Fast Temporal Projection Using Accurate Physics-Based Geometric Reasoning

Lorenz Mösenlechner
Intelligent Autonomous Systems
Technische Universität München, Germany
moesenle@cs.tum.edu

Michael Beetz
Institute for Artificial Intelligence
University of Bremen and TZI*
beetz@cs.uni-bremen.de

***Abstract*— Temporal projection is the computational problem of predicting what will happen when a robot executes its plan. Temporal projection for everyday manipulation tasks such as table setting and cleaning is a challenging task. Symbolic projection methods developed in Artificial Intelligence are too abstract to reason about how to place objects such that they do not hinder future actions. Simulation-based projection is fine-grained enough but computationally too expensive as it is not able to abstract away from the execution of uninteresting actions (such as navigation). In this paper we propose a novel temporal projection mechanism that combines the strengths of both approaches: it is able to abstract away from the execution of continuous but uninteresting actions and provides the realism and fine grainedness needed to reason about critical situations.**

I. Introduction

When robots perform everyday manipulation activities such as setting the table or cleaning up, the difficulty of individual pick and place tasks critically depends on where the robot exactly places the objects and on the order in which the robot puts and takes away objects. If the robot places objects too early or at inappropriate places, these objects might hinder subsequent pick and place tasks.

Robot action planning [1], which is the computational task of deciding on the course of action based on predicting and reasoning about the future consequences of the intended plan, aims to avoid such problems. Unfortunately, symbolic action planning methods, mostly based on PDDL (Planning Domain Description Language) derivates ([2], [3]), are too coarse grained to be informative with respect to whether a reaching or placing action is easy or challenging [4]. Motion planning ([5], [6]) is fine grained enough but very limited in the ways to reason about how scenes can be manipulated in order to simplify actions (but see [7] for a notable exception). Several approaches define more realistic models of world states that consider reachability as the existence of a motion plan [8] or models that are learned from experience [9] but these approaches are limited to navigation actions. Simulation-based projection methods ([10], [11]) are fine grained enough to predict and reason about such effects but they are very resource intensive.

In this paper, we propose a novel robot plan projection method that combines abstract symbolic projection for actions with detailed geometric and simulation-based reasoning for handling the action aspects described above. The system we present allows for inferring plan parameters such as locations for putting down objects under the constraints defined by the current task and future actions. Instead of projecting a plan by applying a sequence of logical rules that update a symbolic world state, our system uses a geometric representation of the world to calculate predicates such as *Visible* and *Reachable* on demand. The whole system is integrated in CRAM [12] and is based on plans that are specified in the CRAM plan language. Similar to purely symbolic approaches for projection, actions are implemented as symbolic rules for updating the internal representation of the world. However, by using a geometrically accurate 3D representation of the world our approach provides a good compromise between fast but very abstract purely symbolic projection and computationally expensive but very accurate simulation of complete actions. This paper is based on the work presented in [13]. The main contributions are extensions to support temporal reasoning based on time lines, the execution of plans in projection to generate these time lines and the definition of behavior flaws.

This paper is structured as follows. First we give an overview of our system that is capable of generating plan parameters such as locations for placing objects or where to stand not only based on static assumptions of the world but based on the future course of actions. Then we introduce the physics-based reasoning engine that we use, followed by an explanation of reasoning about plans, time line generation and the temporal calculus we use to represent behavior flaws. Finally we demonstrate the expressiveness the system by defining various behavior flaws.

II. Related Work

Related publications in the area of planning using geometry and physics simulation include [14] where the authors integrate a physics engine to calculate state transitions in planning. In [15], the authors combine symbolic and geometric planning by calculating predicate values for a high level planner using a geometric planner. However, the authors in both publications do not integrate high-level concepts such as reachability or visibility. Planning under uncertainty, integrating a geometric representation of the world including free and occluded areas has been shown in [16]. In [17], visibility simulation is integrated into a motion planner and in [18] the authors show similar visibility calculation as presented in this paper in the context of human-robot interaction and perspective taking.

*The Institute of Artificial Intelligence is part of both the Unversity of Bremen and the Centre for Computing Technologies (TZI).

While all these publications show the implementation of reasoning in geometric domains, none of them provides means for *generating* parameters such as destination poses for objects or poses for the robot to stand. In [19], the authors present capability maps, i.e. pre-calculated maps to quickly check if poses are reachable or where to place a robot to be able to reach a specific pose.

