Task and Motion Scheduling for Dual-Arm Robots from Natural Language and Demonstration

Jan Kristof Behrens[1] Karla Štěpánová[2] Ralph Lange[1] Radoslav Škoviera[2]

Abstract— **Multi-modal robot programming with natural language and demonstration is a promising technique for efficient teaching of manipulation tasks in industrial environments. In particular with modern dual-arm robots, which are designed to quickly take over tasks at typical industrial workplaces, the direct teaching of task sequences hardly utilizes the robots' capabilities. We therefore propose a two-staged approach that combines linguistic instructions and demonstration with simultaneous task allocation and motion scheduling. Instead of providing a task description and demonstration that is replayed to a large extent, the user describes tasks to be scheduled with all relevant constraints and demonstrates relevant locations and storages relative to workpieces and other objects. Constraint optimization is used to schedule task and motion sequences to minimize the makespan. Naming and grouping enables systematic reuse of sub-tasks ensembles and referencing of relevant locations. The proposed approach can generalize between different workspaces and is evaluated with gluing showcases from furniture assembly.**

Index Terms— **learning from demonstration, dual arm manipulation, multi-modal robot programming, task scheduling**

I. INTRODUCTION

Although today's production plants and shop floors are unimaginable without robots, programming them is still a complex, time-consuming task. We believe, this complexity stems from the large variety of different domains a robotic program must govern. Deliberation, kinematics, collision-avoidance and coordination, geometry, and perception are very different in their nature and are per se challenging to formulate for humans, because we handle many of these topics subconsciously. One approach to take the mental burden from the instructor, is to tap into that resource by employing multi-modal specification methods. Sequencing and naming of subtasks can be easily expressed in natural language, geometric concepts like poses and paths can be demonstrated using pointing-devices or gestures, whereas force-torque controllers can be best trained by physical demonstrations with suitable devices [1].

As example showcases, we use tasks inspired by furniture assembly. The showcases consist of actions like applying glue or picking bolts and inserting them in holes in a board (see Fig. 1, left), which are preparation steps for the final assembly. As in typical instruction booklets, different constraints on the action execution are given: The step-by-step instruction induces a partial ordering on the execution, e.g. glue has to be applied before a bolt is placed, but leaves open in which order unrelated actions have to happen, e.g. the order of bolts to insert, and even which bolt is inserted into which hole. The optimal robot program for these tasks is highly dependent on the actual setup of the robot workspace, which can be subject to regular change, the robot's kinematics, and the evolving task definition. Using planning and scheduling techniques has the advantage of yielding high-quality solutions which can easily adapt to a variance in the mentioned parameters.

Especially in the case of dual-arm robots, the user is often not able to specify the optimal task execution. Next to the works about manipulation planning (e.g., [2]), also dual-arm motion scheduling is an active field of research [3], [4].

We designed a system that allows the user to descriptively define task scheduling problems by demonstration and linguistic input. Natural language is used to define and add sub-tasks, locations of these sub-tasks as well as to add constraints on them. We employ a custom context-free grammar which enables the definition of new actions and hierarchical action compositions. The demonstration of the action in parallel with natural language instructions enables the parametrization of sub-tasks by key words, e.g. the word *here* references a demonstrated location relative to the workpiece. Linguistic constructs like *first-then* are used to constrain on the order of the subtasks.

Then, we employ our simultaneous task allocation and motion scheduling (STAAMS) solver [5][1] to deploy the task to the robot's resources and optimize the execution for efficiency. This solver's task modeling abstraction *Ordered Visiting Constraints* (OVCs) can be used to represent a wide range of industrial task and motion scheduling problems in an effective and robot-agnostic way. An OVC models a sequence of actions to be executed at different – possibly variable – locations by a manipulator. Ordering and temporal constraints within and between OVCs allow to take dependencies between the different actions into account as well as to synchronize them. By utilizing the remaining degrees of freedom in the task execution, idle times of the manipulators can be avoided and thus the makespan, i.e. the execution time, often can be significantly reduced.

The remainder of this paper is organized as follows: We discuss related work in Section II, before we briefly explain the OVC formalism and the corresponding planner in Section III. In Section IV, we describe an OVC-based

[1]Robert Bosch GmbH, Corporate Sector Research and Advance Engineering, Renningen, Germany, jan.behrens@de.bosch.com, ralph.lange@de.bosch.com

[2]Czech Technical University in Prague, Czech Institute of Informatics, Robotics, and Cybernetics

[1]manuscript available on the web page associated with this paper: http://imitrob.ciirc.cvut.cz/planning.html

