Knowledge-Driven Robot Program Synthesis from Human VR Demonstrations

Benjamin Alt[1,2] , **Franklin Kenghagho Kenfack**[2] , **Andrei Haidu**[2] , **Darko Katic**[1] , **Rainer Jäkel**[1] , **Michael Beetz**[2]

[1]ArtiMinds Robotics, Karlsruhe, Germany
[2]Institute for Artificial Intelligence, University of Bremen, Germany
{benjamin.alt, darko.katic, rainer.jaekel}@artiminds.com, {fkenghag, haidu, michael.beetz}@uni-bremen.de

Abstract

Aging societies, labor shortages and increasing wage costs call for assistance robots capable of autonomously performing a wide array of real-world tasks. Such open-ended robotic manipulation requires not only powerful knowledge representations and reasoning (KR&R) algorithms, but also methods for humans to instruct robots what tasks to perform and how to perform them. In this paper, we present a system for automatically generating executable robot control programs from human task demonstrations in virtual reality (VR). We leverage common-sense knowledge and game engine-based physics to semantically interpret human VR demonstrations, as well as an expressive and general task representation and automatic path planning and code generation, embedded into a state-of-the-art cognitive architecture. We demonstrate our approach in the context of force-sensitive fetch-and-place for a robotic shopping assistant. The source code is available at https://github.com/ease-crc/vr-program-synthesis.

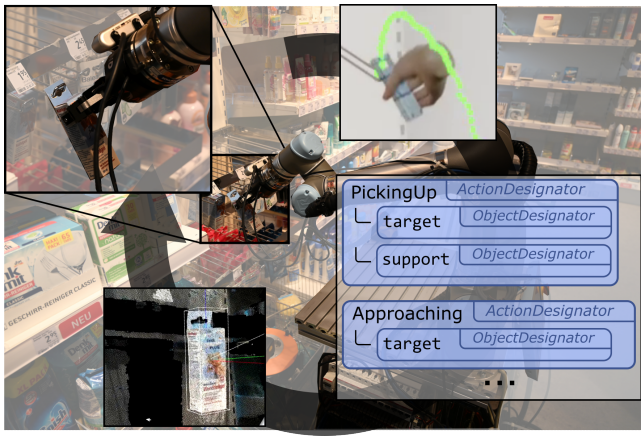

Figure 1: We propose a knowledge-driven approach to convert human demonstrations in virtual reality (top right) to executable robot programs by leveraging semantic task knowledge (center right) and knowledge-augmented perception (bottom).

1 Introduction

Robots are universal manipulators: Their versatility promises a future in which robots assist humans not just on industrial assembly lines, but also in everyday situations such as shopping or elderly care. Many everyday assistance tasks such as setting a table or fetching objects in a supermarket involve open-ended manipulation, requiring the robot to reason about tasks which cannot be exhaustively specified. It would be impossible or prohibitively expensive, for example, to provide a shopping assistance robot with a dedicated control program for each product it may be asked to fetch from a shelf. Instead, the robot should be programmed in a way that is sufficiently general to cover a wide array of objects, while at the same time allowing fine-grained control over important task-specific details such as force limits when handling fragile items like glass or fruit.

The adoption of robots for everyday assistance tasks will require novel methods of programming. Non-experts should be able to specify tasks, goals and constraints in an intuitive manner. Based on high-level human input, robot control programs should be automatically generated and specialized to meet the requirements of the demonstrated task and environment. Program synthesis for open-ended manipulation tasks requires reasoning about objects and their physical properties as well as adaptability to changes in the environment and the task requirements. ackrr approaches permit the design of cognitive systems capable of solving such open-ended tasks in a data- and compute-efficient manner.

We contribute a system that is capable of inferring executable robot control programs from a single human VR demonstration of a manipulation task. We decompose program generation into two steps – action interpretation and task execution. *Action interpretation* refers to the inference of an underspecified task such as "Hanging an object onto a hanger", which captures the human's intention, from low-level VR data. *Task execution*, in turn, generates a fully specified, executable robot control program for the inferred task. By parsing VR data into a semantically rich knowledge representation, and by designing and implementing a collection of general ontological knowledge about robot tasks as well as their preconditions and effects, the unification algorithm can be leveraged to infer a robot control program in order to perform the demonstrated tasks. We experimentally validate our system on two challenging real-world assistance tasks in a supermarket environment. Due to the principled use of knowledge representation and reasoning, our approach generates robust robot programs from a single VR demonstration, without requiring training data. More-

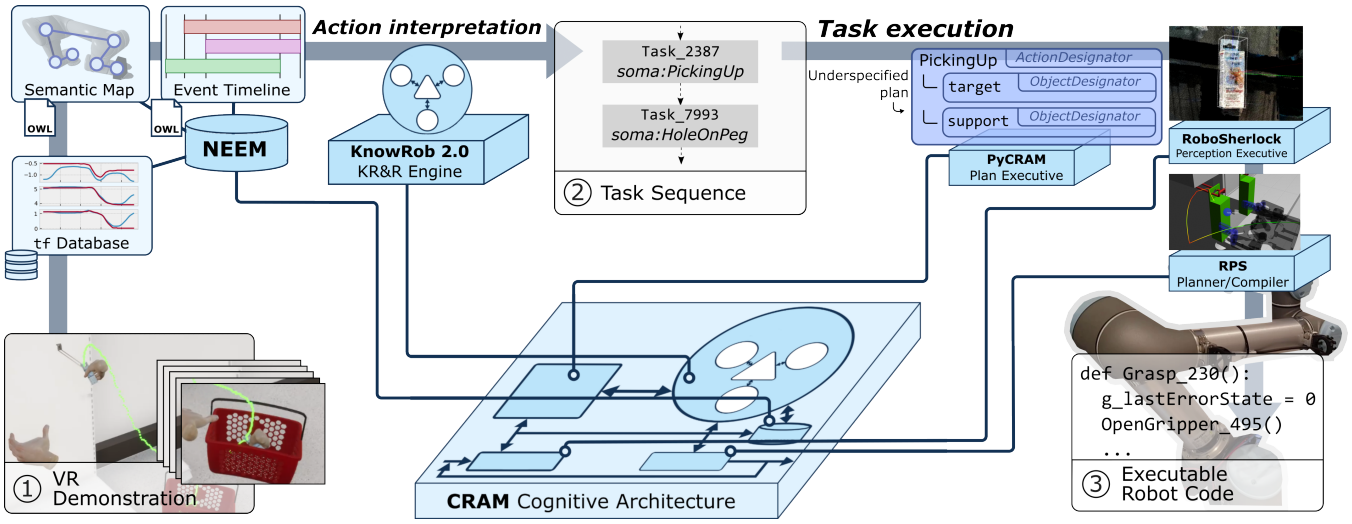

Figure 2: Overview of knowledge-driven robot program synthesis from virtual reality (VR) demonstrations. Humans demonstrate complex manipulation actions in VR (1). Using common-sense knowledge, an knowledge representation and reasoning (KR&R) engine can interpret this experience data as a sequence of underspecified tasks (2). This sequence is then translated into a plan, which is grounded to the real world via high-level reasoning, knowledge-based perception, motion planning and code generation. The resulting control program can be directly executed on the robot (3).