Temporal projection is a well studied field in formal logic. Given a sequence of actions, temporal projection tries to infer the state of the world after executing these actions. In [20] and [21], the authors deal with the problem of uncertain knowledge about the initial state of the world. McDermott introduces the generation of time lines, similar to the work presented in this article. However, purely symbolic approaches are often too abstract to represent geometric properties of the world, for instance occlusions. While simulation based projection as presented in [11] and the authors' previous work in [10] provide similar functionality for geometric reasoning and reasoning about actions as presented in this paper, they suffer from high computational complexity and extremely long run times.

III. System Overview

The purpose of the system described in this paper is to generate poses for placing the robot's base while manipulating objects and for placing objects, for instance on a counter. The system not only uses the current state of the world to generate such poses but also takes into account future actions of the current plan.

In CRAM plans, locations, objects and actions are specified using designators, instances that are built from conjunctions of symbolic constraints specified as key-value-pairs. For instance, we specify a location for a plate on the counter that is visible and reachable for the robot as follows:

(a location (for plate) (on counter) (reachable-for robot) (visible-for robot))

To resolve these designators, they are compiled into Prolog programs and solutions for them are generated by executing them. To improve readability, instead of writing actual Prolog programs, in this paper we will use a first-order representation that is equivalent to the corresponding Prolog code. The above designator is compiled to the following logical expression:

$$\begin{aligned}&\text{PosesOn(Counter, Cup, ?Poses)} \wedge \text{Member(?Pose ?Poses)}\\ &\quad \wedge \text{AssertPose(Cup, ?Pose)} \wedge \text{Reachable(Cup, Robot)}\\ &\quad \wedge \text{Visible(Cup, Robot)}\end{aligned}$$

The *PosesOn* predicate generates candidate poses by drawing samples from a probability distribution that is built from the designator. If all predicates hold for such a candidate pose, it is a valid pose under the constraints defined in the designator and it is considered a solution. In addition to explicitly defined constraints, as shown in the example above, a number of implicit constraints need to hold and are added to the Prolog program. For instance, an object of interest must stand stable at a put-down location, i.e. it must not be in collision with any other objects. In contrast to classical Prolog, the predicates used in designator resolution are computed by using an accurate simulated 3D representation of the world as a Prolog database. For instance, if the Prolog inference process needs to prove if the *Stable* predicate holds for a specific object instance, a physics simulation is used to check if the object is moving at its current location. If not, it is considered stable and the stable predicate holds.

While proving predicates in a static world database finds locally valid solutions, it does not take into account future actions and thus cannot infer if a specific solution negatively influences the performance of future actions. However, by projecting a plan, and analyzing the generated time line, we can evaluate if specific designator solutions will cause problems in future actions.

Projection is the execution of a plan in a lightweight simulation environment. It generates a time line that allows for reconstructing different world states along the course of actions that can be reasoned about. By matching behavior flaws, i.e. logical definitions of conditions in the world that should be avoided on these time lines, we can define an objective function and evaluate the quality of different solutions of a specific designator.

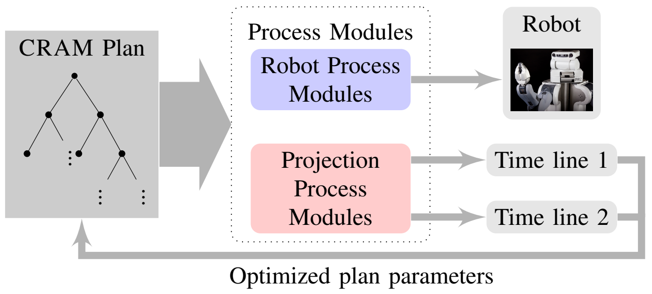


Fig. 1. System overview of plan projection for designator resolution.

Let us consider a simple example: the robot's task is to grasp a cup on the counter and put it on the table and grasp a plate that is on the table and place it on the counter. Apart from being on the counter or table respectively, the put-down locations are unspecified. However, good solutions will increase plan performance. For instance, if the put down location for the first object is close to the other object, the robot does not need to move to grasp the second object. However, the put down location of the cup should not be too close or in front of the plate since it would block the pick up action because the plate cannot be detected by perception or it cannot be grasped because the grasp position needed by the pick-up action is blocked by the cup.

Figure 1 gives an overview of the system design. When the system starts to execute a plan, a number of projections are started in parallel. Each projection spawns off from the current execution and world state, i.e. if the cup has already been grasped, all subsequent projections start with the cup in the robot's gripper. Each projection generates a time line. The time line contains events that indicate changes in the simulated world and for each event, we store a copy of the simulated world state for later reasoning and analysis. In addition to this time line, a separate task tree is stored for each projection that contains information about plan execution such as which task has been executed when and why, what the task's status at a given time was and what the outcome of the task was, including information about failures. When one projection thread finishes, the time line is evaluated by matching predefined flaw specifications. Based on the result, a performance value for all location designator solutions is generated. If the solutions in one episode are better than the previous ones, they are cached and used by the actual plan that is executed on the robot. Examples for behavior flaws include the distance the robot has to drive between actions, if objects that are to be manipulated are visible and if objects are blocking other objects.

