Specializing Underdetermined Action Descriptions Through Plan Projection

Gayane Kazhoyan and Michael Beetz*
{kazhoyan, beetz}@cs.uni-bremen.de

Abstract—Plan execution on real robots in realistic environments is underdetermined and often leads to failures. The choice of action parameterization is crucial for task success. By thinking ahead of time with the fast plan projection mechanism proposed in this paper, a general plan can be specialized towards the environment and task at hand by choosing action parameterizations that are predicted to lead to successful execution. For finding causal relationships between action parameterizations and task success, we provide the robot with means for plan introspection and propose a systematic and hierarchical plan structure to support that. We evaluate our approach by showing how a PR2 robot, when equipped with the proposed system, is able to choose action parameterizations that increase task execution success rates and overall performance of fetch and deliver actions in a real world setting.

I. INTRODUCTION

There have been remarkable demonstrations of autonomous mobile manipulation robots performing everyday activities such as folding clothes [1] or washing dishes [2]. However, the robot control programs for executing these tasks are only applicable in the specific settings that they are implemented for. To enable the robots escape laboratory settings and enter unstructured environments such as human households, they need to be able to autonomously manipulate a big variety of objects in a multitude of task contexts, while dealing with differences in the environments and constant failures due to inaccuracies in sensors, actuators and the world representation.

To generalize robot control programs towards different objects, tasks, environments and robot platforms, we introduced the concept of entity descriptions [3]. These are abstract underspecified symbolic descriptions of task-relevant entities (objects, locations, actions, etc.) that are being grounded during execution into robot’s environment through perception and reasoning. During grounding they are augmented with symbolic and subsymbolic data that specializes them to the environment at hand. Here is an example action description:

(an action (type picking-up)
 (object (the object (type cup)
 (pose a-pose )))

which gets augmented with information such as which arm to pick up with and with which trajectory:

(an action (type picking-up)
 (object (the object (type cup)
 (pose a-pose )))
 (arm left)
 (trajectory pose-1 pose-2 ...)

Listing 1. Grounded description of a pick up action

*The authors are with the Institute for Artificial Intelligence, University of Bremen, Germany.

There can be different groundings for the same entity description which result in different outcomes, including a variety of failures that can happen. For example, for picking up a certain object, the chosen grasp type, robot base location, arm to use, the grasping force etc. decide if the action will be successful or, on the contrary, if the object will slip out, be out of reach, or if the trajectory will result in the robot colliding with the environment or knocking objects over (see Figure 1).

Fig. 1. Projecting a plan for ”fetch object of type milk” action:
(top-left) step 1: robot approaches object – base collides with furniture;
(top-right) step 2: backtracking, relocate and try left arm – arm in collision;
(middle-left) step 3: try right arm – no IK solution for next pose in trajectory;
(middle-right) step 4: relocate and try left arm again – successful;
(bottom) step 5: execute chosen parameterization on real robot.
Robot control programs written with entity descriptions profit from a clean separation between the control flow and decision making. The control flow specifies the ordering of actions and failure handling behaviors. Decision making performs the reasoning necessary to find groundings and parameterizations of entities, which lead to successful task execution. Introducing entity descriptions into programs makes them more scalable and generalizable towards different contexts as well as more compact and readable, thereby decreasing code development and maintenance efforts. However, the effort of specializing the program to the specific execution environment is thereby shifted from the programmer onto the robot itself.

This paper concentrates on reasoning required to adapt a general plan to a specific one tailored towards a known environment and scenario at hand by the means of reasoning into the future. The adapting of the plan is implemented by specializing the underspecified entity descriptions contained in the plan. Intelligence here lies not in finding the correct sequence of actions but in the choice of action parameterizations that avoid failures and lead to successful task execution. For example, the robot might need to answer a question such as: "If I grasp the cup from the handle with my left arm, will I be able to place it at the designated location without having to regrasp?" and based on the answers choose the arm to grasp with. We present a fast plan projection mechanism\(^1\) which can be used to evaluate how well a plan with a particular parameterization will perform. By thinking ahead of time through light-weight plan projection, the parameters of actions can be optimized by executing multiple runs of the same part of the plan with different parameterizations in the projection environment and choosing the best ones according to a cost function. The cost function can be based on the number of occurred handled or unhandled failures, on lengths of trajectories etc. Projection results can then be easily integrated into the robot’s plan to execute the optimized part of the plan in the real world right away.

