Using Physics- and Sensor-based Simulation for High-fidelity Temporal Projection of Realistic Robot Behavior

Lorenz Mösenlechner and **Michael Beetz**
Intelligent Autonomous Systems Group
Department of Informatics
Technische Universität München
Boltzmannstr. 3, D-85748 Garching
{moesenle,beetz}@cs.tum.edu

Abstract

Planning means deciding on the future course of action based on predictions of what will happen when an activity is carried out in one way or the other. As we apply action planning to autonomous, sensor-guided mobile robots with manipulators or even to humanoid robots we need very realistic and detailed predictions of the behavior generated by a plan in order to improve the robot's performance substantially.

In this paper we investigate the high-fidelity temporal projection of realistic robot behavior based on physics- and sensor-based simulation systems. We equip a simulator and interpreter with means to log simulated plan executions into a database. A logic-based query and inference mechanism then retrieves and reconstructs the necessary information from the database and translates the information into a first-order representation of robot plans and the behavior they generate. The query language enables the robot planning system to infer the intentions, the beliefs, and the world state at any projected time. It also allows the planning system to recognize, diagnose, and analyze various plan failures typical for performing everyday manipulation tasks.

Introduction

Consider a household robot (see Figure 1) performing pick-and-place tasks in a kitchen environment. The robot uses its camera to recognize the objects it is required to manipulate where the objects are described by partial and possibly inaccurate object descriptions. Objects may slip out of the robot's hand depending on the friction of the objects and robot grippers. The success of grasps also depends on the trajectories computed by the motion planner and the accuracy with which the arm and gripper controller can follow those trajectories. Other critical factors include the position from which the robot is picking up the object and how cluttered the surrounding is.

The important conclusions that we draw from our scenario are the following ones. First, the success of actions and therefore the high-level plans critically depends on low-level details such as object recognition, selecting standing positions, grasps, objective functions for the motion planner etc. Second, improving the performance of robot plans through action planning requires action planners to adjust low-level behavior and reason about the consequences of these adjustments. As a consequence, the temporal projection of robot action plans must be much more accurate, realistic, and detailed than those performed by most current action planning systems.

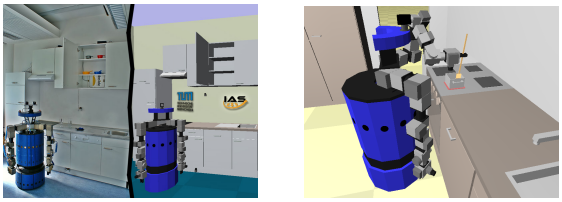


Figure 1: Household robot in reality and simulation.

The temporal projection mechanisms investigated in this paper enable autonomous service robots with manipulators to improve their performance by revising general purpose default plans into tailored optimized ones. Consider, for example, the table setting task. The default plan instructs the robot to put items on the table one after the other. Thus, for this task the robot can revise the default plan in various ways to improve its performance. For example, the robot can stack the plates to carry them more efficiently, it can leave doors open while setting the table, it can slightly change its position such that more objects are in reach, it can transport the cups in an order such that it moves these objects that are obstacles when grasping other ones first.

We formulate the optimization of robot manipulation plans for everyday activities as a transformational planning problem. The basic idea is to apply plan transformations to plan candidates to generate promising plan candidates. In a second step the new candidate plans are projected to predict probable execution scenarios, which are then used to assess the plan's performance, strengths, and weaknesses.

In this paper we propose the use of physics- and sensor-based simulation engines for high fidelity temporal projection of realistic robot behavior. In contrast to current planners that use simulation only for navigation and motion panning, our approach enables the planner to reason about whether the robot manipulates the right object, whether it misses objects, whether objects are slipping out of the gripper, etc. We log the simulation data and use the logged data

to instantiate first-order symbolic representations of the projected plan execution. To enable the robot to *symbolically reason* about robot behavior, flaws of the behavior, and diagnose the reason for these flaws, we contribute in the following ways to the high-fidelity temporal projection of robot action plans.

We show how symbolic representations of the behavior, the beliefs, and the intentions of the robot can be grounded in logged simulation data and internal data of the plan interpreter. Further, we propose a suitable set of predicates for reasoning about the failures and flaws of robot control plans based on temporal projections. Finally, we show that physics-based temporal projection allows to reason about plan execution at a level of detail that has not been demonstrated with transition-based symbolic models.

The remainder of this paper is organized as follows. After motivating our approach for temporal projection, we discuss classical approaches and the differences to our approach. Then, we give an overview over the concepts of our approach, followed by a formalization in first-order logic. We evaluate our approach by showing the expressive power by formalizing examples of flaws in program execution. Finally, we shortly discuss related approaches.

An Example Projection of a Robot Plan

Planning everyday manipulation tasks requires a robot to reason about its behavior at different levels of abstraction. Let us consider a table setting task as an illustrative example.

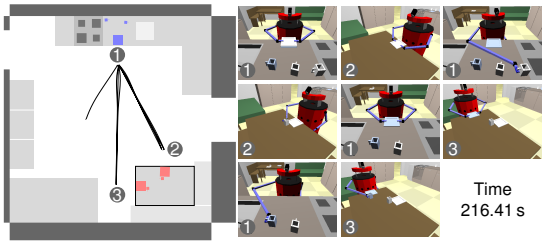


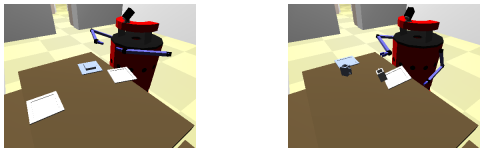
Figure 2: View of the execution of the set-the-table plan.