IV. Physics-Based Reasoning

Projection and inference of flaws is based on a physics based reasoning engine as explained in [13]. Instead of proving all predicates based on a purely symbolic fact base, the truth values and bindings of certain predicates are calculated by using the Bullet physics engine, OpenGL off-screen rendering and inverse kinematics calculation and simplified inverse reachability map calculation. We use physics based reasoning instead of a purely symbolic knowledge base because in our domain of a human household, a symbolic representation would be too abstract. Many problems could not be solved. For instance, if an object is visible or not for the robot not only depends on the current location of the robot and its distance from the object but also on all other objects and their shape. In addition, since we need to find solutions for location designators, we need to have a generative model that yields poses that are valid solutions for the respective designator. We solve this by first drawing random samples based on a probability distribution that is generated from the designator's constraints and then using the physics based reasoning engine to prove the validity of such a sample, considering all explicit and implicit constraints of the designator.

For reasoning, we use a three-dimensional representation of the environment the robot operates. This representation contains all information the robot has about the environment. All static objects such as cupboards, the refrigerator and counter tops are provided by a semantic environment map as described in [22]. The robot's perception routines assert objects in the database when they are seen. The world database is kept consistent by removing objects that should have been visible but could not be seen anymore. The robot is equipped with a huge database of 3D meshes for objects including the PR2's household objects database [1].

To explain the inference process, let us consider a simple location designator for placing a cup on the counter top at a location where it is visible for the robot from its current location. The following first-order logic expression shows how that designator can be resolved:

$$\text{PosesOn(Counter, Cup, ?Poses)} \wedge \text{Member(?Pose, ?Poses)} \\ \wedge \text{AssertPose(Cup, ?Pose)} \wedge \text{Visible(Cup, Robot)}$$

[1]http://www.ros.org/wiki/household_objects_database

The inference engine processes the different terms sequentially and tries to find bindings for yet unbound variables. If a predicate fails, the engine backtracks and retries with a different solution for an unbound variable until a solution has been found or fails if all possible bindings for a variable have been tried. The first step is to generate a sequence of pose candidates that are possible solutions for the designator. The predicate *PosesOn* generates a probability distribution as shown in Figure 2. Since all poses on the counter top might be valid solutions, each pose on the counter has equal probability. The *PosesOn* predicate binds the (virtual) list of all samples to the variable *?Poses*.

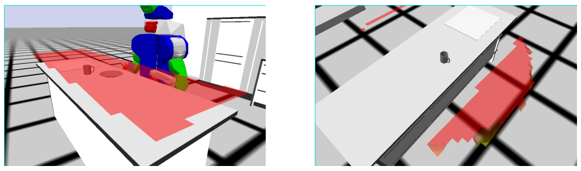


Fig. 2. Probability distributions used for sampling poses that are on the counter top (left) and for the robot to stand to reach the cup (right). The left distribution has an equal probability for all poses on the counter top while the right distribution is based on an inverse reachability map for the robot.

The next step is to draw one sample from the sequence of poses and to put the cup at the corresponding pose. Finally, the *Visible* predicate calculates if the object is visible for the robot at its current pose by rendering the scene from the position of the robot's camera and counting how many pixels of the object are visible.

For generating locations for objects and the robot and for inferring flaws on time lines, we define predicates for *collisions*, *stability* reasoning, *visibility* reasoning and *occlusions* and *reachability* reasoning and inference of objects that might be *blocking* a grasp action. The following list gives an overview of the predicates we implemented.

- *Contact(?W, ?O_1, ?O_2)* holds if the two objects *?O_1* and *?O_2* are in contact in the world *W*.
- *Stable(?W, ?O)* holds if all forces on object *?O* are canceled out, i.e. it does not move in the corresponding world *?W*.
- *Visible(?W, ?P, ?O)* holds if the object *?O* is visible from pose *?P* in the world *?W*.
- *Occluding(?W, ?P, ?O_1, ?O_2)* holds if object *?O_2* is occluding the object *?O_1* when the camera is at pose *?P*.
- *Reachable(?W, ?R, ?O)* holds if the robot *?R* can reach the object *?O* in the world configuration *?W*.
- *Blocking(?W, ?R, ?O, ?B)* unifies *?B* with the list of objects that might be blocking a grasp for object *?O*.

