Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Handling Complex Commands as Service Robot Task Requests

Vittorio Perera and Manuela Veloso
Carnegie Mellon University
Pittsburgh, PA
{vdperera,mmv}@cs.cmu.edu

Abstract

We contribute a novel approach to understand, dialogue, plan, and execute complex sentences to command a mobile service robot. We define a *complex command* as a natural language sentence consisting of sensing-based conditionals, conjunctions, and disjunctions. We introduce a flexible template-based algorithm to extract such structure from the parse tree of the sentence. As the complexity of the command increases, extracting the right structure using the template-based algorithm decreases becomes more problematic. We introduce two different dialogue approaches that enable the user to confirm or correct the extracted command structure. We present how the structure used to represent complex commands can be directly used for planning and execution by the service robot. We show results on a corpus of 100 complex commands.

1 Introduction

As robots are moving out of factories and labs into our everyday life, humans need to be able to specify complex tasks in an intuitive and flexible way. In this paper, we enable users to give complex instructions to a robot using natural language.

Simple instructions refer to simple commands, such as "Please, take this book to the lab." Natural language however enables the specification of requests in a more elaborate manner. The user may request a robot to perform a set or a sequence of tasks, give options to the robot or ask to perform a task, only if certain conditions are met. We view such elaborate natural language as complex commands. In order to handle these types of complexity, we introduce a flexible template-based approach able to break a complex command into atomic components and connectors.

Due to the complexity of natural language, the approach we propose is, inevitably, not always able to correctly resolve a complex command in its atomic components. Therefore we design two dialogue systems for the user to refine and correct the extracted command structure, guided by the robot.

We then show how breaking a complex command into its atomic components can be leveraged to improve the task execution by the robot. By rearranging the order in which each atomic task is executed, while at the same time enforcing the constraints imposed by the structure of the sentence, our algorithm can substantially reduce the distance traveled by the robot to execute complex commands.

In order to evaluate our template-based algorithm, we gathered a corpus of 100 complex commands. The approach proposed is able to correctly break 72% of the commands in their atomic components. The two dialogue approaches proposed are either able to recover the correct structure of a complex command by asking a number of questions linear in the number of atomic tasks involved in the original command; or able to improve the accuracy by asking for rephrased commands. Finally, we evaluate the robot execution planning algorithm on a set of 150 randomly generated complex commands. We compare our approach to a base line and show that we can consistently improve on it.

2 Related Work

In one of the earliest works on natural language understanding, SHRDLU [Winograd, 1971], natural language instructions were processed and executed in a virtual environment. Following up on SHRDLU, many researchers tried to extend its capabilities to real-world systems, and soon started tackling natural language processing also for robotic systems.

We can now find many examples of robots executing natural language instructions for a variety of tasks, such as manipulation of small objects [Zuo *et al.*, 2010], mapping of unknown environments [Ghidary *et al.*, 2002; Kruijff *et al.*, 2007], or following navigation instructions [MacMahon *et al.*, 2006; Marge and Rudnicky, 2010; Tellex *et al.*, 2011].

In this work, we focus on a mobile service robot executing tasks in an office-like environment for prolonged periods of time, similarly to the work of [Biswas and Veloso, 2013]. The robot tasks are represented using semantic frames [Fillmore, 1985]. Semantic frames have been extensively applied both to linguistics [Baker *et al.*, 1998] and robotics [Kollar *et al.*, 2013; Thomas and Jenkins, 2012].

Complex commands have been considered as a combination of LSNL (Limited Segments of Natural Language) also for a service robot [Chen *et al.*, 2010]. We build upon this work and move forward by reducing the limitations and increasing the flexibility on the language used by the user.

3 Complex Commands

We represent the tasks a robot can execute with *semantic frames*. We define a separate frame for each task with its own set of frame elements. As an example, consider a robot that can execute two tasks only: navigate to a room in the building and deliver objects. These two tasks can be represented with the following frames: *GoTo* and *Delivery*. These frames have, respectively, the following frame elements: {*Destination*} and {*Object, Destination*}.