In order to improve table setting plans, the robot can apply transformation rules such as the following:

- **IF** the robot — when executing this plan in order to transport multiple objects from one place to another one — *might* drop objects that it stacked to carry the objects more efficiently, **THEN** change the plan such that the objects are not stacked any more.
- **IF** the robot — when executing this plan — *might* overlook objects that are occluded, **THEN** change the perception subplan to actively search occluded areas.

To predict behavior flaws our robot temporally projects what will happen if the plan gets executed using a high-fidelity physics- and sensor-based simulation. To account for nondeterminisms in the execution and for uncertainties, the robot projects its plans multiple times, applying probabilistic noise models of its sensors. To project the plans, they are executed in a realistic simulation environment (Figure 2).

The reasons why such accurate and detailed predictions are necessary are illustrated in Figure 3. Figure 3(a), for example, shows the plate lying upside down because it slipped out of the gripper. This is a situation where a simulated physical event (slippage) caused a situation in which the robot cannot pick up the plate any more and therefore not achieves the user command. This example shows that variations at a detailed level decide whether or not the goals at an abstract planning level are achieved. Using abstract action models, as used by most action planners, such plan failures could not be predicted and therefore not planned for.



(a) (b)

Figure 3: The results of laying the table. Due to the detailed simulation, the execution of plans often results in qualitatively different situations.

Based on the temporal projections generated by the sensor- and physics-based simulation, our planner ([Beetz, 2000; Müller, Kirsch, & Beetz, 2007]) can infer answers to queries concerning the world states, the intentions, the beliefs, the results of plan interpretations and the interactions between these concepts. Thus, concerning the world states the robot can for example answer: what objects were on the cupboard at the beginning? What objects are on the table at the end? Where did the robot stand in order to pick up objects? Are all objects placed accurately at the end? Did the actions cause unwanted side-effects like the displacement of other objects?

In addition to queries about the world states that our projector can answer, like many others can do, it can also answer queries about the beliefs of the robot at various states of plan interpretation as well as interpretation of plan steps. Examples of such queries are the following ones: what did the robot see when it looked for the cups on the table? Did the robot see all cups on the table? Why did the robot pick up the object with two hands? How often did picking up the object fail before it succeeded?

To sum up, our example shows that our temporal projection mechanism can predict robot behavior very realistically because of the use of sensor- and physics-based simulation. It can also answer queries about the beliefs and computational states during plan execution, which is essential in robotic applications because robots have in most cases only partial and uncertain information about the world.

Temporal Projection for Robot Planning

Most researchers use a model-based approach to action planning, which is depicted in Figure 4. They model control routines that are intended to perform a specific task, such as navigating to a specified destination, as actions in a symbolic language. The models of actions are typically represented as a transition system in which an action trans-

fers a state into sets of successor states. The different approaches, such as action logics for example, differ with respect to the assumption they make about the underlying transition system: whether transitions are deterministic, nondeterministic, conditional on state properties, representing the concurrent execution of actions, caused by exogenous events, etc. A variety of logical representation formalisms including numerous extensions of the Situation Calculus, ADL, event calculus, and fluent calculus are the result, to name only a few.[1]

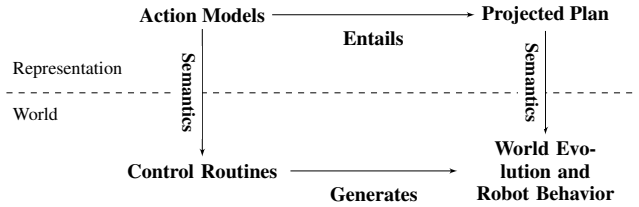


Figure 4: Model-based robot action planning

Researchers make these ontological commitments in order to provide representation and reasoning mechanisms for planning systems that satisfy the basic relationship depicted in Figure 4. In contrast, we want to represent the control routines of the robot such that we can, in the representation framework, symbolically infer the consequences of executing a plan. We consider the meaning of the inferred consequences to accurately (approximately) represent what is expected to happen in the real world when executing the respective control program.

Almost all logical action representations, for instance PDDL [Ghallab *et al.*, 1998], work on the basis of conceptualizing an underlying transition system for an action domain.

There are problems with this modeling approach when applying it to autonomous manipulation. First, in autonomous robot control the effects of actions are the result of the interaction of the robot's behavior with its environment. Indeed autonomous robot control addresses this issue through feedback control and by monitoring action effects and reactively recovering from local problems and failures. The second problem is quantification. Because action models are typically universally quantified, even if modeled probabilistically or content specific, and because the effects of actions are caused by the subtle interplay of actions and situations, it is difficult to define the models realistically. Indeed the difficulty of representing robot actions realistically is mirrored by the huge number of action logics proposed in the literature.

Interestingly, the modeling problems that result from quantifying over models and abstracting away from some interactions between actions and context seem to be artifacts that do not arise in physical robot simulators based on physics engines such as ODE, PhysX, or Bullet. The reason is that these simulators do very little abstraction, time has a high resolution, and the state update rules modeling physics take all current and relevant state variables into account. However, the realism and accuracy comes at a price. Simulations can only sample possible episodes but not quantify over all possible ones.

