Robot Programming from Demonstration, Feedback and Transfer

Yoan Mollard[†] Thibaut Munzer[†] Andrea Baisero[‡] Marc Toussaint[‡] Manuel Lopes[†]

***Abstract*— This paper presents a novel approach for robot instruction for assembly tasks. We consider that robot programming can be made more efficient, precise and intuitive if we leverage the advantages of complementary approaches such as learning from demonstration, learning from feedback and knowledge transfer. Starting from low-level demonstrations of assembly tasks, the system is able to extract a high-level relational plan of the task. A graphical user interface (GUI) allows then the user to iteratively correct the acquired knowledge by refining high-level plans, and low-level geometrical knowledge of the task. This combination leads to a faster programming phase, more precise than just demonstrations, and more intuitive than just through a GUI. A final process allows to reuse high-level task knowledge for similar tasks in a transfer learning fashion. Finally we present a user study illustrating the advantages of this approach.**

I. Introduction

New societal and economical challenges demand new robotic systems that are more adaptable to different environments and tasks, and easier to use. Robots are starting to be used at home, and in flexible industrial cells, both applications that demand a constant change between tasks and an easy way to instruct and personalize the behavior of the robotic system. Assembly tasks represent a large set of challenging tasks to be taught to a robot that have immediate real life applications in the context of small scale industrial cells. New interest in personalized furniture and other consumer goods require an increased dexterity and frequent task switching. A major challenge is to provide intuitive and fast ways to program robots so that a larger part of the population can work with those robotic systems. For this end, a set of desired properties of a system that can be instructed and executed in a friendly way will include:

easy to program where a simple demonstration of a task allows, in most cases, to program it;

easy to instruct where corrections of what has been learned are easy to provide;

shared task awareness where a joint comprehension of the task being executed, the world state and the role of each partner, is shared;

shared initiative where the robot does not require a step-by-step instruction that renders the interaction very tiring, the system is able to start its own behavior and wait when it is uncertain about the task, which allows considering that the user might want to correct or change the behavior at any point in time;

hierarchical learning where all parts of the task are easy to program. Even assuming that the robotic system is already equipped with many skills, at all levels there will be limitations that must be recoverable at runtime.

Similar lists have already been introduced in different robotic roadmaps [10], [11], [18].

An easy way to program a task is just to show it. Several learning from demonstration formalisms and applications exist but they need to be completed with other techniques before being deployed to the real world. Mainly learning from demonstration does not achieve good enough precision and needs to be made more robust to mistakes in the demonstration, to compensate the lack of robotic expertise of the operator, and to allow fast task switching. For this reason we propose to improve the interaction protocol to allow the user to better understand what the robot is learning and trying to execute. We do this by means of a graphical user interface (GUI) in the programming loop, showing learned information: sequence of manipulation steps, constraints between objects and assembly plan. Other modalities of the task could be also learned and visualized, for example motor primitives. These interaction tools create a shared task awareness that allows the user to understand what the robot has learned, its representation, and to provide feedback and corrections to the robot.

In this paper we present our approach for robotic programming that considers demonstrations, feedback and transfer. Our approach can be decomposed in two distinct phases as illustrated in Fig. 1. The initial phase *Task Learning* creates a task representation from demonstrations. The second phase *Task Refining* is an iterative and interactive process allowing the task representation learned by the robot to converge to the task desired by the user. Our main contribution is to show that to instruct robots in an intuitive and precise way we need to combine several forms of interaction that have complementary advantages. Other contributions include a hierarchical approach to learn and represent assembly tasks, a new perspective on knowledge transfer, and a validation of the impact of each form of learning.

The paper is organized as follows. After describing the related work, we present each phase in section 3 and 4 respectively. The section 5 presents a user study evaluating the impact of the proposed approach. The study demonstrated that the first process speeds up the programming phase while the second one allows programming more precise plans.