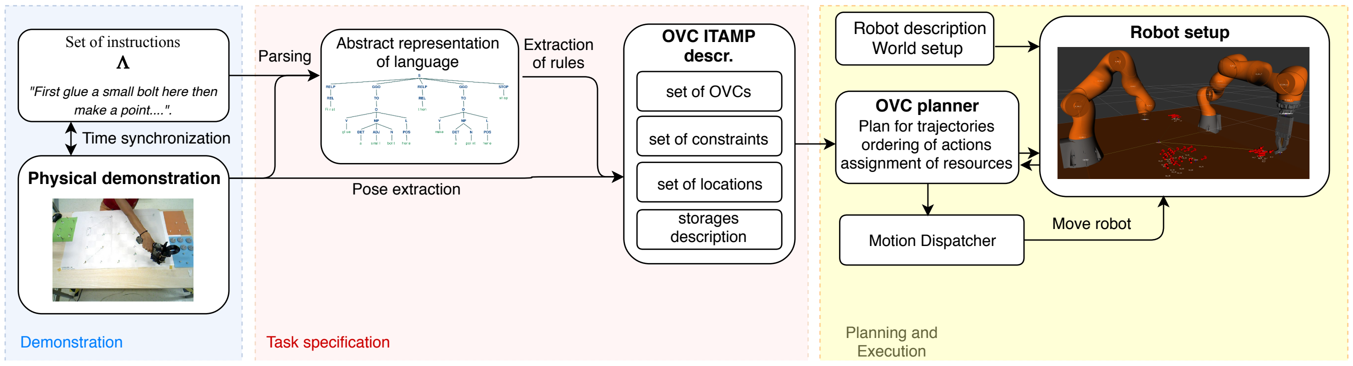

Fig. 1. Overview of the whole system.

task scheduling problem specification which is obtained from natural language and demonstrations, with all relevant details on language processing, simultaneous identification of locations, teach-in of individual tasks and task templates and linguistic specification of constraints. The implementation and setup of our system is described in Section V, followed by experimental results in Section VI. Finally, the paper is concluded in Section VII with a summary and outlook.

II. Related Work

This work is related to two (heavily overlapping) fields: robot learning from demonstration and multi-modal robot programming.

Robot learning from demonstration can roughly be divided into learning of individual motions and skills versus learning of complex tasks. Important works in the field of motion and skill learning are [6], [7], [8]. Works on learning of complex tasks focus on automated segmentation (e.g., [9], [10], [11]). For example, Jenkins and Mataric [12] focused on finding repetitive patterns in the data (motion primitives) using a dimensionality reduction via a spatiotemporal isomap.

In the approach proposed in this paper, we assume a predefined, extendible set of primitive actions. Motion and skill learning is an effective technique for teaching new actions and therefore considered as an important foundation. However, regarding the high-level tasks we aim at explicit task and motion planning to use the full capacity of dual-arm robots, as argued in the previous section. Therefore, learning-based methods that generalize from demonstrations are not appropriate.

In the field of multi-modal robot programming for manipulation, most works aim at obtaining task descriptions for natural language combined with pointing or teach-in techniques. For example, in an early work in 1996, Hwang et al. [13] proposed a comprehensive system for specifying a hierarchical task decomposition by natural language using a fixed grammar. At the same time, the system allows to teach primitive actions and the corresponding poses [14]. On contrary, our custom grammar allows a huge variability in linguistic instructions. Furthermore, we allow interleaved learning of templated actions, which can be directly reused after specification. In the PRACE project, an extension to ABB Robot Studio has been developed that allows to specify a task skeleton – an assembly graph – in natural language, which can then be refined using the graphical programming interface [15]. In [16], Mohseni-Kabir et al. presented an interactive system for specifying hierarchical task networks (HTNs) for manipulation tasks. The user can explain the tasks in a top-down manner and the system asks him/her for task names that are unknown to it as well whether it should generalize tasks to other objects of the same type in the scene. A similar approach based on behavior networks instead of HTNs is presented by Rybski et al. in [17]. In contrast to our approach, the demonstrated order of subtasks is strictly replicated, which neglects the chance to optimize the order of the subtasks. Also they do not show, how ordering constraints on individual subtasks could be specified via spoken language.

Only a small body of works considers the multi-modal input of task scheduling problems. Kirk et.al [18] use natural language to define goals of diverse tasks and explore teaching goals and how agents can afterwards utilize these as strategies. The system was tested on simple riddles and games. No motion planning on top of the provided constraints was performed. In [19], Ekvall and Kragic proposed a robot learning system that uses a STRIPS-like planner whose input is obtained from imitation learning and a dialogue-based approach that allows the teacher to add constraints while demonstrating the task. In the implementation, the constraints were hard-coded in the planner. Also, path planning was not integrated with task planning as in our approach. Suddrey et al. propose a similar approach based on the HTN planning in [20]. Their system uses the OpenCCG parser for natural language processing of the user's explanation of the task decomposition. The system asks for the specification of unknown subtasks in a dialogue-based manner. Grounding of arguments is performed by transforming each argument into a first-order logic query and matching it against the perceived world model. Preconditions from the primitive tasks are propagated along the task hierarchy, but there is no support for natural-language-based input of further constraints. Again, motion planning is not integrated with task planning.

Unlike the last mentioned works, the system presented in this paper allows to easily specify advanced task scheduling problems for dual-arm manipulation tasks, including temporal and causal constraints amongst others and directly transfer them to the robotic environment. The proposed planning approach based on constraint optimization, deeply integrates task allocation and motion scheduling to solve the problems efficiently.

III. Ordered Visiting Constraints for Simultaneous Task Allocation and Motion Scheduling

In this section, we first introduce the OVC model and solver as a tool for the simultaneous task allocation and motion scheduling (STAAMS) problems in dual-arm manipulation settings. In the second part of this section, we describe the modeling with OVCs in further detail and explain selected constraint types for advanced manipulation planning problems.