The same holds for very expressive and probabilistic action representations. McDermott (1997), for example has proposed a powerful and expressive action language capable of representing the concurrent execution of durative actions with interfering events in a probabilistic setting. Beetz & McDermott (1997) have shown that by using such sets of sampled plan projections, realistic robot plans can be reliably improved on the basis of a reasonably small set of sampled projections.

[1] See [Thielscher, 2009] for a discussion and an effort of unifying some aspects of the action logic research field.

Simulation-Based Temporal Projection

Instead of modeling the actions of the robot at an abstract level as a transition system, we propose to simply interpret a robot control program in a sensor- and physics-based simulator, record all the necessary data, and then translate the recorded data into a first-order representation of the episode.

To do so, the interpretation and the simulation run in two coupled loops, where the simulation continually adds the simulated sensor data to the sense data queue of the program and the interpreter sets the control signals for each motor and sensing commands for the simulator (see Figure 5).

Physics-based Simulation

In a nutshell the physics-based simulator works as follows. It receives the control signals for the respective motors and the commands for the robot's sensors as its input. It then periodically updates the state of the simulated world based on the dynamics of the simulated system and physical laws. To do so the simulator performs four steps in each iteration:

1. compute the forces applied at each motor based on the current motor state and the control signal;
2. determine the objects that the forces apply to;
3. for each object sum the forces applied to it and calculate the effects of the forces based on the object's current dynamic state and the object properties including friction, weight, center of gravity, etc;
4. calculate the sensor data by applying the sensor models of the activated sensing processes to the current state of simulation.

In each iteration, the simulator updates the list of collisions, applies forces to contact points and motor joints and updates the location of each object and it's velocity vector.

The central data structure that the simulator works on is the set of objects. For example, the robot consists of the robot base, the different arm modules, the grippers, etc. For each object the data structure contains the position, mesh, etc. Models of objects include:

1. 3D-models of all body parts;
2. the position, orientation and velocity of the object;
3. physical properties such as friction, mass and elasticity;
4. joints representing connecting points between bodies where forces can be applied. Thus, motors are modeled as joints and control signals are translated into forces applied to the corresponding joints;
5. the list of collisions between body parts.

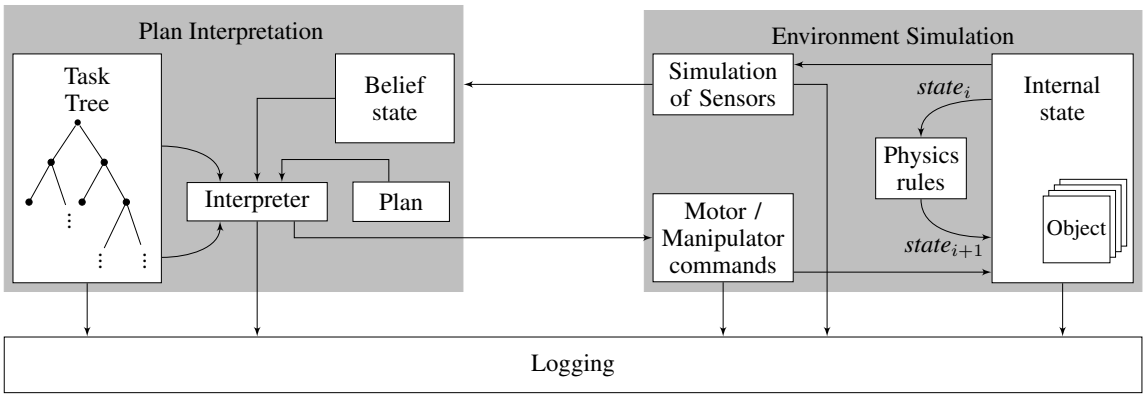


Figure 5: Termporal projection using simulation

This is the most simplistic version of a physics-based simulator: a simulator that simulates rigid objects in an environment where only the robot changes the environment. Modern physics-based simulators also support additional processes acting in the world, soft objects, liquids, etc. (PhysX, Bullet). Some of them [Johnston & Williams, 2008] are so advanced that they can even simulate the infamous egg cracking problem in the commonsense problem-solving community [Morgenstern, 2001].

In physics-based simulation the effects of actions are computed based on physics rules that map forces and object properties into effects (sometimes including some randomization to account for aspects that are not sufficiently modeled). Consequently, a physics-based simulation does not have problems when computing the effects of picking up a pile of plates (based on mass), the consequences of a wet gripper (changed friction), or another object falling on the pile while it is carried (interfering effects of concurrent events). Without these "details" a kitchen robot cannot plan to carry fewer plates after washing its grippers because of the reduced friction in its fingers. Modeling such aspects in abstract first-order representations is tedious and results in huge axiomatizations. This is well illustrated in different formalizations of the egg cracking problem in commonsense reasoning [Morgenstern, 2001; Johnston & Williams, 2008].

