Communications in Information and Systems
Volume 17, Number 1, 45–64, 2017

Nerva: Automated application synthesis for humanoid robot from user natural language description

Hao Li, Yu-Ping Wang and Tai-Jiang Mu*

With the development of computer technology, humanoid robot becomes more and more popular. The user often hopes to use the robot to perform different kinds of tasks as needed. However, the robot manufacturer only provides limited number of robot applications in its application store, which cannot satisfy user demands in many cases. Furthermore, it is often difficult for the user to develop new robot applications as needed, because much robotic programing knowledge and manual work are required. To solve this problem, we propose a practical framework named Nerva, which can automatically synthesize robot applications from user natural language descriptions. There are three phases when Nerva works. Firstly, Nerva converts user natural language descriptions into syntax trees using natural language processing (NLP) techniques. Secondly, Nerva uses the syntax trees to synthesize intermediate language scripts. Finally, Nerva translates the intermediate language scripts into target robot applications based on robotic system APIs. We have implemented Nerva on the NAO robotic system. The experimental results show that Nerva can automatically and successfully synthesize 78.9% user-level robot applications in the NAO robot application store with a short time usage. Moreover, Nerva can also automatically synthesizes practical and useful applications for user-defined tasks which are not in the store.

1. Introduction

Nowadays, the humanoid robot has been a promising and important industry. A humanoid robot is often expected to do different kinds of tasks for the user. However, the robot manufacturer only provides limited number

* Tai-Jiang Mu is the corresponding author. This work was supported by Tsinghua University Initiative Scientific Research Program (20141081140).

of robot applications in its application store, which cannot satisfy user demands in many cases. Expecting the user (especially end user) to develop new robot applications as needed is often impractical, because much robotic programing knowledge and manual work are required. Thus, it is necessary to help user to synthesize robot applications as needed with less manual work.

To help the user to synthesize robot applications with less manual work, many previous approaches have been proposed. Some approaches [5, 7, 18] uses visual programming systems to provide abstract building blocks to embody robot behavior. But when using these approaches, the user needs to understand how to use each building block and how to combine those blocks visually in a logical order. To further reduce manual work of the user, some approaches [9, 10, 13] provides natural language interaction for the user to instruct the robot. They use some predefined command patterns to make the robot perform simple instructions, such as "sit down" and "open your □ hand" (where □ can be "left" or "right"), etc. However, these approaches use predefined keywords or grammars, instead of semantic information, to understand the user natural language description. But the user is often more willing to say more natural sentences to generate more complex robot applications (like triggered task: "When the bell rings, open the door."), instead of simple instructions. In this situation, these approaches cannot well understand the user's demands, and cannot generate corresponding robot applications according to the descriptions.

In this paper, we propose a practical framework named Nerva, which can automatically synthesize robot applications from user natural language descriptions. There are three phases when Nerva works:

1) Nerva converts user natural language descriptions into syntax trees with text chunk nodes, which are tagged with part-of-speech(POS) information.
2) Nerva transforms the syntax trees into formal intermediate language scripts. Text chunk nodes are mapped to tokens in the intermediate language and the syntax structure information is also preserved.
3) Nerva translates the intermediate language scripts into final executable robot applications.

Compared to previous approaches, Nerva has three main advantages:

- Nerva allows the user to use natural language with compound sentence, instead of simple instructions.

- Nerva can synthesize more complex robot applications to handle triggered tasks.
- Nerva provides a robot-independent intermediate language, which can support different robotic systems.

In this paper, we make three main contributions:

- We propose Nerva, which uses NLP and program synthesis techniques to automatically synthesize robot application from user natural language description. Users can describe their tasks naturally with minimal vocabulary or grammar restrictions.
- We have implemented Nerva on the NAO humanoid robotic system.
- By using Nerva, we have automatically and successfully synthesized 78.9% user-level robot applications in the NAO robot application store, with a short time usage. Moreover, we also use Nerva to automatically synthesize practical and useful robot applications for user-defined tasks.

The rest of this paper is structured as follows. Section 2 shows the motivation. Section 3 introduces Nerva in detail. Section 4 introduces our implementation of Nerva on the NAO humanoid robotic system. Section 5 shows the evaluation of generating NAO robot applications. Section 6 shows the limitations and future works. Section 7 introduces the related works. Section 8 concludes this paper.