To enable the robot to infer if any of its executed actions were successful or triggered failures, i.e. to be able to find causal relationships between action parameterizations and task success, the robot needs to have means for introspection. The importance of being able to reason about own execution has been argued, among others, by Ronald Brachman in his seminal talk titled “Systems That Know What They Are Doing” [4], where he explained how the ability of a system to comprehend what it is doing and why makes it robust in the face of unanticipated circumstances. To implement introspection, we store plan-relevant information during real-world and projected execution into a data structure and provide the robot or its programmer with an interface for querying and reasoning with this knowledge.

Plans have to be well-structured such that relevant information — action outcomes, failures that happened, parameterizations used, order of action execution etc. — is available and easily retrievable to support introspection. We propose systematic and hierarchical plan structure, based on the entity descriptions concept [3], and demonstrate how such plan structure is advantageous for introspection on the example of fetch and deliver plans.

The novel contributions of this paper are:

- an approach to structure robot plans such that they are easily introspected;
- a model for fetch and deliver plans and their implementation;
- fast plan projection mechanism for a mobile manipulation robot as means to generate different behavior for the same symbolically-represented action;
- mechanisms to incorporate plan projection online into a specific segment of the robot control program.

We evaluate our approach by showing how a PR2 robot is able to choose action parameterizations that increase task execution success rates and overall performance of fetch and deliver actions in a real world setting by using our system.

II. RELATED WORK

In cognitive science, predicting how motions affect the future evolution of the environment was found to play an essential role in human manipulation capabilities. An example of this line of thought is Hesslow’s simulation theory of cognition [5], which proposes that thinking is imagined interaction with the environment, during which behavior is simulated by activating motor and perceptual stimuli of the human but execution is suppressed such that it is not visible on the outside. Szpunar et al. [6] introduce the term prospection to describe future-oriented cognition and propose a taxonomy thereof, which defines four modes of future thinking: simulation, prediction, intention and planning. These can include subsymbolic as well as semantic reasoning. A brain-inspired cognitive architecture based on these ideas has been proposed by Shanahan [7]. It is based on a dynamical approach [8] to cognitive science: in contrast to traditional approaches, which use language and symbolic reasoning as conceptual foundations for their cognitive architectures, the author makes use of an “analog” representation, which is realized through a neural network and is structurally closer to the perception-action domain of a cognitive agent. The agent runs parallel simulations with each action it can execute to estimate which one would bring to a state with highest reward. The architecture is implemented with a large-scale neural network and is applied to a simple robot that can turn and perceive colorful cylinders.

In classical AI, symbolic plan projection is applied to predict the future state of the world. Projection is considered on the basis of axiomatized models of actions, which are atomic entities that have preconditions and effects. State-space planners, such as STRIPS [9] and more recent HTN-based planners such as SHOP2 [10], search through state transition systems with atomic transitions to find a sequence of actions, which is predicted to lead to the goal state. In the domain of mobile manipulation, the choice of action parameterization is crucial for task success. There is a large number of parameters and most of them are subsymbolic.

---

\(^1\)Open-source code and tutorials: [http://cram-system.org](http://cram-system.org)
Unfortunately, classical AI planners have a difficulty in handling such complex domains, therefore, they abstract away from motions. For example, an atomic grasping action is assumed to have an effect of an object necessarily being in hand after its execution, if certain preconditions have been met. In real world, grasping trajectories, reachability, occlusions, friction forces etc., which are abstracted away from in classical AI, are crucial for successful action execution.