Plan Interpretation

The interpretation of a plan is completely (but not necessarily deterministically) determined by the program state: the program counter and the variable values. In program interpretation these data are usually kept in a stack of task interpretation frames. Thus, everything that the robot "believes" in is at sometime somewhere in its interpretation stack. An example of a stack frame, which we call a *task* data structure is depicted in Figure 6. The task data structure contains the following data. The task environment contains the variables in the scope of the task, their values at different stages of execution, and the state of plan interpretation when these values were assigned. Thus the local variable *OBS* was initialized to *()* and then set to the set of object descriptions *(DES-17 DES-18)* at the end of task *t-7*. The task status contains the change of the task status during the projected plan interpretation and when the status changed.

TASK *T-6*

SUPERTASK	T-4	
TASK-EXPR	(ACHIEVE (OBJECT-CARRIED DES-17))	
TASK-CODE-PATH	((POLICY-PRIMARY) (STEP-NODE 1) (COMMAND 1) (STEP 2))	
TASK-ENVIRONMENT	OBS	(BEGIN-TASK T-4) () (END-TASK T-7) (DES-17 DES-18)
TASK-STATUS	TI10 CREATED	TI10 ACTIVE
	TI13 DONE	

Figure 6: Conceptual view of a projected task.

Let us now consider how the "belief state" of the robot is encoded in control programs using the position estimate of our robot as an example. The probability distribution over the robot's position at time instant t is computed by a particle-based Bayesian filtering approach [Beetz *et al.*, 1998]. For instance, when the robot is navigating in an office environment, the distribution may show that the robot has two probable position estimates: one (the correct one) at the right side of the corridor and a symmetric position at left. This belief state is abstracted into three program variables that are used by the control program: *RobotPos*, *PosAccuracy* and *PosAmbiguity* that store the belief about the pose of the robot, the accuracy of the position estimate in the global maximum of the probability distribution for the robot's position and the number of local maxima with a probability higher than a given threshold.

Interaction between Simulation and Interpretation

The robot control program interacts with the simulator through a middleware layer that is also used to communicate with real robotic hardware. Motors are simulated by providing the corresponding command interface and calculating the respective forces to be applied at the motor joints of the simulated object. Sensory data is also provided by the middleware interface and the behavior of the sensors is calculated from the internal simulator data structures, the simulator's rendering engine and models of the sensors. For instance for laser sensors, additional noise models of the real sensor can be applied to make simulation more realistic.

Logging

The simulation of a plan generates sub-symbolic data streams containing the data from plan interpretation as well

as data from the physical simulation. For instance, while the robot is navigating, for every simulator time step, the new location of the robot in the simulator data structures and the new values of it's self-localization are recorded.

As shown in Figure 5, it receives information from the interpreter, in particular the complete belief state at every point in time, the plan that is executed, and the task tree. Furthermore, the internal state of the simulator with information on all objects, the control signals received by the simulator and the sensory information is recorded by the logging mechanism. In particular, the logged internal state includes information such as the list of all collisions, represented as a pair of object names, the location of all objects, and visibility information. That means, not only simulated noisy sensor data but also the corresponding ground truth information is recorded. This is necessary for the analysis of misperceptions caused by noisy sensor data.

Representing Projected Execution Scenarios

Let us now consider our first-order representation of projected execution scenarios that allows for detailed reasoning and diagnosis of plan failures. This representation, which is generated from the logged data on demand is based on *occasions*, *events*, *intentions* and *causing relations*, which are introduced below.

Occasions are states that hold over time intervals where time instants are intervals without duration. The sentence $\mathit{Holds}(occ, t_i)$ represents that the occasion holds at time specification t_i. The term $\mathit{During}(t_1, t_2)$ indicates that the occasion holds during a subinterval of the time interval $[t_1, t_2]$ and $\mathit{Throughout}(t_1, t_2)$ specifies that the occasion holds throughout the complete time interval.

The second concept are events. Events represent temporal entities that cause state changes. Most often, events are caused by actions that are performed by the interpreted plan. We assert the occurrence of an event *ev* at time t_i with $\mathit{Occurs}(ev, t_i)$. Events happen at discrete time instances.

Occasions and events can be specified over two domains: the world and the belief state of the robot, indicated by an index of *W* and B for the predicates *Holds* and *Occurs* respectively. Thus, $\mathit{Holds}_W(o, t_i)$ states that o holds at t_i in the world and $\mathit{Holds}_B(o, t_i)$ states that the robot believes at time t_i that the occasion o holds at t_i. Syntactically, occasions are represented as terms or fluents. By giving the same name o to a occasion in the world as well as to a belief, the programmer asserts that both refer to the same state of the world. Thus, an incorrect belief of the robot can be defined as

$$\forall o, t_i . \mathit{IncorrectBelief}(o, t_i) \Leftrightarrow \\ \mathit{Holds}_B(o, t_i) \ \wedge \ \neg \mathit{Holds}_W(o, t_i)$$

The meaning of the belief and the world states is their grounding in the log data of the task network and the simulator data respectively. For our application domain we use the occasions shown in Table 1.