over, it immediately generalizes to arbitrary object poses, shapes, and environment configurations.

2 Overview

Fig. 2 provides an overview of the program synthesis system. Given a human demonstration of a manipulation task in VR, an executable robot program is automatically synthesized via a three-step process:

Knowledge extraction In a first step, the VR demonstration is automatically parsed into an episodic memory: A semantically rich representation of human or robot actions which contains both a symbolic log of detected events in terms of an ontology as well as a subsymbolic record of the motions of the agent and objects in the environment (see Sec. 3). This episodic memory is inserted into the knowledge base of a KR&R engine. Parsing VR demonstrations into episodic memories connects the demonstration to the extensive background knowledge provided by upper-level domain and application ontologies.

Action interpretation Episodic memories describe sequences of events, which may be contingent on the agent's (here, the human demonstrator's) abilities and the particular configuration of the VR environment. In order to generate executable programs for robots acting in real-world environments, the action sequence is *lifted* from the action (event) to the task (concept) level, discarding event-specific information such as durations or locations and inferring task-relevant information such as roles or constraints (see Sec. 4). This process is not one-to-one: The interpretation of an action can yield a set of possible tasks, depending on the context and the amount of available information. For example, an action during which a lid is moved onto a container can be interpreted as a `Closing` task, but also as a `Placing` task. Consequently, the interpretation of an entire demonstration typically yields more than one candidate task sequence.

Task execution Each inferred task sequence can be automatically mapped to a *plan*, which contain a list of actionable steps for the robot to fulfill the tasks (see Sec. 5). These initially underspecified plans are refined into executable robot programs via a flexible reasoning, perception and planning pipeline (see Fig. 5). Inferring underspecified plans enables learning solutions to tasks at a very high level, such as "placing an object onto a shelf" or "taking an object from a hanger", reducing the amount of required demonstrations to a minimum. The program generation pipeline will produce executable robot code specialized to the current environment, hardware and robot capabilities from these plans.

3 Knowledge Representation

Enabling robots to synthesize their own control programs requires reasoning about possible tasks, available actions and the properties of objects in the environment. We leverage methods of explicit knowledge representation to facilitate such reasoning.

3.1 Ontological Model of Tasks and Events

A knowledge-driven approach to robot program synthesis requires a semantic model of the meaning of tasks, actions and events and their relation to each other. We adopt the model proposed by the SOMA ontology (Socio-Physical Model of Activities) (Beßler et al. 2020). The goal of robot programs is to cause some `Events`[1] in the world (a robot

[1]We use a fixed-width font (e.g. `Grasping`) to indicate classes, properties etc. as defined in SOMA or other ontologies.

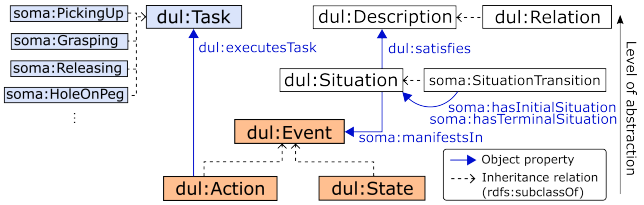

Figure 3: The SOMA model of `Tasks`, `Events` and `Situations` (Beßler et al. 2020).

grasps a bag of chips, lifts it, then puts it into a basket), which bring about one or more `Situations` (the bag of chips is in the basket) seen as consistent with a `Task` (shopping for chips). In this model, `Tasks` are `Concepts`, which describe how an `Event` should be interpreted. In the opposite direction, when a `Task` is executed, it results in one or multiple `Events` (see Fig. 3). Both the interpretation of `Events` as `Tasks` and the execution of `Tasks` to produce `Events` is highly context-dependent: The robot's gripper making contact with an object can imply `Grasping` or `Pushing`, depending on what happens afterwards. Similarly, the task of opening a container can be performed in a variety of different ways, depending on the capabilities of the robot.

Generating executable robot control programs for one or more ambiguously specified high-level tasks is the focus of most prior work on task-level programming. A comprehensive approach to program synthesis, however, must also consider ways for humans to express which tasks the robot should solve (e.g. via natural language or demonstrations in VR). Most such modalities require *interpretation*: In the case of natural language, the user's words (descriptions of intended actions) must be interpreted to tasks while taking the larger context into account. "Smash the button" may mean destroying it in the context of recycling or vigorously pushing it in the context of machine tending. In the case of VR demonstrations, the user takes actions in the VR environment, which must be interpreted to tasks depending on the types of objects in VR, their properties and the domain: A VR demonstration of the same screwing task, for example, will look very different if the demonstrator is using a powered or a manual screwdriver. For these reasons, we consider both the translation of actions into tasks ("interpretation", see Sec. 4) and the translation of tasks into actions ("execution", see Sec. 5).

3.2 Representing Demonstrations: Episodic Memories

To facilitate reasoning over human demonstrations, we propose to represent them as narrative-enabled episodic memories (NEEMs), semantically enriched execution traces in the CRAM ecosystem (Beetz et al. 2018). A NEEM has three components:

1. A *semantic map*, which is an ontology (derived from SOMA) containing a description of all relevant agents and objects in the environment, their properties and interrelationships.
2. An *event timeline*, which is an ontology (derived from SOMA) containing a set of timestamped `Event` individuals (`States` and `Actions`), as well as semantic annotations of the `Situations` manifesting in them (see Fig. 4). This realizes the model of tasks and events described in Sec. 3.1. `Situations` are descriptive contexts linking `Events` to descriptions of the world which hold over their duration - examples are `Relations` between objects (contact, support, containment, ...) or context-dependent object properties (affordances).
3. A *database* containing the poses of all relevant agents and objects over time, as well as the motions of their links.

