Specifying Dual-Arm Robot Planning Problems through Natural Language and Demonstration

Jan Kristof Behrens*,[1], Karla Stepanova*,[2], Ralph Lange[1], and Radoslav Skoviera[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 designed to quickly take over tasks at typical industrial workbenches, the direct teaching of task sequences hardly utilizes the robots' capabilities. We therefore propose a two-staged approach that combines natural language instructions and demonstration with simultaneous task allocation and motion scheduling based on constraint programming. Instead of providing a task description and demonstrations that are replayed to a large extent, the user describes tasks to be scheduled with all relevant constraints and demonstrates relevant locations relative to workpieces and other objects.**

With explicitly stated constraints on the partial ordering of tasks, the solver allocates the tasks to the robot arms and schedules them in time while avoiding self-collisions and reducing the makespan in our experiment by 33%. The linguistic concepts of naming and grouping enable systematic reuse of sub-task ensembles. The proposed approach is evaluated with four variants of a gluing use-case from furniture assembly in user studies with ten participants. In these user studies, we observed a speed-up for the task definition of more than 6 times compared to a textual specification of the planning problems using the Python-based planner API.

***Index Terms*—Task scheduling, multi-modal robot programming, dual arm manipulation, learning by demonstration.**

I. Introduction

MANUFACTURING tasks for dual-arm robots in industrial use-cases are prime examples for the need for integrated task and motion planning (ITAMP) due to the huge state space spanned by the many degrees of freedom on task and motion level. Despite the great advances in ITAMP in the recent years (e.g., [1], [2]), relevant tools generally require deep understanding of the underlying planning mechanisms and they may undisputedly be referred to as expert tools. Multi-modal methods based on natural language and demonstration are a promising approach to bring such advanced planning techniques to the shop-floor. We hypothesize that multi-modal methods lower the inhibition threshold for the use of planning techniques and reduce the mental burden of the robot programmer/instructor.

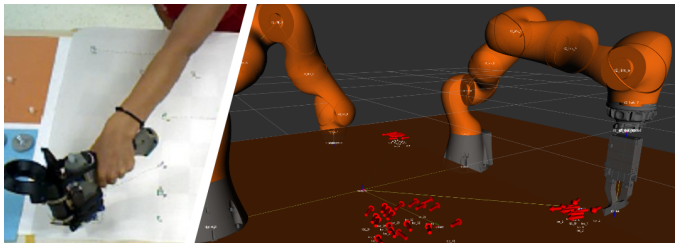

Fig. 1. Illustration of the running example of gluing bolts into a board. On the left hand side, the instructor demonstrates poses for the application of glue on a white board using a glue gun while explaining the corresponding tasks and constraints in natural language. From this input, a planning problem specification expressed by Ordered Visiting Constraints is generated [3]. After solving this planning problem for two manipulators, the task is performed in simulation as depicted on the right hand side.

Most works in the very active research in the fields of robot learning from demonstration and multi-modal robot programming start with the learning/teaching of a directly executable plan and then consider generalization and transferability to related scenarios. Only very few works (e.g., [4], [5]) propose multi-modal techniques for the specification of planning *problems* and then employ a planner – fed with further information on the scenario – to retrieve an executable plan. Therefore, we propose a multi-modal input method for a subclass of ITAMP problems as explained next.

Example. As a running example, we consider a showcase from furniture construction depicted in Figure 1: In this showcase, glue has to be applied into the holes of a large board and then the corresponding bolts have to be picked up and inserted. The order in which the glue is applied and in which the bolts are inserted can be chosen by the planner to optimize the makespan. Also, the allocation of these tasks to the two robot arms is subject to optimization. In such an optimization, the working ranges of the arms have to be considered and collision-free trajectories have to be planned.

Even this small example exhibits a huge combinatorial complexity given by the Cartesian product of possible task sequences and task allocations to the robot arms. The complexity is further increased by alternative joint configurations to reach the desired poses, alternative paths between these configurations and possible time-scaling of the trajectories.

At the same time, this example illustrates some typical characteristics of industrial workplaces and manufacturing tasks.