A. OVC Model and Solver

Simultaneous task and motion scheduling is concerned with scheduling and allocating high-level actions, while taking constraints on the motion level like collisions, robot kinematics, and joint limits into account simultaneously. As a result stands a time-scaled trajectory for every robot arm that does not violate any constraints and reaches the set task goal. Our solver (cf. Fig. 2) is based on constraint programming (programmatically utilizing Constraint Satisfaction Problems). A Constraint Satisfaction Problem (CSP) is generally specified by a triple (X, D, C), where X is a set of variables, D a set of domains, and C a set of constraints. The solution of a CSP is a complete assignment of values to variables that satisfies all constraints C. To find such a solution, the underlying solver performs a backtracking search over the variables with suitable value selection heuristics. For the subsequent optimization of the makespan, a series of CSPs with additional constraints $t_{\text{end}} \leq c_i$, where $c_{i+1} < c_i$, is solved. Our solver supports the definition of tasks by an abstraction called *Ordered Visiting Constraints* (OVC), which are defined by tuples

$$\omega = (A, [P_1, ..., P_l], [L_1, ..., L_l], [I_1, ..., I_l], C_{\text{intra}}). \quad (1)$$

An OVC ω models a sequence of actions to be executed at different locations by a manipulator. A is a variable representing the *active component*, i.e. a manipulator. The P_i define the action type, e.g. applying glue or picking up an object, to execute at the *end-effector locations*, i.e. 6-DoF poses, denominated by the variables L_i. The I_i are *time interval* variables (with variables for start time, end time and duration) modeling the time windows for the action execution. C_{intra} is a set of constraints to model arbitrary relations between OVC interval variables.

The robot's motions are represented as a series of n joint configurations, n intervals modeling the time spent in these configurations, and $n-1$ intervals modeling the traveling time between the configurations. Precomputed roadmaps [21] are used to discretize the configuration space per arm (see Fig. 2 right). Therefore, the domain of the configuration variables is the set of all roadmap nodes of the corresponding arm. Path planning is performed by graph search on these roadmaps. A *collision table* lists all pairs of roadmap nodes of the two arms that preclude each other, which is used to cast disjunctive constraints on conflicting time intervals.

B. Modeling with OVCs

OVCs provide a lot of modeling flexibility as the robot arms, the locations to visit and their order are modeled by CSP variables and very generic constraints among them. Similarly, constraints between OVCs may be defined using propositional logic, Allen's interval algebra, and set-theoretic expressions.

Programming a task with OVCs means to create for each segmentable subtask or series of subtasks an OVC and constrain its variables. To create an OVC, we need the set of arms that should be considered to execute the subtasks (e.g., because they have the required tool for the task mounted) and which actions should be executed. Then, a set of locations per L_i is used to constrain location variables. Lastly, we have lower and upper bounds on the action durations and the action type for every action. In Eq. 2, an exemplary function call to create an OVC for a pick-and-place task is presented.

$$\begin{aligned}\text{addOVC}((r_1, r_2), (\text{pick}, \text{place}), ((\text{loc}_{23}, \text{loc}_{17}), (\text{loc}_2)),\\ ((2.0, 7.0), (1.5, 10.0)))\end{aligned} \quad (2)$$

Here, a pick action shall be performed either at loc_{23} or at loc_{17} and a place action at location loc_2. As OVCs provide the flexibility to specify a set of possible locations for a location variable, we can – in the case that multiple equivalent pieces to be picked are available – leave the final assignment to the solver. To relate multiple OVCs, various constraints can be added. For example, the interval relation

$$\text{addOvcCt}(\text{StartsAfterEnd}, (\text{OVC}_i, \text{OVC}_j)) \quad (3)$$

can be used to prescribe an order among two OVCs. Yet, it is also possible to synchronize two OVCs for different arms to perform a joint action with both arms. In this manner, the task model (see upper half in Fig. 2, left) is declared.

IV. Specifying OVC Scheduling Problems by Linguistic Input and Demonstration

In our system, the set of OVCs for a specific use-case – and thus scheduling problem – is specified using natural language and simultaneous demonstration. From the natural language input, we extract action types and constraints to subsequently transform them to an OVC task definition. To process linguistic input we make use of a custom context-free grammar. The grammar contains production rules (see Fig. 3) for multiple syntactic categories (e.g., noun phrases, verb phrases, prepositions, constraint relations, etc.). We first parse each sentence using a recursive descent parser to retain the tree structure and abstract meaning of the sentence according to the production rules of the grammar. By analysis on this tree, we extract information about tasks and constraints (e.g., temporal constraints, temporal intervals,