A simplified example for a NEEM is shown in Fig. 4. Unlike raw VR data, this semantically rich representation of human demonstrations enables robots to reason about demonstrated action sequences as if it was their own experience. Because NEEM experience is expressed in terms of ontologies, differences between the VR world and the real setting such as kinematic differences between the human VR avatar and the robot or differences in object types or poses can be handled gracefully by the reasoner using ABox reasoning.[2]

3.3 Representing Task Knowledge

Our approach to program synthesis hinges on the interpretation of demonstrated actions as abstract tasks. In addition to a semantically rich representation of the demonstrations, this requires a similarly rich representation of task types and their meaning. In natural language, a task such as `Reaching` can be described as "moving the hand toward an object". A structured knowledge representation enable robots to perform common-sense reasoning about what tasks are feasible ("a robot can only grasp if its gripper is empty") or what makes a task successful ("after placing an object, the object is supported by a surface"). This knowledge representation must be sufficiently universal to cover arbitrary domains, and avoid requiring excessively specific demonstrations or descriptions, but it must also allow to draw specific inferences about what the robot must do to achieve a task in a given environment. We propose to describe tasks using a hybrid representation in which the task taxonomy is defined in terms of an ontology, while task semantics are defined as a set of Prolog rules over the ontology.

Task ontology To define what a task is, which classes of tasks exist (e.g. `Placing`, `Grasping`, `Opening`), which inheritance relationships exist between the different task types (e.g. `Grasping rdfs:subclassOf Manipulating`) and how this task hierarchy relates to other concepts such as actions or parameters, we use and extend the SOMA ontology. A `Task` is an `EventType` *classifying* an `Event`. Defining tasks in terms of SOMA ensures compatibility with other tools and frameworks in the CRAM

For readability, namespace prefixes are omitted or replaced by short-form prefixes (e.g. `Grasping` or `soma:Grasping` rather than `http://www.ease-crc.org/ont/SOMA.owl#Grasping`).

[2]The source code for generating NEEMs from VR demonstrations is available at https://github.com/ease-crc/vr-neem-converter.

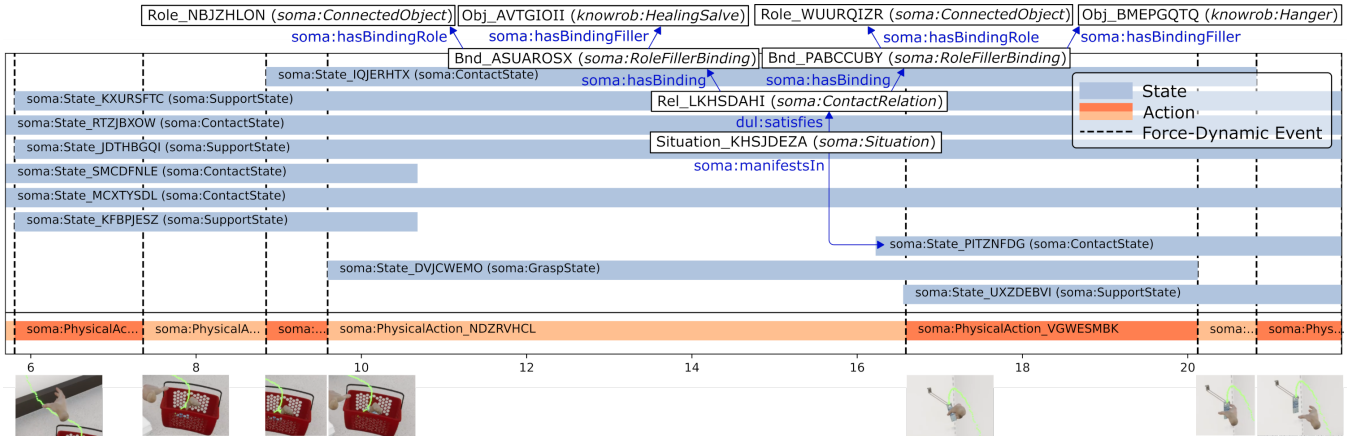

Figure 4: Timeline of a VR demonstration segmented by force-dynamic events.

Algorithm 1 Pre-, runtime and postconditions for a `PickingUp` task.

```
satisfies_pre(Act, soma:'PickingUp') :-
  % precondition 1: Some object O1 grasped
  has_initial_situation(Act, S1),
  object_grasped(O1, S1),
  % precondition 2: O1 supported by
      something
  has_initial_situation(Act, S2),
  object_supported(O1, O2, S2).

satisfies_run(Act, soma:'PickingUp').

satisfies_post(Act, soma:'PickingUp') :-
  % postcondition 1: O1 still grasped
  has_terminal_situation(Act, S1)),
  object_grasped(O1, S1),
  % postcondition 2: O1 not supported
      anymore
  forall(has_terminal_situation(Act, S2)),
    \+ object_supported(O1, O2, S2)).
```

cognitive architecture. When new domain- or application-specific tasks are added (e.g. for the experiments in 6), they are declared as subclasses of `soma:Task`.

Task semantics The ontology only specifies which task types are known and how `Tasks` are related to other entities in SOMA such as `Actions`, `Situations` or `Objects`. We define the task *semantics* (what distinguishes `Opening` from `Closing`? How are observed actions related to the specific tasks they execute?) as a set of Prolog rules in the KnowRob KR&R engine (see Fig. 1 for an example). This set of Prolog rules can be understood as axiomatic knowledge about tasks and how they relate to (observed) actions. We follow the established method of defining different tasks in terms of their preconditions, runtime conditions and postconditions. For a given task type TT and action A, a task of type TT is executed by A if

1. the state of the world just before A begins are consistent with the preconditions of TT,
2. the state of the world during A is consistent with the runtime conditions of TT,
3. the state of the world just after A is finished is consistent with the postconditions of TT.

