Combining Task and Path Planning for a Humanoid Two-arm Robotic System

Lars Karlsson and **Julien Bidot** and **Fabien Lagriffoul** and **Alessandro Saffiotti**
Centre for Applied Autonomous Sensor Systems (AASS), Örebro University, Sweden

Ulrich Hillenbrand and **Florian Schmidt**
German Aerospace Center (DLR), Oberpfaffenhofen, Germany

Abstract

The ability to perform both causal (means-end) and geometric reasoning is important in order to achieve autonomy for advanced robotic systems. In this paper, we describe work in progress on planning for a humanoid two-arm robotic system where task and path planning capabilities have been integrated into a coherent planning framework. We address a number of challenges of integrating combined task and path planning with the complete robotic system, in particular concerning perception and execution. Geometric backtracking is considered: this is the process of revisiting geometric choices (grasps, positions etc.) in previous actions in order to be able to satisfy the geometric preconditions of the action presently under consideration of the planner. We argue that geometric backtracking is required for resolution completeness. Our approach is demonstrated on a real robotic platform, Justin at DLR, and in a simulation of the same robot. In the latter, we consider the consequences of geometric backtracking.

Introduction

The robot Justin, which has been developed at the institute of Robotics and Mechatronics at the German Aerospace Center (DLR) in Oberpfaffenhofen, is an advanced humanoid robot with two arms with four-fingered human-like hands, a head with two video cameras for stereo vision, and a base with four wheels mounted on extensible legs. The upper body of Justin has 43 degrees of freedom: 7 degrees of freedom for each arm, 12 for each hand, 3 for the torso and two for the neck (Ott et al. 2006).

Justin, like other complex manipulators, was until recently dedicated to performing only tasks involving prespecified objects and action sequences, at least at an abstract level. In this article, we present the ongoing efforts to provide Justin with a higher degree of autonomy within the scope of the EU FP7-project GeRT (see `http://www.gert-project.eu`). Here, we focus on planning for tasks, but there is also work in the project on perception and grasping. The overall aim of the project is that Justin should be able to generalize from existing programs for specific tasks and from known objects of certain classes, to perform new tasks consisting of the same types of basic operations but combined in new ways, and with new objects belonging to the same classes. In this paper, we report on a prototype of a planner for Justin. Of course, this problem is not limited to Justin, but applies to other advanced robotic systems as well.

There is a large body of work on task planning which represents the world in logical terms (Nau, Ghallab, and Traverso 2004). But such representations are insufficient for modeling the kinematic and geometric properties of a robot such as Justin and its environment. When planning with Justin, one must take into account how it can move its arms and hands, how it can grasp different objects, and whether there are obstacles that can block its movements.

There is also a large body of work on path and motion planning (LaValle 2006). These algorithms plan in continuous state spaces, and include kinematic (or even dynamic) models of the robotic system as well as models (often in terms of polyhedrons) of obstacles.

While a path planner can find collision-free paths for various movements, it is not able to decide whether such a movement is a step in solving a complex task. A path planner is in general incapable of the kind of means-end reasoning that a task planner can do.

What Justin needs is a combination of task and path planning. It could be achieved by first solving the task planning problem and then for each action in that plan solving the corresponding path planning problem. However, it might happen that no path can be found for an action in the task plan. For instance, the presence of obstacles such as a big box in the middle of the workspace, may render certain parts of the workspace inaccessible for one or other of the arms. Yet, the task planner has no way of determining that, and may generate plans where the wrong arm is chosen for e.g. picking up an object from such a position. Hence, a solution with task planning first and path planning after might result in plans that are invalid at the geometric level. Instead, task and path planning must be integrated. A hybrid approach is required.

In this article, we present the hybrid task and path planning system we have developed for Justin. It is the first time that hybrid planning has been used for such a complex robotic system. Previous work has mainly relied on simulation and sometimes very simplified models. Thus, the fact that we adopt hybrid task and path planning for a real twoarmed humanoid robotic system is the first contribution of this paper. We present how the planner works together with other components of Justin's software, in particular for perception and execution, and we provide details of how the planner works. In particular, we show how several different solvers for path planning, linear interpolation of paths, and inverse kinematics need to be combined in order to find paths. This is the second contribution of this paper. We also consider the issue of geometric backtracking. Choices of how to perform actions at the geometric level may have negative consequences for later actions. Geometric backtracking is the process of revisiting geometric choices in previous actions in order to be able to apply the action presently under consideration. For instance, if the task is to place two cups on a small tray, the first cup may be placed in the middle of the tray, leaving insufficient space for the second cup. When the action to place the second cup is found to be inapplicable, one needs to go back to the first place action and reconsider where the first cup is to be placed. We show how geometric backtracking is performed in our planner, including how alternative geometric choices are sampled. We also argue that geometric backtracking is necessary for achieving resolution completeness for the hybrid planner. This is the third contribution of the paper. Finally, we present a number of demonstrations performed on the Justin platform, and experiments on a simulated version of Justin. The former demonstrate that our approach actually works on a real robotic system, and the latter investigate the benefits and costs of hybrid planning and in particular geometric backtracking. The experiments include a large obstacle (which makes path planning essential), and a number of objects that are put in a limited space (requiring geometric backtracking). In the demonstrations and experiments, the robot only manipulates objects for which it has a priori models, which implies that grasping can be done with previously stored grasps. While this paper focuses on Justin, we believe that what can be learned from planning with Justin can also be applied to other advanced robotic systems.