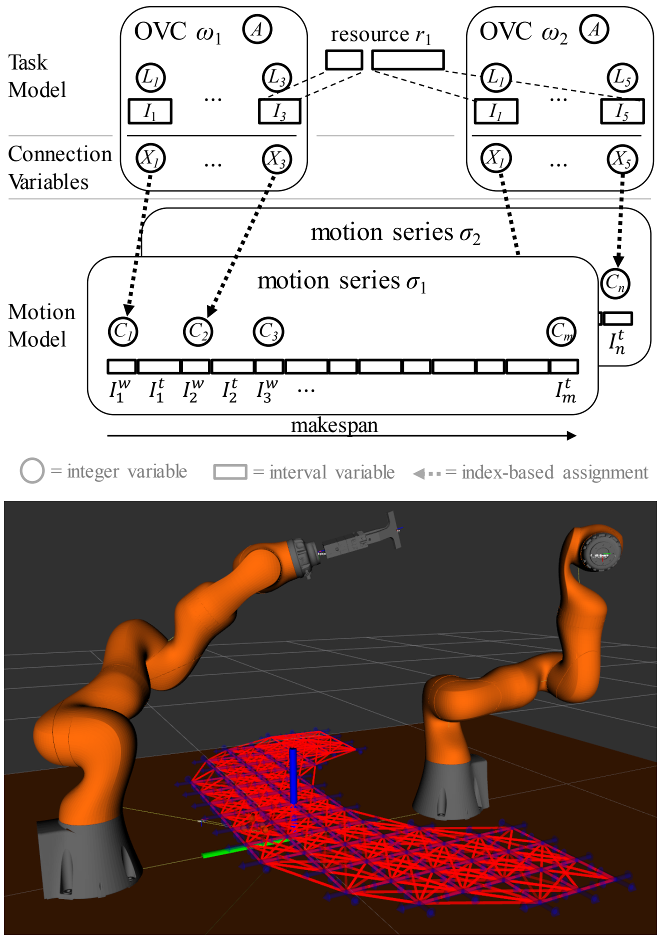

Fig. 2. CP model for task and motion scheduling (top) and simulated robot setup of two kuka iiwa arms. A roadmap of one robot is displayed (bottom).

types of actions, etc.). Each sentence can be: (1.) a definition of new task template (AG), (2.) a definition of locations (a *storage* (HS)), (3.) a grouped task (GO), or (4.) multiple grouped tasks joined by relations (e.g., *first*, *then*, *before*, *after*). Relations (REL) can contain additional temporal constraints (e.g., *within 5 seconds*). Using relations we extract relevant ordering constraints. Task templates are defined in a hierarchical way so they can cover either an unordered set of subtasks, or ordered set of subtasks joined by relations. Each of the individual task (O) is describing a task performed in a given location with the possibility of adding time constraints on the length of the task.

A. Locations

Locations are 6-DoF poses of an appropriate end-effector in the reference system of the workpiece. Their unique names are either given during the linguistic instructions or created automatically, when no name is provided. Locations can be added to the system in two ways: First, by implicit definition during task demonstration using a location denominator like *here*. For example, the instruction "Glue a point *here*" will lead to adding a location with values extracted from the current glue gun pose. Implicit poses are stored relative to the workpiece. Second, locations can be added by explicit

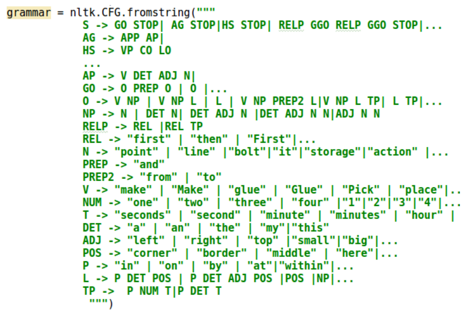
```
grammar = nltk.CFG.fromstring("""
              S -> GO STOP| AG STOP|HS STOP| RELP GGO RELP GGO STOP|...
              AG -> APP AP|
              HS -> VP CO LO
              ...
              AP -> V DET ADJ N|
              GO -> O PREP O | O |...
              O -> V NP | V NP L | L | V NP PREP2 L|V NP L TP| L TP|...
              NP -> N | DET N| DET ADJ N |DET ADJ N N|ADJ N N
              RELP -> REL |REL TP
              REL -> "first" | "then" | "First"|...
              N -> "point" | "line" |"bolt"|"it"|"storage"|"action" |...
              PREP -> "and"
              PREP2 -> "from" | "to"
              V -> "make" | "Make" | "glue" | "Glue" | "Pick" | "place"|..
              NUM -> "one" | "two" | "three" | "four" |"1"|"2"|"3"|"4"|...
              T -> "seconds" | "second" | "minute" | "minutes" | "hour" |
              DET -> "a" | "an" | "the" | "my"|"this"
              ADJ -> "left" | "right" | "top" |"small"|"big"|...
              POS -> "corner" | "border" | "middle" | "here"|...
              P -> "in" | "on" | "by" | "at"|"within"|...
              L -> P DET POS | P DET ADJ POS |POS |NP|...
              TP ->  P NUM T|P DET T
               """)
```

Fig. 3. A sample of our custom grammar including multiple production rules.

definition for later use. It is possible to define reference frames during the location definition to enable different arrangements of parts in the final robot setup.

Definition: A Storage S is a set of locations with its own origin (frame). Individual locations $l \in S$ are obtained from demonstration. The name of the storage is defined in the corresponding utterance after the keyword *showing*. The origin o of the coordinate system is evaluated from n edge points (e_i) coordinates which are also extracted from demonstration: $o = \sum_1^n e_i/n$.
The corresponding utterance in natural language is: "Showing [*Small bolt storage*] corner [*here*] corner [*here*]... location [*here*] location [*here*]..."

All locations (L) and storages (S) are stored in dictionaries with a unique names as key. This allows later referencing to the locations and storages using their names. Additionally, we use a last-in-first-out data structure to track the order of added locations. This enables for example linguistic references to previously added locations without knowing their name like "First glue a point [*here*] and then place a bolt to [*the same location*]" leads to the creation of a location loc_1, which is, after the insertion in our bookkeeping, recalled for a *place* action by using the order of insertion. Future references to locations and storages are more naturally performed using the name – e.g. "Pick a bolt from the small bolt storage" corresponds to the action $pick(bolt, l)$ (see Section IV-B) which can be executed on any location $l \in S_i$, where S_i represents a *small bolt storage*.