In principle, tasks can be defined in terms of pre-, runtime and postconditions directly in the ontology via TBox axioms or SWRL rules. This would make it very easy to test if an observed action can be interpreted as executing a given task (because then, the ontology would remain consistent). But it makes it very hard to infer the set of tasks which an action possibly executes: Due to the open world assumption, the pre-, runtime and postconditions of some task would be satisfied unless something about the action violates them, and the set of preconditions must cover the infinitely large set of states of the world which preclude the task. By contrast, in a closed-world reasoning system like Prolog, the pre-, runtime and postconditions of tasks are by default violated. In such a system, defining the preconditions of a task only requires predicates covering the much smaller set of states of the world in which the task *can* possibly be executed.

An additional advantage of using a hybrid representation is is that tasks can be specified in very abstract and concise terms. To specify e.g. the preconditions of `PickingUp`, it suffices to state that some object must currently be grasped, and that this object is supported by some other object (see Fig. 1). The reasoning system can infer the proper interpretation of "being grasped" or "being supported" depending on the capabilities of the agent and the properties of the involved objects.

4 Interpretation of Demonstrated Action Sequences

In order for a robot to automatically generate its own control programs from tasks, it must first *interpret* the demonstrated human actions as tasks. Informally, action interpretation aims to uncover the intent behind a demonstration: "What did the human want to demonstrate?" In our experiments, we consider human VR demonstrations represented

as NEEMs. In principle, real-world human demonstrations or even experience data from other robots can also be used as inputs for program synthesis.

The hybrid task representation outlined in Sec. 3.3 permits the automatic interpretation of action sequences to high-level tasks. Because the task definition comprises a set of Prolog predicates stating a task's preconditions, runtime conditions and effects, Prolog's unification algorithm can be used to generate the set of tasks an action can be interpreted to execute, using the following simple inference rule:

Algorithm 2 Semantic interpretation of an `Action`

```
interprets_to(Act, Tsk) :-
  has_task_type(Tsk, TskType),
  satisfies_pre(Act, TskType),
  satisfies_run(Act, TskType),
  satisfies_post(Act, TskType),
  parameterize_task(Act, TskType, Tsk).
```

If the pre-, runtime- and postcondition for a `Task` Tsk of type TskType are satisfied, `parameterize_task` will instantiate a `Task` individual for the given `Action` Act. Note that `interprets_to` can be queried repeatedly for the same Act, and will yield individuals for all possible tasks Act can interpret to. This is to be expected - many actions are inherently ambiguous, and distinguishing between them would require highly specific task types with very narrow task definitions. We leave it to the program generation and execution pipeline to resolve such ambiguities (see 5).

Querying `interprets_to` for all `Actions` in a NEEM yields a *task sequence* (a list of `Task` individuals). Querying `interprets_to` exhaustively will yield all possible task sequences for the NEEM:

Algorithm 3 Semantic interpretation of a NEEM

```
actions_interpret_to(ActionSeq, TaskSeq) :-
  findall(Act,
          (member(Act, ActionSeq),
           interprets_to(Act, Tsk)),
          TaskSeq).
```

From the resulting task sequences, executable robot programs can be automatically generated (see Sec. 5).[3]

5 Program Generation & Execution

Given a `Task` individual or a sequence of `Task` individuals, we propose a program generation and execution pipeline translating underspecified plans to fully specified, executable robot programs. The pipeline combines and extends several elements of the CRAM cognitive architecture, such as the CRAM planning language, the KNOWROB KR&R engine and the ROBOSHERLOCK perception framework.

[3]The source code for the task representation and our reasoner is available at https://github.com/ease-crc/vr-program-synthesis.

5.1 Underspecified Plans

The CRAM planning language (CPL) (Beetz, Mösenlechner, and Tenorth 2010) supports hierarchical task-level programming via the concept of *action designators*. A *designator* is a data structure that represents (designates) an entity and associates symbolic and subsymbolic information with it (McDermott 1991). A sequence of (possibly nested) action designators (e.g. `PickingUp`, `Transfer`, `PuttingDown`) forms an *underspecified plan*, where some information required for execution is unknown and must be inferred by a reasoner from a knowledge base (e.g. in which direction a door opens) or determined by perception (e.g. where a target object is located) at runtime. When a CRAM plan is executed, all designators are resolved in turn, possibly yielding other designators, until arriving at the lowest-level *motion designators*, which are resolved by executing a planned motion on a robot. We refer to (Beetz, Mösenlechner, and Tenorth 2010) for a more detailed overview of CPL.

With the task representation proposed in Sec. 3.3, translating a sequence of `Tasks` into an underspecified plan is straightforward. Because `Tasks` and CPL action designators share the same level of abstraction, converting a task sequence to an underspecified plan reduces to instantiating one action designator per `Task`, as well as object designators or location designators for the paramers of the `Task`, depending on their `Role` in the task. A `PickingUp` task, for example, has two objects associated with it: A `Locatum` (the primary object, here the picked up object) and a `Relatum` (the secondary object, here the object supporting or containing the `Locatum`). It can be directly translated into a `PickingUp` action designator with a "target" property (the `Locatum`) and a "support" property (the `Relatum`) (see Fig. 5). For this work, we extended PyCRAM[4] to support a total of 27 different task types, ranging from prehensile manipulation (`Grasping`, `PickingUp`, `Placing`) to force-controlled insertion and retraction (`HoleOnPeg`, `Sliding`, `Retracting`).

5.2 Generation of Executable Robot Control Programs

The generated underspecified plan contains action, object and location designators, which must be grounded (*resolved*) in order for a real robot to perform actions in the real world.

Object and location designator grounding A `HoleOnPeg` action designator, for example, encapsulates the series of actions required to thread an object with a hole onto a peg (e.g. hanging a product with a perforated tab onto a hanger in a supermarket). `HoleOnPeg` is parameterized with two object designators, *hole* (the object to be hung) and *peg* (the hanger). The location of the *hole* is generally not known when the plan is generated, and must be detected at runtime. Moreover, the precise type or instance of the *hole* object may also not be known; if the plan was generated from a VR demonstration, the object

[4]The Python implementation of CRAM (https://github.com/cram2/pycram).