We consider the intentions of the robot to be the tasks on the interpretation stack. By naming a control routine *Achieve(s)* the programmer asserts that the purpose of the routine is to achieve state s, i.e. the corresponding occasion. Thus, if there is a control routine *Achieve(s)* on the interpretation stack the planner can infer that the robot currently has the intention to achieve state *s*. The planner can also infer that the tasks on top of *Achieve(s)* help to achieve *s* and that *Achieve(s)* is a sub-goal of the tasks deeper in the interpretation stack. Intentions are important since they cause actions which lead to events in the world state and in the robot's belief state. Finally, we provide two predicates $\mathit{Causes}_{B \to W}(\mathit{task}, \mathit{event}, t_i)$ and $\mathit{Causes}_{W \to B}(o_W, o_B, t_i)$ to represent the relations between the world and beliefs. The former asserts that a task causes an event whereas the latter relates two occasion terms, one in the world state, one in the belief state, to each other. In other words, it allows to infer that a specific belief was caused by a specific world state.

$\mathit{Contact}(obj_1, obj_2)$	Two objects are currently colliding
$\mathit{Supporting}(obj_1, obj_2)$	obj_i is standing on obj_b
$\mathit{Attached}(obj_1, obj_2)$	obj_1 and obj_2 are attached to each other.
$\mathit{Loc}(obj, loc)$	The location of an object
$\mathit{Loc}(\mathit{Robot}, loc)$	The location of the robot
$\mathit{ObjectVisible}(obj)$	The object is visible to the robot
$\mathit{ObjectInHand}(obj)$	The object is carried by the robot
$\mathit{Moving}(obj)$	The object is moving

Table 1: Occasion statements.

$\mathit{Holds}_W(occ, t_i)$	Occasion assertion in the world state.
$\mathit{Holds}_B(occ, t_i)$	Occasion assertion in the belief state.
$\mathit{Occurs}_W(\mathit{event}, t_i)$	Assert the occurrence of an event in the world state.
$\mathit{Occurs}_B(\mathit{event}, t_i)$	Assert the occurrence of an event in the belief state.
$\mathit{Causes}_{B \to W}(\mathit{task}, \mathit{event}, t_i)$	Causing relation between a task and an event.
$\mathit{Causes}_{W \to B}(s_W, s_B, t_i)$	Causing relation between a world state and a belief state.
$\mathit{SimulatorValueAt}(\mathit{name}, t_i)$	Access simulator-internal data structures.
$\mathit{VariableValueAt}(\mathit{name}, t_i)$	Access interpreter variable (belief state).

Table 2: Basic Predicates and Function statements.

The predicates defined above are implemented by accessing the recorded data structures of the execution log. To perform this low-level access, we define two functions: $\mathit{SimulatorValueAt}(\mathit{name}, t_i)$ to get the value of the simulator data structure identified by *name* at time instant t_i and $\mathit{VariableValueAt}(\mathit{name}, t_i)$ to get the value of the belief state variable *name* respectively.

Table 2 summarizes the basic predicates and functions used to make inferences in the logged execution scenario.

Force-dynamic States

We use force-dynamic states to define the basic physical properties of states in the context of pick-and-place tasks. As proposed by Siskind ([Siskind, 2003]) we represent stable physical scenes in terms of three basic relations:

- $\mathit{Contact}(obj_1, obj_2)$: the two objects are touching each other, i.e. there exists a contact point.
- $\mathit{Supporting}(obj_1, obj_2)$: obj_1 is supporting obj_2, i.e. obj_2 is standing on obj_1. More specifically, this state is described by asserting that obj_2 is above obj_1, a contact between both objects exists and obj_2 is not moving.
- $\mathit{Attached}(obj_1, obj_2)$: obj_1 is attached to obj_2, i.e. a movement of obj_2 causes the same movement of obj_1.

Using these predicates we can define the actions of interest, such as pick up or put down, analogously to Siskind who did it in a modal logic. To do so, we define state terms, used in the *Holds* predicate that mirror the semantics of Siskind's relations but are grounded in the recorded simulator data structures. As an example, consider the *Contact* relation. Contacts are similar to collisions. Therefore, the corresponding *Holds* term is defined as follows:

$$\begin{aligned}&\mathit{Holds}(\mathit{Contact}(\mathit{obj}_1, \mathit{obj}_2), t_i) \Leftrightarrow \\ &\quad \mathit{Collisions} = \mathit{SimulatorValueAt}(\mathit{Collisions}, t_i) \\ &\quad \mathit{Member}(\langle \mathit{obj}_1, \mathit{obj}_2 \rangle, \mathit{Collisions})\end{aligned}$$

The other two state terms are defined accordingly and grounded in the data structures generated by simulation.

Deriving Symbolic Representations from Logged Simulations

Our temporal projection module automatically generates the symbolic representations introduced above on demand, that is when queried for the respective information. To do so, the programmer has to define for each symbolic state how the symbolic state can be inferred from the simulation data and how the respective belief can be reconstructed from the logged interpretation stack. The programmer also has to define how the actions that matter for her planning application can be inferred from the temporal evolution of force-dynamic states. Specifying robot specific procedures is necessary for grounding the predicates of the planner in the data structures of the robot. Other planning systems do not use such procedures because they do not ground their reasoning. Instead, they assume that the semantics can be axiomatized. As stated in Section "Temporal Projection for Robot Planning", these axioms often prevent realistic modeling of autonomous robots.