Manuscript received: September, 10th, 2018; Revised: November, 29th, 2018; Accepted: January, 12th, 2019.

This paper was recommended for publication by Editor Dongheui Lee upon evaluation of the Associate Editor and Reviewers' comments. 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.

*Both authors contributed equally to this work.

[1]Jan Behrens and Ralph Lange are with Robert Bosch GmbH, Corporate Sector Research and Advance Engineering, Renningen, Germany, behrens.jk@gmail.com and ralph.lange@de.bosch.com.

[2]K. Stepanova and R. Skoviera are with Czech Technical University in Prague, Czech Institute of Informatics, Robotics, and Cybernetics, karla.stepanova@cvut.cz and radoslav.skoviera@cvut.cz.

Digital Object Identifier (DOI): see top of this page.

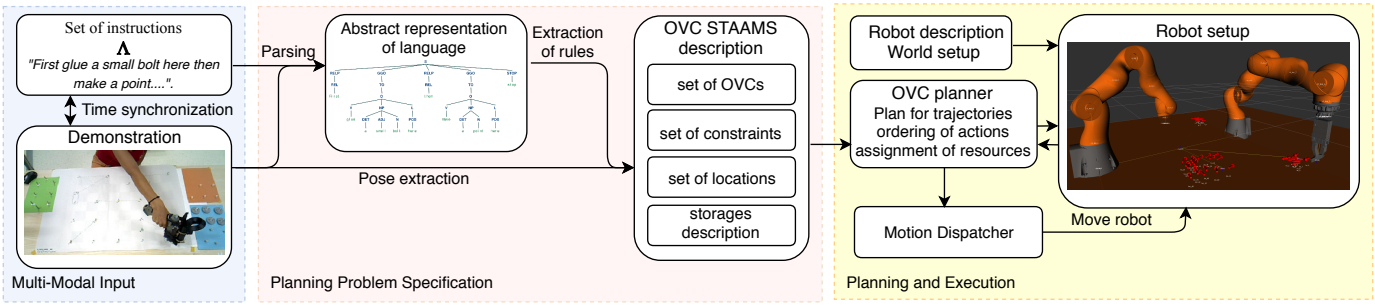

Fig. 2. Overview of the system.

Two key characteristics are: First, the task decomposition is already defined with the design of the workpieces. Second, the workspace is largely unobstructed and there exists a collision-free subset that does not alter over time and allows to reach all relevant locations with at least one robot arm. As a consequence, ITAMP for dual-arm robots in such scenarios boils down to *simultaneous task allocation and motion scheduling* (STAAMS). Details on these characteristics and STAAMS as well as a description of our STAAMS solver based on constraint optimization can be found in [3]. The solver processes STAAMS problems formulated using an abstraction called *Ordered Visiting Constraints* (OVCs). An OVC models in an effective and robot-agnostic way a sequence of actions to be executed by a manipulator at different locations. Ordering and temporal constraints within and between OVCs can be specified as well as resource constraints on tools, workpiece holders, etc. Also, bimanual tasks can be modeled.

In this paper, we propose a multi-modal method combining natural language and demonstration for the specification of STAAMS problems using OVCs. Our main contributions are:

1) A flexible grammar for specifying OVCs with corresponding ordering and temporal constraints. An important feature of this grammar is the option of interleaved teaching of task templates with the overall problem specification, which can be even faster than one-shot learning of a directly executable plan.
2) Integrated mechanisms based on pointing gestures and natural language to define locations, poses and even polygonal places.
3) Suitable abstractions for translating an OVC-based STAAMS problem specification to a different workplace and robot setup.
4) Lowered inhibition threshold and reduced mental burden during task demonstration using the proposed method compared to textual programming of STAAMS problems using a Python-based OVC API, as indicated by a user-study.