Figure 5: The proposed program generation pipeline. A sequence of `Tasks` is translated into an underspecified plan (here, the real-world poses of the healing salve and shopping basket are not known a priori). Via reasoning, perception and collision-free motion planning, the the plan is ultimately resolved to native robot code.

instance will certainly be unknown (the virtual object will not be present in the real world) and the object type in the demonstration will likely be different from the objects present in the supermarket. The missing properties (pose, type, individual) must be resolved using contextual knowledge and sensory information from the actual environment at the time of execution. We connected PyCRAM to the KNOWROB KR&R engine to automatically infer object types, the relative poses of object features such as holes or perforated tabs, as well as affordance-related concepts such as supporting surfaces from an ontological representation of the environment. We also implemented a resolution mechanism for object poses based on ROBOSHERLOCK, which uses neural object detection and pose estimation models to determine the current poses of objects in the environment.

Object pose estimation with RoboSherlock RoboSherlock[5] is a taskable knowledge-driven perception system based on the UIMA (Unstructured Information Management Architecture) (Ferrucci et al. 2009). RoboSherlock provides a query-answering mechanism, which acts as an interface between the perception system and the robot control program and allows the control program not only to access the world in a selective manner (i.e., faster) but also to provide prior information to RoboSherlock (e.g., the pose of a specific object given that the object is blue, cubic, ...). Given this specific query with provided prior information, RoboSherlock will plan a perception pipeline based on the capabilities and requirements of each perception expert in the knowledge base to answer the query. For instance, a 6D-pose estimation expert can provide the 3D-position and 3D-orientation of an object in the world, but requires the semantic segmentation and recognition of that object.

For this work, a core perception expert was RobotVQA (Robot Visual Question Answering) (Kenghagho Kenfack et al. 2020), a deep learning model which is autonomously trained from physico-realistic virtual worlds (for embodied and situated training data) to infer semantic graphs of cluttered and occluded scenes. RobotVQA can be constrained to focus on a specific set of objects described in terms of their attributes (e.g., color, shape, material, category). For experiment 6.3, it has been constrained to detect and estimate the pose of the healing salve product depicted in Fig. 1. In case of multiple similar objects in the robot's viewport, an ID resolution expert is tasked to identify and track object instances (see Fig. 6). In (Kazhoyan et al. 2020), the approach has been validated in a very challenging home setting.

Figure 6: Detection and pose estimation of objects from supermarket shelf hangers. In the three scenarios, color image illustrates the detection while point cloud image illustrates pose estimation.

Given the challenges posed by object 6D pose estimation for deep learning models (e.g., symmetry, occlusions, lack of ground truth, accuracy), RobotVQA uses a knowledge-augmented approach to overcome the problem. When RobotVQA estimates attributes of objects (e.g., shape, color, category), the corresponding 3D models of these objects are retrieved from the knowledge base. Secondly, the point cloud of each object is extracted from the depth image based on the segmentation mask returned by RobotVQA. Given that this segmentation is not perfectly accurate, often the extracted point cloud of the object contains outliers, which are then removed through distance-based hierarchical clustering. Then, the ICP (Iterative Closest Point) algorithm is applied given the object's 3D model and its point cloud. Given that ICP is extremely sensitive to occlusion and initial poses, we first estimate the three main axes of the object with PCA and then launch 12 instances of ICP in parallel, each with a different annotation of the main axes (i.e., XYZ, XZY, ...). Finally, the pose returned by the ICP instance with the highest score is considered. Given that some annotations are

[5]https://robosherlock.org/publications.html

not physically plausible in some scenarios, such as for the experiments for this work where the object on the hanger can only have two possible orientations (i.e., front-facing or back-facing), a more accurate decision can be taken.

Action designator grounding Once all objects and locations for a given action designator have been resolved, the action designator itself can be resolved to a sequence of motion designators, atomic (possibly force-controlled) motions to be executed by the robot. `HoleOnPeg` resolves to a collision-free approach motion, a linear contact motion until contact with the peg is detected, a spiral search motion to thread the grasped object onto the peg, and a force-controlled motion to push the object onto the peg. For this work, a resolution mechanism for motion designators was implemented, which combines KNOWROB with an industrial robot motion planner, compiler and execution environment.[6] Motion designators are resolved in three steps:

1. *Parameterization of motion planner and force controller:* Using the reasoning capabilities of KNOWROB, parameters for motion planners such as target poses, maximal velocities or force controller parameters are inferred based on knowledge of the manipulated objects and the environment.
2. *Offline motion planning:* The current belief state of KNOWROB pertaining to the state of the environment (most notably the current poses of objects and their 3D meshes, if available) are loaded into a motion planner and a collision-free motion is planned, subject to the inferred parameters and constraints.
3. *Robot code generation:* Executable, manufacturer-specific native robot code is generated for the target robot platform, including any peripheral devices (grippers, force-torque sensors).

After resolution of all motion designators, the generated code can be executed directly on the robot.

6 Experiments

To assess the validity of the proposed approach for real-world robotic manipulation tasks, two application scenarios are considered. In a first scenario, the robot is tasked with taking a product down from a hanger and placing it into a shopping basket. In a second scenario, the inverse problem is considered, where the robot has to grasp a product and thread it onto a hanger. Both applications require a high degree of dexterity involving force-controlled insertion or extraction as well as sophisticated collision-free motion and grasp planning in response to changing environments.

6.1 VR NEEM Collection

VR demonstrations are collected using Unreal Engine 4 (Epic Games 2019) and the RobCoG[7] framework (Haidu et al. 2018), utilizing photorealistic rendering and accurate physics simulation. To evaluate the robustness of our approach with respect to variations in the demonstrations, 30 human demonstrations were recorded for each of the two application scenarios. Three box-shaped target objects, differing in width, height and depth, were used in the demonstrations to evaluate the generalization capacity of our approach across different target objects. The recorded demonstrations were stored into the KNOWROB KR&R engine as NEEMs (see Sec. 3.2 and Fig. 4), using the automatic semantic annotation pipeline presented in (Haidu and Beetz 2021).

6.2 Quantitative Evaluation

In a first set of experiments, program synthesis is performed for all 60 demonstrations, the resulting program candidates are evaluated in a simulated environment and then executed on real hardware under controlled conditions. For each human demonstration, the demonstrated action sequence (NEEM) is interpreted to a set of task seq uence candidates. To that end, the Prolog reasoning engine in KNOWROB generates all possible sequences of `Tasks` consistent with the NEEM and background knowledge about `Action` and `Task` semantics (see Sec. 4). This knowledge stems from TBox axioms in the SOMA ontology (Beßler et al. 2020) and an additional Prolog rulebase containing definitions of 14 different general-purpose `Tasks`, according to our proposed task semantics (see Sec. 3.3). Due to possible ambiguities of the human demonstrations, the interpretation of an action sequence typically yields more than one candidate task sequence. The experiments chiefly test the capability of the proposed system to robustly infer executable robot programs for a wide array of human demonstrations.

