Teaching Robots Parametrized Executable Plans Through Spoken Interaction

Guglielmo Gemignani[*]
Sapienza University of Rome
gemignani@dis.uniroma1.it

Emanuele Bastianelli[*]
University of Rome
Tor Vergata
bastianelli@ing.uniroma2.it

Daniele Nardi
Sapienza University of Rome
nardi@dis.uniroma1.it

ABSTRACT

While operating in domestic environments, robots will necessarily face difficulties not envisioned by their developers at programming time. Moreover, the tasks to be performed by a robot will often have to be specialized and/or adapted to the needs of specific users and specific environments. Hence, learning how to operate by interacting with the user seems a key enabling feature to support the introduction of robots in everyday environments.

In this paper we contribute a novel approach for learning, through the interaction with the user, task descriptions that are defined as a combination of primitive actions. The proposed approach makes a significant step forward by making task descriptions parametric with respect to domain specific semantic categories. Moreover, by mapping the task representation into a task representation language, we are able to express complex execution paradigms and to revise the learned tasks in a high-level fashion. The approach is evaluated in multiple practical applications with a service robot.

Categories and Subject Descriptors

I.2.9 [**Artificial Intelligence**]: Robotics

General Terms

Algorithms

Keywords

Human-robot interaction, communication and teamwork; Robot planning and plan execution.

1. INTRODUCTION

While operating in domestic scenarios, robots will necessarily encounter situations not envisioned by their developers at programming time. Often, they will therefore require to be specialized and/or adapted to the needs of specific users and to specific environments. In these environments, learning how to operate interacting with non-technical users seems a key enabling feature to support the introduction of robots in our everyday lives.

To this end, multiple authors have proposed to rely on natural language interaction and dialogue to enable robots to learn how to accomplish complex and tasks. There are, however, various challenges that still need to be tackled for instructing robots using natural language. Robust speech recognition, mapping the given commands into robot behaviors, or resolving the ambiguities arising from natural language are among the problems to be faced to build a reliable system.

In this paper we attempt to tackle some of such challenges by presenting a novel approach for learning, through the interaction with the user, task descriptions that are defined as a combination of primitive actions. We use a grammar-based approach to first acquire the description of the action, later converted into a plan readily executable by the robot. The action learnt by the robot in this way can involve sequences of actions, conditional branches and iterations that can be characterized by multiple open parameters specified at run-time. The two main contributions of our paper are the following:

- A novel approach for learning the syntactic structure of the taught actions, represented as parametric tasks that can be instantiated at run-time;
- A new language representation, called Task Description Language, mapped to the Petri Net Plans (PNP) formalism [11]. This mapping allows us to give a clear execution semantics and to express complex execution paradigms, such as parallel actions.

In addition, we show how to revise the tasks learned, allowing the user to refer to their composing actions in a high-level fashion.

In the remainder of the paper, we first give a brief overview of the related work, underlining the major differences with respect to other techniques proposed in literature. We then describe in detail the proposed approach and how it has been deployed on a service robot. After presenting a set of tests designed to validate our method, we then conclude the paper with a discussion of the presented work and with some hints on possible future work.

2. RELATED WORK

The problem of teaching a robot new tasks through Natural Language Interaction is a new research topic that has emerged in the last decade. Usually, the tasks to be learnt are represented as a composition of primitive behaviors that the user can use as background knowledge to teach the robot. The sequence of such actions is often represented in a graph-like structure that is encoded in a specifically designed language. For example, in [5] and [2], the authors use a language called Robot Control Language (RCL) and compound action specifications, respectively, for representing and executing route instructions, parsed from natural language commands. These works, although focusing on understanding and executing the description of a command from the natural language interaction with

[*]The first two authors have contributed equally to this paper

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

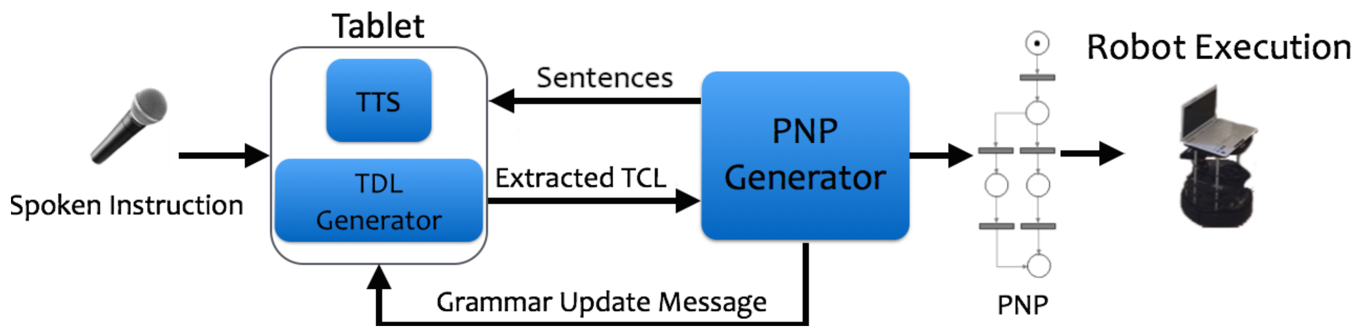


Figure 1: Interaction scheme designed for processing verbal instruction and for generating executable plans.

a user, do not analyze the problem of learning new actions to be used in later stage.