As can bee seen, all predicates require a world database as first variable. By letting this parameter unbound, a default database is used. In this article we can only briefly explain the implementation of these predicates. Details can be found in [13]. The predicates *Contact* and *Stable* are implemented using the Bullet physics engine[2]. Contacts are inferred by using the Bullet's collision engine and for inferring stability, we simulate for a short period of time and compare the poses of objects to check if they moved. If an object changed its location, it is not stable. The predicates *Visible* and *Occluding* are inferred using OpenGL. Each object is rendered in a

[2]http://bulletphysics.org/

different color and by counting pixels, the system can infer how much of an object is visible and by which other objects it is occluded. Reachability is calculated using a standard solver for inverse kinematics and blocking objects are objects the robot is in collision with when reaching for an object. Blocking objects are objects that potentially hinder grasping actions and make motion planning harder.

V. Temporal Projection of CRAM Plans

The reasoning system explained in Section IV has no notion of time. However, projection requires the integration of reasoning about sequences of actions over time and their consequences and we need to extend the system by integrating a temporal calculus.

In CRAM, we consider robot control programs that have annotations in a first-order logic as plans because the annotations allow us to reason about their purpose. Projecting a plan means executing it in projection mode. In projection, instead of interacting with the actual robot's hardware, plans interact only with the world database that we use to implement our physics-based reasoning engine as described in the previous section. Each action generates a sequence of events where each event indicates a change in the world database at a specific point in time. By storing the sequence of events and copies of the corresponding world database at the time the event happened, we construct a time line. By adding predicates for temporal reasoning on these time lines, including predicates for relating events to parts in the plan, we can specify flaws in the robot's behavior as logical expressions.

In this section we show how plans must be constructed to allow for projecting them, how time lines are generated and how reasoning about them is implemented.

A. Reasoning About Plan Execution

In CRAM, plans are robot control programs that cannot only be executed but also reasoned about. Programmatically inferring what an arbitrary control program does is intractable at best if not impossible and it gets even more difficult in highly dynamic changing environments such as a human household. However, by providing annotation describing the semantics of a certain part of a plan, reasoning about the semantics of a complete plan becomes possible.

Execution of a CRAM plan generates a plan tree, i.e. a tree that contains all plans and sub-plans that have been executed, their status at any point in time, when they started, when they terminated and their result or failure description. The semantics of a sub-plan are made transparent for a reasoning engine by providing semantic annotations. For instance, a plan that is supposed to place an object *?obj* at a location *?loc* is called *Achieve(Loc(?obj, ?loc))*. The implication of a terminal status *succeeded* is that the robot believes that the object is at the destination location. The term *Loc(?obj, ?loc)* is called an occasion and is defined in the system's reasoning engine. In other words, using this naming scheme, we define the purpose of plans using predicates of an underlying reasoning system which allows for relating plans to the corresponding world states that are stored on the time line.

To reason about plan execution, we define predicates that query the plan tree. For instance, we define the predicate *Task(?tsk)* to unify a variable with a task object. The predicate *TaskGoal(?tsk, ?goal)* can be used to query a task object for its goal where the goal is for instance an achieve expression as introduced before. In addition, the system provides predicates to access the task's status, result, errors, position in the tree including sub-tasks and start and times. A complete list and a more detailed explanation of the system can be found in [23].

B. Generation of Timelines

CRAM plans need to be written in a general way to allow for executing the same code on the actual robot hardware as well as in projection. This is achieved by a clear, well-defined, minimal and most importantly purely symbolic interface between high-level plans and the low-level components of the robot or projection that execute the actual actions. All actions that are to be performed by the robot are described by action designators, i.e. key-value-pairs specifying the action's parameters in order to execute it. For instance, the following action designator is used to grasp a cup:

```
(an action (type navigation) (goal (location (to see) (obj Cup))))
```

The input for all process modules as shown in Figure 3 are action designators. Each process module is activated at the beginning of plan execution and deactivated after a plan finished. The status can be monitored using special signal slots and process modules update the world database using events.

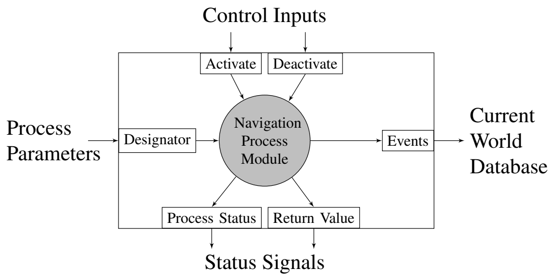


Fig. 3. Process module encapsulating a navigation control process. The input is an action designator specifying that the action is a navigation action and the containing the goal location as represented by a location designator, e.g. *(a location (to see) (obj cup))*.