B. Teach-in of new tasks

The system allows teaching of two types of tasks:

1. Simple task: A task a is a tuple: a = *(name, type, object, location, time constraint)*, where *name* is a unique name of the task, *type* represents an action primitive or a simple/single task such as {glue, pick, place, make, ...} , object $\in$ {ADJ + [point, it, bolt,...]}, location $\in$ L and time constraints $T = (\min, \max)$. The utterance describing a task has the following format *"<Type><Object><Location><Time constraints>"* (e.g., *"Glue a small bolt here within 5 seconds"* or *"Pick up a big bolt from the big bolt storage."*), see Fig. 4. Each single task or a sequence of tasks which requires to be performed by the same resource (e.g., a Pick and Place action) is represented by a single OVC (see Eq. 1).

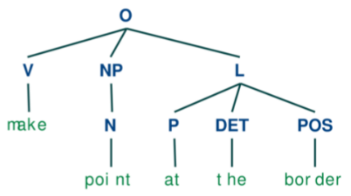

Fig. 4. A simple task: gluing a point in a given location.

2. Task template: A task template AG is a set of OVCs and constraints saved as a template for future reuse. The creation of such a template is triggered whenever a set of demonstrated tasks (e.g., *"Glue a point here then pick up a small bolt from a small bolt storage and place it to the same location."*) is followed by ("This task is called *[glue a small bolt]*"). After the task template is created and its name registered to the system, it can be reused by using its name in the same way as a simple types of tasks (e.g. *"First [glue a small bolt] then glue a point here."*) (see Fig. 6 and Fig. 7). When the task is reused, the template is copied, filled with the location parameters from the current demonstration, and appended to the task description.

C. Ordering and Temporal constraints

Our system supports voice entry for ordering constraints (Allen interval relations) of OVCs as well as quantitative temporal constraints between OVCs. (Note that the actions within an OVC are always sequentially ordered.)

1. Inter-OVC ordering constraint: Ordering constraints are indicated in the abstract representation of a sentence by a subtree 'REL' (REL = {first, then, before, after}) or 'RELP' (RELP = *<REL><Time constraint>*). An example is: *"First glue a point [here] then within 5 seconds first glue a point [here] then pick a small bolt and place it to the same location."* These constraints are transfered to the OVC planner as a set of StartsAfterEnd Constraint (see Eq. 3 and Section VI-A). As can be seen, our system can also handle nested (additional constraints within an already open constraint) and join ordering constraints. A join constraint *First [X] then [Y] then [Z]* (e.g., *First glue a big bolt [here] then glue a small bolt [here] then make a point [here].*") is transfered to two StartsAfterEnd constraint: StartsAfterEnd(OVC_1, OVC_2) and StartsAfterEnd(OVC_2, OVC_3), where OVC_1, OVC_2 and OVC_3 correspond to task X, Y and Z, respectively.

2. Intra-OVC ordering constraint: The constraint on ordering of two or more successive tasks that must be processed by the same resource (typical example is Pick and Place, where pick task induces that place task of the same object will be performed with the same resource, creating an OVC of two consequence locations). An example is a sentence *"First pick a bolt from [L] and then place it to [L]."*, which will be represented as an OVC shown in Eq. 2). Please note, that location L can correspond to a single location as well as to a set of possible locations (e.g., a storage).

1. Intra-OVC temporal constraints: Time constraint corresponding to parameter I in an OVC (see Eq. 1) defining a maximum length of the action (e.g., gluing a point has to be performed in approximately 2 seconds to apply a correct amount of glue). These are indicated within an action as an optional time constraint - e.g., *"Glue a point [here] for 2 seconds."* (resulting range for I will be (2-x, 2+x)).

2. Inter-OVC temporal constraints: These constraints are in the sentence indicated as an additional time parameter within a relation 'RELP' and is passed to the OVC planner as a parameter T in StartsAfterEnd constraint (see Eq. 3) (e.g., *"First pick a small bolt then within 5 seconds place a big bolt."*).

D. Further Editing of OVC-based Scheduling Problem Specifications

Linguistic instructions from the input are processed to retain a list of named locations, a list of named storages, a list of named task templates, a named list of OVCs, and a list of temporal constraints. The algorithm's output is tailored for the OVC-solver, but is also saved as a formatted text file[2]. This file is parsed by the solver. Before that, advanced users can edit this file to add further or more complex constraints, which were impossible to explain by natural language.

V. Implementation and Setup

Fig. 1 gives an overview of the whole system. As the OVC planner has been described in Section 3 already, we focus in this section on the acquisition of voice and demonstration data, its processing and the extraction of OVCs, and the execution of the task and motion plan (currently simulated).

A. Setup for data acquisition

The setup consists of a table top with a calibration checkerboard, two Asus Xtion 3D sensors, an HTC Vive VR set, and a microphone for audio recording. The data from all sensors are broadcasted via the Robot Operating System[3] (ROS). The HTC Vive supplies the 6DOF positions of the two controllers at 60 Hz, both Asus Xtion cameras produce 640x400 RGBD images at 30 Hz. To capture the demonstration of the gluing tasks, one of the HTC Vive controller is attached to a gluegun, while the second HTC Vive controller is used to indicate the ground truth segmentation of the demonstrated tasks. In the experiments, we assumed the workpiece to be fixed to the checkerboard, but in order to define auxiliary coordinate frames, we included special language commands and demonstrations. All sensor data is synchronized using ROS timestamps.