Such a problem is instead faced by [7], where the authors present a first and simple method for learning by from demonstrations of high level tasks, which are built from a pre-existing behavior knowledge. In this work, the learning approach is divided in two main phases: first, the user gives a demonstration of the action to be learnt, while in a second phase the robot has the opportunity to refine the acquired capabilities, by practicing for a small number of trials under the teacher's supervision.

The problem of teaching new tasks to a robot through natural language is also analyzed in [8], where the authors introduce a method for teaching tasks in the form of directed acyclic graphs, composed of available action primitives. In their work, the task is verbally described and interpreted through a grammar-free Automatic Speech Recognition (ASR). As in previous works, the task to be learnt is first demonstrated by the user. Next, the robot has the possibility of engaging the human in a spoken language dialog to query any unspecified effects of conditional alternatives. Differently from previous works, the authors focus the learning process on natural language interaction, also allowing the user to teach behaviors that involve conditional branches. A similar contribution is also given by [10], where the problem of teaching soccer skills to a team of robots via spoken language is addressed. In this latter approach, the authors design a predetermined set of actions and queries that are associated with a predefined grammar. This grammar specifies all the possible natural language commands understood by the robot. The proposed vocabulary includes a set of soccer behaviors (e.g., shoot and pass), and if-then-else control expressions. An interesting aspect of this framework is the possibility of directly querying the robot for its internal state. However, the expressive power of this work is limited, not being able to handle loops and not allowing the user to correct previously taught actions.

These two contributions have been presented in [6], where the authors describe an enhanced approach that allows the user to teach actions involving loops. Their speech interface is based on the Google free-form ASR and they represent actions as Instruction Graphs. This system also enables the user to correct an existing behavior by letting the robot step through the sequence of actions of a chosen task. This revision approach works really well for simple behaviours, but may become cumbersome as task complexity increases.

The above-cited approaches make significant steps forward in extending the language used for task specification. However, they do not address the problem of teaching a robot parametric tasks. It has been shown, by strong linguistic theories based on cognitive studies [4], that when humans talk about an action, they refer to a general structure representing its underlying concept, which is characterized by a set of arguments participating in it. Therefore, when teaching a new task to a robot, in order to make the learning process more intuitive, some authors have suggested to refer to general action structures, instead of teaching instances of more general concepts.

For example, in [3], a simple approach for learning parametric actions is presented. By exploiting the Microsoft ASR Engine the authors show how a robot can be taught how to poke a general object. Open parameters are expressed by using chosen keywords and they are specified at run-time. However, the proposed method is limited to a sequence of actions that do not include branches or loops. Moreover, the actions learnt can be characterized only by a single open parameter, leaving the open question of how to generalize the method to multiple parameters.

A final example tackling the problem of learning parametric tasks is presented in [9]. In this work, the authors describe a three-tier representation that supports both the conversion of natural language into robot actions and the application of existing planning algorithms. However, this framework does not allow to represent conditionals and loops and it does not enable the user to revise previously taught tasks.

Differently from these works, our approach allows a user to teach or specify to a robot new possibly parametrized actions by combining primitive behaviors. Our system makes a significant step forward by allowing the user to refer to multiple parameters during the teaching process, allowing for a rich task specification language, inspired by the Robot Control Language [5]. The Task Description Language, in fact, differently from RCL, allows to represent also conditionals, parallel actions as well as variables. Moreover, by associating execution plans in PNP, we rely on an execution semantics that let us capture more expressive task specifications (e.g., action parallelization). This double representation of the tasks allows us to decouple the problem of capturing the wide variety of linguistic expressions uttered by a user from the problem of defining a clear operational semantics. Finally, we present an innovative approach for task revision that allows the user to refer to the primitive actions that constitute a task in a high-level fashion.

3. APPROACH

In this work we propose an approach to dynamically teach a robot new complex task descriptions through user interaction. With the term *complex* we refer to tasks that are compositions of predefined robotic *action primitives* (i.e., atomic behaviors that are not further decomposable in terms of simple actions). Once learnt, a complex task will instantiate a specific plan and, for this reason, in the rest of the paper, we will refer to complex learned actions either as *tasks* or *plans*, while their composing actions will be called *primitives* or *primitive actions*.

Unlike previous works, in our approach the task learned by a robot take as input arguments that are not statically associated to a specific instance, e.g. *the red book*, but they are filled with possible semantic categories, e.g. *object*. For example, a possible implementation of a *bringing* task could be represented by the general concept of *bringing* an *object* to a *location*. In each task, the arguments can assume different values every time a command is given to the robot, e.g. "*bring the book to the office*".

Our learning process is based on a spoken dialog with a user that incrementally explains to the robot how the task is composed in terms of primitives. To this end, the Task Description Language (TDL) has been designed in order to give a structure to the task descriptions, enabling for the use of different patterns of execution of the involved primitives, e.g. conditional branches or while loops. The interaction with the user is mainly based on spoken inputs, however the proposed approach also allows for the use of examples to drive the learning process (e.g., it is possible to tell the robot *"go to the location for example the office"* to check if the robot correctly executes the command). Moreover, we also investigate the possibility of using dialogs to update a previously learned task by removing, inserting and replacing some of its parts with new TDL constructs. Generally, our interactive and action learning approach takes place in three consecutive phases: Processing verbal instructions; Generating executable plans; Updating plans. Figure 1 reports the interaction scheme of the first two processes that we designed.