When users give commands to a robot, they are asking the robot to perform one of its tasks. Therefore semantic frames can also be used to represent the command given to a robot. When the command refers to a single frame and each frame element is uniquely instantiated, we call it an *atomic command*. An example of an atomic command is: "Please robot, go to the lab." This command refers to a single frame, *GoTo*, and its only frame element, *Destination*, is instantiated as "the lab" and therefore we consider it an atomic command.

When a command is not atomic, we call it a *complex command*. We identify four different types of complexity that can arise in a command, as we now introduce.

Set of tasks: The user may ask the robot to perform a set of tasks, for which the command refers to multiple frames.

Example 1. "Go to the lab and bring these papers to my office."

With this command, the user is asking the robot to perform two tasks, *GoTo* and *Deliver*.

Disjunctive task elements: The command might refer to a single frame but some of the frame elements are not univocally instantiated.

Example 2. "Bring me some coffee or tea."

This command refers to the *Delivery* frame but the *Object* can be instantiated both as "tea" or "coffee".

Explicit sequence of tasks: The user may ask the robot to perform an ordered sequence of tasks. Users can refer to a sequence of tasks explicitly in their command, as in:

Example 3. "Go to the lab and then to my office."

Conditional sequence of tasks: The user may ask the robot to perform an ordered sequence of tasks, while using conditionals. For example:

Example 4. "Bring me some coffee if it's freshly brewed."

Assuming that we have a frame to represent the action of checking if coffee is freshly brewed, this command then refers to a sequence of two tasks, where the second might or might not be executed depending on the outcome of the first one.

Our goal is to represent a complex command with a set of atomic commands. In order to preserve the original meaning of the command, we use four operators to connect the atomic commands extracted from the original command. The operators are **AND**, **OR**, **THEN**, **IF**. Each of these operators corresponds to one of the types of complexity just introduced:

- **AND** is used for commands referring to a set of tasks, therefore the command in Example 1 becomes:

  [Go to the lab] **AND** [Bring these papers to my office]

- **OR** is used when an element of the frame can be instantiated in multiple ways. Example 2 would then become:

  [Bring me some coffee] **OR** [Bring me some tea]

- **THEN** orders tasks in a sequence. Accordingly, Example 3 can be rewritten as:

  [Go to the lab] **OR** [Go to my office]

- **IF** is used for the sequence of tasks involving conditionals. Example 4 then becomes:

  [Coffee is freshly brewed] **IF** [Bring me some coffee]

For the **IF** operator, as in this last example, the condition is always moved to the beginning of the sequence, as the robot needs to check it before proceeding.

Finally, in order to measure the complexity of each command, we use the number of atomic commands it contains. Accordingly an atomic command has a complexity level of one and all the examples given in this section have a complexity level of two.

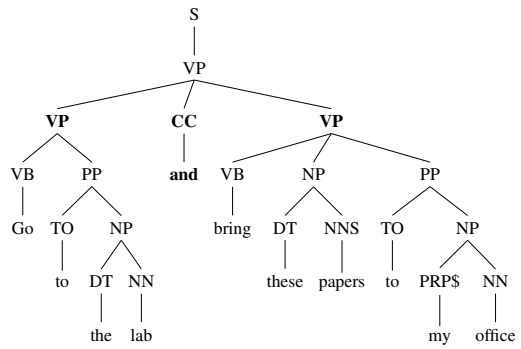
4 Detecting Complex Commands

In the previous section, we have identified the different types of complexity a command can present. In Section 4.1, we present a template-based algorithm to break complex commands in their atomic components and, in Section 4.2, we show the results on a corpus of 100 commands.

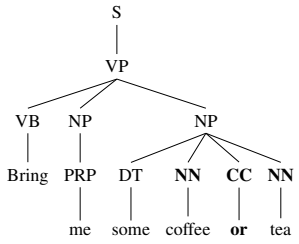
4.1 A Template-Based Algorithm

In order to detect complex commands and break them down in their atomic components, we leverage the syntactic structure of the sentences. We define a *template* as a specific structure in the parse tree of the command. We identify one or more templates for each of the operators defined. Each *template* not only defines the structure associated with a specific operator, but also the rules to break the complex command into its components. Figures 1(a), 1(b), 1(c), and 1(d) show the templates found in the four examples presented in Section 3. Highlighted in boldface are the templates used to break the complex commands down into atomic commands.