From the point of view of a high-level plan, process modules can be seen as black boxes. That allows us to keep the high-level plan general enough to work with process modules developed for different robot platforms, simulation or projection. Specifically, when projecting a plan, we implement a process module that just makes assertions and retractions in a projection-local copy of the world database of the reasoning engine instead of sending commands to the actual robot hardware. Depending on the application and the robot, the number of process modules may vary. On the PR2, we currently implement four process modules: manipulation, navigation, perception and a process module for moving the cameras mounted on the robot's head.

To understand how projection and the creation of a time line works, let us have a closer look at the manipulation projection process module as an example for a rather complex process module. In the current implementation it supports the actions *grasp*, *lift*, *put down*, *open* and *close*. When the process module receives a respective action designator it makes assertions in the world database, emits events and increments the simulation time. The following example shows an action designator for grasping a cup:

(an action (to grasp) (obj (object (type cup))))

When the process module executes it, it generates the events *RobotStateChanged()* and *ObjectAttached(Cup, "r_gripper_wrist_link")* Since projection is supposed to be fast, we do not perform a complete simulation of a trajectory which would also require motion and grasp planning. Instead, we define *key points* for trajectories. For grasping, we define only one key point, the final pose the robot needs to reach in order to grasp the object. For simplicity, we use one out of the following grasps: a side grasp, a front grasp or a top grasp. Which grasp is used depends on the type of the object that should be grasped. For instance, cups are grasped with front or side grasps while plates always require a side grasp. After the *RobotStateChanged* event, the robot's gripper is at the cup's pose. The assertion in the world database that corresponds to the *RobotStateChanged* event positions the links of the robot's arm according to an inverse kinematics solution for placing the gripper at the cup's pose. The *ObjectAttached* event notifies the system that the object should be moved whenever the gripper link moves as long as the object is attached.

ObjectPerceived(?obj, ?s)	An object ?obj (described by an object designator) has been seen in a specific sensor ?s.
RobotStateChanged()	The robot has changed its state, i.e. it changed its position or the position of some links.
ObjectAttached(?obj, ?link)	An object ?obj has been attached to a specific link.
ObjectDetached(?obj, ?link)	An object ?obj has been detached from a specific link.
ObjectArticulationEvent(?obj, ?d)	An object changed its articulation status, e.g. a drawer or a cupboard has been opened by distance ?d.
ActionStarted(?m, ?d)	The process module ?m started to execute the action designator ?d
ActionFinished(?m, ?d)	The process module ?m finished executing the action designator ?d

TABLE I

OVERVIEW OF THE EVENTS USED FOR GENERATING A TIME LINE.

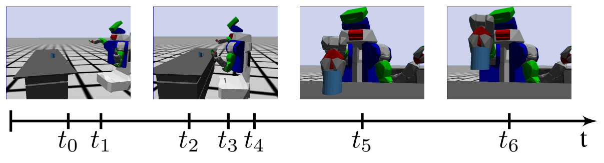
Table I gives an overview of the events that are used in the current system. The perception process module only generates *ObjectPerceived* events. The manipulation process module generates events of type *RobotStateChanged*, *ObjectAttached*, *ObjectDetached* and *ObjectArticulationEvent*. Navigation and PTU (Pan-Tilt-Unit) only generate *RobotStateChanged* events. When they start executing, all process modules generate the event *ActionStarted* and when they finish they generate the event *ActionFinished*.

Handling of time and incrementing time is especially critical in projection. While we want to produce predictions of the outcome of a specific plan quickly, we must not sacrifice the ability to project actions that happen concurrently. For instance, suppose the robot tries to see an object on the table while it is putting down an object next to it. If the destination location for the put-down action happens to be in front of the object, the run time of both actions determines if perception succeeds or not and projection needs to account for that.