3.1 Processing verbal instructions

The first step in learning new actions through user interaction consists of acquiring a natural language description of the command and its interpretation, associated with a structured representation. Specifically, taking inspiration from the Robotic Control Language proposed in [5], we designed a Task Description Language (TDL), to create a first representation of the tasks to be learnt.

TDL is a high-level representation for complex procedure specifications, described using natural language. TDL acts as a bridge between the instructions as expressed by humans and the final plan of the robot, and it is specified by the grammar reported in Table 1. In TDL, tasks are described as a composition of procedural components that can be primitive actions or structural elements describing different execution patterns, that embed other TDL constructs in a recursive way. Starting from the capabilities of our robotic platforms, we first define a set of primitive actions so that every complex TDL structure is expressed as a function of them. Below, we list the primitives we defined to test our approach:

- `goTo`: the action of going from a point to another point in the space. It takes one parameter: the destination.
- `follow`: the action of following someone or something in front of the robot, in an open-loop fashion.
- `takePicture`: the action of taking a snapshot of the scene, acquired through some visual sensing device. It outputs the name of the picture taken, which can be stored in a variable (`@picture`) to be recalled in the dialogue by the user.
- `say`: tells the robot to call the Text-to-Speech service to synthesize a string into voice. Its parameter is the sentence to be said.
- `sendEmail`: the action of sending an email to an address with a content. Its two arguments are: the content of the mail and the address where to send it.
- `pickUp`: the action of picking up an object. It takes the object to be picked up as a parameter.
- `drop`: the action of releasing an object.

Additionally, we defined two conditions to be checked in the `if-then-else` and `do-until` constructs:

- `in`: to check whether the robot is located in a particular location or not.
- `isPerceived`: to verify whether an object is perceived by the robot or not. If the *anyone* parameter is passed as an argument, a person detector functionality is activated.

The list of parameters (see Table 1) passed to the primitives can either hold variables, denoted with an @ character followed by a string, or instances, represented by strings. Such lists are used to instantiate the arguments of the primitive actions, and are bounded to the parameters of the task being described. This feature is fundamental for teaching the robot general concepts of actions, where arguments are only referenced as possible semantic categories and not as specific instances of them.

For example, consider the case of teaching a robot the general concept of the *bringing* action. We can assume that, for a particular instance, this action will require two arguments: an *object* to be brought and a *location* where the *object* must be brought. This general conceptual structure of the action is instantiated at the beginning of the learning interaction, after the user tells the robot a phrase like "*I'll teach you how to bring an object to a location*". Consequently, two variables are created (i.e. `@object` and `@location`) for the current task. During the explanation, the user can refer to these variables as arguments of prompted primitive actions. Let us assume that the task corresponding to the *bringing* action is composed by the sequence of four primitives: a `goTo` action needed to reach the object to be brought, followed by a `pickUp` action, then another `goTo` action to reach the final location and, finally, a `drop` action to release the object. When referring to a primitive that requires one or more arguments, it is possible to bind them with the task variables, e.g. `goTo:[@location]`. The final TDL structure for this particular instance of the *bringing* action therefore will be:

```
( do − sequentially
        ( goTo : [@object] )
        ( pickUp : [@object] )
        ( goTo : [@location] )
        ( drop : [@object] ) ).
```

In order to acquire such a description of the action we propose an interaction scheme that is based on a spoken dialog with a user that prompts step by step the elements composing the final TDL. The process is structured in three steps:

- Automatic Speech Recognition to translate the user vocal input into text.
- Interpretation, where the transcribed input is translated into a TDL construct.
- Decision about how to proceed in the dialog flow (e.g. asking for confirmation about the learned action or possible clarification about the input received).

The ASR phase is realized using a grammar-based engine that allows us to implement specific grammars for describing the TDL. The grammars drive the recognition process: by attaching a proper semantic output to each grammar rule, we obtain a representation of the linguistic semantics of a recognized utterance, which is later

Table 1: The constructs defined in Task Description Language.

	TDL Form	Example
Condition	*<Condition>:[Parameter List]*	*(isPerceived:[door])*
Primitive Action	*<Action>:[Parameter List]* *<Variable>=<Action>:[Parameter List]*	*(goTo:[office])* *(@picture=takePicture:[])*
Sequence of Actions	*do-sequentially <TDL$_1$ > ... <TDL$_n$ >*	*(do-sequentially (goTo:[office]) (say:[hello]))*
Conditional	*If <Condition> then <TDL$_1$ > else <TDL$_2$ >*	*(If in:[Office] then (say:[hello]) else (say:[I am lost]))*
Counting Loop	*do-n-times <TDL> <Integer>*	*(do-n-times (turn:[around]) 10)*
Do-until Loop	*do <TDL> until <Condition>*	*(do (follow:[@person]) until (in:[Office]))*

translated to a specific TDL. This representation is based on the *frame* concept inspired by the notion defined in the *Frame Semantics* linguistic theory [4]. The general meaning expressed by each frame can be enriched by semantic arguments that are part of the sentence and provide additional meaning to the action. As an example, the command *"go to the office"* will be mapped to the MOTION frame, while the sub-phrase *"to the office"* will fill the specific frame element GOAL representing the destination of the MOTION. The representation will then be translated in the following TDL structure:

$$(\texttt{goTo} : [\texttt{office}]).$$

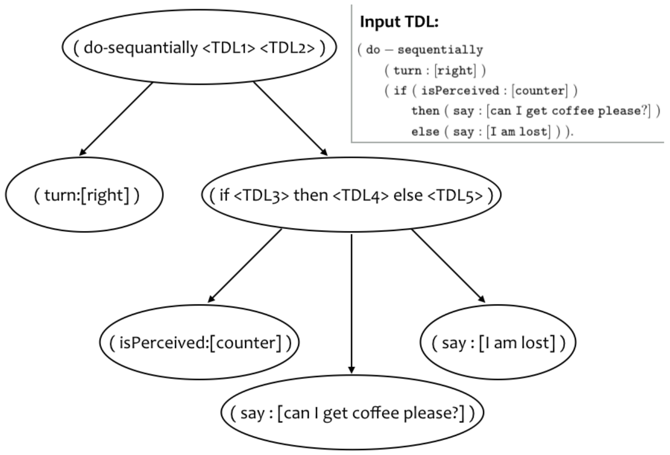
The adopted ASR provides also for the possibility of using special grammar rules that trigger a free-form dictation ASR engine. We use this feature to recognize the names of new tasks that are not predefined in the grammar, as well as messages to be uttered when performing the `say` primitive.

The process of learning new tasks does not only consist of converting spoken instructions to TDL and consequently instantiating a new plan. In fact, the acquisition of knowledge about new tasks also involves the learning of new linguistic forms in order to verbally recall them. To this end, after a new plan has been correctly built, the ASR grammars are augmented with new syntactic-semantic structures corresponding to the learned task, e.g. *bring the @object to the @location* for the task "bring the object to the location".

3.2 Generating Executable Plans

Once the spoken description of the task given by the user has been converted in its corresponding TDL structure, it is processed to obtain an executable plan described with the Petri Net Plans (PNP) formalism [11]. PNP, is a plan-representation framework based on Petri Nets, which includes a rich set of features suitable for expressing executable actions: non-instantaneous actions, sensing and conditional actions, action failures, concurrent actions, interrupts, and action synchronization in a multi-agent context are among the aspects that can be described by such a formalism. This versatility allows us to represent and precisely characterize within PNP not only the multiple details of the plan uttered by the user, but also every action and dialog between the user and the robot. Moreover, with this additional representation we are able to decouple the problem of understanding user utterances from the issue of executing a task on the robot.

Starting from the TDL structure representing the plan extracted from the utterances of the user, in order to obtain an executable plan, a tree structure is created by recursively decomposing it in its elementary parts. Such tree is composed by nodes labeled with the TDL forms contained in the input TDL and by edges representing the constituency relation (see Figure 2). Once the tree has been generated, starting from its leaves, each TDL construct is replaced by the corresponding PNP structure. Figure 3 shows the PNP structure


(do-sequantially <TDL1> <TDL2>)
(turn:[right])
(if <TDL3> then <TDL4> else <TDL5>)
(isPerceived:[counter])
(say : [I am lost])
(say : [can I get coffee please?])
Input TDL:

```
( do − sequentially
    ( turn : [right] )
    ( if ( isPerceived : [counter] )
        then ( say : [can I get coffee please?] )
        else ( say : [I am lost] ) ).
```

Figure 2: Tree structure created from an example input TDL.

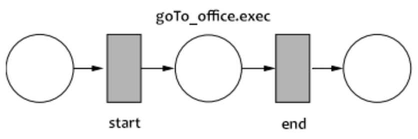
corresponding to each TDL constructs shown in Figure1. By querying a KB that holds the information about the primitive actions (i.e., their name and their input and output semantic categories), each component of the PNP structures is then renamed to form a correct executable plan. In particular, when dealing with a parametric primitive action, the system is able to query the KB in order to bind the variable used in the TDL structure with the variable used in the primitive action. By converting the TDL structure into a specific PNP, an exact mapping between natural language and executable PNPs is obtained, thus providing a suitable execution semantics.

Once the PNP representation of the action has been created, an update message is sent to the speech recognition module and to the KB to flag that the new plan has been learnt. Specifically, this message is used to store in the KB the number of variables used by the learnt plan, their name and the associated TDL structure needed during the update phase, as described in the following section.

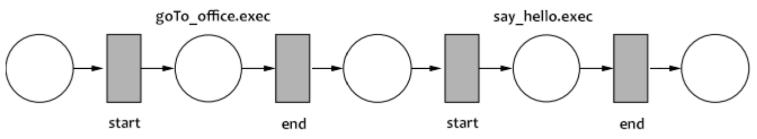
3.3 Updating Plans

A feature of our approach consists of allowing the user to correct parts of the taught plan as desired. The work by [6] is the only example of plan revision in a task teaching framework: the authors propose a task correction mode, during which the robot describes the action sequence and asks, at each step, whether the user wants to replace a primitive action with another one. We propose a way for referring to each primitive action composing the newly learnt plan in a high-level fashion, allowing the user to add replace or delete a particular action in a task.