On the other hand, also in classical AI prediction of the future has been considered as a useful tool and has been researched as a separate component of the planning system. One of the first works in that direction is by Hanks [11], where he argues that the classical AI approach of constraining the search space of possible outcomes by simplifying the world state and action representations may not generate accurate enough projection results to be practically applicable. Instead, he suggests to consider comprehensive world and action representations but restrict the search space to only “important” or “significant” outcomes. Continuing this line of work, Beetz at al. [12] present a plan revision technique that improves the behavior of agents by eliminating probable execution failures. They estimate the frequency of occurrence of failures and apply plan transformation rules to forestall the most probable ones, based on running a small number of plan execution samples in projection. The work demonstrates advanced techniques, however, the application domain of the system is a simulated delivery robot in a 2D grid world, whereas in the real world domain it is very difficult to construct a realistic probabilistic model of robot’s actions and their effects. In future, we are planning to learn such models using large amounts of robot experience data.

In the robotics community, the idea of using simulators to improve execution is not new. Rockel et al. [13] show a system where simulation is integrated into the planner, such that the latter can choose the appropriate action and parameters based on simulation. This allows the robot to learn a new skill such as balancing an object on a tray. Kunze et al. [14] present a temporal projection system that translates naive physics problems into parameterized simulation tasks with support of first-order representation reasoning over the execution results. With this system the robot can estimate parameters of actions, e.g., for manipulating an egg. Abelha et al. [15] use a simulator to estimate how a particular tool performs in a given task: they vary the parameters of the action of using a tool to estimate the best parameterization based on a “task function”. The difference between the aforementioned works and our approach is that they concentrate on short time span tasks and simulations thereof, whereas our approach implements fast temporal projection over multiple plan steps and can infer a full set of parameters at once. From a practical perspective, traditional simulation-based approaches are computationally expensive and have a low real-time factor, whereas our plan projection is very fast with respect to the pace of action execution (see Section VII).

The closest related work that deals with large time span temporal plan projection is by Mösenlechner et al. [18]. The system described in [18] considers simple sequences of actions designed for simulation. Our approach aims at complete plans, which run on a real robot and are, therefore, much more complex than those used in simulation. This requires an approach to structure plans such that they are easily introspected, which is presented in this paper. Additionally, running projection on a real robot during execution and integrating results of projection-based reasoning back into the executive poses another challenge, which has been tackled in this paper.

III. PLAN ARCHITECTURE

To enable convenient performance introspection, plans have to be nicely structured, i.e. be modular, explicit and transparent. We have developed plans for fetching and delivering objects that have such a structure. In these plans, the control flow is separated from the reasoning necessary to ground entity descriptions into the environment at hand. The control flow of fetch is illustrated in Figure 2.

![Fig. 2. Task tree of a fetching action](image)

The fetch plan consists of sequentially executing four other subplans, which can generate failures of 5 different types that fetch has to be able to handle. Some of the failures relevant for the fetch and deliver domain are listed in Table 1.

| Failure Type                        | Description                                                                 |
|-------------------------------------|-----------------------------------------------------------------------------|
| perception-object-not-found         | Perception system returned no matching object                                |
| object-nowhere-to-be-found          | Object was not found at the search location despite all failure recovery     |
| navigation-pose-unreachable         | Navigation trajectory is blocked                                            |
| navigation-pose-in-collision        | Navigation goal results in a collision with the environment                 |
| navigation-goal-not-reached         | Navigation controller finished but goal was not reached                      |
| put-goal-unreachable                | Look goal tries to twist robot’s neck                                       |
| manipulation-pose-unreachable       | No IK solution exists for pose                                              |
| manipulation-goal-not-reached       | Manipulation controller finished but goal was not reached                   |
| manipulation-pose-in-collision      | Manipulation trajectory generates a collision with the environment         |
| gripper-closed-completely           | Gripper closed completely although an object was expected to be grasped     |

TABLE 1

**COMMON FAILURES FROM FETCH AND DELIVER DOMAIN**

The default failure recovery strategy of fetch is to sample a new robot base location and retry, as illustrated with the red arrows in Figure 2. The pick-up subplan consists of four other subplans, two of which are executed in parallel, as illustrated with the blue box. If fetch cannot handle a failure locally, it throws an object-unfetchable failure to the higher level of the plan hierarchy.
The plans in our system are implemented using the CRAM Plan Language (CPL) [17], which is a domain-specific language that provides syntactic sugar for implementing parallelism and synchronization, contains failure handling constructs targeted at robotic applications and implements the entity descriptions concept mentioned above. The fetch plan, written in CPL, is simple and concise. The knowledge required to execute the plan successfully in a given environment is inferred through the reasoning rules for grounding entity descriptions. Table II shows all the knowledge preconditions of the fetch plan. These rules define the search space of plan projection, from which the sampling is done.