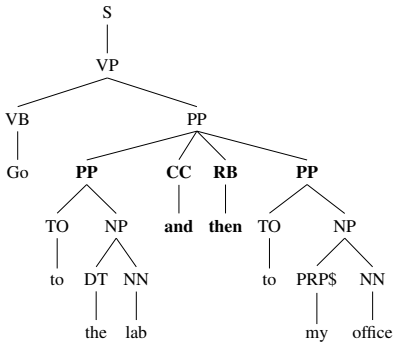
Our approach is to first parse the received command and then inspect the parse tree for the defined templates. In all the given examples, each complex command is composed by only two atomic commands. However, this is not always the case. Therefore, we propose to break down a command into simpler components and then recursively check each of them until we get atomic commands. Algorithm 1 shows the details of the proposed approach. The `DECOMPOSE` function takes as input a command s and parses it. The loop in line 3 checks, in breadth-first fashion, if any sub-tree of the parse tree matches any of the structures defined templates and, if so, calls the `BREAK_SENTENCE`.



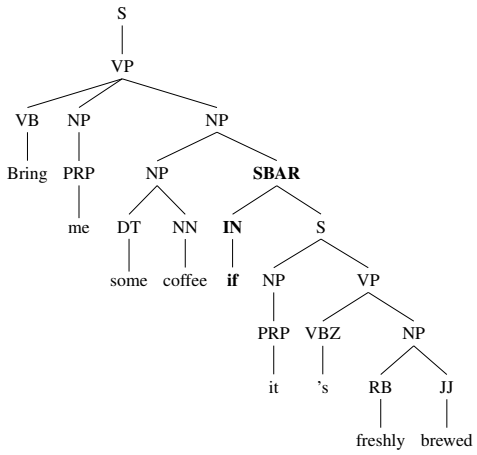
(a) [Go to the lab] **AND** [Bring these papers to my office]



(b) [Bring me some cofee] **OR** [Bring me some tea]



(c) [Go to the lab] **THEN** [Go to my office]



(d) [Coffee is freshly brewed] **IF** [Bring me some coffee]

Figure 1: Examples parse trees, and corresponding textual parenthetic representations. Used templates are in boldface.

The BREAK_SENTENCE function, in line 5, takes as input the command s and the matching template, applies the rule specified by the template to decompose the input command, and returns three values: two simpler commands, called left-hand (LH) and right-hand (RH), and the operator. Finally, in lines 6 and 7, the function is called recursively on the two simpler commands extracted.

Algorithm 1

```
1:  function DECOMPOSE(s)
2:      p = parse(s)
3:      for node in p do
4:          if node == template then
5:              LH,O,RH, = BREAK_SENTENCE(s, template)
6:              L = DECOMPOSE(LH)
7:              R = DECOMPOSE(RH)
8:              return [L O R]
9:          end if
10:     end for
11:     return s
12: end function
```

Now consider a complex command such as "If the door is open go to the lab and to my office" and let us look on how the DECOMPOSE function operates on it. Once the sentence is parsed the function finds a template for the **IF** operator and breaks it into "the door is open" as the left-hand, and "go to the lab and to my office" as the right-hand. Next the function is called recursively. The left-hand is a simple commands so the function returns the sentence as it is. For the right-hand the function, after parsing, finds a second template matching the **AND** template. The function breaks the sentence and the new left-hand and right hand are, respectively, "go to the lab" and "go to my office". These are returned to the initial call of the function that can now end and returns:

[the door is open **IF** [go to the lab **AND** go to my office]]

4.2 Evaluation

We gathered a corpus of 100 complex commands by asking 10 users to give each 10 commands. (This corpus is available at *http://www.cs.cmu.edu/∼vdperera/corpora/*) The users, graduate students at our institution, had different backgrounds ranging form math and stats to computer science and robotics. The exact instructions given to the users were:

> The commands can contain conjunctions (i.e., "Go to the lab and then to Jane's office"), disjunctions (i.e.,"Can you bring me coffee or tea?") and conditionals (i.e., "If Jane is in her office tell her I'll be late"). A single sentence can be as complex as you want, for instance you can have conjunctions inside of a conditional (i.e., "If Jane is in her office and she's not in a meeting tell her I'm on my way"). Please while the sentences can be as complex as you want, we are looking for sentences that you would realistically give to the robot (both in length and content).

