component, which processes the plans generated by PKS, one action at a time. For robot actions, plan execution calls a trajectory generator, which performs a quintic interpolation of the robot paths in configuration space. Once generated, these trajectories are then executed on the physical robot. For sensing actions that give rise to contingent (branching) plans, the outcome of the sensing is used to determine which contingency (branch) of the plan should be followed [27]. In our evaluation scenario, the only run-time sensing action involves a simple colour-based vision component (see Figure 4) that reports if a bottle is full or empty. We note that the spatial locations of the bottles are also visually detected by this sensing component. However, since this information is also needed for motion planning, it may already be available at planning time. The plan execution component also acts as a recovery mechanism that allows plans to be rebuilt in response to unexpected changes in the environment. For example, in our bartender domain, bottles that are newly detected at run time could trigger a replanning stage, which would lead to new (but few) geometric volume computations. While we do not use replanning in our evaluated system, in general, this mechanism provides a useful tool that can improve the overall robustness of the system in many domains.

IV. Evaluation

We performed an evaluation of our KVP implementation both in simulation and on a bimanual robot system, whose

\(^2\)http://roblib.sf.net/
\(^3\)http://bulletphysics.org/
Run-time execution time 68 s 68 s

Table II shows the running times for various aspects of KVP, under alternative search methods used by the PKS planner for plan generation. While the results indicate depth-first search is more efficient than breadth-first search for scenarios more complicated than our evaluation domain, breadth-first search may yield significantly shorter plans. Furthermore, our results illustrate that the total planning time is adequately short for the evaluated scenario, and does not significantly impact on the overall execution of the system. For depth-first search, planning is an order of magnitude faster than both the offline simplification of static volumes and the actual execution of robot actions, making it an efficient technique for solving problems in this scenario.

V. Conclusion and Future Work

In this paper, we describe an approach to robot task planning that combines reasoning about complex geometric volumes with general-purpose knowledge-level planning techniques. We demonstrate the effectiveness of our KVP framework on a realistic two-robot setup that includes run-time perception and physical robot actions in a bartender scenario. Overall, we believe that volume-based task planning is applicable to a broad range of robot tasks, and may prove effective in structured and partially known environments, including automation, robot-aided manufacturing, and mobile manipulation, involving arbitrary numbers of manipulators.

We also view the use of a general-purpose planner that has not been explicitly optimised for planning in robotics domains as an advantage of our approach. First, our framework profits from future updates and improvements to the actively-developed PKS planner. Second, by treating planning as a black box we keep our framework sufficiently modular, allowing us to consider other symbolic planners from the planning community which can be substituted for PKS and tested in the KVP framework with minimal effort.

As future work, we plan to explore other symbolic planners, as well as extensions to PKS to improve its performance in more complex robotics domains. We will also investigate the notion of volumes for perception further, which neatly fits into the KVP framework, and may enable us to identify initially unknown or dynamic objects and obstacles.

References


Fig. 6. Action sequence of a solution in the evaluation scenario: In this example, the left arm is the only one to reach the bottle (image B). However, only the right arm can reach the desired dishwasher location. Therefore, the robot’s left arm moves the first bottle to a location where the other arm can reach it (image D), and the right arm passes it on to its final location (images G and H). We note that this behaviour has not been pre-programmed but, rather, arises purely from symbolic planning.

A video on this scenario is available in the file attachment of this paper and on the webpage http://youtu.be/9MzX8hfr88s.