In projection, all the motions of the robot are not continuous, as in traditional simulators, but discrete, so the robot goes through key poses of motions by "teleporting". This is the level of abstraction sufficient for our plan projection framework for making realistic predictions about action outcomes: physics-based methods provide fine-grained information sufficient to perform geometric reasoning. Opposite of the precision requirement, projection also should not significantly delay execution, i.e. it should be much faster than realtime, hence, ensuring the correctness of motion controller trajectories is out of its scope. We assume that low-level controllers generate motions that satisfy the constraints given by the plan, and in case the controllers throw a failure, those are handled by well-designed failure recovery strategies. Thus, we achieve modularization and ensure that our plans satisfy the design requirements, while maintaining necessary accuracy by considering low-level motions through their key poses (see more related discussion in Section VIII).

We use the same geometric world for robot’s belief state representation and for projection. Due to this tight integration, it is easy to initiate projection with the current belief state of the robot at any point in time and manipulate it for projecting into the future, then reset it back to the original state representing the real world once projection is over.

We apply the projection library to our carefully designed plans and do performance introspection on them. As opposed to the typical model-based approach to action planning, where control routines are modeled in a purely symbolic way, our system represents the control routines in a subsymbolic way but at the same time such that it would be possible to symbolically infer consequences of executing a plan. This is described in the next section.

V. PERFORMANCE INTROSPECTION

The main data structure in which plan-relevant information is stored during execution is the task tree (see Figure 4). The nodes of the tree correspond to tasks. A task is a represen-
tation of the runtime state of an annotated segment of the robot control program that is semantically meaningful in the context of plan execution and is important for introspection purposes. The most common task is the representation of an action description that is performed within the plan. Every node in the task tree contains a unique path, which is used for indexing and searching, a status, which can be any of created, running, suspended, succeeded, evaporated or failed, pointers to the parent node and children nodes, the code expressions of the task, the parameters with which it has been called, information about its failures etc. The task tree is automatically generated at runtime while tasks are being executed. To access the task tree and to reason on it, an API consisting of first-order logic predicates is defined. The ones relevant for this paper are listed in Table III.

Let us consider a transporting action (see Figure 5).

![Fig. 5. Task tree of transporting action](image)

We would like to project its plan with different parameterizations of the fetch and deliver actions and choose the one that leads to successful execution. We might also want to compare the parameterizations based on a certain cost function, e.g., a function that compares lengths of trajectories the robot would have to execute. Thus, we use the task tree Prolog API described above in the following way:

```
successful_fetch_and_deliver_params(ParentTaskPath, PickNavAction, PickAction, PlaceNavAction, PlaceAction) :-
  action_subtask(ParentTaskPath, fetching, FetchTask, FetchAction),
  task_path(FetchTask, FetchTaskPath),
  action_subtask(FetchTaskPath, picking-up, PickTask, PickAction),
  task_outcome(PickTask, succeeded),
  action_task(previous_sibling(FetchTaskPath, PickTask, navigating, PickNavTask),
  action_task(DeliverTaskPath, succeeding, DeliverTask),
  task_outcome(DeliverTask, succeeded),
  action_subtask(DeliverTaskPath, placing, PlaceTask, PlaceAction),
  task_outcome(PlaceTask, succeeded),
  action_task(previous_sibling(DeliverTaskPath, PlaceTask, navigating, PlaceNavTask),
  action_task(DeliverTaskPath, navigating, PlaceNavTask, PlaceNavAction).
```

We extract the fetching and delivering tasks from the task tree and make sure that their outcomes are succeeded. If not, the rule fails and we do not get any parameter bindings, which means that the projection run was not successful. Next, we extract the picking up action from the fetching subtree and the action description corresponding to that task. As we are applying introspection after execution has finished, in case of successful pick up the action description has to be already grounded. Therefore, we can access all the parameters of that grounding, including the arm that was used, the grasp type, even the trajectories. Once we have the picking up action, we find the navigating action that last preceded the pick up. That action contains the location description that was used to position robot’s base.