II. Related Work

Learning from demonstration (LfD) algorithms [27], [5] have focused on limited forms of demonstrations, in most

Work supported by the 3rdHand project, funded by the European Union under the FP7 programme (FP7-ICT-2013-10610878)

[†]Flowers Team, Inria, France. firstname.lastname@inria.fr

[‡]Machine Learning and Robotics Lab, University of Stuttgart, Germany. firstname.lastname@ipvs.uni-stuttgart.de

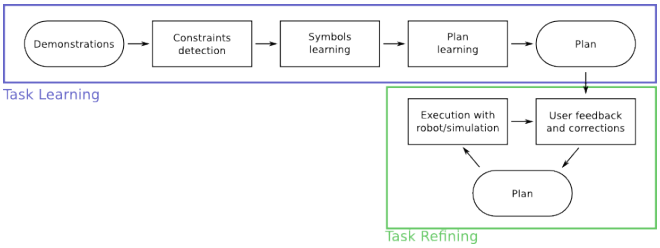

Fig. 1. Workflow for learning a task by demonstration, feedback and transfer. Blue frame: task learning from demonstrations. Green frame: iterative and interactive process for task refining.

cases just working with a specific type of demonstration. Recently, some approaches started to consider a larger variety of demonstrations that go beyond trajectories acquired by means of optical tracking or kinesthetic teaching. These forms include keypoints [1], [2], learning from failures [14], active approaches [17], [9], and even the use of loosely specified protocols [13].

New approaches start to consider how to improve the knowledge acquired from a demonstration through different processes: self-improvement and scalability [15], training the user to provide better demonstrations [8], correcting the task by the means of a visual programming language [4], allowing the system to request specific demonstrations or clarifications. The last option has several advantages: theoretically it allows faster learning [19], it provides information to the user about what the system understands and potentially increases the trust he can have in the system. In many cases we see a trend of unifying the training and execution steps where the instructor is free to interrupt the system when a correction is needed [20], and also allows the robotic system to request further instruction when necessary [9].

Another domain of interest is learning hierarchical representation of tasks. In the robotic domain this consists into learning high-level task plans and also low-level motor controllers. Works have mostly considered these levels separately even if several approaches already consider the learning and execution of such hierarchical systems. Notable examples are [23], [22] that presented an integrated approach to segment manipulation demonstrations into actions, each represented by a Dynamic Movement Primitive (DMP); or [26] that start from low-level demonstrations to learn a high-level representation of actions using logic-based approaches.

Some works focus on learning a high-level representation by learning symbols. These symbols can be the labeling of key features of a state [16], or even the meaning of instruction signals [13]. On the opposite, some work focus on learning a low-level representation: [25] studied LfD in an industrial context by first manually splitting the overall task in sequence of actions, then individually taught to the robot. Low-level motor primitives represented using DMPs can also be stored in a library and then called by a high-level planner [24].

Learning efficiently from demonstration also requires strong communication between human and robotic coworkers. In human teams, communication is part of an information system that makes people aware about what are the activities of others. In case things are unclear, people naturally ask questions to clarify the situation. In the field of machine learning, clarifying questions has already been studied in active learning [17], [9], [19]. In this context both a human and a robot working together should be aware of what the other coworker knows and be able to improve each other's knowledge, creating common awareness. However, this has been little researched [3] in a context of LfD.

III. Assembly Task Learning from Demonstration

In this section we show how to learn an assembly task from demonstrations. Our learning from demonstration approach is shown in Fig. 1. The sequence of steps is the following: i) identify and detect the key events in a demonstration, i.e. the sequence of constraints that are made on objects; ii) identify an abstract representation of such sequences of events in a relational formulation; and iii) identify a high-level plan that fulfills the assembly task.

Many assembly tasks on rigid objects can be concisely described as sequences of actions that establish (or destroy) *kinematic constraints* between pairs of objects. For instance, the assembly of a chair is, at a symbolic level, the act of attaching four *leg* pieces, and a *back* piece, to a *seat* piece. The importance of extracting constraints from a set of demonstrations is manifold. Constraints represent symbolic transferable representations of tasks. Moreover, the motion of the agents and object pieces from before and until the creation of a constraint can be used to aid motion primitive learning for the geometric assembly plan. These geometrical constraints will then be converted in a set of symbols that can then be used to learn a high-level assembly plan.

A. Detections of Key Events

We make use of a linear-chain Conditional Random Field (CRF), a discriminative graphical model, paired with motion-based features to detect and extract the rigid constraints which arise between pairs of objects [6].

A linear-chain CRF models a log-linear distribution:

$$p(\boldsymbol{x} \mid \boldsymbol{y}) = \mathcal{Z}(\boldsymbol{y})^{-1} \prod_{t=1}^{T} \psi_t(\boldsymbol{x}, \boldsymbol{y}) \tag{1}$$

$$\psi_t(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(\phi_t(x_t, x_{t-1}, \boldsymbol{y})^\intercal \theta\right) , \tag{2}$$

where $\boldsymbol{x}$ is a sequence of latent binary variables, $\boldsymbol{y}$ is a sequence of observations, $\mathcal{Z}$ is the partition function, ϕ_t is a *feature vector* and θ is the model parameters vector.