Related work

The approaches to combining task and path planning we have encountered in the literature can roughly be divided into two categories, defined in terms of how the task and path planning components relate to each other.

Path planning guided by task planning. In these approaches, path planning is primary, and task planning secondary. The planners mainly work on a path planning problem, but there is also a symbolic interpretation of the domain which can be used to structure the path planning problem and determine where to direct the search. These approaches include aSyMov (Cambon, Alami, and Gravot 2009) and SamplSGD (Plaku and Hager 2010). I-TMP (Hauser and Latombe 2009) should also be mentioned here, although it strictly speaking does not involve a task planning algorithm but a given task graph which represents a set of potential plans. These approaches address path planning problems involving a number of movable objects and/or multiple robots and/or a robot with many links. Such path planning problems have high-dimensional configuration spaces. In order to reduce that dimensionality, the problem is divided into tasks or actions corresponding to lower-dimensional subproblems. Such actions can for instance be to move one single object to a specific position while all other objects remain in position. The role of the task planner is to determine what actions/subproblems are to be explored. For instance, aSyMov only invokes the task planner as a heuristic for selecting actions.

Task planning querying path planning. In these approaches, a task plan is generated, and some of the actions involve path planning problems which are solved by dedicated path planners. Each path planning problem is solved separately. These approaches include Guitton and Farges (2009), Alili et al. (2010), SAHTN (Wolfe, Marthi, and Russell 2010), semantic attachments (Dornhege et al. 2009a; 2009b), and HTN and motion planning (Kaelbling and Lozano-Perez 2010). Typically, specific clauses in the preconditions and/or effects invoke calls to a path planner. For instance, the semantic attachments represent a general approach to invoking external solvers. A precondition clause such as *([check-transit ?x ?y ?g])* may invoke a call to a path planner. Information about the current robotic configuration is encoded in the states of the task planner by terms *q1 ... qn* and the transformation matrix for the pose of object *o* is encoded by terms *p0(o) ... p11(o)*.

It is noteworthy that these approaches have rarely been applied to real robots. With the exception of I-TMP, which has been demonstrated on a climbing Kapuchin robot, they are (as far as we know) only tested on simulated systems or very simple robots.

Our approach for Justin belongs to the second category. Besides being aimed at an advanced real robot, it also distinguishes itself by using geometric backtracking: only an extended abstract by Alili et al. (2010) appears to address that topic before (and only briefly, so there is not sufficient information to make a comparison). However, the first category of planners such as aSyMov (Cambon, Alami, and Gravot 2009) can perform similarly by exploring multiple paths between states.

Task and path planning

In task planning (Nau, Ghallab, and Traverso 2004) a state s is a set of atomic statements $p(c_1, \ldots, c_n)$ where p denotes a property of or a relation between objects denoted by names c_i. An action a has preconditions P_a (a logical combination of statements) that specify in what states a is applicable, and effects E_a (for instance a set of literals) that specify how a state changes (i.e. what statements are added or deleted) when a is applied. A planning domain D consists of a set of actions, and a planning problem is comprised of a domain, an initial state s_0 and a goal formula g. A plan is a sequence of actions $P = (a_1, \ldots, a_n)$. The result of a plan is the state s obtained by applying the first action a_1 to get a state s', and then recursively applying the rest of the plan to s'. In a valid plan, each action a_i is applicable in the state obtained by applying the preceding actions $(a_1, \ldots, a_{i-1})$ starting from the initial state s_0. A solution to a planning problem is a plan P with actions from D which when applied to the initial state s_0 results in a state s in which the goal g is satisfied.

Fig. 1 shows an example of an action schema from one of the domains for Justin.

act: pick(h,g,o)
pre: empty(h) and graspable(o) and can-move-pick(h,g,o,τ)
eff: not empty(h) and grasped(h,o) and is-picked(h,g,o,τ)

Figure 1: Action schema for Justin, representing a set of *pick* actions. The parameter *h* indicates what hand to use (*left* or *right*), *g* indicates the type of grasp (e.g. *top*), *o* is an object (e.g. *cup1*), and τ represents a path to be followed by the hand *h* during a *pick* action.