For each showcase (see Section VI), a separate data file (ROS bag file) is recorded. Besides the timestamped RGBD data from the ASUS Xtion and the 6DOF poses from the HTC Vive controller, each bag file contains the button press information for both HTC Vive controllers, speech-to-text

[2]see our web page `http://imitrob.ciirc.cvut.cz/planning.html`
[3]`http://www.ros.org/`

transcripts acquired through Google Speech API and the transformations between the individual coordinate frames. Although, the captured motions are synchronized with the recorded speech. the human demonstration is likely to be not perfect, i.e. the demonstration of a task will not appear in the exact same time as the key word in the linguistic input. Hence, our system matches the demonstrations with the closest appearance of a location keyword. As ground truth, the pose of the glue application is recorded based on the glue gun button press. For each showcase we further provide sentence-wise separated bag files with a checked transcript. [4] To process linguistic input we make use of the Python library NLTK 3.3 (Natural language processing toolkit) [22].

B. Setup for simulation-based execution

We consider a setup consisting of two KUKA LBR iiwa robots mounted on a table with largely overlapping workspaces (see Fig. 2). While we evaluate this paper in simulation, we have the real robotic system available for future experiments. The robots are simulated and visualized using ROS and RVIZ. The OVC solver uses services provided by MoveIt! to make kinematic queries, collision checks, and path planning, but since the planner outputs a trajectory for every arm, the execution of plans is implemented using the follow joint trajectory action.

The setup of the two arms is defined in an URDF file. In the simulation, we add the workpiece and storage frames for the task using a Scene Graph Manager. In a real-robot setup, the user would place the parts (i.e. storages and workpieces) directly in the robot's workspace. Their reference frames might be detected e.g. by a on-top 3D sensor.

VI. Experimental results

As evaluation, we created four showcases (see Fig. 5) and programmed them using our system. For the first showcase, we demonstrated applying glue to several locations, while explaining what should be done. The second showcase contains the same tasks as the first one, but the language provides a number of constraints on the execution order of individual tasks and on groups of tasks. Showcase three illustrates the creation of task templates, teaching separate reference frames, and pick&place actions. Showcase four is similar to showcase three, but features some ordering constraints[5]. In the following, we explain a typical OVC representation of a showcase, discuss the transfer to different workspaces, and finally visualize the solver performance on our showcases.

We show, that the solver finds and optimizes solutions to the modeled tasks and show some statistics of the solutions. In the second part of this section, we discuss the intermediate task description, which is the planner input, but is also adjustable by expert users to incorporate advanced constraints or correct mistakes in the extracted task specification.

[4]see our web page `http://imitrob.ciirc.cvut.cz/planning.html` for source bag files

[5]Please, see the attached video for a visual demo of our system. Video can also be downloaded from our web page `http://imitrob.ciirc.cvut.cz/planning.html`.

A. OVC problem representation example

Type of the task: Group of ordered simple tasks and tasks templates $a_1, ..., a_n$

Sentence: *First glue a small bolt here [showing location/pose] and then within 20 seconds make a point here [showing location] and then make a point here [showing location] in 2 seconds.*

OVC representation:

$$O_1 = [[LA, RA], (gluepoint), [loc_{11}]]$$
$$O_2 = [[LA, RA], (pick, place), [[loc_5, ..., loc_{10}], loc_{11}]]$$
$$O_3 = [[LA, RA], (gluepoint), [loc_{12}], (1.7, 2.3)]$$
$$StartsAfterEnd(O_1, O_2)$$
$$StartsAfterEnd([O_1, O_2], 20)$$
$$L = \{\text{'loc}_1\text{'} : poseStamped, ..., \text{'loc}_1 2\text{'} : poseStamped\}$$
$$HS = \{\text{'small bolt storage'} : \{\text{corners} = [loc_1, ..., loc_4], \text{locations} = [loc_5, ..., loc_{10}]\}\}$$
$$AG = \{\text{'glue a small bolt'} : [O_t, O_{t+1}], StartsAfterEnd(O_t, O_{t+1})\}$$

L - location dictionary

HS - storage dictionary

AG - tasks templates dictionary

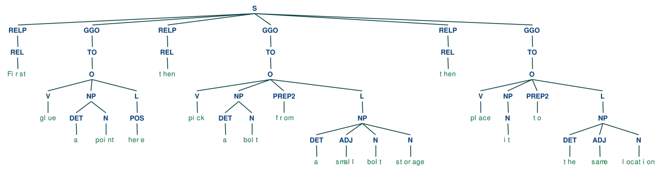
S
RELP REL First
GGO TO O V glue NP DET a N point L POS here
RELP REL then
GGO TO O V pick NP DET a N bolt PREP2 from L NP DET a ADJ small N bolt N storage
RELP REL then
GGO TO O V place NP N it PREP2 to L NP DET the ADJ same N location

Fig. 6. A task template "Glue a small bolt" described by a sentence "First glue a point here then pick a bolt from a small bolt storage then place it to the same location".