In our application, the features are computed as various measures of variance which quantify the amount of individual and relative motion within a pre-defined window of time.

The model is applied separately to each individual pair of objects to infer the existence of a constraint between them over time. The model inputs (observations) are the trajectories of the objects during the demonstration, which are used to compute the mentioned features. The model's sequence of binary latent variables represents the existence of a rigid constraint between the pair.

The model is trained on labeled demonstrations which contain the creation of rigid constraints between objects. Because the model is applied to pairs, each demonstration contains a number of individual training sequences which grows quadratically with the number of objects; for this reason, single training demonstrations on objects composed of multiple parts have shown to suffice for training purposes.

During testing, the model is applied to new scenarios, generating *multiple* and potentially *overlapping* segmentations tied to the multiple and potentially overlapping actions that the human performs during the whole demonstration to accomplish the assembly task. The segmentation directly highlights the moments in time in which the constraints are being formed, while the actual constraints, each encoded as a translation vector and a quaternion, are extracted as the average transformation between the objects during the periods in time where the constraints are detected.

B. Constraint and Symbolic Learning

In order to learn and then execute the assembly plan, we propose the use of high-level reasoning relying on relational representations and leveraging new algorithmic developments. Before being able to learn abstract plans of a task, it is necessary to find an abstract representation of the context where those actions are executed. Since in this work we consider tasks as sequences of kinematic constraints, we can use a generic 3-ary relation $is\_constraint(O1, O2, C)$ where $O1$ and $O2$ are two objects and C is a constraint symbol. A constraint is modeled as the transformation between $O1$ and $O2$ frames.

If we have multiple demonstrations we can use a standard clustering algorithm (in our case k-means) to identify whether the constraints that are enforced in each demonstration are similar. To finally create symbols as 3-ary relations, we can attribute a symbol to each cluster. Using unsupervised learning allows to find the symbols that represent the relevant constraints for each task.

C. Plan Learning

From the symbolic relations found in the previous step, we can represent each demonstration as a sequence of states in a relational domain. We want now to learn which policy is able to fulfill such task in a robust way.

The high-level task is represented as a relational policy in a relational Markov Decision Process (MDP) framework. The state is defined as the set of active constraints. For each state the agent executes one of several possible relational actions $action(O1, O2, C)$. Executing this action will therefore apply the constraint C between $O1$ and $O2$.

The starting state s_0 is the empty set and is modified in the following way: if the constraint $is\_constraint(O1, O2, C)$ is present at time t then $s_{i+1} = s_t \cup \{is\_constraint(O1, O2, C)\}$ and $a_t = action(O1, O2, C)$. By this way we can learn a relational policy $\pi \in S \rightarrow A$.

We propose to use the TBRIL algorithm [21] to learn the policy. TBRIL is an algorithm that learns a policy close to the one demonstrated by minimizing a likelihood cost function with gradient boosting [12]. As a weak learner it uses TILDE [7], a relational version of classical greedy top down induction of decision tree algorithms.

Since we know the model of the environment dynamics we can then unroll the policy to produce an assembly plan.

IV. Iterative and Interactive Task Refining

The LfD framework *Task Learning* presented in Sec. III is able to create a task representation but it has three major shortcomings: i) it is not capable of error correction; ii) it does not communicate to the user what the system learned; and iii) it is not sufficient unto itself in the sense that describing the task does not ensure it is actually executable in any environment. Indeed, at runtime, new parameters come into play such as the workspace, the limits of the robot, the reachability of objects, or the presence of obstacles, that must be taken into account to create a feasible plan.

To improve it, we propose the *Task Refining* process based on a graphical user interface (GUI) able to show to the user, and allow the user to correct, the elements of the task that the system learned. These include: i) the constraints between objects; ii) the symbols learned; iii) the sequence of actions to be made; iv) a 3D simulation of the plan execution.

The user is able to interact with the system by providing corrections or further information and by visualizing what the system learned. Such a GUI allows to correct a wrong interpretation of the task but also enables knowledge creation from scratch and knowledge transfer. For this reason *Task Learning* is a bootstrap of *Task Refining* in the sense that the latter can also be run alone from no demonstration. The simulation step allows then to verify the corrected plan.