Path planning (LaValle 2006), on the other hand, considers a continuous space. There is a world space $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$. The obstacles in the world space are defined by the obstacle space $\mathcal{O} \subseteq \mathcal{W}$. There is a robot which can be a rigid body $\mathcal{A}$ or a collection of connected links $\mathcal{A}_1, \ldots, \mathcal{A}_n$.

The configuration space $\mathcal{C}$ is determined by the various translations and rotations that the robot (or its links) can perform. $\mathcal{A}(q)$ is the space occupied by the robot transformed according to the configuration q (and equivalently for $\mathcal{A}_1, \ldots, \mathcal{A}_n$). $\mathcal{C}_{obs}$ is the obstacle region in the configuration space, defined as the set of configurations where the interior (int) regions of $\mathcal{O}$ and $\mathcal{A}$ intersect. $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$ is the free space where the robot can move.

A path planning problem is defined by the above entities (the domain), a start configuration $q_1 \in \mathcal{C}_{free}$ and a goal configuration $q_G \in \mathcal{C}_{free}$. A solution to a path planning problem as defined above is a continuous path $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$ where $\tau(0) = q_1$ and $\tau(1) = q_G$. This is the most basic version of the path planning problem. There might for instance be parts (objects) that the robot can transport, or there might be multiple robots. In the case of Justin, the path planning problems addressed concern one 7-degree arm at a time, and any transported parts simply follow the hand. Hence, $\mathcal{C}$ for each path planning problem effectively consists of 7 parameters. The obstacle space $\mathcal{C}_{obs}$ comprises the table surface, objects on the table, and the other arm.

Task planning and path planning representations can be linked through the object names and atomic statements at the task planning level: certain names correspond to parts, positions or regions at the path planning level, and certain statements correspond to properties of or relations between the parts and/or the robot. A state s then consists of a symbolic component σ_s (a set of statements) and a geometric component γ_s (containing the current configuration and poses of objects). The preconditions P_a of an action a can refer to both the symbolic and geometric state components, and the effects E_a can alter them both.

In Fig. 1, the predicates *can-move-pick* in the preconditions and *is-picked* in the effects are geometric: the former concerns the existence of a path τ in γ_s from the present arm configuration for the selected arm to one where the selected object can be picked, and the latter updates γ_s to such a target configuration.

A statement that is interpreted geometrically can be true in many different geometric states. This implies that a geometric effect can be realized in many different ways. For instance, if an object o is to be positioned in a certain region r, then $in(o, r)$ can be achieved by a set of different poses which is constrained by the borders of r, the presence of other objects in r and so on. Hence, an action with an effect such as $in(o, r)$ can be implemented in many different ways geometrically. In addition, how it is implemented may affect subsequent actions. For instance, if a later action has the effect $in(o_2, r)$, it will be constrained differently depending on the selected pose for o, and in some cases may even be infeasible.

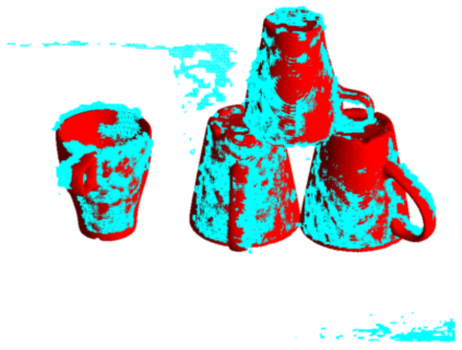

Figure 2: Point clouds (cyan) from a Kinect camera together with best matching models (red).

System overview

Here, we present the relevant modules of Justin: perception, world model, planning and execution.

The *perception module* is based on analysis of point clouds obtained from stereo processing or some other range sensor. Given geometric models of possible objects in the actual scene, an interpretation of that scene in terms of these objects is computed. In a first step, a set of hypotheses (several tens) is computed for each object through pose clustering (Hillenbrand 2008; Hillenbrand and Fuchs 2011). In the second step, these hypotheses are tested by aligning the object models with the data (Fig 2) and scoring the inliers by proximity to the model surface and similarity of surface orientation. Finally, collisions of models are detected and lead to pruning the hypotheses with the lowest scores.

The *world model* receives information from perception about classes, shapes and poses of objects in the scene. Presently, the world model has access to polygon mesh models of the corresponding example objects. When novel objects are introduced, these will have to be generated online from the point clouds obtained from range sensing. In particular, the objects' poses come with some uncertainty, and the robot's actions need to be robust enough to compensate for that.

In addition, the world model has access to a set of grasps for each object. These grasps are represented in terms of configurations for the individual fingers and the relative pose of the *tool center point* (TCP) which is roughly in the center of the wrist.

The *planner* queries the world model in order to get geometric information about the planning problem addressed. From the world model, it can construct an initial geometric state with the 3D models of the objects positioned in the correct poses. Purely symbolic information is given in a problem file, as is the goal. Objects in the geometric state are automatically given names from the symbolic states.