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 STAAMS solver in Section III. In Section IV, we describe how OVC-based STAAMS problem specifications are obtained from natural language and demonstrations, with the 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

In this section, we first give a brief overview to works on robot learning from demonstration and natural-language-based (NL-based) robot programming. Then, we discuss existing works on NL-based specification of planning and scheduling problems, from general works to robotic-specific approaches.

a) Robot learning from demonstration: This field 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]). In the approach proposed in this paper, we assume a predefined, extensible 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 and scheduling to use the full capacity of dual-arm robots, as argued in Section I.

b) NL-based robot programming: Most works in this field use a multi-modal approach, combining natural language with pointing or teach-in techniques. For example, in an early work in 1996, Hwang et al. [12] 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. An overview of methods for NL-based human-robot cooperation is provided in [13]. According to the division in the paper, our method is a grammar model integrating temporal, spatial, and ordering relations and allowing a wide variability of linguistic instructions. Compared to the mentioned works, our system allows teaching of immediately reusable task templates during task specification. This makes our grammar extensible by abstract NL expressions like in empirical association models and thus eliminates a limitation of fixed grammar models. 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 [14]. In [15], Mohseni-Kabir et al. presented an interactive

system for specifying hierarchical task networks (HTNs) for manipulation tasks. The instructor explains the tasks in a top-down manner while the system asks for unknown task names and whether it should generalize tasks to similar objects in the scene. A similar approach based on behavior networks instead of HTNs is presented by Rybski et al. in [16]. 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.

c) NL-based planning and scheduling problems: Only a small body of work considers the NL-based specification of planning and scheduling problems. Kirk et al. [17] show the benefits of using visual demonstration of goal states amended with human instructions to define diverse tasks. Lindsay et al. [18] propose a method to generate PDDL planning domain models from natural language. Both systems were tested on simple riddles and games which are far from the complexity of STAAMS or ITAMP, not involving any motion planning for manipulators.

In [5], Ekvall and Kragic proposed a robot learning system for a mobile manipulator that uses a STRIPS-like planner which obtains its input 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 [4]. 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 NL-based input of further constraints. Again, motion planning is not integrated with task planning.

This paper's approach differs in several aspects from the mentioned works: First, it is based on a planning concept and method that deeply integrates the task level with the motion level. Second, it allows to specify advanced temporal and ordering constraints as well as teaching immediately reusable task templates by natural language. Third, it combines demonstration techniques and natural language into an effective multi-modal approach.

III. Ordered Visiting Constraints for STAAMS

In this section, we first describe the OVC model and solver as a tool for modeling and solving simultaneous task allocation and motion scheduling (STAAMS) problems in dual-arm manipulation settings, as proposed by Behrens et al. [3]. In the second part of this section, we explain 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 of 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 on the task and motion level while executing the action according to the given task decomposition. The solver (see Fig. 3) is based on constraint programming (CP) (programmatically utilizing Constraint Satisfaction Problems) and constraint optimization (sequentially solving constraint programming 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. The input for the STAAMS solver is a CP-based task specification by one or more *Ordered Visiting Constraints* (OVC) (see Fig. 3 top). An OVC is defined as a tuple

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

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. the 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 time-scaled trajectories representing 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 (see Fig. 3 bottom). Precomputed roadmaps [19] are used to discretize the configuration space per arm. 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 generic constraints among them. Similarly, constraints among OVCs may be defined using propositional logic, Allen's interval algebra, and set-theoretic expressions.

Specifying a task with OVCs means to create for each segmentable subtask or series of subtasks an OVC and constrain

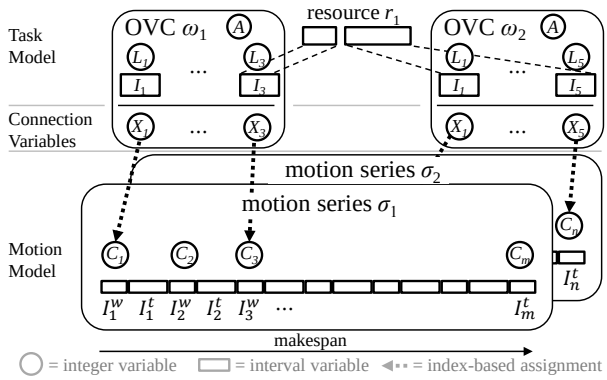

Fig. 3. CP model for task and motion scheduling.

its variables. To create an OVC, a set of suitable arms (i.e., arms with the appropriate end-effector) and a set of actions should be specified. 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.