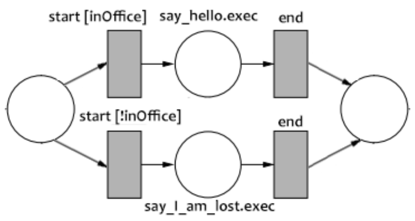
To update a previously taught task, the user can vocally select it and subsequently refer to its primitive actions. By naming the action that needs to be modified, the desired operation, and the optional parameters needed, the plan is modified accordingly. For



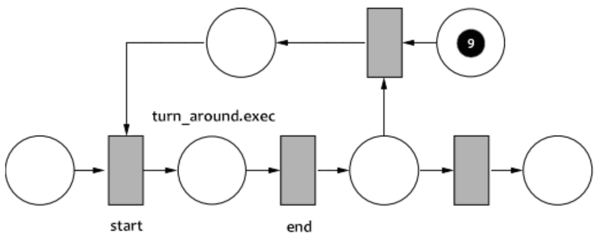
(a) Primitive Action



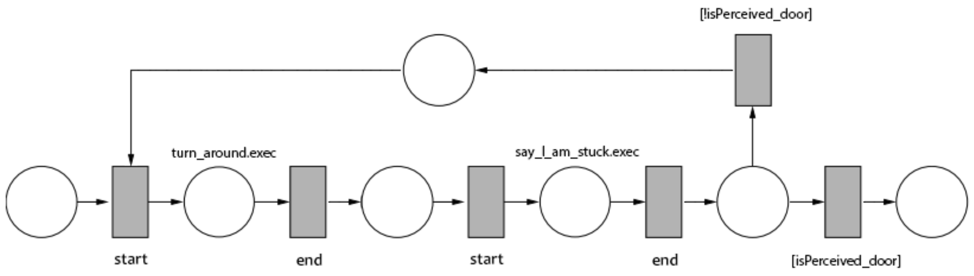
(b) Sequence of Actions



(c) Conditional



(d) Counting Loop



(e) Do-until Loop

Figure 3: Examples of PNP structures for each TDL construct.

example, assuming that the user needs to replace a `say` action with a `goTo` action, this can be achieved by telling the robot: *"I want you to perform the go to location action instead of the say action"*. When multiple actions of the same kind are instead present in a plan to be modified, the user can distinguish them by referring to their parameters (e.g., *"the go to location"*) and/or by specifying the ordinal number with which they appear in the TDL structure (e.g., *"the second go to"*). Once the updating command has been acquired by the robot, the old plan TDL structure is retrieved from the KB. If the command univocally identifies an operation in the plan (i.e., there are not any ambiguities in the given command) the old TDL structure is modified accordingly, otherwise a clarification dialogue is started by the robot. Finally, the new TDL is translated into its corresponding PNP to enable its execution on the robot.

4. SYSTEM DEPLOYMENT

Our goal in this section is to demonstrate the effectiveness of the approach, showing that our teaching methodology can be adopted to learn high-level parametric plans, also allowing to represent multiple execution paradigms expressed using natural language. To this end, we have deployed our system on a robotic platform to show the advantage of using an intermediate representation, namely TDL, together with the PNP formalism to model the learning, revision and execution processes. To ground the semantic categories expressed in TDL, a semantic map built with the aid of the user has also been given to the robot [1]. The proposed approach has been implemented on a Videre Design platform equipped with a Hokuyo laser range finder and a Kinect. For localization and navigation, standard software packages of the ROS framework have been adopted. Finally, for the ASR engine, we used the Microsoft Speech Platform running on an external tablet with a push-to-talk microphone, used to implement the connected speech interface. Figure 4 shows an image of the robot interacting with a user.

4.1 Interaction Example

In order to better understand the process of learning a complex plan, we report a video showing an example interaction that guides the teaching process of the *bringing* an *object* to a *location* plan[1].

H: *I'll teach you how to perform the "bring an object to a location" plan*.
R: *Ok, tell me how to perform this action*.
H: *First, go near the object, for example "the DVD"*.
R: [moves near the DVD]
R: *did I perform the right action?*.
H: *Yes*.
R: *Should I do anything else afterwards?*.
H: *Yes, pick up the object*.
R: *Ok, can you put the DVD on the tray?*
R: [waits for the user]
U: [confirms through a dialog box]

... [The action is added to the task and the interaction continues]

H: *bring the book to the office*
R: [performs the new instance of the action]

The interaction starts with a sentence that instantiates the struc-

[1]`https://vid.me/HXnG`

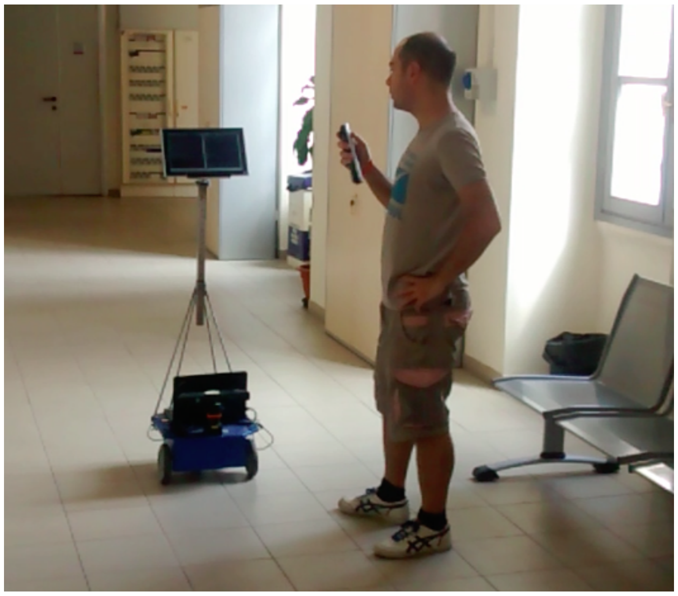


Figure 4: The Videre Design platform used in the experiments interacting with a user.