A. Feedback and Correction of Learned Data

1) Constraint representation and correction: Constraints between objects can be easily represented in a 3D simulation. Based on their visual representation, the user is able to correct or enrich them, such as improve precision or convert rigid constraints into rotational or prismatic constraints. If a constraint between two objects is not rigid then the extra degrees of freedom will help the robot's motion planner by allowing a larger set of goal positions. The GUI must provide visual and intuitive tools to help the user focusing on the wrong constraints and must invite the user to correct them naturally. Asking the user to concentrate on measures and to interpret their meaning would require a lot of effort on his part and would increase the risk of forgetting to correct a wrong information. In this sense visual cues and automatic animations decrease the effort to provide and increase the chance of converging quickly to a correct information. We do this by animating each constraint according to the uncertainty and degree of freedom obtained. Under our general perspective on robot instruction, in some cases it might be even easier to directly enter the numeric values of the constraints, possibility that we also allow in our interaction. An example of such a GUI is proposed in Fig. 2.

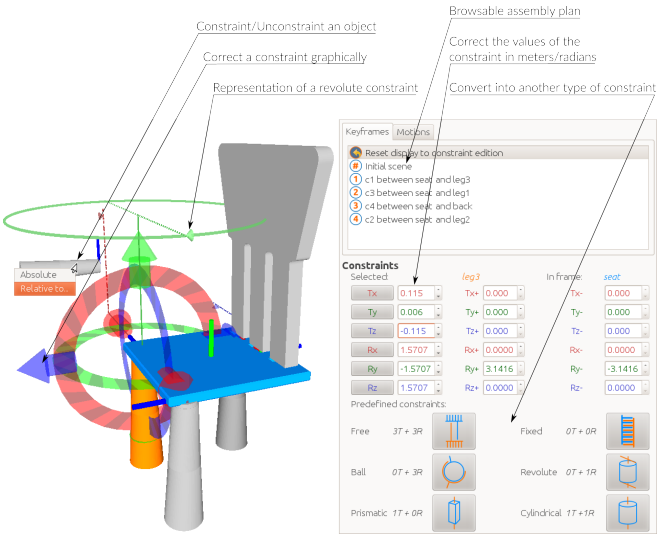

Fig. 2. Screenshot of a GUI showing constraints and assembly plan, able to invite the user to focus on the wrong data and allowing their correction

2) Assembly plan representation and correction: The assembly plan is a list of ordered actions applying constraints to objects. It can be visually represented as consecutive snapshots of all objects, constrained or unconstrained. Each snapshot is a view of the result of the execution of an action creating a new constraint between objects, like in Fig. 3.

The user is able to correct the plan by reordering the list or by changing the parameters of the action, especially by re-assigning the given constraint to another pair of objects. Indeed a plan could be correct at a high level, but could not be implemented at the current environment due to geometric constraints. By reordering the plan the user is able to help the system adapt to the current situation. The GUI showed in Fig. 2 allows these representation and changes.

B. Knowledge Transfer and Creation from Scratch

Since such a GUI allows to correct wrong constraints and plan, this can also be used in other ways to: i) program the task from scratch assuming that no demonstration has been provided – condition *cold start*; and ii) program the task using initial data coming from another task – condition *knowledge transfer*.

Indeed, by assuming that some knowledge has been learned for a first task e.g. *assembly of a chair*, it can be reused for a different but similar task e.g. *assembly of a bench* after small corrections. Since our approach does not make the system aware of the geometry of objects, the difference between a chair and a bench is provided by the user through the GUI.

C. Geometrical Plan Validation and Execution

From the data learned, created from scratch or transferred, and then corrected in the GUI, the user needs to be sure that the plan is actually executable on a robot. His plan is thus loaded into a simulated environment and the system tries to execute it.

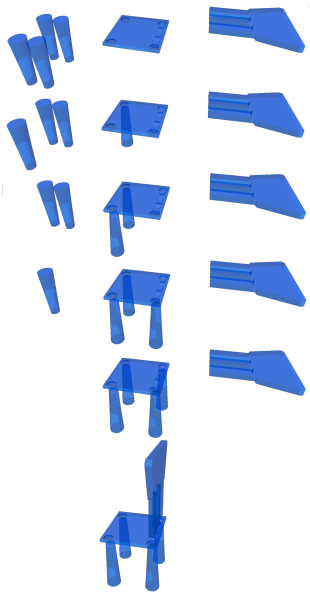

Fig. 3. Assembly plan of a chair shown to the user after learning the plan and the constraints. The user is able to browse the list of steps and visualize their corresponding snapshot

The approach presented in this paper represents an assembly task as consecutive pick-and-place operations that can be executed using a geometrical motion planner. Unlike pick-and-plug approaches we do not consider here the final insertion or screwing and focus on positioning and orienting the objects right before their attachment. Thus, execution amounts to applying all pick-and-place actions in the order of the assembly plan using the given constraints and objects. Right after picking an object the geometrical planner is given as input the initial pose p_{init} of the object in world coordinates, and the rigid constraint being the placing pose $p_{constraint}$ of this object in the frame of its parent. It outputs a collision-free motion going from the starting pose p_{init} to the goal pose $p_{constraint}$ corresponding to the actual assembly motion.