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

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

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

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. 3, left) is declared.

IV. Specifying OVC scheduling problems by linguistic input and demonstration

In the proposed architecture, 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. 4) 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, types of actions, etc.). Each sentence can be: (1.) a definition of new task template (AG), (2.) a definition of

```
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"
          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"|...
          P -> "in" | "on" | "by" | "at"|"within"|...
          L -> P DET POS | P DET ADJ POS |POS |NP|...
          TP ->  P NUM T|P DET T """)
```

Fig. 4. A sample of our grammar including multiple production rules.

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 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.

A storage S is a set of locations relative to its own coordinate 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 frame 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

We allow teaching of two types of tasks:

a) 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. 5. Each single task or a sequence of tasks which requires to be performed by the same arm (e.g., a Pick and Place action) is represented by a single OVC (see Eq. 1).

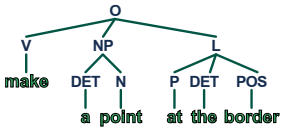

Fig. 5. A simple task: gluing a point in a given location.

b) 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 named, 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. 8 (top)). 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 approach 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 ordered sequentially.)

a) 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 transferred 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.

b) Intra-OVC ordering constraint: The constraint on ordering of two or more successive tasks that must be processed by the same resource Typical application 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 consecutive locations. An example for an OVC with such a constraint is the 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).

c) 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 duration for I will be 2 s).

d) Inter-OVC temporal constraints: These constraints are in the sentence indicated as an additional time parameter within a relation 'RELP' and are 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. 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 the 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.[1]

E. Editing of OVC-based STAAMS Problem Specifications

The 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. 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.

[1]see http://imitrob.ciirc.cvut.cz/planning.html for the audio to sentence converter, user-study tutorials and editable template

V. Implementation and Setup

Figure 2 gives an overview of the system. As the OVC planner has been described in Section III 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[2] (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 is recorded. Besides the RGBD data from the ASUS Xtion and the 6DoF poses from the HTC Vive controller, each recording contains the button press information of both HTC Vive controllers, speech-to-text transcripts acquired through Google Speech API and the necessary coordinate transformations. Although, the captured motions are synchronized with the recorded speech, the human demonstration is likely to be imperfect, i.e. the demonstration of a location will not appear in the exact same time as the location denominator in the linguistic input. Hence, our system has to match the demonstrated locations with the closest appearance of a location denominator. Furthermore, we check for mismatches between demonstration and linguistic input (e.g., in a case of more denominators than demonstrated locations, an error is reported). As ground truth, the pose of the glue application is recorded based on the glue gun button press. Sentence-wise separated files with a transcript are provided.[3] To process linguistic input we make use of the Python library NLTK 3.3 (Natural language processing toolkit) [20].

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. 3). 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 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 custom management interface for the scene graph. 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 for instance by an on-top 3D sensor.

[2]http://www.ros.org/

[3]see our web page http://imitrob.ciirc.cvut.cz/planning.html for source bag files, visual demo of our system and detailed description of showcases

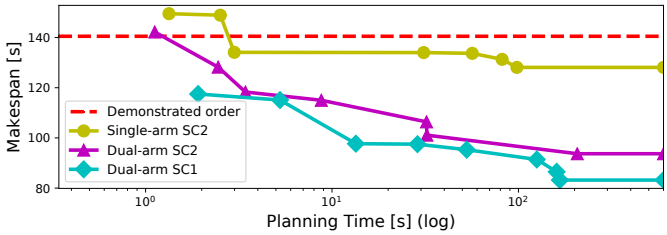

Fig. 6. Makespan vs. solving time for showcase 2 (gluing action in given locations with constraints on the the ordering) compared to the showcase 1 (dropping the ordering constraints), single-arm execution, and single-arm execution in the demonstrated order.