Because the VR environment differs from the real-world environment, the resulting task sequence candidates are underspecified. In this experiment, object and location designators such as the manipulated object and its location or the pose of the shopping basket are resolved by querying the knowledge base; experiment 6.3 uses a perception system for designator resolution. For execution, a Universal Robots UR5 robot arm was used, equipped with an ATI Axia80 force-torque sensor and a Robotiq 2F-85 robotic gripper. For this set of experiments, one supermarket hanger was mounted in a fixed position (see Fig. 7 (bottom)). The real-world target object is a tube of healing salve in a rectangular carton box (see in Fig. 1), which has been reinforced with 3D-printed plastic to withstand repeated grasping.

Scenario 1: Fetching an Object From a Hanger An extract of the human demonstration and the execution of the corresponding program candidate is shown in Fig. 7. Quantitative results are provided in Fig. 8. For 29 out of 30 demonstrations, action interpretation produced at least one candidate task sequence, with the majority of demonstrations resulting in 2 or more candidates. In the one remaining demonstration, a glitch in the VR engine caused the object to briefly disappear from the human avatar's hand, spawn on the ground and then re-appear in the avatar's hand, resulting in wrong semantic annotations (sudden contact with the ground plane). This prevented the reasoning engine from finding a viable task sequence for the demonstration. For two of the three objects, simulation of 7-13% of the generated robot programs failed, generally due to unreachable ap-

[6] ArtiMinds Robot Programming Suite, https://artiminds.com
[7] Robot Commonsense Games, https://robcog.org

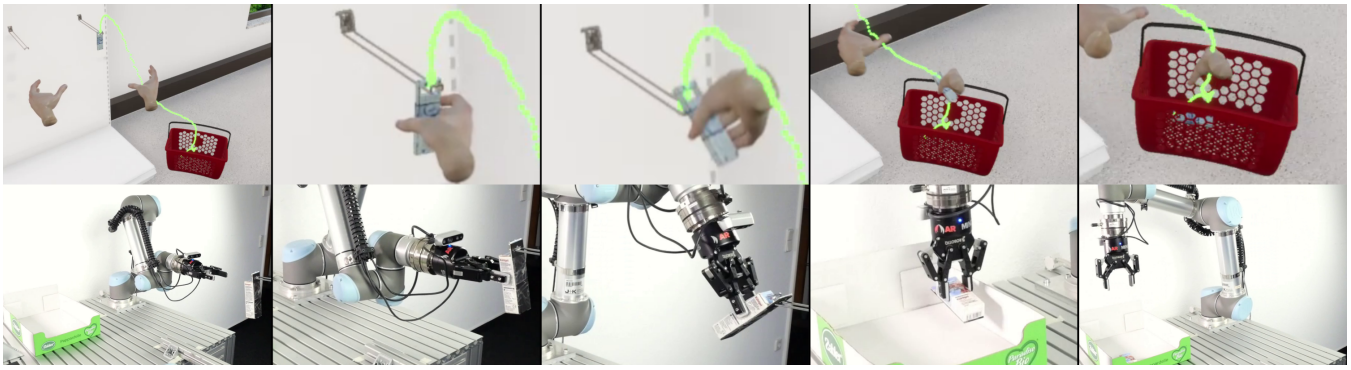

Figure 7: VR human demonstration (top) and execution of the synthesized program (bottom) for force-sensitive fetch-and-place.

	Demonstr.	Interp.	Simulation		Exec.
	# Demos	# Cands.	Plan succ.	Task succ.	Succ.
O1	10	29	27 (93%)	27 (93%)	26 (90%)
O2	10	60	52 (87%)	44 (73%)	35 (58%)
O3	9	18	18 (100%)	18 (100%)	18 (100%)

Figure 8: Quantitative evaluation results for force-sensitive fetch-and-place. For three different objects (O1, O2, O3), the majority of synthesized programs achieve the task in simulation and real-world execution.

	Demonstr.	Interp.	Simulation		Exec.
	# Demos	# Cands.	Plan succ.	Task succ.	Succ.
O1	10	22	20 (91%)	18 (82%)	18 (82%)
O2	10	20	20 (100%)	20 (100%)	15 (75%)
O3	10	25	25 (100%)	25 (100%)	25 (100%)

Figure 9: Quantitative evaluation results for inverse peg-in-hole. For three different objects (O1, O2, O3), the majority of synthesized programs achieve the task in simulation and real-world execution.

proach, grasp or depart poses. Depending on the object, 75-100% of the generated robot programs successfully placed the object into the basket in the simulation. For each of the 29 valid demonstrations, at least one synthesized program could be successfully executed, extracting the object from the hanger and placing it into a basket while avoiding collisions. The generated programs successful in the simulation sometimes failed in real-world execution due to exceeded force limits during extraction, or by colliding with the hanger or basket due to slight differences in the 3D models (used for collision-free planning) and the real-world objects. This can be solved by more robust parameterization of the force controller and the use of perception (e.g. 3D cameras) to ensure the planning scene matches the real-world more precisely.

Scenario 2: Threading an Object Onto a Hanger A visualization of the simulation and real-world execution for scenario 2 is shown in Fig. 10. Quantitative results are provided in Fig. 9. For all human demonstrations, action interpretation produced at least one candidate task sequence, with the majority of demonstrations resulting in 2-4 candidates. In this scenario, all but two candidates could be simulated successfully, and all but four candidates successfully placed the object onto the hanger in the simulation. Execution was successful for 75-100% of generated programs. The main challenge was robustly finding the tip of the hanger using force-controlled search. Integration of visual perception to precisely detect the pose of the hanger can further improve results. As for the first application scenario, at least one successfully executable robot program could be generated for each human demonstration.