Figure 2 shows the number of commands for each level of complexity, and Figure 3 shows, for each complexity level, one example of the sentences contained in the corpus.

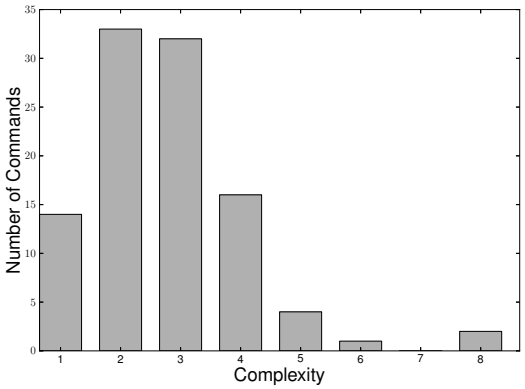


Figure 2: Number of commands per complexity level.

1	Bring some coffee to my office please.
2	If you have no tasks scheduled, go explore the 5th floor.
3	Go to the supply room and, if you find a stapler, bring it to me.
4	Please go to the lab and if the door is closed go to John's office, and ask him to send me the memory stick.
5	I need you to first bring me a cup of tea, or a bottle of water, or a soda, and then go to Chris office and ask her to order more bottled water.
6	If Jane is in her office, ask her when she wants to go to lunch, go to Chris office and tell him her reply, then come back here and tell Jane's reply to me.
8	If Christina is in her office, pick up a package from her, deliver it to Jane, then go to the lab and say that the package has been delivered. Otherwise, go to the lab and say that the package has not been delivered.

Figure 3: Examples of commands in the corpus for each complexity level.

While most of the commands have a complexity level between one and three, people also use more complex instructions, and occasionally long and convoluted sentences.

To measure the accuracy of our approach, we broke each command in its atomic component manually and compared the extracted structure with the result returned by our algorithm. The overall accuracy of the algorithm was 72%. Figure 4 shows the percentage of matching commands, for each complexity level.

As to be expected, our approach does not have any problem with atomic commands, out of 14 only one is not understood correctly. The only sentence not correctly understood is: "The 3rd floor lab has taken our gaffer tape and screw driver. Please bring it to the 7th floor lab." In this sentence, the algorithm incorrectly recognizes the template for an **AND** operator even if the user is asking only for one of the tools.

For commands of complexity two and three, the algorithm is able to correctly decompose 76.9% of the complex commands. As the complexity increases, the accuracy decreases, but so does the corresponding number of commands. The main reason we identify for the lower accuracy on more complex commands is the lack of appropriate templates. As an example consider the following sentence: "If the person in

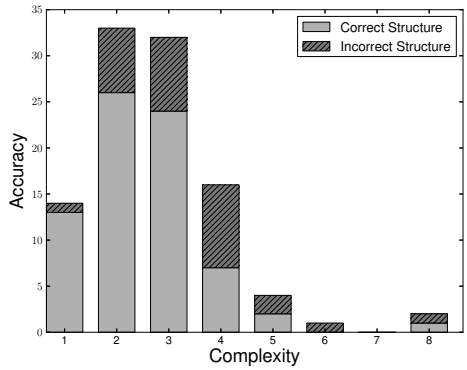


Figure 4: Complex commands correctly decomposed for each complexity level.

Office X is there, could you escort me to his/her office if you're not too busy?". This sentence can be represented as [person in office X is there **AND** not too busy] **IF** [escort me to his/her office]. Our templates allow conditions of an **IF** operator to be at the beginning or at the end of the command but not in both. Therefore our current set of templates were not able to properly break down this type of command, while they successfully covered a wide range of other commands. Inevitably, templates will not be able to cover all the variations of language. Our approach is to combine the template-based automation with dialogue.

5 Dialogue

After processing our corpus using Algorithm 1, we are still left with few complex commands that we are not able to understand correctly. Nonetheless, in order to execute a complex command correctly, the robot needs to recover the correct structure representing it. In this section we introduce two models for a dialogue that allow to recover the structure of a complex command.