Each set of changes in the world database must cause an increment of the projection clock. This assumes that the time spent while executing code in a high-level plan takes almost no time. In fact, executing actual actions on the robot is the most time consuming part. For instance, a navigation action will take several seconds depending on the distance to drive and planning and executing an arm trajectory can take 30 seconds and more while triggering these actions in a high level plan can be done in fractions of seconds. By only allowing time increments in process modules, we keep plans general since they do not have to have explicit support for projection. The specific projection clock we use increments at a maximal rate, for instance 20 milliseconds. In other words, increments happen at most at 50 Hz. However, the amount of time by which the clock is incremented can vary depending on the action that is projected. That way, each action will take exactly 20 milliseconds real time while still adding events to the time line at approximations of the action's run time. This implementation is a good compromise to allow for concurrent execution and good performance. The projected run time of actions is calculated using heuristics and random numbers. For instance, navigation time is approximated by a linear function over the distance between the starting point of the action and the goal and noise generated by a random number generator. The same is done for manipulation while perception is modeled using a constant function. The heuristics are not deterministic to increase the variation in different projections of the same plan.

C. *Reasoning on Timelines*

After a plan has been projected, the system needs to analyze the time line and search for flaws. We define two predicates, *Occurs* for reasoning about events and *Holds* for reasoning about world states over intervals of time. Figure 4 shows an example time line generated by a simple pick-up plan. The robot navigated to a location close to the table and picked up a cup.



Events:

t_0	start
t_1	ActionStarted(Navigation, NavDesig)
t_2	RobotStateChanged(), ActionFinished(Navigation, Desig), ActionStarted(Perception, PerceptionDesig)
t_3	ObjectPerceived(Cup), ActionFinished(Perception, PerceptionDesig), ActionStarted(Manipulation, GraspDesig)
t_4	RobotStateChanged(), ObjectAttached(Cup, "r_wrist_roll_link"), ActionFinished(GraspDesig), ActionStarted(Manipulation, LiftDesig)
t_5	RobotStateChanged(), ActionFinished(Manipulation, LiftDesig)

Fig. 4. Sample projection of a simple pick-up plan. The robot first navigates to the table, then moves the gripper to the location of the cup and finally lifts the cup.

The time line in our example contains four events. First the robot navigates to a location where it can reach the cup. Navigation generates a *RobotStateChanged* event since the robot changed its location in the map. Perceiving the object generates an *ObjectPerceived* event. Grasping generates another *RobotStateChanged* event since the arm moved. In addition, the object is attached to the right gripper. Finally, lifting moves the robot arm again, so another *RobotStateChanged* event is generated. As already mentioned, the ordered sequence of events creates a time line. More specifically, the time line not only contains the events that were generated in the process modules and corresponding time stamps but also snapshots of the world database at the time when an event occurred. In contrast to classical purely symbolic projection, our approach does not require logical expressions that represent the world state to be added or removed when actions are performed. Instead, we calculate the truth values and variable bindings of predicates on demand, based on the stored geometric world databases. This approach avoids certain problems purely symbolic approaches suffer from. For instance, consider the predicate *Visible* as defined in the previous section before and after a put-down action. The put-down action might cause occlusions. In a purely symbolic representation of the world that only consists of logical formulas, it is hard to infer that an object is occluded by another object after a put down action, in particular because the put-down action only contains a reference to the manipulated object. In contrast, our system proves visibility using OpenGL and on demand and the problem of deciding which conditions do not hold anymore after executing an action is avoided.

To reason about events, we define the predicate *Occurs(?event, ?time)*. It unifies an event with a time stamp. For instance, to find all objects that have been grasped and the corresponding times we can state the following query:

Occurs(ObjectAttached(?o, ?l), ?t)

The resulting bindings of this query in the above example timeline are:

$$\text{?o} = \text{Cup},\ \text{?l} = \text{“r\_wrist\_roll\_link”},\ \text{?t} = t_2$$

The implementation of the *Occurs* predicate just iterates over the sequence of events on the time line and unifies the variables *?event* and *?time* with the corresponding event pattern and time stamp.

Events are transitions in the world database, i.e. they indicate changes in the world. In other words, occasions (i.e. states) hold over time intervals and change only at events. For instance, if an object has been attached, the Prolog expression *ObjectInHand(?object)* holds until the object is detached again. To reason about these conditions and the corresponding time intervals they hold in, we define the predicate *Holds(?occasion, ?interval)*. The system tries to prove a given logical expression *?occasion* at the time points described by *?interval*. Please note that *?occasion* and *?interval* cannot be free variables since we cannot enumerate all possible logical expressions and time intervals. However, *?occasion* and *?interval* can be terms that contain free variables. *?interval* must be one of the following three expressions:

- *during*(t_0, t_1) indicating that the occasion must hold at least once in the interval $[t_0, t_1)$.
- *at(t)* for specifying a single point in time, i.e. an interval with length 0.
- *throughout*(t_0, t_1) indicating that the occasion must hold throughout the complete time interval $[t_0, t_1)$.