6.3 Real-World Validation

A second set of experiments aims at validating the proposed approach in a realistic supermarket environment. To that end, we realized both application scenarios in a laboratory for retail robotics (Costanzo et al. 2020), which accurately models the furniture, products and lighting conditions in a real supermarket. For execution, Donbot[8] is used, which features a mobile platform and a Universal Robots UR5 collaborative robot arm. For this experiment, an ATI Axia80 force-torque sensor, a Robotiq 2F-85 robotic gripper and a flange-mounted Intel RealSense camera have been added. Both scenarios were solved successfully in this more realistic setting (see Fig. 10 for a visualization of the experiments for Scenario 2). RoboSherlock (see Sec. 5.2) was used to ensure that changes in the pose of the manipulated objects were reflected in the knowledge base. Our framework generated successful robot control programs for each task given a single human demonstration, even when the pose of the manipulated object was changed.

7 Related Work

Most recent work in program synthesis is dedicated to one of two central challenges: *Action (or task) recognition*, the process of identifying the intention behind e.g. a human demonstration or a natural-language instruction, and *task execution*, the process of generating robust low-level robot

[8] https://ai.uni-bremen.de/research/robots/donbot

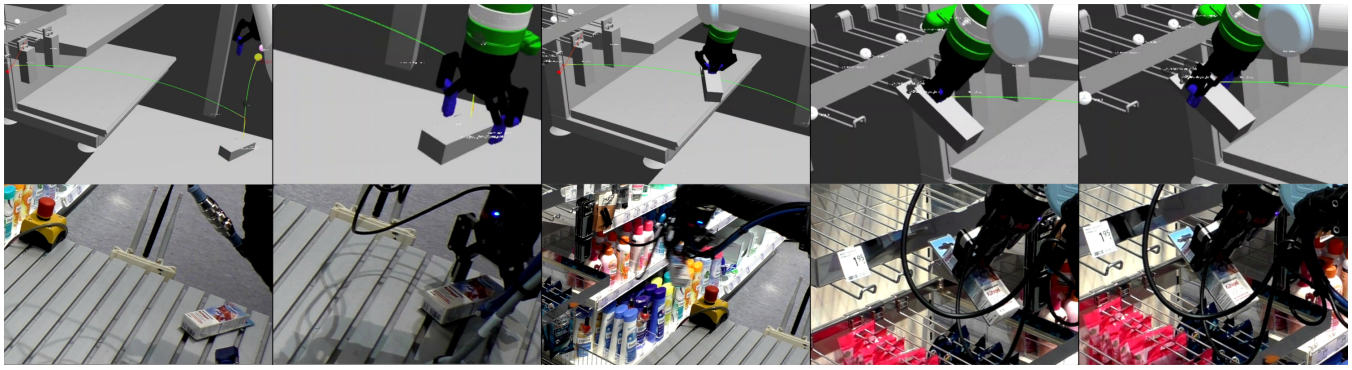

Figure 10: Simulation (top) and execution of the synthesized program (bottom) for inverse peg-in-hole in a realistic supermarket environment.

control commands to solve high-level tasks in real environments. Several action recognition frameworks based on deep neural networks have been proposed (Sun et al. 2022; Kong and Fu 2022). However, for the purpose of program synthesis, it is highly advantageous to detect not only the human activity itself, but also to extract rich semantic information about the agent, objects and execution context. To that end, knowledge-based approaches to human activity recognition have been proposed (Ramirez-Amaro, Beetz, and Cheng 2017; Haidu and Beetz 2016; Bates et al. 2017). NEEMs as a universal exchange format for semantically rich, annotated human or robot experience data have been proposed in (Beetz et al. 2018). In (Haidu and Beetz 2021), Haidu et al. propose a mechanism to automatically parse human demonstrations to NEEMs. The present work is the first to leverage NEEMs as an input modality for program synthesis.

Task execution has been traditionally addressed by mapping tasks to robot skills in a skill library (Ramirez-Amaro, Beetz, and Cheng 2017; Ramírez-Amaro, Beetz, and Cheng 2015). Alternatively, Task and Motion Planning (TAMP) approaches have been proposed, which derive a set of constraints from a task description and solve a multi-horizon planning problem (Kaelbling and Lozano-Perez 2011; Schmitt et al. 2019; Diehl, Paxton, and Ramirez-Amaro 2021). Underspecified plans (Beetz, Mösenlechner, and Tenorth 2010) and plan specialization (Koralewski, Kazhoyan, and Beetz 2019) suggest an alternative approach, whereby high-level plans are incrementally specialized to executable programs by way of reasoning and perception. In prior work (Kazhoyan, Niedzwiecki, and Beetz 2020; Kazhoyan et al. 2020), underspecified plans were written manually upfront. This work automatically generates underspecified plans directly from human demonstrations.

Recent work has increasingly leveraged deep learning for program synthesis in the domain of robotics (Xu et al. 2018; Sun et al. 2018). Large Language Models (LLMs) have been used to this end with particular success (Liang et al. 2023; Singh et al. 2022). However, in complex long-horizon force-controlled manipulation scenarios, purely neural program synthesis approaches struggle to address the dual requirement of generating policies which generalize well, but at the same time solve a given task with high precision. The knowledge-based action interpretation and task execution mechanisms proposed in this work provide cognitive mechanisms which address these challenges.

8 Conclusion

This work proposes and describes a knowledge-driven approach to robot program synthesis for real-world open-ended manipulation tasks. Embedded into a state-of-the-art cognitive architecture, it leverages sophisticated knowledge representations and reasoning algorithms to interpret the task-level intentions of human demonstrations in VR, generate a generalized motion plan and transform it into executable robot code via reasoning, path planning and knowledge-enabled perception.

8.1 Discussion and Future Work

Designing a program synthesis system around a common knowledge representation and using a shared reasoning engine permits a high degree of generalization and integration. The SOMA foundational ontology, NEEMs as an exchange format for experience data and integration in the CRAM cognitive architecture allow for sharing knowledge and reasoners between components. Moreover, ontology-based knowledge representation allows our system to be highly generalizable, as task semantics can be specified at a very general level (see Sec. 3.3): In our experiment, the same task knowledge could be used to infer robot control programs for both fetch-and-place and peg-in-hole tasks. The same mechanisms permit generalization to domains beyond supermarkets, such as industrial or household settings.