Moreover, if the constraint is non-rigid (revolute or prismatic), it can be interpreted as having multiple possible goals for the motion planner. Any of these goals is valid from the point of view of the task, but the planning of some of them could fail for some reason (goal requires an impossible Inverse Kinematics solution or is not reachable by avoiding collisions), in this case other goals are picked and tried until motion planning succeeds.

Therefore, this execution step allows to verify quickly that *Task Learning* and *Task Refining* processes led to an actually feasible program. If an error is detected (i.e. an object falls at some point or motion planning fails for any reason mentioned above with no other possible goal) the user is always able to correct again by continuing the *Task Refining* process.

V. User Study

To evaluate the proposed approach, we conducted a user study to show how our framework allows users to program a robot. We want to evaluate *Task Learning* and *Task Refining*,

how they allow to reduce the time and effort needed to program the task, and their impact on the final precision.

A. Experimental protocol

We recruited 14 subjects (aged 22 ± 3) from a technical background but most of them without great experience with real robots or CAD systems. We evaluate what is the impact of each process into the overall efficiency of the system from the high-level knowledge acquisition to its execution in a MoveIt[1] simulated environment of the Baxter robot. Four different exercises, labeled from A to D, are considered and they all consist into programming a robot to assemble either a chair or a bench. In all exercises the subject has to understand the environment around the robot and select a plan that avoids obstacles and workspace limits of the robot. The four exercises are listed below:

- A) **Chair cold start:** In this exercise the subject builds the chair by starting directly with the *Task Refining* process from no demonstration. He needs to create all constraints and plan from scratch using only the GUI. Our hypothesis is that this process is not intuitive, will take a longer time and require more effort from the user.
- B) **Chair bootstrap:** This exercise uses the *Task Learning* process to compute constraints and assembly plan from demonstrations and then bootstrap the *Task Refining* process. The subject is asked to perform corrections on the data learned from his own demonstrations in order to assemble the chair successfully in simulation. Our hypothesis is that the initial demonstration process is very intuitive and will give a good quality overall plan, but will not have enough precision nor consider the runtime environment to fulfill the task. So the second process of refining will give this extra advantage.
- C) **Bench cold start:** Similarly to the first exercise, this one consists into assembling the bench from no demonstration, from scratch through the GUI only.
- D) **Bench bootstrap:** This exercise illustrates knowledge transfer from the chair to the bench. The subject is asked to assemble the bench using the plan of the chair that he has already corrected. We preload the plan corresponding to the most precise constraints between exercises A or B. The system automatically transfers the constraints of the legs and ignores the one associated to the back since the bench has no such object. Thus the only corrections that the user needs to provide correspond to the difference between a chair and a bench in terms of geometrical constraints, that cannot be computed automatically. Our hypothesis is that using the demonstrations of a different object allows nevertheless to save time and effort compared to a cold start.

Pictures illustrate the overall process in Fig. 4 and the protocol in Fig. 5. The subject starts by recording 3 demonstrations of the chair assembly acquired with an Optitrack tracking system [2].

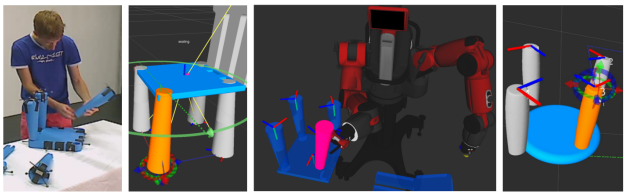

Fig. 4. Data recording, correction step, and execution step (video: http://vimeo.com/120726868); as well as transfer of constraints to another object (video: http://vimeo.com/109165300)