Next, the planner searches for a plan: how that is done will be described in the next section. If successful, the plan is used to generate a robot program in the form of a sequence of Python scripts. Each action model in the domain corresponds to one parameterized script segment, and each action in the plan generates one segment in the final script, instantiated with the appropriate objects and poses, grasps and possibly also paths. The scripts may contain, among other things, arm motions to specific frames, arm motions according to a given path, finger motions to given configurations, guards for specific conditions such as resistance due to contact with some object, perceptual operations such as looking for a specific object, and exception handling. These scripts are then executed in the Justin execution environment. Python was already extensively used in the execution environment, and provides an expressive and efficient high-level interface language between planning and execution. Fig. 3 shows one such Python script. In the future, we intend to add execution monitoring and recovery techniques to the system.

The planner

Our hybrid task and path planner is based on forward chaining task planning in combination with bidirectional rapidly exploring random tree (RRT) planning (LaValle 2006). Currently, we are using the hierarchical task network planner JSHOP2 (Nau et al. 2003) as the task planning component; it was chosen because it is a progressive planner and searches among fully specified states. From the perspective of the task planner, two modifications are made:

- The state is augmented with a geometric component, which contains information about the (predicted) configurations of the robot and of any movable objects, as well as their shapes (the latter are the same for all states).
- Atomic statements with certain predicates are not evaluated in the symbolic component of the state but in the geometric one. When such a statement is encountered while testing a precondition of an action, a method is called that evaluates whether it is true in the geometric state component. When a statement with a geometric predicate is encountered while adding the effects of an operator, a method is called which updates the geometric state accordingly.

Thus, the application of an action results in updating both the symbolic state, by adding/removing statements, and the geometric state, by invoking the associated methods. Notice, that the interaction between task and path planning occurs exclusively through the geometric predicates, in the pre- and postconditions of operators and possibly when the goal is evaluated. Thus, the only modifications of the task planner are how preconditions and effects are applied, and the inclusion of a geometric component into the state.

A method for the geometric state may be of two kinds. It may involve a simple computation, e.g. if it concerns the position of a certain object. It may also involve a more complex computation such as searching for a path in the configuration space of one of the arms. The latter is done as follows for a statement with the predicate *can-move-pick*:

1. The present configuration in the geometric state is the initial configuration for the path planning.
2. The goal configuration is computed by first determining a desired pose for the tool center point of the selected arm. There are typically several alternative grasps and hence several TCP poses that constitute a sample. An inverse kinematic solver for Justin's arms then computes a set of arm configurations that puts the tool center point in the desired pose. These are tested for collisions, and one of those found to be collision free will be the goal configuration.
3. Inverse kinematics is also used to generate a configuration at some distance from the object, and this will be the approach configuration. Passing through this configuration reduces the risk of failing the grasp due to e.g. unexpected collisions between the hand and the object.
4. A bidirectional RRT planner attempts to find a collision-free path for the arm between the initial and approach configurations. The RRT planner employs a forward-kinematic model for projecting Justin's arm into the work space.
5. A path between the approach configuration and the goal configuration is computed by linear interpolation.
6. The fingers are closed according to a grasp-specific configuration.
7. If a path is found, it is stored for later use, and the statement is considered true in the state.

The predicate *can-move-pick* is used in preconditions. The corresponding effect predicate, *is-picked*, updates the geometric state such that the selected arm and hand are set to the target configuration generated by *can-move-pick*. In addition, the grasped object is constrained to follow the hand.

For the predicate *can-move-place*, which is used when grasped objects are moved to a new position, the sequence is: select a target pose (there may be many, if the target is an extended area), compute inverse kinematics for this pose, compute inverse kinematics for an approach pose and a lift pose, do linear interpolation between the start and lift configurations, call the RRT planner for a path between the lift and approach configurations, and do linear interpolation between the approach and target configurations.

As mentioned before, each type of action is associated with a parameterizable Python script that can be executed on the robot. The parameters include paths found during path planning, and these are subject to smoothing in the script before they are executed. The scripts may also contain guards in order to detect e.g. when an object that is to be placed has contact with the table. The scripts for the actions in the plan

```
### user header 'pyrs_source'
# move hand to pregrasp config with side left, type top and object 4
path0 = [[0.2443, 0.0873, 0.1745, -0.0, 0.0, 0.6109, -0.0, 0.0, 0.5236, -0.0, 0.0, 0.6109]]
execute_path(path0, path_is_for_manipulators=['left_hand'])
# move arm to REAL pre-grasp (RRT-path which needs shortening):
path1_1 = [[-0.7907, -1.4714, 0.239, 1.6179, 0.7105, -0.6163, 0.6261],... ]
execute_path(path1_1, path_is_for_manipulators=['left_arm']) # <-- with joint path shortener!
# move arm to REAL grasp (along a Cartesian line without shortening):
path1_2 = [[-0.9226375374193561, 1.3244301190878176, 0.9000000000000015, 1.473463845957323,
2.441791818050337, 0.8452778932497086, -0.12911968091025275],...]
execute_path(path1_2, path_is_for_manipulators=['left_arm'], skip_optimization=True)
# grasp it!
path2 = [[0.2443, 0.2793, 0.3142, -0.0, 0.384, 0.6109, -0.0, 0.384, 0.5236, -0.0, 0.384, 0.6109]]
execute_path(path2, path_is_for_manipulators=['left_hand'])
rave.bind('leftArm', 'mug1_1')
execute_path(path1_2[::-1], path_is_for_manipulators=['left_arm'], skip_optimization=True)
exit('out')
```