2. Motivation

In the traditional way, the user describes a task to robot developer and the developer implements a robot application to achieve the task. This process is time consuming and requires a lot of manual work. Nerva aims to automatically synthesize robot applications from user natural language descriptions directly.

Here are some challenges for Nerva to achieve this goal.

Firstly, the user usually describes a task in compound sentences. These sentences othen consist of phrases, clauses and conjunctions. Previous works focus on "understanding" a simple sentence that can be handled by an instruction, such as booking a flight, setting a meeting remember, etc. However, many tasks cannot be described by one instruction, such as "If it is sunny tomorrow, book me a flight to Beijing." For this kind of descriptions, not only each simple sentence but also conditional relations between them should be well understood.

Secondly, according to the understanding of these sentences, Nerva should be able to synthesize correct applications on target robot.

Thirdly, Nerva should have good scalability when supporting different kinds of robotic systems.

Finally, Nerva should be friendly for users without programming knowledge. The generation process should be effective and automatic.

In the next section, we will give the approach Nerva used to overcome these challenges.

3. Approach

We will first give a high level architecture of our approach, then use an example to explain the detail of each phase in the approach.

Figure 1 shows the high level architecture of Nerva. There are three phases when Nerva works:

1) Nerva accepts the user's task description in natural language. It then uses the Stanford parser [4] to convert the description into a POS tagged syntax tree.
2) Nerva discovers all key components, particularly *SBAR clause*, *Verb Phrase*, *Noun Phrase*, and structure information from the tree. Then, mapping these components to corresponding tokens defined in our Nerva intermediate language, which are *Event*, *Action* and *Value.* Nerva finally combines them together and synthesizes an intermediate robot script by using a bottom-up algorithm and the structure information extracted from the tree.
3) Nerva translates the intermediate scripts to applications on target robotic systems.

In this section, we illustrate the three phases of Nerva via the following example:

Example 1: Suppose you have a baby and you have to leave for a moment. You wish to consign your baby to the care of your domestic robot: *Keep an eye on my baby. If he is out of your sight, shout out "baby lost" please.*

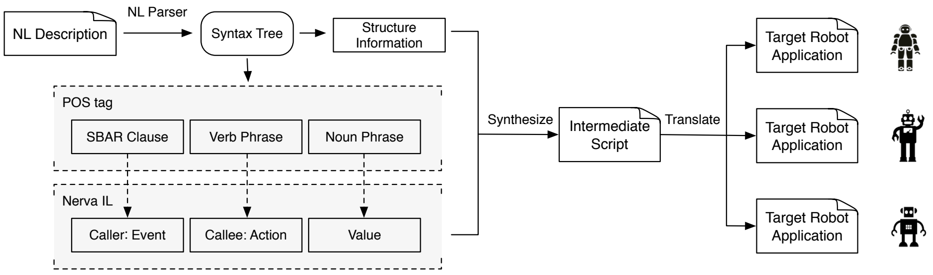


Figure 1. Architecture of Nerva.

3.1. Parse NL description to syntax tree

In this phase, Nerva accepts NL descriptions and annotates each sentence using the Stanford NLP tools. For Example 1, the parser produces a syntax tree as shown in Figure 2. Items in bold type in the tree are POS tags for phrase level items. Items in italic type are POS tags for word level items. **VP**, **NP**, and **PP** are the abbreviations of "Verb Phrase", "Noun Phrase" and "Prepositional Phrase". **SBAR** represents a clause introduced by a subordinating conjunction. The phrase level units are the basic text chunks used in our synthesis algorithm.

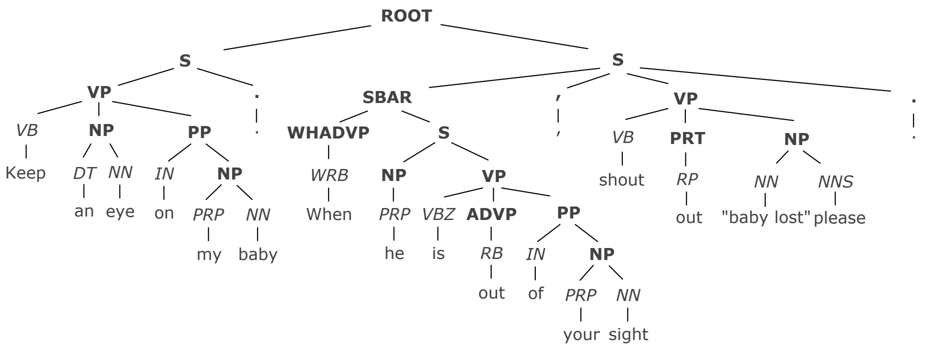


Figure 2. Stanford NL Parsing for Example 1.