VI. Experimental Results

For the evaluation, we defined four showcases illustrated in Fig. 7 based on the running example introduced in Section I. The complexity of the STAAMS problem is increasing with each showcase and thus the difficulty of the linguistic constructs to be used by the instructor. In detail, the showcases are: (1.) Demonstration of the points for the application of glue without any constraints. (2.) Introduction of ordering constraints between some of these glue points. (3.) Introduction of task templates for the insertion of the bolts and temporal constraint between glue application and the bolt insertion. (4.) Combination of ordering constraints between glue points and temporal constraint on bolt insertion.[3]

Figure. 6 illustrates the significant benefits of STAAMS optimization in this example. In the showcases 1 and 2, the OVC-based STAAMS solver reduces the makespan by 41% and 33%, respectively, compared to a typical fixed task order given by an instructor - namely the order used for multi-modal problem specification. Even for a single-arm robot, the STAAMS solver reduces the makespan by 9%. This optimization is only legitimate, because the constraints are explicitly specified in the verbal instructions. In the following, we explain a typical OVC representation of the showcase 4 (see Fig. 7), present the results of our user-study conducted on 10 participants (7 novice and 3 trained), and finally discuss the translation to different workspaces.

A. OVC problem representation example

Type of the task: Group of ordered simple tasks and tasks templates $a_1, ..., a_n$ (showcase 4, see Fig. 7)

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

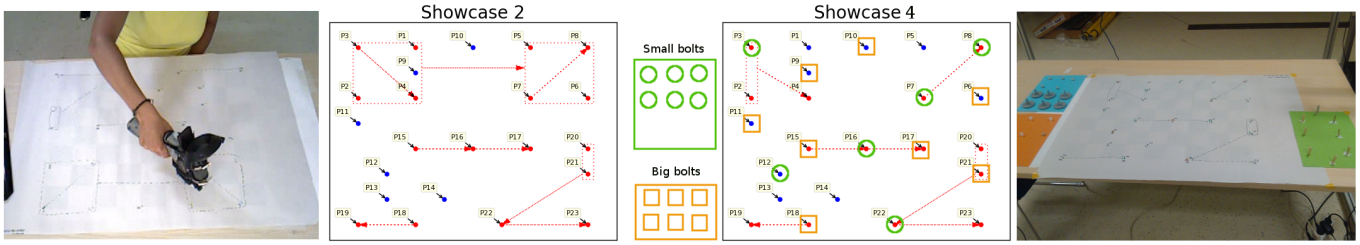

Fig. 7. Showcases: (a) demonstration and (b) abstract visualizations of showcase 2 and (c,d) showcase 4: Showcase 2 is a partially ordered set of point tasks (gluing a point), Showcase 4 represents a partially ordered set of 'gluing a point' tasks, 'gluing a small bolt' task (green circle) and 'gluing a big bolt' task(orange squares). The task templates 'gluing a small bolt' is visualized in Fig. 8 (top).

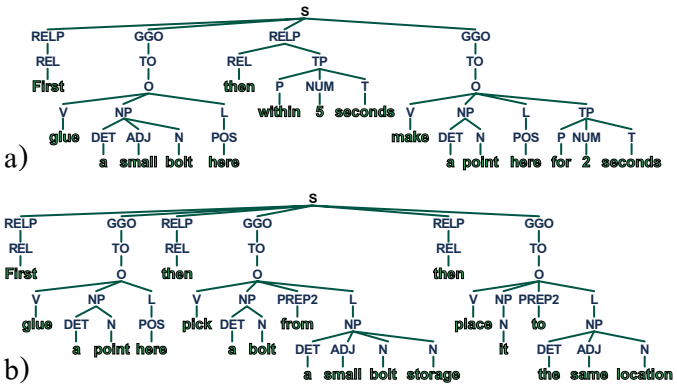

Fig. 8. (a) A task template "glue a small bolt" described by a sentence "First glue a point here then pick a bolt from a small bolt storage then place it to the same location". (b) Resulting tree (abstract representation of sentence) gained from parsing the sentence "First glue a small bolt here then within 5 seconds make a point here for 2 seconds." The task template "glue a small bolt" is reused in this sentence.

location] and then make a point here [showing location] in 2 seconds.