Figure 3: Python script generated by planner for a grasping action (paths have been truncated). Note the different phases: pregrasp configuration for hand and then for arm, grasp configuration for arm, and actual grasp with hand.

are then executed in a sequence, making the robot perform the plan.

Geometric backtracking

When the planner selects an action, it not only chooses the type of action and what objects and locations are involved. It also needs to make geometric choices that are not visible to the task planner, but are related to the interpretation of certain geometric statements. These might be the exact TCP to use when grasping an object as in the statements *can-move-pick(h,g,o,τ)* and *is-picked(h,g,o,τ)*. These might also concern the exact pose when an object is put down at a given location, as in *can-move-place(h,g,o,τ)* and *is-placed(h,g,o,τ)*. As mentioned before, such choices may very well affect the applicability of actions later on. However, these choices are done locally, and are not informed about constraints imposed by subsequent actions. Hence, it is important that they can be reconsidered. Otherwise, the planner would be incomplete relative to its sampling at the geometric level. For instance, consider the example with the small tray and two cups from the introduction. If the planner only tries to put the first cup in the center of the tray (this might be the first sampled placement), then it will never find a placement for the second cup and will ultimately fail to find a plan.

To sample statements with geometric predicates in a systematic manner, we use the van der Corput and Halton sequences (Kuipers and Niederreiter 2005). These sequences guarantee a uniformly distributed sampling over $[0,1]^n$, and can straightforwardly be used to sample a bounded n-dimensional space.

Backtracking occurs along a single sequence of actions/states

$$(a_n, s_n, a_{n-1}, s_{n-1}, \ldots, a_{n-k}, s_{n-k})$$

where a_n is the most recent action. Other parallel search branches at the task planning level are unaffected. Backtracking is triggered when a_n is not applicable because some particular geometric statement related to motion was false. The most recent van der Corput indices that were used for geometric sampling for each action are also maintained: $(i_n, i_{n-1}, \ldots, i_{n-k})$. The most recent index i_n is incremented, giving a new geometric sample for the previously failed geometric predicate. If that fails, the procedure is repeated. If a maximal value for the index has been reached, it is reset to 0, and we move up one step to a_{n-1} and increment its index i_{n-1}, giving a new geometric sample at that level and for the relevant geometric predicates there. If successful, we update the geometric state γ_{s_n} and move downwards to a_n and i_n again. If after repeated failures at a_{n-1} the index i_{n-1} reaches its maximal value, it is reset to 0 and we move up yet another level and so on, in a recursive manner. Fig. 4 shows an example of geometric backtracking.

In our current implementation, we use the van der Corput sequence to sample (1) orientations of TCP relative to target object for the *can-move-pick* predicate where the domain $[0,1]$ of the van der Corput sequence is mapped to $[0,2\pi]$ (radians), and (2) position (*x* and *y*) and (optionally) orientation for the *can-move-place* action. In general, the sampling schema is built into the interpretations of the various predicates.

Another important factor is the sample size, which determines the resolution at the geometric level. Presently, we have a fixed size for each predicate. This is basically where we define the balance between task planning and path planning: a large sample size will effectively result in more effort being spent on path planning and inverse kinematics tests for each action.

Overall, we consider how to perform the sampling and how large to make the sample size as central questions in hybrid task and path planning. This issue will be discussed further within the context of our experiments, where we also give some concrete examples of the utility of geometric backtracking.