3.2. From syntax tree to Nerva intermediate script

This phase aims to describe the user task in a robot-independent script, so that Nerva can support various robot platforms. Nerva transforms the syntax tree to a Nerva intermediate script using an assembly line. We first introduce the Nerva Intermediate Language (NIL), then give the detail of the three steps in the assembly line:

1) The **data identifier** performs initial data collection.
2) The **analysis engine** tries to discover actions and events from the syntax tree.
3) The **intermediate language synthesizer** constructs the script using structure information from the syntax tree.

3.2.1. Nerva intermediate language. NIL is designed to represent a wide variety of humanoid robot tasks, and keep the structure information from NL.

The syntax of NIL is shown in Figure 3. A module $\mathcal{M}$ in NIL represents a new task. $\mathcal{P}$ is a set of initial values. $\mathcal{B}$ is the main body of the module. $\mathcal{S}$ consists of a sequence of instructions that executed in order. A condition $\mathcal{C}$ consists of an event for a triggered task and a $\mathcal{S}$ to be executed when the event raised. Each instruction $\mathcal{I}$ is a basic robot action.

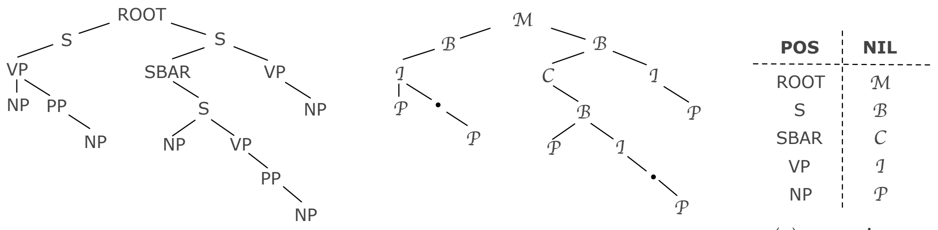
Module	$\mathcal{M}$	::=	$\mathcal{P}\,\mathcal{B}$
Parameter	$\mathcal{P}$	::=	Input(p_1, p_2, … , p_n) | ε
Body	$\mathcal{B}$	::=	$\mathcal{S}\,\mathcal{B}$ | $\mathcal{C}\,\mathcal{B}$ | ε
Sequence	$\mathcal{S}$	::=	$\mathcal{I}\,\mathcal{S}$ | ε
Condition	$\mathcal{C}$	::=	when Event($\mathcal{P}$) do $\mathcal{S}$
Instruction	$\mathcal{I}$	::=	Action($\mathcal{P}$)

Figure 3. Syntax of Nerva intermediate language.

Figure 4 shows the relation between syntax tree and NIL. (a) is the skeleton of key components in the syntax tree for Example 1. (b) is the corresponding expression in NIL. (c) is the mapping from POS tags to NIL tokens.

3.2.2. Data identifier. The data identifier accepts the syntax tree from the NL parser and extracts initial data from each NP for further analysis. In natural language, NP implies an object. Nerva uses both regular expressions and a static mapping table to identify them and converts them into typed values.

The two identification strategies are:



POS	NIL
ROOT	ℳ
S	ℬ
SBAR	𝒞
VP	ℐ
NP	𝒫

(a) skeleton of syntax tree (b) corresponding expression in NIL (c) mappings

Figure 4. Relation between Syntax Tree and NIL.

- **Regular expression**. Some types of data can be easily recognized using regular expressions, such as integers, dates, phone numbers, etc.
- **Static Mapping Table**. The static mapping table maintains a list of named-entity, such as "London" which mapped to a city type value, "last one" which mapped to an integer type value and "my baby" which mapped to a person type value.

There are 5 NP nodes in Example 1 (see Figure 5). "Baby lost" is mapped to string type value by regular expressions. Others are mapped by our static mapping table except the node "he" in the when-clause. "he" is a personal pronoun. It should be mapped to a person type value. However, in the NP node, we can not determine that value. The data identifier uses a heuristic search strategy within the tree. The Data Identifier accesses its parent node recursively to find the closest and most suitable person type value previously extracted in the input. In this case, "he" would be assigned the value "myBabyID".