OVC representation:

$O_1 = [[LA, RA], (gluepoint), [loc_{11}]]$
$O_2 = [[LA, RA], (pick, place), [[loc_5, ..., loc_{10}], loc_{11}]]$
$O_3 = [[LA, RA], (gluepoint), [loc_{12}], (1.7, 2.3)]$
$StartsAfterEnd(O_1, O_2)$
$StartsAfterEnd([O_1, O_2], 20)$
$L = \{$'loc$_1$' : $poseStamped, ...,$ 'loc$_1$2' : $poseStamped\}$
$HS = \{$'small bolt storage' :
$\{$corners $= [loc_1, ..., loc_4]$,
locations $= [loc_5, ..., loc_{10}]\}\}$
$AG = \{$'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

B. User-studies: Ease of use and benefits of multimodality

To evaluate the ease of use of our system conveyed by the features described in Section IV-D, we conducted a user-study on 7 novice users. We provided them with an instructional video (8 min. in duration) of our system and a list of linguistic 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. As a part of the instructional video, we showed three simple examples with an increasing complexity. Users then had to fulfill all given tasks and were corrected on each mistake. We measured the number of trials until success and time of performance for individual showcases (see Table I). Average time per defining a single OVC or Constraint for all showcases was between 1-3 s.

TABLE I
EASE OF USE EVALUATION ON 4 SHOWCASES (7 NOVICE USERS).

Showcase	SC1	SC2	SC3	SC4
# of actions	23	29	61	68
# Trials to success	1	2.4	2	1
Time (novice) [s]	37± 13	70 ± 7	158± 34	173 ± 5
Time per action [s]	1.6	2.4	2.59	2.54

TABLE II
BENEFITS OF MULTI-MODALITY (7 NOVICE, 3 TRAINED USERS).

SC3	Multi-modal spec. (our)		Textual spec.	speed-up
Time [s]	Templates	No templates		
Novice	158 ± 34	186 ± 23	911 ± 242	(6.4±3.1)x
Trained	59 ± 4	113 ± 12	540 ± 36	(9.2±0.2)x

To compare teaching using our multi-modal input method to the conventional textual specification, we performed user studies on 7 novice users and 3 trained users, who had to specify the showcase 3 using the simplified textual notation similar to the STAAMS solver's python API. We provided the users with a visual diagram of the showcase (see Fig 7) including the location names necessary to specify the task. Thus, assuming the locations are already known to the system. This enables the user to concentrate on the task of specifying OVCs and constraints, solely. The experiment with this simplification was used to find a lower bound on the time needed for the textual specification of the showcase (see results in Table II).

The usage of templates reduced the time for the multi-modal task specification by 17% and 48% (novice and trained users) compared to the specification without templates. Overall, the proposed multi-modal method was approximately 8x quicker than textual specification. As can be seen in Fig. 9, the definition of task templates takes significant time for novice users in the beginning of the demonstration, but allows more

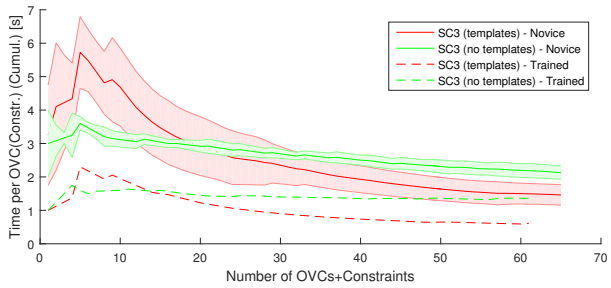

Fig. 9. Cumulative time per OVC/constraint vs. number of defined OVCs/constraints in showcase 3 for 7 untrained (average and standard deviation) and trained users with/without using action templates. It can be seen, that using task templates significantly reduces the time for the overall task specification for all users alike.

than 25% reduction of the time for item definition. Trained users specify new templates very effectively. All users reported that using templates was very convenient and they wouldn't like to use a system without this feature.

C. Translation 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 an RGBD sensor overlooking the workspace) or defined manually.