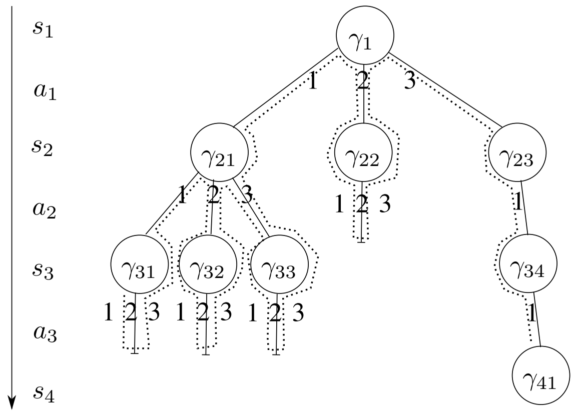

Figure 4: Geometric backtracking. The arrow shows the temporal order of the (partial) plan being explored. The circles with γ_k are different geometric states generated that belong to the hybrid states s_k (the symbolic components of the latter do not change), a_k are actions with geometric preconditions and/or effects, and the numbers on the lines are van der Corput indices that are used for sampling. The lines ending with a horizontal stroke are failed attempts to satisfy the corresponding p_k. The dashed line shows the order of traversal, starting from γ_{31} with the failed application of action a_3 and ending successfully in γ_{41} with the same action. Note, that we always consider the same sequence of actions: what varies is the sampling at the geometric level.

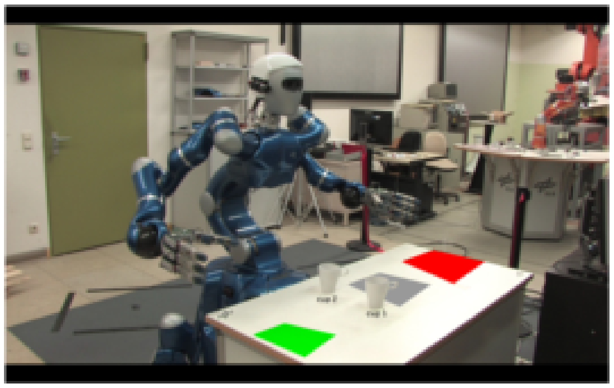

Figure 5: The setup of the first two scenarios. The regions where the cups should be placed are indicated.

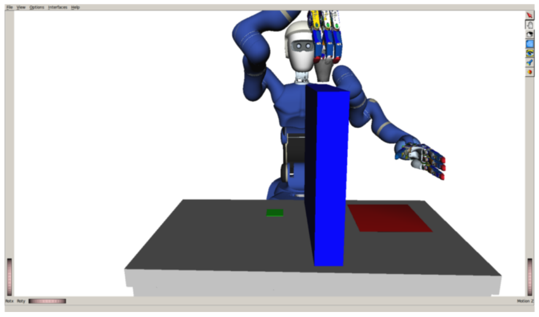

Figure 6: Simulated Justin using the top of the box as a temporary placement for the cup during the first series of experiments (P1).

Demonstrations on real Justin

A number of demonstrations have been performed on the real Justin robot. The purpose with these demonstrations is to provide evidence that our hybrid planning approach is relevant to a real robot, and not just limited to simulation. They show how we have connected perception, planning and execution on the Justin robot.

We worked on two scenarios where small cups are to be manipulated by Justin. The initial geometric state was the same: 2 cups were placed on the table in front of Justin (see Fig. 5). Information about this geometric state was obtained through perception, as explained above. The planner had no a priori information about the geometry, but knew what objects would be present.

In the first scenario, the 2 cups are to be placed in a region on the table on the left side of Justin, which is shown with red in Fig. 5. Justin plans and successfully executes 4 symbolic actions with the left arm:

pick(left,top,cup2), place(left,red-region,cup2), pick(left,top cup1), place(left,red-region,cup1).

Of course, this only shows the symbolic actions. In reality, the plan also includes specific paths for the arm and specific grasps, and those imply specific poses for the objects.

In the second scenario, the 2 cups are to be placed in two different regions on the table. *Cup2* is to be placed in a region on the right side of Justin (green). *Cup1* is to be placed first in a region in front of Justin (grey) and then it is to be placed in a region on the left side of Justin (red). Justin plans and successfully executes 7 symbolic actions involving both arms:

pick(right,top,cup2), place(right,green-region,cup2), pick(right,top,cup1), place(right,gray-region,cup1), move-hand-away(right,cup1), pick(left,top,cup1), place(left,red-region,cup1).

Note how *cup1* is handled by both hands, and how the right hand is moved away from it before it is grasped by the left hand.

Experiments in simulation

In addition to the demonstrations on the real Justin, a number of experiments in simulation have been conducted. The aims of these experiments are (1) to test the planner with more challenging tasks, and (2) to explore the benefits and costs of using geometric backtracking. The planner was running in Java 1.6.0 RTE with 32-bit native libraries used for collision detection (VCOLLIDE), inverse kinematics, and forward kinematics computation. The computer had an Intel CORE i5 vPro processor (2.5GHz, 64 bits, 2 cores, 3MiB for the cache memory), 8 GB memory and Linux kernel 2.6.38.