ture of the plan, defining its general name (e.g. *bring*), and its arguments, (e.g. *object* and *location*). Afterwords, the explanation phase begins: the use of predefined keywords enables for recalling constructs of TDL associated with them. For example, the word "*first*" at the beginning of the second sentence uttered by the human specifies to the system that the prompted primitive is part of a sequence. The resulting interpretation of this statement will be a `do-sequentially` construct, where the first action to be performed is a `goTo:[@object]`:

$$(\texttt{do} - \texttt{sequentially} \quad (\texttt{goTo} : [@\texttt{object}])).$$

In this example, an instance of the semantic category object, e.g. the *DVD*, is provided so that the robot can show the user that the action he prompted has been correctly understood. The action is then executed using the example as input argument. Note that instead of using predefined keywords to point out the end of a TDL construct, e.g. *endif*, we rely on a question-answer modality to determine its scope boundaries. In this example, the question "*Should I do anything else afterwards?*" is used to understand if the next instruction belongs or not to the current TDL construct, i.e. the `do-sequentially`. The interaction continues with the second action. This time the user refers only to the semantic category, but because of the binding between variables and instances provided in the previous example, the robot asks directly the user to put the *DVD* on the tray, as the `@object` has been locally instantiated before. Doing so, the `pickUp` primitive is simulated and the learning process continues by inserting it in the current TDL construct, e.g.

$$(\texttt{do} - \texttt{sequentially} \quad (\texttt{goTo} : [@\texttt{object}]) \quad (\texttt{pickUp} : [@\texttt{object}])).$$

It is important to underline that the use of examples while explaining a task is not mandatory. In fact, it is possible to refer only to the semantic categories involved in the task, without using direct instances and skipping the demonstration about the action to perform. Using or not using examples during the teaching phase does not affect the final plan instantiation.

At the end of the interaction, the final TDL structure is processed yielding the PNP plan corresponding to it. Accordingly, the language of the robot is augmented by inserting in the ASR grammar the semantic-syntactic structure corresponding to the command "*bring an @object to a @location*". Now it is possible to call the plan for any location and any object defined in the ASR grammar.

5. SYSTEM VALIDATION

In order to validate the system, three different kinds of analysis have been carried out. First, we taught our robot five different tasks. During this teaching phase we measured the accuracy of the system as well as the time needed for teaching each task. We then counted the number of non parametric tasks that should have been taught to act on a particular set of instances, comparing it with the number of parametric tasks needed to act on the same set of targets. For example, following an unparameterized approach, in order to learn the *bring an object to a location* task valid for every object and location in the environment, one should have taught the robot a single task for each combination of instances of *object* and *location*. Next, we replicated the tests performed on different mobile bases by the authors of related works. This test aimed at verifying the expressive power of our approach with respect to the others found in literature. Finally, since we adopted a formalism able to represent a wide variety of execution paradigms, we attempted to show how other kinds of tasks could be learned by the robot. In particular, we successfully managed to teach the robot a task involving parallel actions. This is a first example of the several possible extensions that are achievable by creating new TDLs and providing them with a suitable execution semantics using PNP.

5.1 Expressivity and Effectiveness Evaluation

The aim of our work is to create an interactive task teaching framework, based on a natural language interaction with the user; however, we do not evaluate our approach through a user study, since usability is not the purpose of this paper. Rather, we make an assessment of the expressivity and effectiveness of our approach. In particular, to understand the impact of learning parametric actions and to measure the efficiency of our approach, we first taught our robot five basic tasks, each involving one or two parameters. The five tasks we selected for the evaluation are just illustrative examples of what our system can learn, and they are reported in the following list:

- *enter @location*: this action allows to add another possible linguistic reference for the `goTo:[@location]`.
- *take an @object*: This task is represented by `goTo:[@object]` followed by `pickUp:[@object]`. As the `pickUp` primitive, this task requires the robot to be near the target object.
- *bring an @object to a @location*: (see above).
- *give @person an @object*: this task represents the action of bringing an object that the robot already carries to a person, assuming that the position of the person is known.
- *check an @object for a @person*: this task consists of going in front of an object, taking a snapshot of it and sending an email with the picture to the person.

In order to show the overall behavior of the system while learning this set of tasks, we asked 5 users not familiar with robots to test the teaching framework. Each user was first given a brief explanation of the task teaching framework and the semantic map and

Table 2: Results obtained for the evaluation of the proposed task teaching framework.