5.1 A Structure-Based Dialogue

The first dialogue model we introduce aims at recovering the structure of a complex command. When receiving a command, the robot executes Algorithm 1 and offers the user the extracted structure, as its corresponding textual parenthetic representation (see Figures 1(a)-1(d)). If the offered structure correctly represents the command broken in its atomic components, the user can confirm and the robot starts executing the command. Otherwise the robot enters in a dialogue with the user to get the correct structure. Algorithm 2 shows the details of the structure-based dialogue.

First the algorithm checks if the command is simple or complex. If it is a simple command, the robot asks for confirmation for it and then executes it. If it is a complex command, the robot needs to recover all of its components. In the dialogue, the robot first asks for the operator and then for left and right-hand commands. Since both the left-hand and the right-hand commands can be themselves complex commands the dialogue will recur on each of them.

Algorithm 2

```
1: function ST_DIALOGUE( )
2:     cmpl = ASK_COMPLEX( )
3:     if cmpl then:
4:         o = ASK_OPERATOR( )
5:         rh = ST_DIALOGUE( )
6:         lh = ST_DIALOGUE( )
7:         return [rh, o, lh]
8:     else
9:         return ASK_SIMPLE( )
10:    end if
11: end function
```

The ASK_COMPLEX function asks the user if the command is complex or not and saves the answer in the boolean variable cmpl. If cmpl is false, the robot, using the function ASK_SIMPLE, asks for a simple command. Otherwise, if cmpl is true, the robot enters in the recursive step of the dialogue and asks for a connector and two simpler commands. This dialogue generates, in the worst case, a total of $2n - 1$ questions to recover a command of complexity n.

5.2 A Rephrasing Dialogue

Instead of asking for the correct structure of the command, our second dialogue model asks for a rephrased command.

Similarly to the structure-based dialogue, the rephrasing dialogue asks the user for confirmation on the correctness of the structure extracted, using a textual parenthetic notation. If the user finds the structure incorrect, the dialogue algorithm requests the command to be rephrased using one of the known templates, by giving short examples of the four complex command operators.

To respond to this rephrasing dialogue approach, we rephrased the sentences with an incorrect structure. We achieved an improvement from 72% to 88% in accuracy. Figure 5 shows the results for each complexity level.

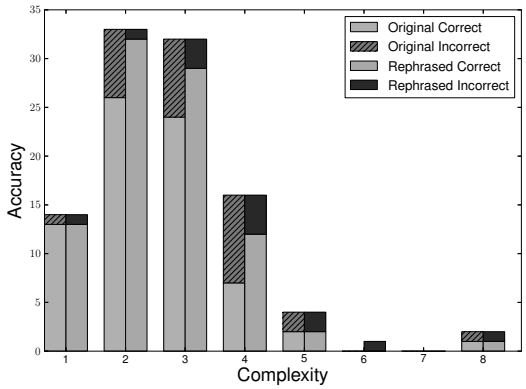


Figure 5: Complex commands correctly decomposed for the original command and the rephrased one.

As we can see using a rephrasing dialogue improves the accuracy but, still does not cover all the sentences. If the rephrasing dialogue is not able to extract the correct structure, it can be followed by a second, structure-based dialogue to ensure all the commands are correctly understood.

6 Execution

Before being able to execute a complex command, the robot needs to ground it. To ground a command, for a service robot, means to extract from a natural language sentence a representation of the tasks it should execute (i.e., a semantic frame) that allows the robot to perform the command received. The grounding is carried out, for each atomic command, independently. While a detailed discussion of the grounding process is outside of the scope of this paper, our approach builds upon the work by [Kollar *et al.*, 2013]. We briefly report it here for completeness.

We consider the problem of grounding an atomic command as a joint probabilistic inference. Our model is composed by the possible groundings Γ, the command C and its parse P. The inference is made possible by a Knowledge Base (KB) that gathers previous grounding of natural language expressions. We formally define the inference problem as:

$$\arg\max_{\Gamma} p(\Gamma, P, C|\text{KB})$$

In the rest of this section we assume the correct structure representing a complex command has been extracted and each atomic command has been grounded. Next we present an algorithm to compute an optimal plan to execute all the atomic commands in a complex command and evaluate it.