Thus, with a small number of queries we can access all the parameterizations of the general plan that were used to specialize it to the environment at hand. This mechanism is only made possible due to the design of plans that we follow, in which the control flow is separated from the reasoning processes.

| Table III | Predicates for accessing task tree data |
|-----------|----------------------------------------|
| task(Task) | Binds Task to any task of the current task tree |
| task(SubtreePath, Task) | Binds Task to any task of subtree defined with SubtreePath |
| task_path(Task, Path) | Gives the unique path of the task node defined with Task and binds it to Path |
| task_outcome(Task, Outcome) | Binds the result of Task to Outcome |
| task_failure(Task, Failure) | If Task failed, binds its failure object to Failure |
| task_created(Task, Time) | Binds the timestamp of creation of Task to Time |
| task_started(Task, Time) | Binds the timestamp of when Task started execution to Time |
| task_ended(Task, Time) | Binds the timestamp of when Task execution ended to Time |
| action_subtask(SubtreePath, Task) | Binds all tasks from SubtreePath corresponding to action descriptions to Task and their action description to Action |
| action(SubtreePath, ActionType, Task) | Binds all tasks from SubtreePath corresponding to action descriptions of type ActionType to Task |
| action_task(previous_sibling(SubtreePath, Task, ActionType, Task) | For an action task Task in SubtreePath finds the previous action task of type ActionType and binds it to PrevTask |
| action_task_next_sibling(SubtreePath, Task, ActionType, Task) | Binds the next action of type ActionType of an action task Task in SubtreePath to NextTask |

These predicates can be used by the robot as building blocks for answering questions such as “What was the last action I was trying to perform?”, “Which parameters did I use?”, “Was the action successful?”, “What were the failures?” etc. For example, if a placing action failed, the robot could crawl the task tree for the picking up action that preceded the failed placing action to see if the source of failure could have been that the object was picked up in a wrong way. As the input parameters of tasks are stored in the task tree, the robot can access all the action parameterizations that it used during execution and reason about them by reading out the results of grounding the entity descriptions that were used as parameters of action tasks. To keep introspection queries simple and straightforward it is crucial for the task tree to be well structured. This is achieved automatically if the plans are designed in a structured and systematic way, as is, for example, the case with our fetching and delivering plans.
VI. INCORPORATING PROJECTION INTO EXECUTION

To run projection for finding plan parameterizations that lead to successful task execution we have implemented the with-projected-task-tree construct. It is wrapped around the segment of the robot control program that we would like to project. For example, in the transport plan illustrated in Figure 5 projection is run after the search action has been executed and the object has been found: as we would like to optimize the parameters of the picking up action and the placing action, the location of the object has to be known.

The signature of with-projected-task-tree is as follows:

\[
\text{(with-projected-task-tree entity-descriptions-to-optimize number-of-projection-runs cost-function-to-compare-results code-to-project-and-execute)}
\]

The transporting plan is, therefore, defined as follows:

\[
\text{(def-plan transport (?object ?search-location ?delivering-location)}
\]

\[
\text{(perform (an action)}
\]

\[
\text{(type searching)}
\]

\[
\text{(object ?object)}
\]

\[
\text{(location ?search-location))}
\]

\[
\text{(with-projected-task-tree (?fetch-robot-location ?pick-up-action ?deliver-robot-location ?place-action))}
\]

\[
\text{4 #'pick-best-parameters-by-distance}
\]

\[
\text{(perform (an action)}
\]

\[
\text{(type fetching)}
\]

\[
\text{(object ?object)}
\]

\[
\text{(location ?fetch-robot-location ?pick-up-action)}
\]

\[
\text{(pick-up-action ?pick-up-action))}
\]

\[
\text{(perform (an action)}
\]

\[
\text{(type delivering)}
\]

\[
\text{(target ?delivering-location)}
\]

\[
\text{(robot-location ?deliver-robot-location ?place-action)}
\]

\[
\text{(place-action ?place-action))}
\]

The code segment with fetching and delivering actions will be executed in projection four times and the resulting four parameters will be compared with the best projection run.