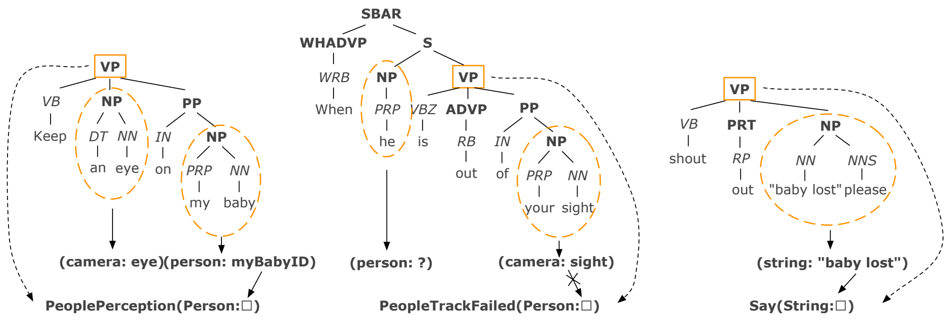


Figure 5. Mappings: NP → Value, VP → Action / Event.

3.2.3. Analysis engine. The analysis engine tries to map VP to actions or events.

Actions are a set of predefined basic functions for humanoid robot. A sequence of actions combined in different order can perform different user tasks. Events are special functions that monitor whether certain conditions are satisfied.

Action and event are annotated with:

- *keyword tags* used by the mapping algorithm. Nerva use a synonym service from WordNet[1] to generate an extended set for each keyword.
- *parameter type signatures* used by the synthesis algorithm to synthesize only type-safe scripts.

Example 1 has 3 VP nodes. The analysis engine find 2 actions and 1 event mapping to them (see Figure 5 and Table 1).

No.	**VP node**	**Result**	**Type**	**Parameter**	**Tags**
1	keep an eye on my baby	PeoplePerception	Action	*Person*	HUMAN, TRACK, FACE
2	is out of your sight	PeopleTrackFailed	Event	*Person*	HUMAN, LOST, OUT SIGHT
3	shout out “baby lost” please	Say	Action	*String*	SPEAK, SAY, SHOUT

Table 1. VP Mapping Result for Example 1.

The analysis engine works in three steps as shown in Algorithm 1.

1) The algorithm searches the syntax tree to determine whether mapping the VP node to action or event. For each VP node, if it’s parent is an SBAR node, the analysis engine searches the set of possible events E to find the most likely event. Otherwise, the analysis engine will search the action set A to find an action mapping (Lines 1-5).

   The first and third VP nodes in Example 1 should be mapped to actions, and the second should be mapped to an event.

2) Find all candidate mappings for the VP node. A candidate mapping should satisfy the **type constraint** that its parameters have values, extracted from the VP’s parent sentence node, of suitable types (Lines 7-15). The candidates are stored in $\mathcal{T}$.

   Take “is out of your sight” for an example. There are 2 typed values in its parent sentence node, camera type value “your sight” and person type value “he”. There are 5 mapping candidates in this case, such as

[1]A lexical database of English—see `https://wordnet.princeton.edu/`

Algorithm 1 Analysis Engine

```
Input: ${A, E, node_{VP}, V_S}$
    action set $A$, event set $E$, verb phrase node $node_{VP}$, values extracted by the
    data identifier from the VP's parent sentence node $V_S$.
Output: $t_{top}$, the best mapping candidate.
 1: if $node_{VP} \in node_{SBAR}$ then
 2:     $\mathcal{C} \leftarrow E$
 3: else
 4:     $\mathcal{C} \leftarrow A$
 5: end if
 6: $\mathcal{T} \leftarrow \emptyset$
 7: foreach $c \in \mathcal{C}$ do
 8:     $P_{arg} \leftarrow GetParameterTypeList(c)$
 9:     $P_{fail} \leftarrow \{p|p \in P_{arg} \wedge \forall v \in V_S, \nexists Type(v) = Type(p)\}$
10:     if $P_{fail} = \emptyset$ then
11:         $Distance \leftarrow \sum_{p \in P_{arg}} dist(node_p, node_{VP})$
12:         $Relation \leftarrow RelationBetween(c_{tags}, node_{VP})$
13:         $\mathcal{T} \leftarrow \mathcal{T} \cup \{< c, Distance, Relation >\}$
14:     end if
15: end for
16: if $\mathcal{T} \neq \emptyset$ then
17:     $t_{top} \leftarrow ChooseBestMatch(\mathcal{T})$
18: else
19:     $NotFoundHandling(e)$
20: end if
21: return $t_{top}$
```

"CameraStarted" which needs a camera type value and "PeopleTrack-Failed" which needs a person type value.