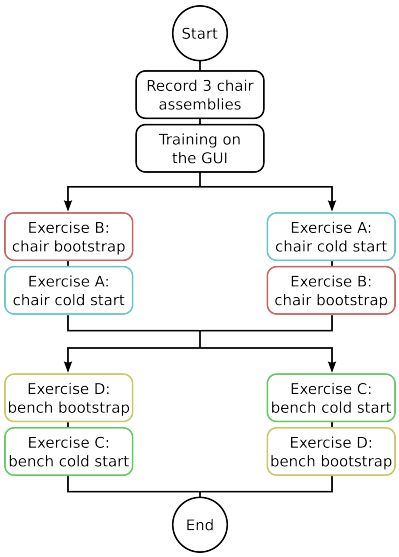

Fig. 5. Experimental protocol completed by 14 subjects. After recording 3 demonstrations and being trained to the GUI, subjects are assigned randomly to one of the two conditions A then B or B then A. Once both are completed they are assigned randomly one of the last two conditions C then D or D then C.

The second step of the protocol consists into training the subject to the GUI during 20 minutes with synthetic example constraints and plans. These data contain all possible cases to train into all features of the GUI: create and delete a constraint, reorder the plan, correct precision, and also get familiar with 3D navigation ensured with a 6D space mouse.

Then the subject performs randomly either exercises A and then B or B and then A. Both exercises must lead to a successful assembly of the chair in simulation. In case of failure, the user performs new corrections and retries as many times as needed. Finally the subject performs randomly either exercises C and then D or D and then C, both exercises leading to a successful assembly of the bench.

B. Task representation quality

We are interested in the quality of the plan and symbols learned by the *Task Learning* process and their quality after correction during the *Task Refining* process. The results are presented in Table I. The quality is estimated with the chair, thanks to the mean number of created symbols, the mean

[1]http://moveit.ros.org/

[2]Repository of recorded data: http://github.com/3rdHand-project/Data

TABLE I

PLAN AND SYMBOLS QUALITY USING THE MEAN NUMBER OF SYMBOLS CREATED, THE MEAN NUMBER OF ACTIONS CREATED FROM THESE IN THE PLAN, THE MEAN NUMBER OF CONSTRAINTS ASSOCIATED TO WRONG OBJECTS, AND THE MEAN POSE ERROR OF CONSTRAINTS

	Symbols learning			Plan learning	
	# of symbols	position error	orientation error	wrong association	# of actions
After *Task Learning*	4.28	25 mm	10.2 deg	0.29	3.5
After *Task Refining*	5	8.8 mm	4.3 deg	0	5

numerical error of symbols in position and orientation, the mean number of constraints associated to wrong objects (*e.g.* applying to the back a symbol corresponding to a leg) and the mean number of actions created in the plan. Since the chair is composed of 6 objects the processes could end up to a maximum of 5 constraints, the seat being the root and so unconstrained. Thus a perfect system would create 5 symbols resulting into 5 actions in the plan, and would never associate symbols to wrong objects. The output of *Task Learning* creates in average 4.28 symbols. All symbols are not necessarily used to create an action in the final plan mainly because of false negatives during the constraint detection phase. That is why the plan is composed of an average of 3.5 actions only, with almost no wrong association. After correction all subjects have correctly constrained the 5 objects by creating or correcting 5 symbols leading to 5 actions.

Table I also shows after each process the error of symbols (in position and rotation) compared to reference constraints defined beforehand. The position error is the cartesian distance to the reference and the orientation error is the angle θ with the reference. This angle between two quaternions $q1$ and $q2$ is defined as $\theta = arccos(2{<}q1, q2{>}^2 - 1)$. This reference is also used by the execution system to simulate object fall. Beyond the tolerance of 15 mm and 20 degrees an object is considered as *fallen* and the assembly fails. We observe that automatic constraint extraction creates an average error of 25 mm in position and 10.2 degrees in rotation, that is more than the tolerance and would lead to a fall without correction. However the user provides correction reducing error up to 8.8 mm and 4.3 degrees.

To sum up, the *Task Learning* process creates a representation of the task that has not quality enough to execute it. The *Task Refining* process allows to improve the plan and its overall precision, making it successfully executable.

We will now compare the effort necessary to provide accurately all information necessary to fulfill the task, i.e. constraints and plan. For this we use two metrics: the time and the number of clicks necessary to complete the task. The results are shown in Fig. 6. We used the Wilcoxon T test to analyze the results.