The first series of experiments involves a large box at the center of the table. The presence of this box causes repeated failures of the path planner when objects placed near the center are to be moved from one side of the table to the other. However, the box can also be used for temporarily placing objects (Fig. 6), and this gives the robot an opportunity to shift hands. Hence, the following plan works well for moving a single cup from one side to the other:

pick(right,top,cup1), place(right,box-top,cup1), move-hand-away(right,cup1), pick(left,top,cup1), place(left,red-region,cup1)

Table 1 presents the results from these experiments (the three lines marked P1, with varying sampling resolution).

What is striking is the amount of time spent on geometric reasoning (inverse kinematics and path planning): it is several orders of magnitude more than the time spent on task planning. We present average path planning times with four decimal precision only to show how little time is spent on task planning. Also note the number of times the geometric reasoning fails to find a solution. This indicates that if we had solely relied on task planning, the risk of obtaining a plan that was not executable due to obstacles and kinematic constraints would have been considerable. Inspections of the logs of the planning process confirm this.

The problem was solved with varying resolution in the sampling for the geometric backtracking. Lower values for resolution where also tried, but then the planner often failed to find a solution. Not surprisingly, the choice of resolution strongly influences the total planning time, and the number of geometric configurations considered. Most of the time by far was spent on geometric backtracking (compare columns *#conf* and *#fail conf* to *#conf bt* and *#fail c bt*).

The second series of experiments involves moving a number of cans (shaped as cylinders) onto a tray positioned on the table. Each tray had a fairly small area which requires planning when putting more than one can on it. We varied both the number of cans — 2, 3 or 4 — and the size of the trays: they could just fit 2, 3 or 4 cans, respectively. We made a series of runs where the position of one cup can trigger a geometric backtracking when later cups are placed. Without geometric backtracking, the problems could not be solved. The planner would simply have considered the second (or third) placement action as inapplicable, and would have backtracked at the task planning level.

The following is a plan generated for putting three cans on a tray large enough for three cans (P3).

pick(right,top,can1), place(right,tray,can1), pick(right,top,can2), place(right,tray,can2), pick(right,top,can3), place(right,tray,can3).

Table 1 shows the results of the experiments. The problems are: P2 with 2 cans and a tray for 2, P3 with 3 cans and a tray for 3, P4 with 4 cans and a tray for 4, P5 with 2 cans and a tray for 4, and P6 with 3 cans and a tray for 4. Again, a considerable amount of time is spent on geometric reasoning and especially backtracking. The task planning problem, on the other hand, requires comparatively little effort. However, for P5 and P6 (with plenty of extra space on the tray), there is none or little backtracking.

A general problem appears to be to determine in advance how much geometric backtracking (if any) is needed. This is due to the complexity of the problem: each arm is a 7-degrees of freedom system with complex kinematics, there are constraints on how the objects can be grasped, there are obstacles that constrain movements (including other objects that can be moved), and the goal positions can be constrained in different ways. Hence, it may be a good idea to for instance iteratively increase the resolution instead of setting a fixed level. This also applies to the maximal number of nodes for path planning. We should also point out that the way we sample at the geometric level may not always be optimal. For instance, in order to fit several objects into a constrained area, it might be better to focus the sampling near the borders of the (remaining) space.

Summary and conclusions

In this article, we have presented a prototype of a hybrid task and path planning system for a humanoid two-arm robotic system. It is as far as we know the first time hybrid task and path planning has been applied to such an advanced robotic system. The planning system integrates a task planner and several methods for generating paths: a bidirectional RRT planner for longer movements and linear Cartesian interpolators for shorter approach and lift motions. The interface between task and path planning consists of a number of geometric predicates that can occur in the preconditions and effects of actions. We have presented how the planner works together with other components of the robot's software, and we have demonstrated that we have a working system. We also consider the issue of geometric backtracking: the process of revisiting geometric choices in previous actions in order to be able to apply the action presently under consideration. Simulated experiments have been made to illustrate the utility of geometric backtracking. We think that efficient methods for geometric backtracking (in particular how to sample and how much to sample) are vital for achieving efficient hybrid planning, and much remains to be done.

The current system has some limitations. First, there is no grasp planning available yet, which restricts us to a priori known objects for which there is a set of directly applicable grasps. Second, although some robustness towards uncertainty has been built into the path planning and the produced Python scripts, a monitoring and recovery component would also be needed. Both these issues will be addressed in the near future.

Acknowledgements The project receives funding from the European Community's Seventh Framework Programme under grant agreement no. ICT- 248273 GeRT.

References

Alili, S.; Pandey, A. K.; Sisbot, E. A.; and Alami, R. 2010. Interleaving symbolic and geometric reasoning for a robotic assistant. In *ICAPS Workshop on Combining Action and Motion Planning*.

Cambon, S.; Alami, R.; and Gravot, F. 2009. A hybrid approach to intricate motion, manipulation and task planning. *Int. J. Rob. Res.* 28(1):104–126.