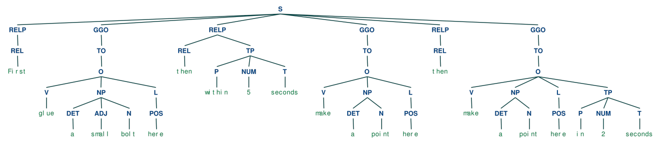
S
RELP REL First
GGO TO O V glue NP DET a ADJ small N bolt L POS here
RELP REL then TP P within NUM 5 T seconds
GGO TO O V make NP DET a N point L POS here
RELP REL then
GGO TO O V make NP DET a N point L POS here TP P in NUM 2 T seconds

Fig. 7. Resulting tree (abstract representation of sentence) gained from parsing the sentence "First glue a small bolt here then within 5 seconds make a point here then make a point here in 2 seconds." (for tree of the task template "glue a small bolt" see Fig. 6).

B. Transfer to different workspaces

The demonstration of the tasks is generally not executed in the robot's workspace and should be transferable to other workspaces. To solve and execute a previously given task with a robot on a different workplace setup, the reference frames of the workpieces and storages have to be detected (e.g., by a RGBD sensor overlooking the workspace) or

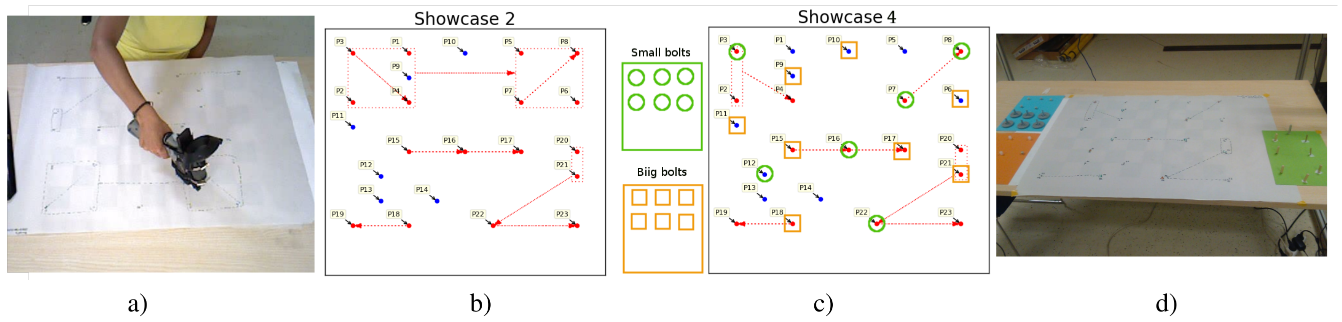

Fig. 5. Showcases: (a) demonstration and (b) abstract visualizations of showcase 2 and (c,d) showcase 4: Showcase 2 is a partially ordered set of point tasks (gluing a point), Showcase 4 represents a partially ordered set of 'gluing a point' tasks, 'gluing a small bolt' task (green circle) and 'gluing a big bolt' task(orange squares). The task templates 'gluing a small bolt' is visualised in Fig. 6.

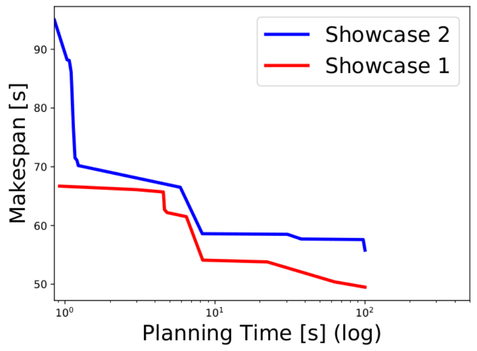

Fig. 8. Makespan vs. solving time for showcase 1 (gluing action in given locations) and showcase 2 (showcase 1 with additional ordering constraints).

defined manually. If a different robot should perform the task, we need to make sure, that the robot's capabilities (i.e. actions each arm can execute) are included in the configuration file. Subsequently, the roadmaps used in the OVC-solver are updated to include the task locations.

C. Experiments

For each showcase, we let the solver run for 500 s. The solver finds solutions of decreasing makespan through changing the assignment of subtasks to arms or choosing a beneficial order of these. In Fig. 8 the makespans of successive solutions of showcase 1 and 2 are compared. It can be seen, that the makespan of the more constrained problem is larger than the unconstrained version.

D. Ease of use

To enable convenient usage of our system, we implemented several features which add to the flexibility of the speech. First, we enable to specify sets of homophones and synonyms of words or phrases, e.g., then:[them], then:[and then, afterwards, after that, and later]. The synonyms allow the user to use richer vocabulary to express the same concept. The homophones are used to correct errors of the voice recognition software. Knowledge of the task and therefore the words likely to be used is utilized when constructing the sets of homophonous and synonymous words. Second, our system is robust to filling words such that "Pick a big bolt from a big bolt storage and then place it here" leads to the same interpretation as "Just take that big bolt from hmm the big bolt storage and afterwards place it directly here.". Third, common variations in the sentences stem from the use of articles (a/the/one), which is recognized and handled directly in the grammar[6].

To evaluate the ease of use of our system, we conducted a user-study on 2 novice users. We introduced them briefly to our system. In particular, we provided them with list of language constructs they can use. We explained to them that their linguistic input has to be accompanied by a physical demonstration and that the location references should roughly occur at the same time and in the same order as the linguistic description. Finally, we showed three examples of demonstrations with an increasing complexity. Users had to then fulfill given tasks based on the instructional booklet and were corrected on each mistake. It took approximately 15 minutes until both users were able to instruct the robot for all showcases.