3) Choose the best matched result from $\mathcal{T}$. Line 17 use two metrics to choose the best match one. One metric is **distance** (Line 11). The parameters used by a candidate action or event come from some NP nodes of the tree. Distance metric is the sum of the distance from these NP nodes to the target VP node on the syntax tree. Distance is the metric for data relevance. The closer, the better the match. We use the lowest common ancestor (LCA) algorithm to calculate the value distance. The other metric is **Relation** (Line 12). We compare all the lexical tokens of the VP node with candidate's tags. Two words are matched if they are the same or synonyms. Relation metric gives the number of lexical tokens found in the candidate's tags.

The second VP has lexical tokens "out of sight" which match the tags of "PeopleTrackFailed". However, "PeopleTrackFailed" needs a person type value from its brother NP node "he". "CameraStarted" has higher distance metric while "PeopleTrackFailed" has higher relation metric. After calculating the final score, the best match is "PeopleTrackFailed".

3.2.4. Intermediate language synthesizer. In the previous two steps, Nerva finds all key components from the NL description, and maps them to tokens in NIL. Based on the initial data, the mapped actions and events, the intermediate language synthesizer now generates the intermediate script using the structure information from the syntax tree.

The structure information used by the intermediate language synthesizer is shown in Figure 6. It starts from the root of the syntax tree and does a depth first search for each child node in turn. Each sentence is used to output a basic code block. If the sentence includes an SBAR node, a *when* clause will enclose the block.

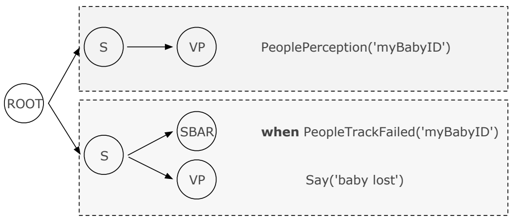

ROOT
S
VP
PeoplePerception('myBabyID')
S
SBAR
when PeopleTrackFailed('myBabyID')
VP
Say('baby lost')

Figure 6. Structure Information

At last, Nerva synthesizes the intermediate script (shows in Code 1) for Example 1.

```
PeoplePerception('myBabyID')
when PeopleTrackFailed('myBabyID') do
    Say('baby lost')
```

Code 1. Intermediate Script for Example 1

3.3. From Nerva intermediate script to target robot application

This phase is robot-dependent. Generally, there are four parts in the translation.

1) Nerva uses a static mapping table to identify initial values. We need to add robot-dependent part to the table, such as the name of joints for target robot.
2) A set of actions and events on humanoid robot are predefined in NIL. Actions are the simple instructions the robot would support, such as "move forward", "turn left", "say hello". Events are functions to check the status of robot (such as the status of battery, WiFi connection) or inputs from devices (such as camera, microphone, tactile sensors). We need to implement these actions and events on the target robot.
3) An event mechanism is needed to support triggered tasks. The event mechanism includes event register and event monitor.
4) A backend translator which translates Nerva intermediate scripts to programs executed on target robots.

In this paper, we implement Nerva on NAO robot, we will discuss the implementation in the next section.

4. Implementation on NAO

We have deployed Nerva on NAO humanoid robot from SoftBank Robotics (Aldebaran). We choose NAO robot because it is a widely used humanoid robot and its system NAOqi has implemented an event mechanism. Besides, NAOqi API provides lots of method modules we can reuse to implement our predefined actions and events.

4.1. NAO joints and NAOqi APIs

NAOqi is the framework used to program NAO robot applications. The robot has 25 joints which can be controlled with NAOqi APIs. We first add the name of them to Nerva static mapping table. For example, the phrase "right hand" is mapped to a particular joint "RHand".

NAOqi is based on a broker-module-method model (see Figure 7). The broker provides lookup services so that any module can find any method that has been advertised by other modules. Each module includes a group of actions or events related to a certain kind of task. For example, the ALMotion module provides actions for making the robot move; the ALDialog module allows you to endow your robot with conversational skills by using a list of rules written and categorized in an appropriate way.