The code segment with fetching and delivering actions will be executed in projection four times and the resulting four parameters will be compared with the pick-best-parameters-by-distance cost function. Finally, the same code segment will be executed on the real robot with parameters from the best projection run.

VII. EXPERIMENTAL ANALYSIS

We evaluated our approach on a breakfast table setting scenario with a PR2 robot. The scenario included fetching 5 different objects and bringing them to the table. We executed it 10 times without our system and 10 times with it. In an effort to reduce the randomness factor in execution we constrained the initial as well as goal locations of objects to be constant in all the runs. The initial configuration we chose was random, with the constraint that objects should be at least 2 cm away from each other and not be completely out of reach of the robot. The setup is shown on Figure 6.

The robot transports the objects one by one in the following order: milk, cup, cereal, bowl, spoon.

The first action in the transport plan is the searching action, so the robot searches for the object of a specific type on the surface of the counter. As it does not know where exactly the object is and as its field of view is limited to the sensor’s image size, it samples random poses on the surface, navigates to a location from where the pose is visible, and moves its head to point at it. Then it calls the perception system. If perception fails, the robot picks a different pose on the surface and retries.

Once the object has been found, next in the plan are the fetch and deliver actions wrapped into with-projected-task-tree as shown in Figure 5. If projection is disabled, the robot samples a location to stand to reach the object, drives to the location, samples an arm and a grasp type to use and tries to reach. If there is no IK solution or there is an occlusion, a manipulation failure occurs, so the robot samples a different location to stand, drives there and retries. This backtracking behavior is time consuming and leaves an impression of incompetent behavior. Additionally, if the object placing orientation is difficult to achieve with a certain grasp and the robot is unlucky to sample that particular grasp, deliver will completely fail. If projection is enabled, the robot executes four runs of projection. We refer to [12] for the proof of an argument that with a small number of randomly generated execution scenarios it is nonetheless very likely that the probable failures will be eliminated. Projection is used to choose the following four parameters: the arm to grasp with, the grasp type and the locations of the robot base for picking up and placing the objects. Action parameterizations in successful projection runs are evaluated based on a heuristic approximation of distances that the robot would have to move and, thus, the best run is chosen. Chosen plan parameters are then used in the real world to execute the fetch and deliver actions.

Our perception system has about 3 cm precision for the objects in the experimental setup, which tends to improve when the robot gets closer to the object. Due to that, the fetching plan reperceives the object directly before grasping. We were faced with two alternatives: either perceive the object once, project, then reuse generated trajectories even if the result of reperception varies highly from the previous result, or not reuse the trajectories but rather only predict the arm and the grasp type that is most likely to succeed. We went in favor of the second, as reusing trajectories proved to be very prone to misgrasping errors.
As the aim of projection is to find the action parameterization that leads to successful task execution, we chose our evaluation criteria to be the success rate of actions and the number of failures that happen. The types of most prominent failures that decided the outcome of actions were the ones related to reachability, including *object-unreachable*, *manipulation-pose-unreachable* and *navigation-pose-unreachable*, and the environment collision failures, including *manipulation-pose-in-collision* and *navigation-pose-in-collision*. Tables below show statistics from the first run of the system without using projection (Table IV) and the first run with projection (Table VI).

### Table IV
**Experimental results of one scenario run without projection**

| Object | Sum failures | Coll. fail. | Reach. fail. | Proj. success | Grasp used | Arm used | Success | Infer. time | Runtime |
|--------|--------------|------------|--------------|---------------|------------|----------|----------|-------------|----------|
| milk   | 45           | 13         | 0            | 68            | yes        | right    | yes      | 179.9       | 138.5    |
| cup    | 52           | 22         | 14           | 95            | yes        | right    | yes      | 137.9       | 137.9    |
| cereal | 22           | 10         | 0            | 68            | yes        | right    | yes      | 137.9       | 137.9    |
| bowl   | 17.25        | 2.8        | 2.8          | 4.0           | yes        | right    | yes      | 137.9       | 137.9    |
| spoon  | 11.8         | 5.4        | 7.0          | 11.09         | yes        | right    | yes      | 137.9       | 137.9    |
| Total  | 55.45        | 100%       | 100%         | 100%          | 91%        | 91%      |          |             |          |