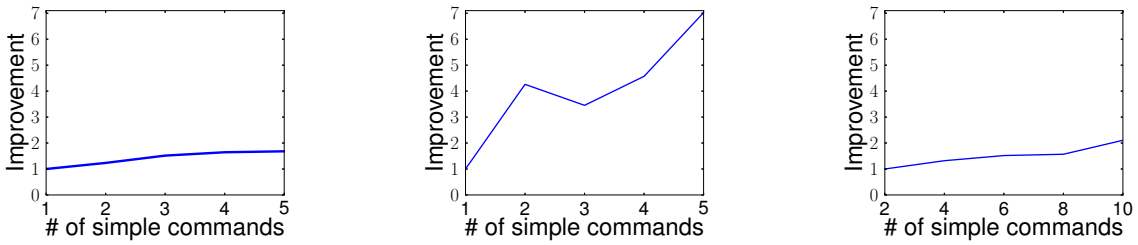
6.1 A Reordering Algorithm

Once all the atomic commands have been grounded, the robot can start executing them. A naïve approach would simply execute each task in the order given by the initial command. We assume that the robot has a measure of the cost to execute each single task. Our goal is, therefore, to find the optimal plan that satisfies the constraints expressed in the original complex command and minimize the overall execution cost.

The idea is to leverage the structure extracted from a complex command. Each of the four operators we use to describe the structure of a complex command allows for different optimizations or specifies a constraint. For each operator we generate different sequences of commands, namely:

- **AND** operators originally refer to a set of commands. We therefore generate a sequence for each permutation of the commands connected.
- **OR** operators give multiple options to the robot. Accordingly we generate a sequence for each command connected, each sequence only contains one of the commands.
- **THEN** operators express a constraint in the order in which tasks should be executed.
- **IF** operators, similarly to **THEN**, express a constraint but the left-hand part of the command is executed only if the conditions expressed in the right-hand part are met.

The algorithm we presented at generates all the possible sequences of tasks that satisfy the complex command, evaluates the cost of each of them and, finally, executes the optimal one. The algorithm takes as input a command C and starts with an empty list of tasks sequences. If C is atomic, the corresponding task is returned. Otherwise, the algorithm considers the



(a) (b) (c)

Figure 6: Comparison between our reordering the commands and executing them in the given order, measured as the ration of the traveled distances. Commands including: (a) only **AND** operators, (b) only **OR** operators and (c) any of the operators.

right-hand (RH) and the left-hand sides (LH) separately, generates all the possible sequences for both of them, combines them accordingly to the operator O, and adds them to the list of possible sequences. Algorithm 3 shows our approach.

Algorithm 3

```
1:  function CREATE_SEQUENCE(C)
2:      if IS_ATOMIC(C) then:
3:          return C
4:      else
5:          LH, O, RH ← C
6:          L = CREATE_SEQUENCE(LH)
7:          R = CREATE_SEQUENCE(RH)
8:          result = []
9:          for all l in L do
10:             for all r in R do
11:                 result.append(COMBINE(l, r, O))
12:             end for
13:         end for
14:     return result
15:     end if
16: end function
```

6.2 Evaluation

In order to test our reordering algorithm, we generated 3 different sets of random commands; the first containing only **AND** operators, the second containing only **OR** operators and a third containing **IF** or **THEN** operators and, for more complex commands, **AND** and **OR** operators as well. The first two sets are composed of commands of increasing complexity from one to five atomic tasks, while the third one contains commands of complexity two to ten. Each set contains fifty commands, ten for each complexity level.

Our approach is compared to a naïve base-line that, for **AND**, **THEN** and **IF**, executes the tasks in the order given and, for **OR**, randomly picks one of the alternatives. To measure the cost of each sequence of tasks we used the travel distance of the robot. We measured the improvement of our algorithm over the baseline as the ratio of the two distances. In measuring the cost of the execution, we assumed, for both the baseline and our approach, that the conditions of commands with **IF** operators are always met.

Figure 6(a) shows the result for the **AND** set. As to be expected for commands of complexity one the baseline and our approach achieve the same result. As the complexity increases, our reordering algorithm consistently improves and, for a command of complexity five, we get a travel distance 1.68 times shorter than the baseline.