Task	Min	# Err	# Mis	# Cor	AT No Err	AT w Err
enter @location	4	0.2	0.4	0	29	38
take an @object	5	0.6	0	0	41	52
bring an @object to a @location	7	0.2	0.6	0.6	57	74
give @person an @object	6	1	0.4	1	49	79
check an @object for a @person	5	0.6	0	0	55	69

primitives available to the robot. Next, the user was asked to teach the robot each of the previously described actions in a random order. For each task we measured the following quantities:

- The minimum number of instructions required to teach a specific task (**Min**).
- The average number of instructions not recognized by the automatic speech recognition (**Err**). In this measure, only the instructions that the ASR could not process were considered.
- The average number of instructions misrecognized (**Mis**) by the natural language understanding.
- The average number of corrections needed to modify a wrongly learned task (**Cor**).
- The average time in seconds needed to teach a specific task when no errors or misrecognitions were encountered (**AT no Err**).
- The average time in seconds needed to teach a specific task when errors or misrecognitions were encountered (**AT w Err**).

Table 2 reports the numerical results obtained for the analysis. Notice that the time measured refers only to the interaction with the user, as we avoided examples requiring the action of the robot.

A first fact that can be noticed from the table is the cost, in terms of time, of a correction process. When one or more corrections have been required, the average time needed to correctly teach a task increases of $16.2s$ on average. Additionally, it can be noticed that the longer a task is, the higher is the probability of having a misrecognition. This expected result is in fact underlined by the highest number of misrecognitions obtained for the two longest tasks (e.g., *bring an @object to a @location* and *give @person an @object*). For these particular tasks we also obtained the highest difference between **AV no Err** and **AV w Err**, respectively of $17s$ and $30s$. The effects of a misrecognition can be rather different: in the starting phase of a procedure it may not require a correction (e.g. for the *enter* task), while in other cases one or more corrections might be needed (e.g for the *bring* and *give* tasks).

Finally, in order to give an idea of the teaching steps saved with this parametric approach, we counted the number of non parametrized tasks that should have been taught to the robot in order to cover the same exact set of possible instantiated tasks. This particular analysis has been carried out considering the scenario presented in [1], built through the direct interaction with a user. Such an environment consists of 23 objects and 10 persons, located in 10 different rooms. In this specific environment, in order to obtain the equivalent of the five parametric tasks, we should have taught the robot 723 single tasks by combining all the instances of the involved semantic categories for each task. The gain in terms of number of interaction is straightforward, especially if we consider the possibility of dynamically extending the number of objects, or in general the knowledge about the environment.

5.2 Comparison with Related Works

A second validation of the learning ability of our system has been carried out by acquiring the same tasks that systems presented in other related works learned. The *dinner is ready* task in [8] is used to test the presented approach. Such a task is composed by a sequence of `goTo` and `say` primitives, interleaved with some conditional branches regarding the presence of specific persons in the environment. We taught our robot the exact same task, resulting in the following TDL:

```
( do − sequentially
    ( goTo : [dining_room] )
    ( if ( isPerceived : [jeremy] )
        then ( say[“Jeremy set the table”] )
        else ( say[“cannot find Jeremy”]) )
    ( goTo : [living_room] )
    ( if ( isPerceived : [kevin] )
        then ( say[“Kevin come to dinner”] )
        else ( say[“cannot find Kevin”] ) )
    ( goTo : [bedroom] )
    ( say[“turn off the television”] )
    ( goTo : [living_room] )
    ( say : [”task complete”] ) ).
```

The average training time for this particular task was 185 seconds. Even though the environment we used for reproducing the task was different, our system was able to learn the same sequence of actions learnt by the other robot. Such a result demonstrates the learning ability of the system, independently from the actual scenario.

In a similar way, we managed to interactively teach the *get coffee* task presented in [6] to our robot. In this case, we had to slightly adapt the task learnt as the set of primitives provided by the CoBot platform, was different from ours. Indeed, due to the lack of specific primitives, we exploited the *do-until* loop to surrogate the tasks that required quantified parameters, e.g. *move forward 5.4 meters*, terminating the loop with a specific perception condition. Instead of referring to generic numbered landmarks, instead, we used objects on the map to trigger some perception checks. The average training time for this task was 92 seconds. The following TDL reports the result obtained:

```
( do − sequentially
    ( do ( getCloser : [] )
    until ( isPerceived : [door] ) )
    ( turn : [right] )
    ( if ( isPerceived : [counter] )
        then ( say : [can I get coffee please?] )
        else ( say : [I am lost] ) ).
```