Asserting States of the World. World states are computed from the simulator data structures using the predicate *SimulatorValueAt*, which is implemented as a method that computes for state variables the respective value of the variable at the specified time. Thus, the programmer can state that the robot is at position $\langle x, y \rangle$ at time instance t_i if the simulator data structures say so:

$$\begin{aligned}&\mathit{Holds}_W(\mathit{Loc}(\mathit{Robot}, \langle x, y \rangle), t_i) \Leftrightarrow \\ &\quad \langle x, y \rangle = \mathit{SimulatorValueAt}(\mathit{RobotPose}, t_i)\end{aligned}$$

The $Holds_W$ predicate is defined for all occasions that are used to describe the state of the world. The most important ones in our household domain are shown in Table 1.

Asserting Beliefs. Analogously, the programmer defines the beliefs using the interpretation data structures instead of the simulator data structures. The robot's belief state is stored in interpreter variables and is accessed with the function *VariableValueAt*. The robot believes to be at pose $\langle x, y \rangle$ if its variable *RobotPose* says so:

$$\begin{aligned}&\mathit{Holds}_B(\mathit{Loc}(\mathit{Robot}, \langle x, y \rangle), t_i) \Leftrightarrow \\ &\quad \langle x, y \rangle = \mathit{VariableValueAt}(\mathit{RobotPose}, t_i)\end{aligned}$$

Please note that in contrast to accessing the simulator state, $Holds_B$ relies on the function *VariableValueAt* which queries interpreter variables. Besides *Colliding* all occasions of Table 1 are also defined as beliefs.

Asserting Intentions. In order to infer the intentions of a simulated plan we have to consider the interpretation stack more carefully. Achieving a state *s* has been an intention if the routine *Achieve(s)* that was on the interpretation stack during the simulation. The robot pursued the goal *Achieve(s)* in the interval between the start and the end of the corresponding task. The purpose of achieving *s* can be computed by contemplating the supertasks of *Achieve(s)*: to represent tasks and the relations between them, we use the predicates and functions listed in Table 3.

$\mathit{Task}(\mathit{task})$	*task* is a task on the interpretation stack.
$\mathit{TaskGoal}(\mathit{task}, \mathit{goal})$	Relates a specific goal to the task
$\mathit{TaskStart}(\mathit{task})$	Returns the start time of the task
$\mathit{TaskEnd}(\mathit{task})$	Returns the end time of the task
$\mathit{Supertask}(\mathit{task}_s, \mathit{task}_c)$	task_s is a super task task_c, i.e. task_s occurs in the call stack of task_c

Table 3: Intention related statements.

Asserting Events. Actions that are performed by the robot cause events. For instance, a manipulation action that intends to achieve the $\mathit{TaskGoal}(\mathit{task}, \mathit{Achieve}(\mathit{ObjectInHand}(\mathit{obj})))$ will cause a *PickUp* event when the object is not already picked up. More specifically, the events that are defined in our systems include $\mathit{Collision}(\mathit{obj}_1, \mathit{obj}_2)$ and its inverse event $\mathit{CollisionEnd}(\mathit{obj}_1, \mathit{obj}_2)$ to state that two objects start or stop touching each other, $\mathit{LocChange}(\mathit{obj})$ to state that the object changed its location and $\mathit{PickUp}(\mathit{obj})$ and $\mathit{PutDown}(\mathit{obj})$ to state the respective manipulation events. Collision events can only be defined for $Occurs_W$ based on simulator data. While the *PickUp* action can be easily defined in the belief state by being generated at the end of the execution of the *ObjectInHand* goal, the definition in the simulator state is defined in terms of force-dynamic states — the *Collision* events and the *Contact* and *Supporting* occasions.

$$\begin{aligned}&\mathit{Occurs}_W(\mathit{PickUp}(\mathit{Obj}_1), t) \Leftrightarrow \exists t_1,\ t_2. \\ &\quad \mathit{Holds}_W(\mathit{Supporting}(\mathit{Table}_1, \mathit{Obj}_1), \mathit{Throughout}(t_1, t)) \\ &\quad \wedge\ \mathit{Occurs}_W(\mathit{Collision}(\mathit{Obj}_1, \mathit{Gripper}), t_2) \\ &\quad \wedge\ \mathit{Holds}(\mathit{Attached}(\mathit{Obj}_1, \mathit{Gripper}), \mathit{During}(t_2, t)) \\ &\quad \wedge\ \mathit{Occurs}_W(\mathit{CollisionEnd}(\mathit{Obj}_1, \mathit{Table}_1), t)\end{aligned}$$

When picking up Obj_1, it is first standing on the table, i.e. supported by the table. Then the gripper approaches the object and grasps it, resulting in a collision event. When grasped, the object is attached to the gripper and the pick-up event is generated when the contact between the table and the object is removed (indicated by a removed collision), i.e. the object is actually picked up. Table 4 shows the most important events defined in our system.

$\mathit{LocChange}(\mathit{obj})$	An object changed its location
$\mathit{LocChange}(\mathit{Robot})$	The robot changed its location
$\mathit{Collision}(\mathit{obj}_1, \mathit{obj}_2)$	obj_1 and obj_2 started colliding
$\mathit{CollisionEnd}(\mathit{obj}_1, \mathit{obj}_2)$	obj_1 and obj_2 stopped colliding
$\mathit{PickUp}(\mathit{obj})$	*obj* has been picked up
$\mathit{PutDown}(\mathit{obj})$	*obj* has been put down
$\mathit{ObjectPerceived}(\mathit{obj})$	The object has been perceived

Table 4: Event statements.