### Table V
**Results averaged over ten scenario runs without projection**

| Object | milk | cup | cereal | bowl | spoon | Total | Per obj. |
|--------|------|-----|--------|------|-------|-------|----------|
| Coll. fail | 6.2  | 1.8 | 2.6    | 6.8  | 22.4  | 4.48  |          |
| Reach. fail | 22.25 | 9.0 | 11.8   | 5.4  | 7.0   | 55.45 | 11.09    |
| Total fail  | 22.25 | 9.0 | 11.8   | 5.4  | 7.0   | 55.45 | 11.09    |
| Success rate | 75%  | 100%| 100%   | 100% | 80%   | 91%   | 91%      |

### Table VI
**Experimental results of one scenario run with projection**

Table VII shows the average number of failures that happen in real world when using fast plan projection.

### Table VII
**Results averaged over ten scenario runs with projection**

Based on experimental data we can conclude that our system improves the success rate of fetch and deliver plans from 90% to 96%, which is not substantial since the robustness of the evaluation scenario is already considerably high. However, we additionally decrease the amount of manipulation-related failures and, therefore, times when robot physically backtracks, from 55.45 per run to 9.82, which is more than a 500% improvement (see Figure 7).

![Fig. 7. Success rate and number of manipulation failures comparison between executions without and with projection.](image-url)

Due to the high variety, we do not consider execution time an important evaluation criteria. In the 10 times we ran the scenario with projection-based reasoning disabled, execution time varied between 696.4s and 1181.4s. There is a vast number of factors that affect the runtime of execution on a robotic system, including the efficiency of computational processes, power of the underlying hardware, time optimality of robot’s controllers etc., as well as the amount of physical backtracking that happens due to, e.g., perception failures or suboptimal action parameterization samples. To give a
meaningful empirical estimate of execution time, the number of runs would have to be significantly higher than what we have. Considering that one scenario run takes about 15min, a large-scale evaluation might not be feasible. However, we do mention some statistics below to give a rough idea of our implementation efficiency for practical reasons. In the 10 runs of the scenario with projection enabled, 196 projection runs have been executed. The average projection time per one run of transporting one object was 6.2s, and 4.1s if excluding all the non-successful runs. This can be considered sufficiently fast with respect to the pace of action execution. The hardware used for projection was a laptop with 8GB RAM, an 8 core i7 CPU and an NVIDIA GeForce GT 650M graphics card. The average runtime of a full scenario run was 877.28s without projection and 823.85s with projection.

VIII. Conclusion, Discussion and Future Work

In this paper we presented the fast plan projection mechanism, which can be used to specialize a general plan towards the environment and task at hand by choosing action parameterizations that are predicted to lead to successful task execution. We showed how carefully designed plan structure can benefit plan introspection and how to apply introspection tools to choose parameterizations of executed actions that were predicted to succeed in the projection environment. We demonstrated how the results are easily integrated into robot’s executive such that the optimized part of the plan can be executed right away in the real world. Finally, we evaluated our approach by showing how a PR2 robot is able to use the system to choose action parameterizations that increase task execution success rates and decrease failure rates of fetch and deliver actions in a real world setting.

In the evaluation section we mentioned one limitation of our approach, which is a general limitation that any system that thinks ahead of time based on the current belief state has: if the belief state representation is inaccurate, projection has a higher chance of producing action parameterizations, which do not lead to successful task execution when transferred onto the real world. We overcome this problem by integrating our projection results only as a suggestion for the planner, and if suggested parameterization fails, execution continues with its default failure handling routines, trying to find a better parameterization without the help of projection-based reasoning. A similar limitation is the danger of the belief state changing while projection performs its inference. In our application scenarios, which happen in semi-controlled environments, external influence and, therefore, unexpected belief state changes happen with a sufficiently low frequency compared to the runtime of the projection-based inference.