One limitation of the presented approach is that the range of tasks that can be solved is limited to the contents of the knowledge base, and that extending the knowledge base requires manual adding of task knowledge (e.g. pre-, runtime- and postconditions). Moreover, the approach implies a trade-off between filtering feasible candidate programs in simulation (which risks rejecting good candidates if the simulation differs from reality), and trying out the remaining candidates in the real world (which requires time and resources). These shortcomings can be addressed by integrating learning approaches such as interactive task learning (Gluck, Laird, and Lupp 2019), which will be the subject of further investigation.

Acknowledgments

This work has been partly funded by the German ministry of education and research (BMBF) as part of the ILIAS project (reference no. 01DR19001B).

References

Bates, T.; Ramirez-Amaro, K.; Inamura, T.; and Cheng, G. 2017. On-line simultaneous learning and recognition of everyday activities from virtual reality performances. In *2017 IEEERSJ Int. Conf. Intell. Robots Syst. IROS*, 3510–3515.

Beetz, M.; Bessler, D.; Haidu, A.; Pomarlan, M.; Bozcuoglu, A. K.; and Bartels, G. 2018. Know Rob 2.0 — A 2nd Generation Knowledge Processing Framework for Cognition-Enabled Robotic Agents. In *ICRA*, 512–519.

Beetz, M.; Mösenlechner, L.; and Tenorth, M. 2010. CRAM — A Cognitive Robot Abstract Machine for everyday manipulation in human environments. In *2010 IEEERSJ Int. Conf. Intell. Robots Syst.*, 1012–1017.

Beßler, D.; Porzel, R.; Pomarlan, M.; Vyas, A.; Höffner, S.; Beetz, M.; Malaka, R.; and Bateman, J. 2020. Foundations of the Socio-physical Model of Activities (SOMA) for Autonomous Robotic Agents. *ArXiv201111972 Cs*.

Costanzo, M.; Stelter, S.; Natale, C.; Pirozzi, S.; Bartels, G.; Maldonado, A.; and Beetz, M. 2020. Manipulation Planning and Control for Shelf Replenishment. *IEEE Robot. Autom. Lett.* 5(2):1595–1601.

Diehl, M.; Paxton, C.; and Ramirez-Amaro, K. 2021. Optimizing robot planning domains to reduce search time for long-horizon planning. *ArXiv211105397 Cs*.

Epic Games. 2019. Unreal engine.

Ferrucci, D.; Lally, A.; Verspoor, K.; and Nyberg, E. 2009. Unstructured information management architecture (UIMA) version 1.0. OASIS Standard.

Gluck, K. A.; Laird, J. E.; and Lupp, J. 2019. *Interactive Task Learning: Humans, Robots, and Agents Acquiring New Tasks through Natural Interactions*. Cambridge, MA: MIT Press.

Haidu, A., and Beetz, M. 2016. Action recognition and interpretation from virtual demonstrations. In *IROS*, 2833–2838.

Haidu, A., and Beetz, M. 2021. Automated acquisition of structured, semantic models of manipulation activities from human VR demonstration. In *2021 IEEE Int. Conf. Robot. Autom. ICRA*, 9460–9466.

Haidu, A.; Beßler, D.; Bozcuoğlu, A. K.; and Beetz, M. 2018. KnowRobSIM — Game Engine-Enabled Knowledge Processing Towards Cognition-Enabled Robot Control. In *IROS*, 4491–4498.

Kaelbling, L. P., and Lozano-Perez, T. 2011. Hierarchical task and motion planning in the now. In *ICRA*, 1470–1477.

Kazhoyan, G.; Stelter, S.; Kenfack, F. K.; Koralewski, S.; and Beetz, M. 2020. The Robot Household Marathon Experiment. *ArXiv201109792 Cs*.

Kazhoyan, G.; Niedzwiecki, A.; and Beetz, M. 2020. Towards Plan Transformations for Real-World Mobile Fetch and Place. In *IEEE Int. Conf. Robot. Autom. ICRA*.

Kenghagho Kenfack, F.; Ahmed Siddiky, F.; Balint-Benczedi, F.; and Beetz, M. 2020. Robotvqa — a scene-graph- and deep-learning-based visual question answering system for robot manipulation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 9667–9674.

Kong, Y., and Fu, Y. 2022. Human Action Recognition and Prediction: A Survey. *Int J Comput Vis* 130(5):1366–1401.

Koralewski, S.; Kazhoyan, G.; and Beetz, M. 2019. Self-Specialization of General Robot Plans Based on Experience. *IEEE Robot. Autom. Lett.*

Liang, J.; Huang, W.; Xia, F.; Xu, P.; Hausman, K.; Ichter, B.; Florence, P.; and Zeng, A. 2023. Code as Policies: Language Model Programs for Embodied Control.

McDermott, D. 1991. A reactive plan language. Research Report YALEU/DCS/RR-864, Yale University, Yale University.

Ramírez-Amaro, K.; Beetz, M.; and Cheng, G. 2015. Understanding the intention of human activities through semantic perception: Observation, understanding and execution on a humanoid robot. *Advanced Robotics* 29:345–362.

Ramirez-Amaro, K.; Beetz, M.; and Cheng, G. 2017. Transferring skills to humanoid robots by extracting semantic representations from observations of human activities. *Artificial Intelligence* 247:95–118.

Schmitt, P. S.; Wirnshofer, F.; Wurm, K. M.; v. Wichert, G.; and Burgard, W. 2019. Planning Reactive Manipulation in Dynamic Environments. In *IROS*, 136–143.

Singh, I.; Blukis, V.; Mousavian, A.; Goyal, A.; Xu, D.; Tremblay, J.; Fox, D.; Thomason, J.; and Garg, A. 2022. ProgPrompt: Generating Situated Robot Task Plans using Large Language Models.

Sun, S.-H.; Noh, H.; Somasundaram, S.; and Lim, J. 2018. Neural Program Synthesis from Diverse Demonstration Videos. In *ICML*, 4790–4799.

Sun, Z.; Ke, Q.; Rahmani, H.; Bennamoun, M.; Wang, G.; and Liu, J. 2022. Human Action Recognition from Various Data Modalities: A Review. *IEEE Trans. Pattern Anal. Mach. Intell.* 1–20.

Xu, D.; Nair, S.; Zhu, Y.; Gao, J.; Garg, A.; Fei-Fei, L.; and Savarese, S. 2018. Neural Task Programming: Learning to Generalize Across Hierarchical Tasks. In *ICRA*, 1–8.