The implementation is based on proving occasions in the different world databases stored together with the events on the time line, assuming that changes in the world database only happen with events. As an example, if we want to find objects that were standing on the table at time t with $t_2 < t < t_3$, the corresponding query can be formulated as follows:

$$\text{Holds(On(?object, Table), at}(t))$$

The *Holds* predicate then finds and loads the world database instance on the time line whose time stamp is directly before t and proves the term *On(?object, Table)*. The implementation of *during* iterates over all valid world databases in the interval and proves the occasion in each of them, yielding all possible solutions. The implementation of *throughout* only generates solutions if the occasion holds in all world databases in the interval with the same variable bindings.

VI. Finding Behavior Flaws

In order to improve plan parameters based on projection, we need to evaluate the time line. The analysis of the time lines finds flaws of various severity. Some flaws are critical for the overall outcome of a plan, for instance occluded objects. Critical flaws indicate that the plan will probably fail. Other flaws such as the distance to drive between different actions, the overall execution time of a plan and blocking objects are only a measure of the quality of solutions and indicate potential problems. In this paper, we will show the definitions of the flaws *“object occluded”* and *“object blocking”*.

The object occluded flaw basically means that an object that is to be perceived later is occluded because the put-down location of another object is in front of it. In the simplest case, an error indicating that an object could not be perceived is thrown during projection. The flaw definition does not require the perception plan to fail though. We first need to find out which object was to be perceived and if it was occluded by an object we put down previously. We define the flaw as follows:

$$\begin{aligned}
&\text{Task}(?task_1) \wedge \text{Task}(?task_2)\\
&\quad \wedge \text{TaskGoal}(?task_1,\ \text{Perceive}(?o_1))\\
&\quad \wedge \text{TaskGoal}(?task_2,\ \text{Achieve(ObjectPlacedAt}(?o_2,\ ?l)))\\
&\quad \wedge \text{TaskStatus}(?task_2,\ \textit{succeeded}) \wedge \text{TaskEnd}(?task_2,\ ?t)\\
&\quad \wedge \text{Holds(Occluding}(?w,\ ?o_2,\ ?o_1),\ \textit{at}(?t))
\end{aligned}$$

Please note that the variable *?w* is unbound in *Holds*. That means that the default database is used which is implicitly bound by *Holds* to match the data base as it was at time t. First, we query the task tree that was generated by plan execution for a failed perception task and a put-down task. Then we assert that the put-down was successful and we bind the time when it finished to the variable *?t*. Finally, we check if the object that was put down is occluding the object that was to be perceived.

We define blocking objects as objects that the robot is in collision with while grasping an object (excluding the grasped object itself). While this flaw not necessarily leads to execution errors on the actual robot where grasp planning and motion planning is used, it is still an indicator for potential problems and performance penalties since a motion planner will need more time to find a valid plan when trajectories need to be complex. We define the flaw to find all objects that are in collision with the robot during the execution of a pick-up plan:

$$\begin{aligned}&\text{Task}(?tsk) \wedge \text{TaskGoal}(?tsk,\ \text{Achieve}(\text{ObjectInHand}(?o)))\\&\quad\wedge \text{TaskStart}(?tsk,\ t_s) \wedge \text{TaskEnd}(?tsk,\ t_e)\\&\quad\wedge \text{Holds}(\text{Blocking}(?w,\ \textit{PR2},\ ?o,\ ?b),\ \textit{during}(t_s,\ t_e))\end{aligned}$$

First we find all pick-up plans and get their start and end times. Then we use the *Holds* predicate to find blocking objects during the pick-up action.

VII. Conclusions and Future Work

In this paper, we presented a system for projecting high-level robot plans for pick-and-place actions in a human household. The system is based on a the CRAM Plan Language execution environment and a Prolog reasoning engine that integrates a geometrically accurate world database and implements predicates for reasoning about stability, visibility and reachability using the Bullet physics engine, OpenGL rendering and inverse kinematics calculation. Projection is implemented by assertions and retractions in the world database and the generation of a time line containing events. The system can predict potential problems in plan execution such as unwanted occlusions or objects hindering future actions and is used to generate plan parameters such as the locations for placing objects or for placing the robot when performing an action.

The use of projected time lines is not restricted to finding plan parameters such as locations at run time. We will implement a transformational planner that will be able to predict and fix behavior flaws by applying structural transformations to a plan.

Acknowledgements: This work is supported in part by the EU FP7 Projects *RoboHow* (grant number 288533) and *SAPHARI* (grant number 287513) and within the DFG excellence initiative research cluster *Cognition for Technical Systems – CoTeSys*, see also `www.cotesys.org`. We thank Joris DeSchutter's group from KU Leuven (the authors of iTaSC) for insightful discussions.