One important assumption that has to be made about the projection mechanisms is that the probability distribution of failing in projection is similar to the real world. In our fetch and deliver scenarios it is the case as most failures happen due to (1) inverse kinematics solver not finding a solution, which is the exact same mechanism in projection and in real world, and (2) the goal poses being unreachable due to collisions with the environment, which are also realistic in the projection environment as we use a high-precision model of the environment. However, some other failures such as an object slipping away from the gripper, are not represented in our system, and that is a limitation. In future, we are planning to learn failure models to use in our projection environment based on data collected from real world experiments.

ACKNOWLEDGMENTS

This work was supported by DFG Collaborative Research Center Everyday Activity Science and Engineering (EASE) (CRC #1320).

REFERENCES

[1] S. Miller, J. Van Den Berg, M. Fritz, T. Darrell, K. Goldberg, and P. Abbeel, “A geometric approach to robotic laundry folding,” The International Journal of Robotics Research, vol. 31, no. 2, 2012.
[2] K. Okada, T. Ogura, A. Haneda, J. Fujimoto, F. Gravot, and M. Inaba, “Humanoid motion generation system on hrp2-jsk for daily life environment,” in IEEE International Conference on Mechatronics and Automation, 2005.
[3] G. Kazhoyan and M. Beetz, “Programming robotic agents with action descriptions,” in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017.
[4] R. J. Brachman, “Systems that know what they’re doing,” IEEE Intelligent Systems, vol. 17, no. 6, pp. 67–71, Nov 2002.
[5] G. Hesslow, “Conscious thought as simulation of behaviour and perception,” Trends in cognitive sciences, vol. 6, no. 6, 2002.
[6] K. K. Szpunar, R. N. Spreng, and D. L. Schacter,”A taxonomy of prospection: Introducing an organizational framework for future-oriented cognition,” Proceedings of the National Academy of Sciences, vol. 111, no. 52, 2014.
[7] M. Shanahan, “A cognitive architecture that combines internal simulation with a global workspace,” Consciousness and cognition, vol. 15, no. 2, 2006.
[8] R. D. Beer, “Dynamical approaches to cognitive science,” Trends in cognitive sciences, vol. 4, no. 3, 2000.
[9] M. Ghallab, D. Nau, and P. Traverso, Automated Planning: theory and practice, Elsevier, 2004.
[10] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman,”SHOP2: An HTN planning system,” Journal of artificial intelligence research, vol. 20, 2003.
[11] S. Hanks, “Practical temporal projection,” in AAAI, vol. 90, 1990.
[12] M. Beetz and D. McInernott, “Fast probabilistic plan debugging,” Recent Advances in AI Planning, 1997.
[13] S. Rockel, S. Konecný, S. Stock, J. Hertzberg, F. Pecora, and J. Zhang, “Integrating physics-based prediction with semantic plan execution monitoring,” in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015.
[14] L. Kunze, M. E. Dohla, E. Guzman, and M. Beetz,”Simulation-based temporal projection of everyday robot object manipulation,” in The 10th International Conference on Autonomous Agents and Multiagent Systems, 2011.
[15] P. Abela and F. Guerin, “Learning how a tool aids by simulating 3d models from the web,” in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017.
[16] L. Mösenlechner and M. Beetz, “Using physics-and sensor-based simulation for high-fidelity temporal projection of robotic robot behavior.” in ICAFS, 2009.
[17] M. Beetz, L. Mösenlechner, and M. Tenorth,”CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments,” in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010.
[18] L. Mösenlechner and M. Beetz, “Fast temporal projection using accurate physics-based geometric reasoning,” in IEEE International Conference on Robotics and Automation (ICRA), 2013.
[19] L. Mösenlechner and M. Beetz, “Parameterizing Actions to have the Appropriate Effects,” in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011.
[20] M. Beetz, F. Balint-Benczedi, N. Błodow, D. Nyga, T. Wiedemeyer, and Z.-C. Marton, “RoboSherlock: Unstructured Information Processing for Robot Perception,” in IEEE International Conference on Robotics and Automation (ICRA), 2015.