Evaluation

In order to evaluate the feasibility and potential of our simulation-based temporal projection framework we discuss four aspects. First, we show that important prediction tasks that are tedious, difficult, or even impossible to answer by abstract temporal projection mechanisms can be handled elegantly and accurately in our approach. Second, we show the feasibility of our approach by referring to the prediction-based debugging of a natural language Internet instruction for table setting. Third, we state the computational resources required by our approach: logging at simulation speed and answer times for queries. Finally, we give evidence that our approach is not suitable for the computation of probability distribution but sufficient for plan debugging.

Diagnosing Unexpected Events. In action logic representations, unexpected events are hard to model because the preconditions stated for actions in transition systems typically exclude the situations for which the effects of abstract actions are difficult to predict. So, typically the model for picking up an object is defined for situations where the hand is empty but not when objects are in the hand. Also, the plans in our approach specify how the robot is to react to sensory input rather than specifying strict plans handling only well-defined contingencies. In our approach events are detected and recognized in simulation data and therefore many more unexpected events can be predicted. Thus, for the actions in our system we specify expected events such as a collision of the robot's gripper with the object to be picked up. All other collisions are unexpected ones. The set of expected events at time *t* can be queried using the function $\mathit{ExpectedEvents}(t)$:

$$\begin{aligned}&\mathit{UnexpectedEvent}(\mathit{event}, t) \Leftrightarrow\\ &\quad \mathit{Occurs}(\mathit{event}, t) \wedge\\ &\quad \neg \mathit{Member}(\mathit{event}, \mathit{ExpectedEvents}(t))\end{aligned}$$

By using unexpected events, our robot is for example able to debug incompletely specified actions. Thus when debugging a natural language instruction for setting the table, the instruction tells the robot to put a plate in front of the chair but does not specify *on the table*. The robot infers by default that the supporting entity is the same one as the one for the reference object — the floor. Now, when the robot projects the action by putting the plate in front of the chair it will detect an unexpected collision which suggests that floor is not the right supporting entity but should be replaced with the table.

Interfering Effects of Simultaneous Actions Consider, for example, mobile manipulation where the robot moves its arm while navigating. In order to predict whether the robot will collide with objects the projection mechanism has to consider the superposition of the effects of navigating and reaching. In general, the interference of effects of concurrent actions can be extremely complex and heterogeneous. Such effect interferences are extremely hard to model in transition systems. The simulation based projection they come for free because the simulator works at a high temporal resolution and applies in each cycle the dynamic rules for all active physical processes.

Again, in order to predict whether the concurrent reaching and navigation will cause a flawed behavior we simply have to ask the simulation whether or not an unexpected collision of the arm and another object occurred.

Diagnosing Incorrect Beliefs. Many flaws of plan execution are caused by incorrect or inaccurate belief states. As an example, we state an incorrect belief of the location of the robot as follows:

$$\begin{aligned}&\mathit{IncorrectBelief}(\mathit{Loc}(\mathit{Robot}, \mathit{pos}_B), t_i) \Leftrightarrow\\ &\quad \mathit{Holds}_W(\mathit{Loc}(\mathit{Robot}, \mathit{pos}_W), t_i)\\ &\quad \wedge \mathit{Holds}_B(\mathit{Loc}(\mathit{Robot}, \mathit{pos}_B), t_i)\\ &\quad \wedge \neg(\mathit{pos}_B =_{pos} \mathit{pos}_W)\end{aligned}$$

Unachieved Intentions. Another example of a flawed belief state is that the robot believes that it has succeeded in navigating to a location because its navigation routine terminated with signaling successful task achievement but in the simulator the robot didn't arrive at the right location. This is stated as follows and can be evaluated on our projected execution scenarios.

$$\begin{aligned}&\mathit{FailedNavigation}(\mathit{task}) \Leftrightarrow\\ &\quad \mathit{TaskGoal}(\mathit{task}, \mathit{Achieve}(\mathit{Loc}(\mathit{Robot}, \mathit{pos}_B)))\\ &\quad \wedge \mathit{TaskStatus}(\mathit{task}, \mathit{Done}, t)\\ &\quad \wedge \mathit{Holds}_B(\mathit{Loc}(\mathit{Robot}, \mathit{pos}_B), t)\\ &\quad \wedge \mathit{Holds}_W(\mathit{Loc}(\mathit{Robot}, \mathit{pos}_W), t)\\ &\quad \wedge \neg(\mathit{pos}_W =_{pos} \mathit{pos}_W)\end{aligned}$$

Plan Debugging Besides the comparatively simple examples shown above, we have evaluated our simulation based projection mechanism by using it in a transformational planner to debug a plan imported from natural language instructions. More specifically, the plan was generated from the table-setting task as described at *www.wikihow.com*[2]. It had major flaws due to underparameterized goal locations, missing actions, etc.

Computational Resources Projection of a plan runs in simulation time and the complete reasoning that is done to infer the behavior flaws of the plan in one step is done in less than 10 seconds on a common PC. Individual queries of behavior flaws run in fractions of a second.

Probabilistic Plan Debugging Because in physics- and sensor-based simulation for high fidelity temporal projection sampling of projected execution scenarios is necessary, the approach is not suitable for computing probability distributions over expected execution scenarios. It is however fully sufficient for probabilistic plan debugging, that is if we consider planning systems that debug inherent behavior flaws that occur with a probability *p* and a detection rate of *e*. Beetz & McDermott (1997) state how many samples have to be drawn depending on the specified *p* and *e*.