VII. Conclusion and Discussion

In this paper, we demonstrated and evaluated a system to program robots using natural language and simultaneous demonstrations. But instead of translating those inputs directly into a definite robot plan, we compile a planning problem using Ordered Visiting Constraints [5]. The definition of such an OVC scheduling problem is particularly well suited for the definition by linguistic input, because the formalism abstracts away any robot dependency and thus reduces the complexity in the necessary language constructs. Subsequently, the solver is used to find an optimized schedule with regard to the makespan by utilizing the robot's resources.

There are several aspects, to further investigate in the future. First, we would like to extend the amount of actions, tasks and constraints which our system can handle. Some visions are trajectory actions (e.g., gluing a line, welding along an edge, etc.), spatial constraints (e.g. constrain all objects on the left from here, etc.), correcting mistakes ('sorry, don't make the last glue point there, but here') or referencing to multiple former tasks (e.g., last 5 subtasks have to be made within 20 seconds). Second, we would like include hand and finger gestures to enrich the interaction.

[6]see our web page `http://imitrob.ciirc.cvut.cz/planning.html` for the audio to sentence converter

As a complementary method, we would like to explore the possibilities to use RGB(D) sensors to track 6DOF pose of the object and hand and avoid usage of the external HTC Vive system. Finally, the optimality of the found solutions and robot behavior are object of future studies.

VIII. Acknowledgment

This work was supported by the European Regional Development Fund under project Robotics for Industry 4.0 (reg. no. $CZ.02.1.01/0.0/0.0/15\_003/0000470$) and TAČR Zeta project Imitation learning supported by language for industrial robots (no. TJ01000470) granted by Technological agency of Czech republic.

References

[1] S. A, Richard, L. Tim, W. Carolee, W. Gabriele, and Z. Howard, *Motor Control and Learning*. Human Kinetics, 2018, google-Books-ID: RpBFDwAAQBAJ.

[2] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, "FFRob: Leveraging Symbolic Planning for Efficient Task and Motion Planning," *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 104–136, Jan. 2018, arXiv: 1608.01335. [Online]. Available: http://arxiv.org/abs/1608.01335

[3] J. Kurosu, A. Yorozu, and M. Takahashi, "Simultaneous Dual-Arm Motion Planning for Minimizing Operation Time," *Applied Sciences*, vol. 7, no. 12, p. 1210, Nov. 2017. [Online]. Available: http://www.mdpi.com/2076-3417/7/12/1210

[4] A. Kimmel and K. E. Bekris, "Scheduling Pick-and-Place Tasks for Dual-arm Manipulators using Incremental Search on Coordination Diagrams," 2017.

[5] Anonymous, "…," in *submitted to IROS*, 2018. *Details omitted for double-blind reviewing.*

[6] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning." in *AAAI*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[7] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 1.

[8] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 4565–4573.

[9] D. Kulić, C. Ott, D. Lee, J. Ishikawa, and Y. Nakamura, "Incremental learning of full body motion primitives and their sequencing through human motion observation," *The International Journal of Robotics Research*, vol. 31, no. 3, pp. 330–345, 2012.

[10] J. Butterfield, S. Osentoski, G. Jay, and O. C. Jenkins, "Learning from demonstration using a multi-valued function regressor for time-series data," in *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*. IEEE, 2010, pp. 328–333.

[11] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, "Robot learning from demonstration by constructing skill trees," *The International Journal of Robotics Research*, vol. 31, no. 3, pp. 360–375, 2012.

[12] O. C. Jenkins and M. J. Matarić, "A spatio-temporal extension to isomap nonlinear dimension reduction," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 56.

[13] Y. Hwang, P. Chen, and P. Watterberg, "Interactive task planning through natural language," vol. 1. IEEE, 1996, pp. 24–29. [Online]. Available: http://ieeexplore.ieee.org/document/503568/

[14] Y. K. Hwang, P. C. Chen, and P. A. Watterberg, "Interactive task planning through natural language," in *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, Minneapolis. Minnesota, USA, Apr. 1996.

[15] M. Stenmark and P. Nugues, "Natural language programming of industrial robots." IEEE, Oct. 2013, pp. 1–5. [Online]. Available: http://ieeexplore.ieee.org/document/6695630/

[16] A. Mohseni-Kabir, C. Rich, S. Chernova, C. L. Sidner, and D. Miller, "Interactive hierarchical task learning from a single demonstration," in *Proceedings of the 10th Annual ACM/IEEE International Conference on Human-Robot Interaction (HRI '15)*, Portland, Oregon, USA, Mar. 2015, pp. 205–212.

[17] P. E. Rybski, K. Yoon, J. Stolarz, and M. M. Veloso, "Interactive robot task training through dialog and demonstration," in *Proceedings of the 2nd ACM/IEEE International Conference on Human-Robot Interaction (HRI '07)*, Arlington, Virginia, USA, Mar. 2007, pp. 49–56.

[18] J. Kirk, A. Mininger, and J. Laird, "Learning task goals interactively with visual demonstrations," *Biologically Inspired Cognitive Architectures*, vol. 18, pp. 1–8, 2016.

[19] S. Ekvall and D. Kragic, "Robot learning from demonstration: A task-level planning approach," *International Journal of Advanced Robotic Systems*, vol. 5, no. 3, Sept. 2008.

[20] G. Suddrey, M. Christopher Lehnert, M. Eich, F. Maire, and J. Roberts, "Teaching robots generalizable hierarchical tasks through natural language instruction," *IEEE Robotics and Automation Letters*, vol. 2, no. 1, pp. 201–208, Jan. 2017.

[21] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[22] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.