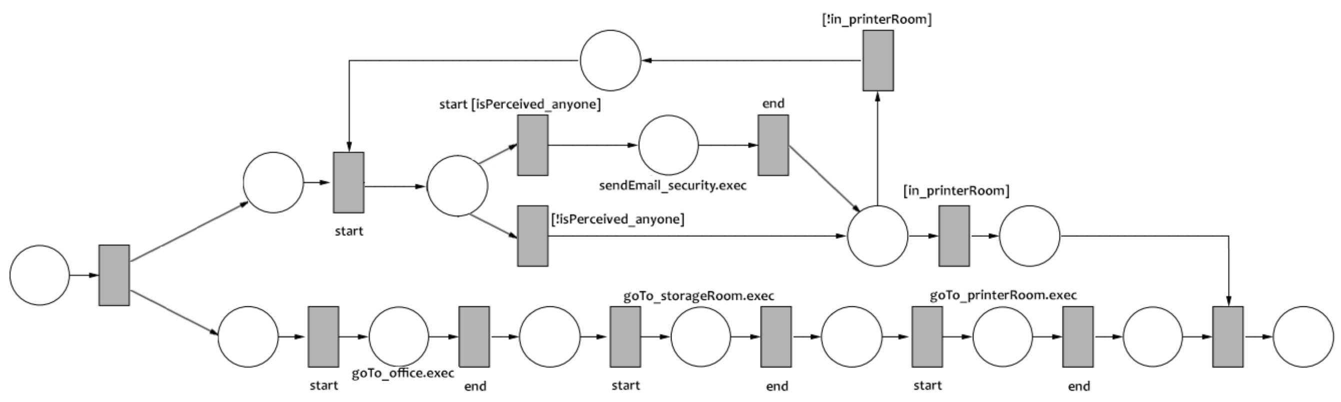


Figure 5: The resulting PNP for the patrolling task.

5.3 Teaching a More Complex Task

As a third and final validation, we used TDL to capture an additional execution paradigm already available in the PNP formalism. To this end, we instructed our robot to execute a complex task involving primitive actions to be performed in parallel. This pattern has been realized using an ad-hoc TDL construct. We therefore designed a non parametric task to *patrol* a wing of our department. Such a task consists of visiting three different rooms, while continuously checking for the presence of persons. If a human is detected, the robot sends an email sent to the security staff. The loop is ended when the robot reaches the final destination. The TDL structure representing the task is the following:

```
( do
    ( do − sequentially
        ( goTo : [office] )
        ( goTo : [storage_room] )
        ( goTo : [printer_room] ) )
 and
    ( do
        ( if ( isPerceived : [anyone] )
            then ( sendEmail : [security] )
            else ( continue ) )
      until ( in : [printer_room] ) ) ).
```

The resulting PNP is instead shown in Figure 5. According to this specific definition of the patrolling task, the two branches are performed in parallel, by exploiting one of the key features of the PNP formalism.

6. CONCLUSION AND FUTURE WORK

In this paper we presented a novel approach for learning, through the natural interaction with the user, executable plans described as a combination of primitive actions. Specifically, our system uses a grammar-based approach to first acquire the description of the action, later converted into an executable Petri Net Plan, passing through an intermediate representation expressed in a specifically developed language called Task Description Language. The proposed approach makes a significant step forward by making task descriptions parametric with respect to domain specific semantic categories. Moreover, by mapping the task representation into a plan representation language, we have been able to express complex execution paradigms and to revise the learnt plans in a high-level fashion. The teaching framework has been deployed on a Videre design robot, showing the advantage of using an intermediate representation, namely TDL, together with the PNP formalism to model the learning, revision and execution processes. With a double task representation we have shown how it is possible to decouple the problem of capturing the wide variety of linguistic expressions uttered by the user and the problem of defining a clear operational semantics.

For future works, a number of additional features need to be addressed. We are first investigating the issues that arise during the possible failure of an action. We would like in fact to enable the user to teach the robot how to recover from a specific action failure. This can achieved by exploiting the PNP formalism that allows to implement actions interrupts based on particular conditions. Moreover, we are investigating the possibility of specifying optional parameters during the teaching phase when using examples. Instead of just telling the robot if it performed the correct action or not, the user could indeed exploit this particular phase for specifying optional parameters specific to the taught plan (e.g., by saying *"Yes but stop a little bit closer"* for the `goTo` action). Finally, we are planning to deploy and monitor our robot in our department for an extended period of time, by letting it freely interact with the users present in the environment.

REFERENCES

[1] E. Bastianelli, D. D. Bloisi, R. Capobianco, F. Cossu, G. Gemignani, L. Iocchi, and D. Nardi. On-line semantic mapping. In *Proceedings of the 16th International Conference on Advanced Robotics*, 2013.

[2] D. L. Chen and R. J. Mooney. Learning to interpret natural language navigation instructions from observations. In *AAAI*, 2011.

[3] J. Connell, E. Marcheret, S. Pankanti, M. Kudoh, and R. Nishiyama. An extensible language interface for robot manipulation. In *Artificial General Intelligence*. 2012.

[4] C. J. Fillmore. Frames and the semantics of understanding. *Quaderni di Semantica*, 1985.

[5] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox. Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, 2013.

[6] Ç. Meriçli, S. D. Klee, J. Paparian, and M. Veloso. An interactive approach for situated task specification through

verbal instructions. In *Proceedings of the 13th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2014.

[7] M. N. Nicolescu and M. J. Matarić. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*, 2003.

[8] P. E. Rybski, J. Stolarz, K. Yoon, and M. Veloso. Using dialog and human observations to dictate tasks to a learning robot assistant. *Intelligent Service Robotics*, 2008.

[9] L. She, S. Yang, Y. Cheng, Y. Jia, J. Y. Chai, and N. Xi. Back to the blocks world: Learning new actions through situated human-robot dialogue. In *15th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2014.

[10] A. Weitzenfeld, C. Ramos, and P. F. Dominey. Coaching robots to play soccer via spoken-language. In *RoboCup 2008: Robot Soccer World Cup XII*. 2009.

[11] V. A. Ziparo, L. Iocchi, P. U. Lima, D. Nardi, and P. F. Palamara. Petri net plans. *Proceedings of the 10th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2011.