Related Work

Many planning systems, in particular partial-order planning systems have their predictive mechanisms deeply integrated into the planning algorithms. Planning algorithms add constraints to plans to ensure that future states will satisfy the preconditions of the actions that are to be executed in those states. The action models used by such planners are typically coarse grained and formulated in the Planning Domain Description Language PDDL [Fox & Long, 2003].

[2] http://www.wikihow.com/Set-a-Table

Our temporal projection mechanism is designed for the use in transformational and case-based planners (such as XFRM [McDermott, 1992]), which completely separate the generation of plan hypotheses from the testing through temporal projection. In the context of these systems, powerful temporal projection mechanisms have been developed.

Hanks (1990) developed an algorithm to compute probabilistic bounds on the states resulting from action sequences. McDermott (1997) developed a very powerful totally ordered projection algorithm capable of representing and projecting various kinds of uncertainty, concurrent threads of action, and exogenous events. This algorithm is used in the planning system system XFRM [McDermott, 1992] that adds capabilities to project the computational state of the robot while executing a plan. Beetz & Grosskreutz (2000) further elaborated on the language for specifying action models and grounded their representation into probabilistic hybrid automata as a formal underpinning. The representation language is rich enough to accurately predict reactive navigation behavior of an autonomous robot office courier.

Asimo [Cambon, Gravot, & Alami, 2004] is a robot action planner that is unique in that the reachability of places at the symbolic representation layer is grounded into the motion planning mechanisms of the robot. Thus, symbolic action planning calls the motion planning algorithm to check whether or not the reachability of a particular place is given.

Most high-end manipulation robots, such as Justin and HRP-2 already come with very accurate simulation engines. These simulation models are designed to be as accurate and detailed as possible in order to transit from simulations to the real robots very smoothly and with minimum effort. We use the Gazebo open-source simulator as our physics- and sensor-based simulation engine.

Conclusions

In this paper we have proposed physics- and sensor-based simulation for high fidelity temporal projection as an alternative temporal projection method for AI planning, which is tailored for applications such as autonomous mobile robot manipulation. Many behavior flaws and failure conditions for robot behavior that are difficult or impossible to represent in transition models widely used in AI planning, come at very low cost in the physics- and sensor-based simulation for high fidelity temporal projection of realistic robot behavior. We believe that our approach to temporal projection will make AI planning applicable to modern mobile manipulation platforms that perform pick-and-place tasks in realistic environments. We expect that this expressiveness of simulated rather than symbolically projected behavior enables us to realize and deploy AI action planning systems on autonomous manipulation platforms that can forestall costly misbehaviors and thereby substantially improve the performance of the robots.

In our ongoing research we apply the techniques to pick-and-place tasks in human living environments and the preparation of meals in simulation. At the same time we run the plan language and individual manipulation plans on the real robot where they prove themselves to be reliable and flexible enough for real robot control.

Acknowledgements The research reported in this paper is supported by the cluster of excellence CoTeSys (Cognition for Technical Systems, *www.cotesys.org*).

References

Beetz, M., and Grosskreutz, H. 2000. Probabilistic hybrid action models for predicting concurrent percept-driven robot behavior. In *Proceedings of the Sixth International Conference on AI Planning Systems*. AAAI Press.

Beetz, M., and McDermott, D. 1997. Fast probabilistic plan debugging. In *Recent Advances in AI Planning. Proceedings of the 1997 European Conference on Planning*, 77–90. Springer Publishers.

Beetz, M.; Burgard, W.; Fox, D.; and Cremers, A. 1998. Integrating active localization into high-level control systems. *Robotics and Autonomous Systems* 23:205–220.

Beetz, M. 2000. *Concurrent Reactive Plans: Anticipating and Forestalling Execution Failures*, volume LNAI 1772 of *Lecture Notes in Artificial Intelligence*. Springer Publishers.

Cambon, S.; Gravot, F.; and Alami, R. 2004. A robot task planner that merges symbolic and geometric reasoning. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, 895–899.

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

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDLthe planning domain definition language. *AIPS-98 planning committee*.

Hanks, S. 1990. Practical temporal projection. In *Proc. of AAAI-90*, 158–163.

Johnston, B., and Williams, M. 2008. Comirit: Commonsense Reasoning by Integrating Simulation and Logic. In *Artificial General Intelligence 2008: Proceedings of the First AGI Conference*, 200. IOS Press.

McDermott, D. 1992. Transformational planning of reactive behavior. Research Report YALEU/DCS/RR-941, Yale University.

McDermott, D. 1997. An algorithm for probabilistic, totally-ordered temporal projection. In Stock, O., ed., *Spatial and Temporal Reasoning*. Dordrecht: Kluwer Academic Publishers.

Morgenstern, L. 2001. Mid-sized axiomatizations of commonsense problems: A case study in egg cracking. *Studia Logica* 67(3):333–384.

Müller, A.; Kirsch, A.; and Beetz, M. 2007. Transformational planning for everyday activity. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, 248–255.

Siskind, J. 2003. Reconstructing force-dynamic models from video sequences. *Artificial Intelligence* 151(1):91–154.

Thielscher, M. 2009. A unifying action calculus. *Artificial Intelligence Journal*.