Results are significantly better both in terms of time, *p-value* $= 0.001$, and number of clicks, *p-value* $= 0.001$, in combining *Task Learning* and *Task Refining* (median time: 6.3 min, median clicks: 81) rather than programming the task from a cold start (median time: 12.9 min, median clicks: 153). It should be noted however that the time to demonstrate is not accounted here. Depending of the time spent to record

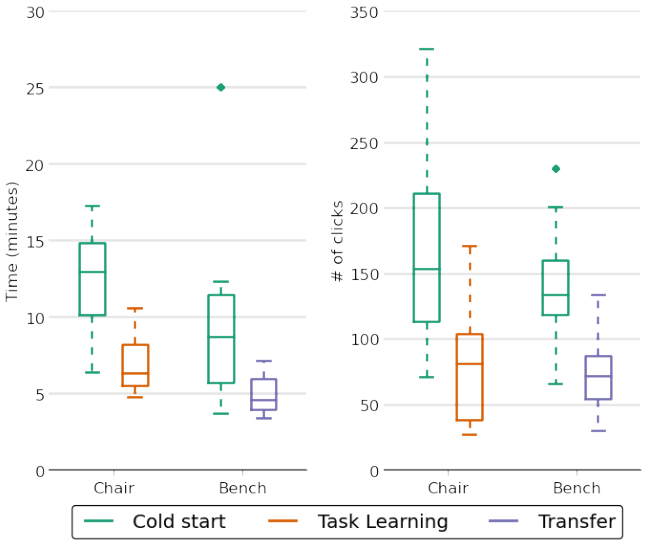

Fig. 6. Time and effort (number of clicks) necessary to complete the tasks from cold start, by bootstrapping from demonstrations, and by transferring knowledge

demonstrations the advantage of using *Task Learning* could decrease. In our case the mean duration of a demonstration over all subjects is 1.03 min.

C. Impact of Transfer

For the bench the bootstrap corresponds to knowledge transfer from the corrected chair. In this case also, we get significantly better results both in terms of time, *p-value* $= 0.004$, and number of clicks, *p-value* $= 0.001$, in combining both processes (median time: 4.6 min, median clicks: 72) rather than programming the task from a cold start (median time: 8.7 min, median clicks: 134). This result is made possible by the use of a high level relational representation of the task.

We conclude that it is more efficient to have knowledge to bootstrap from, even in the case where this knowledge does not correspond exactly to the task we intend to achieve but is inherited from a similar task.

VI. DISCUSSIONS

We proposed a new approach to instruct robots that combines the complementary advantages of demonstrations, feedback and knowledge transfer. Under this approach there are two processes of learning. The first process, *Task Learning* is used to bootstrap the task programming with demonstrations that are very intuitive and fast to provide by

non-expert users. The second process, *Task Refining* gives feedback of what has been learned to the user and allows the user to correct wrong information with reduced effort thanks to a GUI. The GUI gives a feedback of what the robot has learned from the task, allows to correct these data, to transfer them from another object, to create the task from scratch using no demonstration, and also to verify them by executing the task in a 3D simulation. It handles the assembly plan and the constraints between objects, with a 3D representation for ease of visualization and their equivalent values for precision.

Our approach is tailored for general assembly tasks and so we represent object assemblies in the form of sequences of actions that create constraints between pairs of objects. Our method to learn from demonstration considers automatic detection of the constraint creation, decomposition of subparts of the demonstration, and learning of hierarchical assembly plans. We believe that no such process can be perfect nor directly executable in any environment so we rely on a second process of knowledge refining. Refining consists into an iterative process alternating corrections and executions converging to a more precise plan thanks to user inputs.

We evaluated our approach with a user study where 14 subjects completed 4 consecutive assemblies of a chair and a bench. Our system extracts assembly plans and constraints from the demonstrations. We measured the time and effort that subjects needed to provide before being able to execute assemblies successfully. Finally, the results of our user study validated the interest of combining both processes and emphasizes the importance of allowing the user to be aware of the knowledge acquired by the robot to reduce the overall programming time and effort and to increase precision.

We conclude that the combination of both processes is more efficient that any of them in isolation. This effect is true even when the system learns from a different but similar object, by performing knowledge transfer. Demonstrations enable to acquire a fast but imprecise representation of the task, while a more constrained interaction enables to increase such precision and ensures that the task could be executed whatever the robot and the environment is.

Although the end precision is small, open loop execution does not suffice and assembly tasks would require visual servoing to complete on a real robot. Another improvement of this work would also consider all subparts of the demonstrations that do not end up to a new constraint but are key motions in the assembly, for instance returning the seat before assembling the back or moving objects closer to the robot. This could decrease significantly the risk of failures by creating new intermediary motions able to finish the assembly without user intervention. Another enhancement would extend demonstrations to human-human cooperation to assemble more complex objects. Analyzing several people assembling a complex object together could lead to a formalism representing concurrent, sequential or dependent actions. Finally, we noticed that considering object shapes instead of assimilating objects to a single frame would help to also detect degrees of freedom and thus non-rigid constraints (prismatic, rotational) instead of relying on user inputs on this point.