VII. Conclusion and Discussion

In this paper, we demonstrated and evaluated a system to program robots using natural language and simultaneous demonstration. Instead of translating those inputs directly into a definite robot plan, we compile a Simultaneous Task Allocation and Motion Scheduling problem using Ordered Visiting Constraints as introduced by Behrens et al. [3]. Through a user-study on 10 users, we proof that our system is easy and efficient to use. Our untrained participants were able to specify a task using the Python-OVC API in 911 ± 242 s. Using the proposed system, they were 6.4 ± 3.1 times faster (158 ± 34 s). The explicit definition of ordering constraints allows the solver to deviate from the demonstrated action sequence and thus optimize the makespan, which leads in our example to a 9% speed-up for a single-arm robot and to a 33% speed-up for a dual-arm robot.

In our future work, we want to further investigate the following aspects. First, we would like to extend the amount of actions, tasks and constraints, which are teachable through our system (e.g., gluing lines or welding along an edge). Also, referencing tasks with spatial qualifiers (e.g., all tasks right of a location) or using the definition order (e.g., the last 5 subtasks) could further increase the usability of our system. Closing the feedback-loop to the user by a visual user interface and a dialogue system, would enable the user to recognize and correct errors directly during demonstration. Third, we would like to include hand and finger gestures and detect the tools from RGBD data to make the demonstrations more natural and avoid the usage of an external tracking system.

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] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*. New York, NY, USA: Cambridge University Press, 2016.
[2] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *IJRR*, vol. 37, no. 10, pp. 1134–1151, 2018.
[3] J. K. Behrens, R. Lange, and M. Mansouri, "A constraint programming approach to simultaneous task allocation and motion scheduling for industrial dual-arm manipulation tasks," 2019, arXiv:1901.07914 [cs.RO].
[4] G. Suddrey, M. Christopher Lehnert, M. Eich, F. Maire, and J. Roberts, "Teaching robots generalizable hierarchical tasks through natural language instruction," *IEEE-RAL*, vol. 2, no. 1, pp. 201–208, Jan. 2017.
[5] S. Ekvall and D. Kragic, "Robot learning from demonstration: A task-level planning approach," *IJRR*, vol. 5, no. 3, Sept. 2008.
[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 *Proc. of 21st ICML*, 2004.
[8] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Inf. Processing Syst.*, 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," *IJRR*, 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 *Proc. of 10th IEEE-RAS Humanoids*, 2010, pp. 328–333.
[11] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, "Robot learning from demonstration by constructing skill trees," *IJRR*, vol. 31, no. 3, pp. 360–375, 2012.
[12] Y. K. Hwang, P. C. Chen, and P. A. Watterberg, "Interactive task planning through natural language," in *Proc. of ICRA*, Minneapolis, MN, USA, Apr. 1996.
[13] R. Liu and X. Zhang, "A review of methodologies for natural-language-facilitated human-robot cooperation," *arXiv preprint: 1701.08756*, 2017.
[14] M. Stenmark and P. Nugues, "Natural language programming of industrial robots," in *Proc. of 44th ISR*, Seoul, Korea, Oct. 2013.
[15] A. Mohseni-Kabir, C. Rich, S. Chernova, C. L. Sidner, and D. Miller, "Interactive hierarchical task learning from a single demonstration," in *Proc. of 10th ACM/IEEE Int'l Conf. on HRI*, USA, 2015, pp. 205–212.
[16] P. E. Rybski, K. Yoon, J. Stolarz, and M. M. Veloso, "Interactive robot task training through dialog and demonstration," in *Proc. of 2nd ACM/IEEE Int'l Conf. on HRI*, USA, 2007, pp. 49–56.
[17] J. Kirk, A. Mininger, and J. Laird, "Learning task goals interactively with visual demonstrations," *Biologically Inspired Cognitive Architectures*, vol. 18, pp. 1–8, 2016.
[18] A. Lindsay, J. Read, J. F. Ferreira, T. Hayton, J. Porteous, and P. Gregory, "Framer: Planning models from natural language action descriptions," in *Proc of ICAPS'17*, Pittsburgh, PA, USA, Jun 2017.
[19] 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.
[20] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc., 2009.