Figure 6(b) shows the result for the **OR** set. Again, for commands of complexity one, the baseline and the reordering algorithm have the same result. As the complexity level increases, our approach starts improving over the baseline. The improvement is non-monotonically increasing due to the nature of the baseline. Since it chooses the task to execute randomly the improvement cannot be constant.

Finally, Figure 6(c) shows the result on complex commands containing all the four operators. For this set, we start with commands of complexity two (that is a sequence of two tasks). Our approach consistently improves over the baseline.

7 Conclusion and Future Work

In this paper we presented a novel approach to understand and execute complex commands for service robot task requests. We identified four different types of complexity, and designed a template-based algorithm able to break down a complex command in its atomic components. Experiments show that the algorithm is able to correctly reduce a complex command 72% of the time. To further recover the correct structure of a complex command, we introduced two different dialogue approaches. Finally we presented a reordering algorithm able to find the optimal plan to execute a complex command that substantially improves over a naïve baseline.

In this paper we focused on enabling a robot to understand and execute complex sentences. A key component of our approach is the dialogue with users and, in our experiments, we used a parenthetic notation to represent the structure of complex commands. Many other representations can be adopted such as parse trees or verbal description by the robot. In our future work we plan to investigate the optimal way to convey this information.

Acknowledgments

This research is partly sponsored by the NSF under awards NSF IIS-1012733 and IIS-1218932, the ONR under grant number N00014-09-1-1031, and an ERI by the FCT, Portugal. The views and conclusions contained herein are those of the authors only.

References

[Baker *et al.*, 1998] Collin F Baker, Charles J Fillmore, and John B Lowe. The berkeley framenet project. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics, 1998.

[Biswas and Veloso, 2013] Joydeep Biswas and Manuela M Veloso. Localization and navigation of the cobots over long-term deployments. *The International Journal of Robotics Research*, 32(14):1679–1694, 2013.

[Chen *et al.*, 2010] Xiaoping Chen, Jianmin Ji, Jiehui Jiang, Guoqiang Jin, Feng Wang, and Jiongkun Xie. Developing high-level cognitive functions for service robots. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 989–996. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[Fillmore, 1985] Charles J Fillmore. Frames and the semantics of understanding. *Quaderni di semantica*, 6(2):222–254, 1985.

[Ghidary *et al.*, 2002] Saeed Shiry Ghidary, Yasushi Nakata, Hiroshi Saito, Motofumi Hattori, and Toshi Takamori. Multi-modal interaction of human and home robot in the context of room map generation. *Autonomous Robots*, 13(2):169–184, 2002.

[Kollar *et al.*, 2013] Thomas Kollar, Vittorio Perera, Daniele Nardi, and Manuela Veloso. Learning environmental knowledge from task-based human-robot dialog. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4304–4309. IEEE, 2013.

[Kruijff *et al.*, 2007] Geert-Jan M Kruijff, Hendrik Zender, Patric Jensfelt, and Henrik I Christensen. Situated dialogue and spatial organization: What, where... and why. *International Journal of Advanced Robotic Systems*, 4(2):125–138, 2007.

[MacMahon *et al.*, 2006] Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. *Def*, 2(6):4, 2006.

[Marge and Rudnicky, 2010] Matthew Marge and Alexander I Rudnicky. Comparing spoken language route instructions for robots across environment representations. In *Proceedings of the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 157–164. Association for Computational Linguistics, 2010.

[Tellex *et al.*, 2011] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R Walter, Ashis Gopal Banerjee, Seth J Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, 2011.

[Thomas and Jenkins, 2012] Brian J Thomas and Odest Chadwicke Jenkins. Roboframenet: Verb-centric semantics for actions in robot middleware. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 4750–4755. IEEE, 2012.

[Winograd, 1971] Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. Technical report, DTIC Document, 1971.

[Zuo *et al.*, 2010] Xiang Zuo, Naoto Iwahashi, Ryo Taguchi, Kotaro Funakoshi, Mikio Nakano, Shigeki Matsuda, Komei Sugiura, and Natsuki Oka. Detecting robot-directed speech by situated understanding in object manipulation tasks. In *RO-MAN*, pages 608–613, 2010.