Problem	Time	#plans	#sym bt	#acts	Time path	#conf	#fail conf	#conf bt	#fail c bt	Resol
P1	17.3638	10	5	5	17.3597	225.2	1087	215.2	950.8	6×60
P1	5.4522	10	5	5	5.4479	73.6	221.8	63.6	93.4	2×60
P2	4.7092	4	0	4	4.7053	26.1	2484.1	23.1	2362.9	16×120
P2	1.7159	4	0	4	1.7129	13.6	484.4	10.6	423.4	6×60
P2	0.7986	4	0	4	0.7962	8.4	82	5.4	50.9	2×30
P3	11.9506	6	0	6	11.9467	59.2	6064.6	55.2	5822.6	16×120
P3	4.1042	6	0	6	4.1006	30.4	1253.7	26.2	1143.1	6×60
P3	2.2240	6	0	6	2.2215	19.6	461.8	15.6	339.8	2×60
P4	1056.9276	8	0	8	1056.9241	4193.7	459769.7	4186.7	459564.7	16×120
P4	154.1280	8	0	8	154.1246	716	37909.4	709.1	37819.7	12×60
P4	15.0828	8	0	8	15.0792	84	3647	77	3562	6×60
P5	0.3184	4	0	4	0.3162	4	9	0	0	16×120
P5	0.3420	4	0	4	0.3393	4	9	0	0	6×60
P5	0.3175	4	0	4	0.3145	4	9	0	0	2×30
P6	0.7151	6	0	6	0.7121	6.1	89.4	0.2	2.2	16×120
P6	0.5324	6	0	6	0.5284	6.1	30.5	0.2	2.3	6×60
P6	0.6374	6	0	6	0.6349	6.6	40.6	0.7	12.4	2×60

Table 1: Results from simulated experiments. *Time* is the time in second spent during the whole planning process (both task and path planning). *#plans* is the number of partial symbolic plans that were visited during the whole planning process. These plans consist of primitive actions. *#symb bt* is the number of times the task planning process has backtracked. *#acts* is the number of symbolic primitive actions of the solution plan. *Time path* is the time in second spent on path planning. *#config* is the number of geometric configurations that were visited during the path planning process. *#fail conf* is the number of failed attempts to create geometric configurations. *#conf bt* is the number of geometric configurations that were visited during geometric backtracking. *#fail c bt* is the number of failed attempts to create geometric configurations during geometric backtracking. *Resol(ution)* indicates the number of grasps that are tried to pick an object (cup or can) and the number of poses that are tried for the place actions. Each line presents the average of 10 runs. The RRT planner had a limit of 500 nodes per attempt.

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009a. Semantic attachments for domain-independent planning systems. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS09)*, 114–122.

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

Guitton, J., and Farges, J.-L. 2009. Taking into account geometric constraints for task-oriented motion planning. In *Proc. Bridging the gap Between Task And Motion Planning, BTAMP'09 (ICAPS Workshop)*.

Hauser, K., and Latombe, J.-C. 2009. Integrating task and PRM motion planning: Dealing with many infeasible motion planning queries. In *Proc. Bridging the gap Between Task And Motion Planning, BTAMP'09 (ICAPS Workshop)*.

Hillenbrand, U., and Fuchs, A. 2011. An experimental study of four variants of pose clustering from dense range data. *Computer Vision and Image Understanding* 115:1427–1448.

Hillenbrand, U. 2008. Pose clustering from stereo data. In *Proceedings VISAPP International Workshop on Robotic Perception (RoboPerc 2008)*, 23–32.

Kaelbling, L. P., and Lozano-Perez, T. 2010. Hierarchical planning in the now. In *Proc. of Workshop on Bridging the Gap between Task and Motion Planning (AAAI)*.

Kuipers, L., and Niederreiter, H. 2005. *Uniform distribution of sequences*. Dover Publications.

LaValle, S. M. 2006. *Planning Algorithms*. Cambridge, UK: Cambridge University Press.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research, special issue on the 3rd International Planning Competition* 20:379–404.

Nau, D.; Ghallab, M.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Ott, C.; Eiberger, O.; Friedl, W.; Bäuml, B.; Hillenbrand, U.; Borst, C.; Albu-Schäffer, A.; Brunner, B.; Hirschmüller, H.; Kielhöfer, S.; Konietschke, R.; Suppa, M.; Wimböck, T.; Zacharias, F.; and Hirzinger, G. 2006. A humanoid two-arm system for dexterous manipulation. In *Proceedings IEEE-RAS International Conference on Humanoid Robots (Humanoids 2006)*, 276–283.

Plaku, E., and Hager, G. 2010. Sampling-based motion planning with symbolic, geometric, and differential constraints. In *Proceedings of ICRA10*.

Wolfe, J.; Marthi, B.; and Russell, S. J. 2010. Combined task and motion planning for mobile manipulation. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS10)*, 254–258.