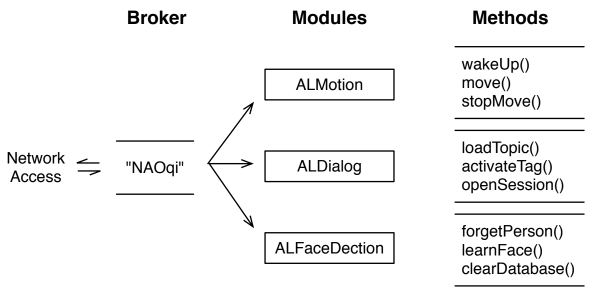


Figure 7. NAOqi Broker-Module-Method model.

4.2. Event mechanism

Figure 8 shows how the NAOqi event mechanism works: **Simple Instructions** are executed immediately. **Triggered Tasks** should be executed when some conditions are satisfied. In the case of simple instructions, Nerva appends them to the robot's task queue directly. For triggered tasks, Nerva generates a caller-callee pair where the *caller* provides code to check whether the conditions are satisfied and the *callee* provides actions that would be appended to the robot's task queue.

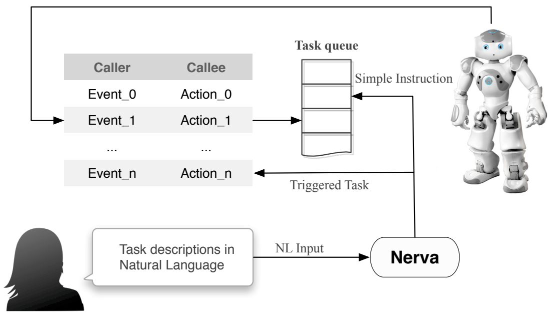


Figure 8. NAOqi Event Mechanism.

The NAOqi event mechanism is based on a centralized memory which stores all key information related to the hardware configuration, the current state of the actuators, and the sensors of NAO robot. The memory works as a hub for the distribution of event notifications. The memory is managed by the NAOqi ALMemory module, which guarantees that read and write

```
1   from naoqi import ALProxy
2
3   memory = ALProxy("ALMemory")
4   PeopleTrackFailedReact = None
5
6   class PeopleTrackFailedReactModel(ALModule):
7     def _init_(self, name):
8       ALModule.__init__(self, name)
9       self.name = name
10      global memory
11      memory.subscribeToEvent("PeopleTrackFailed", self.name,
12                  "onPeopleTrackFailed")
13    def inCall(self):
14      global memory
15      memory.unsubscribeToEvent("PeopleTrackFailed", self.name,
16                  "onPeopleTrackFailed")
17    def offCall(self):
18      global memory
19      memory.subscribeToEvent("PeopleTrackFailed", self.name,
20                  "onPeopleTrackFailed")
21    def onPeopleTrackFailed(self, *_args):
22      self.inCall()
23      TextToSpeechProxy = ALProxy("ALTextToSpeech")
24      TextToSpeech.say("baby lost")
25      self.offCall()
26
27  def run():
28    trackerProxy = ALProxy("ALTracker")
29    myBabyID = 0x01
30    myBabyFaceWidth = getFaceWidth(myBabyID)
31    trackerProxy.registerTarget("Face", myBabyFaceWidth)
32    trackerProxy.tarck("Face")
33    global PeopleTrackFailedReact
34    PeopleTrackFailedReact = PeopleTrackFailedReactModel(
35                      "PeopleTrackFailedReact")
36
37  def main():
38    pip = sys.argv[1]      # The IP address of your robot
39    pport = sys.argv[2]    # The port NAOqi is listening to
40    myBroker = ALBroker("myBroker", "0.0.0.0", 0, pip, pport)
41    run()
```

Code 2. Python Module for Example 1

operations are thread safe. Furthermore, ALMemory notifies all subscribers to an event when the event is triggered.

4.3. Translator

After synthesizing the intermediate script from the syntax tree, Nerva translates the intermediate script into target programs to run on the robot. On NAO robot, the target programs are python modules. Code 2 shows the new python module Nerva generated for Example 1.

In the NAOqi framework, each new module should create a broker to represent itself and use the broker to communicate with other modules. The broker must stay alive until the program exists. The initial code to make this happen must be added first in the main function of our new module (Line 38-40). Then a function named “run()” is called after the initialization. The "run()" function (Line 27-35) is generated by our translator and it is the entry point of our new module.