REFERENCES

[1] B. Akgun, M. Cakmak, K. Jiang, and A. Thomaz. Keyframe-based learning from demonstration. *International Journal of Social Robotics*, 4(4), 2012.

[2] B. Akgun, K. Jiang, M. Cakmak, and A. Thomaz. Learning tasks and skills together from a human teacher. In *AAAI*, 2011.

[3] S. Alexandrova, M. Cakmak, K. Hsiao, and L. Takayama. Robot programming by demonstration with interactive action visualizations. In *RSS*, 2014.

[4] S. Alexandrova, Z. Tatlock, and M. Cakmak. Roboflow: A flow-based visual programming language for mobile manipulation tasks. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 2015.

[5] B. Argall, S. Chernova, and M. Veloso. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5), 2009.

[6] A. Baisero, Y. Mollard, M. Lopes, M. Toussaint, and I. Lütkebohle. Temporal segmentation of pair-wise interaction phases in sequential manipulation demonstrations. In *IROS*, 2015.

[7] H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artificial intelligence*, 101(1), 1998.

[8] M. Cakmak and L. Takayama. Teaching people how to teach robots: The effect of instructional materials and dialog design. In *HRI*, 2014.

[9] S. Chernova and M. Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34, 2009.

[10] H. Christensen, T. Batzinger, K. Bekris, K. Bohringer, J. Bordogna, G. Bradski, O. Brock, J. Burnstein, T. Fuhlbrigge, R. Eastman, et al. A roadmap for us robotics: from internet to robotics. *Computing Community Consortium*, 2009.

[11] S. Forge and C. Blackman. A helping hand for europe. Technical report, European Commission, Joint Research Centre, Institute Technological Studies, 2010.

[12] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 29(5), 2001.

[13] J. Grizou, M. Lopes, and P.-Y. Oudeyer. Robot learning simultaneously a task and how to interpret human instructions. In *ICDL*, 2013.

[14] D. H. Grollman and A. Billard. Donut as i do: Learning from failed demonstrations. In *ICRA*, 2011.

[15] J. Kober and J. Peters. Learning motor primitives for robotics. In *Robotics and Automation, 2009. ICRA'09. IEEE Inter. Conf. on*, 2009.

[16] J. Kulick, M. Toussaint, T. Lang, and M. Lopes. Active learning for teaching a robot grounded relational symbols. In *IJCAI*, 2013.

[17] M. Lopes, F. S. Melo, and L. Montesano. Active learning for reward estimation in inverse reinforcement learning. In *ECML/PKDD*, 2009.

[18] J. Manyika, M. Chui, J. Bughin, R. Dobbs, P. Bisson, and A. Marrs. *Disruptive technologies: Advances that will transform life, business, and the global economy*, volume 180. McKinsey Global Institute San Francisco, CA, 2013.

[19] D. Martinez, G. Alenya, P. Jimenez, C. Torras, J. Rossmann, N. Wantia, E. E. Aksoy, S. Haller, and J. Piater. Active learning of manipulation sequences. In *ICRA*, 2014.

[20] M. Mason and M. Lopes. Robot self-initiative and personalization by learning through repeated interactions. In *HRI*, 2011.

[21] S. Natarajan, S. Joshi, P. Tadepalli, K. Kersting, and J. Shavlik. Imitation learning in relational domains: A functional-gradient boosting approach. In *IJCAI*, 2011.

[22] S. Niekum and S. Chitta. Incremental semantically grounded learning from demonstration. In *RSS*, 2013.

[23] S. Niekum and S. Osentoski. Learning and generalization of complex tasks from unstructured demonstrations. In *IROS*, 2012.

[24] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal. Learning and generalization of motor skills by learning from demonstration. In *ICRA*, 2009.

[25] M. R. Pedersen and V. Krüger. Gesture-based extraction of robot skill parameters for intuitive robot programming. *Journal of Intelligent and Robotic Systems*, 2015.

[26] K. Ramirez-Amaro, M. Beetz, and G. Cheng. Automatic segmentation and recognition of human activities from observation based on semantic reasoning. In *IROS*, 2014.

[27] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6), 1999.