References

[1] D. McDermott, "Robot Planning," *AI Magazine*, vol. 13, no. 2, pp. 55–79, 1992.

[2] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL–the planning domain definition language," *AIPS-98 planning committee*, 1998.

[3] M. Fox and D. Long, "PDDL2.1: An extension of PDDL for expressing temporal planning domains." *Journal of Artificial Intelligence Research*, vol. 20, pp. 61–124, 2003.

[4] D. Smith, Ed., *Special Issue on the 3rd International Planning Competition*, ser. Journal of Artificial Intelligence Research, vol. 20, 2003.

[5] J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer Academic Publishers, 1991.

[6] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[7] M. Stilman and J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," in *Proceedings of the 2004 IEEE International Conference on Humanoid Robotics (Humanoids)*, vol. 1, December 2004, pp. 322–341.

[8] S. Cambon, F. Gravot, and R. Alami, "asymov: Towards more realistic robot plans," in *International Conference on Automated Planning and Scheduling, (ICAPS 2004)*, 2004.

[9] F. Stulp, A. Fedrizzi, and M. Beetz, "Action-related place-based mobile manipulation," in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2009, pp. 3115–3120.

[10] L. Mösenlechner and M. Beetz, "Using physics- and sensor-based simulation for high-fidelity temporal projection of realistic robot behavior," in *19th International Conference on Automated Planning and Scheduling (ICAPS'09).*, 2009.

[11] L. Kunze, M. E. Dolha, E. Guzman, and M. Beetz, "Simulation-based temporal projection of everyday robot object manipulation," in *Proc. of the 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Yolum, Tumer, Stone, and Sonenberg, Eds. Taipei, Taiwan: IFAAMAS, May, 2–6 2011.

[12] M. Beetz, L. Mösenlechner, and M. Tenorth, "CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, Taiwan, October 18-22 2010, pp. 1012–1017.

[13] L. Mösenlechner and M. Beetz, "Parameterizing Actions to have the Appropriate Effects," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, CA, USA, September 25–30 2011.

[14] S. Zickler and M. Veloso, "Efficient physics-based planning: sampling search via non-deterministic tactics and skills," in *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*. Richland, SC: IFAAMAS, 2009, pp. 27–33.

[15] C. Dornhege, M. Gissler, M. Teschner, and B. Nebel, "Integrating symbolic and geometric planning for mobile manipulation." in *IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR)*, 2009. [Online]. Available: http://www.informatik.uni-freiburg.de/~ki/papers/dornhege-etal-ssrr09.pdf

[16] L. P. Kaelbling and T. Lozano-Perez, "Unifying perception, estimation and action for mobile manipulation via belief space planning," in *IEEE Conference on Robotics and Automation (ICRA)*, 2012. [Online]. Available: http://lis.csail.mit.edu/pubs/tlp/ICRA12_1803_FI.pdf

[17] P. Michel, C. Scheurer, J. Kuffner, N. Vahrenkamp, and R. Dillmann, "Planning for robust execution of humanoid motions using future perceptive capability," in *Proceedings of the IEEE/RSJ IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'07)*, October 2007, pp. 3223–3228.

[18] L. Marin, E. A. Sisbot, and R. Alami, "Geometric tools for perspective taking for human-robot interaction," in *Mexican International Conference on Artificial Intelligence (MICAI 2008)*, 2008.

[19] F. Zacharias, C. Borst, and G. Hirzinger, "Capturing robot workspace structure: representing robot capabilities," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007, pp. 3229–3236. [Online]. Available: http://infoscience.epfl.ch/record/109336

[20] S. Hanks, "Practical temporal projection," in *Proc. of AAAI-90*, 1990, pp. 158–163.

[21] D. McDermott, "An algorithm for probabilistic, totally-ordered temporal projection," in *Spatial and Temporal Reasoning*, O. Stock, Ed. Dordrecht: Kluwer Academic Publishers, 1997.

[22] D. Pangercic, M. Tenorth, B. Pitzer, and M. Beetz, "Semantic object maps for robotic housework - representation, acquisition and use," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vilamoura, Portugal, October, 7–12 2012.

[23] L. Mösenlechner, N. Demmel, and M. Beetz, "Becoming Action-aware through Reasoning about Logged Plan Execution Traces," in *IEEE/RSJ International Conference on Intelligent RObots and Systems.*, Taipei, Taiwan, October 18-22 2010, pp. 2231–2236.