The generator generates an immediate function call for each simple instruction. In NAOqi framework, before calling the function, a proxy object is created to represent the module that provides this function. For example, "TextToSpeechProxy" is the proxy for the predefined module "ALTextToSpeech" (Line 23 of Code 2). After that we can call the "say" function of "ALTextToSpeech" module (Line 24).

Under NAOqi event mechanism, a class should be generated to handle event for triggered task. Lines 6-25 of Code 2 show the class generated to handle "PeopleTrackFailed" event. Class function "onPeopleTrackFailed" is registered as the callback of the event. Within the callback function, the "say" function is called to execute the action when the event is raised. At the entry and exit of the callback function, "inCall" and "offCall" are set to ensure the callback is not recursively called during its execution.

5. Evaluation

To show how useful our system is, we tried to synthesize applications which replicate the behaviors of all user-level applications collected from the NAO application store [2]. Then we give some user-defined tasks which are not in the NAO application store to show that Nerva is user-friendly.

Nerva runs on a machine with a 3.2 GHz Intel i5-3470 CPU with 8GB of RAM. The target robot is a NAO with NAOqi framework version 2.1.3.

5.1. Applications from NAO online store

The store provides 27 NAO applications. 19 of them are usr-level applications. Nerva successfully synthesized python modules for 15 of them, a success rate of about 78.9%. Table 2 briefly describes the applications considered; the last two columns indicate the line of final modules and time used.

5.1.1. A successful example. Taking "Go to rest" for an example. The application tells the NAO robot to rest and wait until someone pats his head.

[2]NAO application store, login is required. `https://cloud.aldebaran-robotics.com/`

No.	Application	Description	Success	LOC	T(ms)
1	Dialog move	move according to the user's instructions	√	174	1384
2	Dialog touch	react when his bumpers are touched	√	46	219
3	Grasping	grasp small objects.	√	68	311
4	Ask Nao	talk about himself, the applications and the interface	√	248	879
5	Touch my head	play a small game	√	102	641
6	Taichi dance free	some taichi exercises	√	48	94
7	Night before Christmas	tell the night before Christmas story	√	64	193
8	Walk to the ball	walk to a red ball in your hand	√	129	398
9	Follow me	give you its hand and walks with you	×		
10	Soccer	demonstrate some simple soccer behavior	√	131	252
11	Fall recovery	try to go back to his previous stable posture after falling	×		
12	Blind me	react when it is blinded by somebody	√	62	104
13	Go to rest	sit down and react when touching head	√	104	213
14	Presentation	talk about himself and what he can be used for	√	169	693
15	Exploration	explore its environment using sonar	×		
16	Move recorder	copy some moves made by a human	×		
17	Dialog abilities	list what he can do	√	142	549
18	Dialog about me	answer several questions about himself	√	87	493
19	Dialog greetings	respond to "hello" according to the time of the day	√	242	1293

Table 2. 19 Applications from NAO Application Store.

We can give the task description as "Go to rest. When I touch your head, stand up.". This application is based on a set of motion instructions. The task can be split into two sub-tasks:

1) A simple motion instruction "go to rest". The NAO robot's motion is managed by the "ALMotion" and "ALRobotPosture" module. "ALMotion" module provides different levels of robot body control from joint angles to Cartesian space movement or consistent, stabilized whole body motions. "ALRobotPosture" module provides predefined postures such as "Stand", "Sit", "Rest".
2) A Triggered task "When I touch your head, stand up". In local static mapping table, NAO robot's head is bound to middle head tactile sensor. Nerva subscribes to the event "MiddleTactilTouched" which is raised when the middle head tactile sensor is touched by users.

Code 3 gives the intermediate script. The final python module generated from the intermediate script has 104 lines of code.

```
GoToPosture('Sit')
when MiddleTactilTouched() do
    GoToPosture('Stand')
```

Code 3. Nerva Intermediate Script for Go To Rest

5.1.2. Failed tasks analysis. There are 4 applications Nerva failed to synthesize, because of the following limitations in the implementation of Nerva.

- Nerva does not support event nesting. Nerva uses a simple event-action model to handle triggered tasks, but cannot yet cope with more complex event structures. For example, the application "Fall recovery" requires the NAO robot to go back to a stable posture if he falls. If he fails to get up by self, he calls a human for help. The second event "failed to get up" is raised only when the first event happened that he has fallen down. Such nested event structures cannot at present be represented in NIL, so Nerva does not support them.
- Nerva does not support action loops, so Nerva cannot tell the robot how many times an action should be executed. For example, the application "Move recorder" tells NAO to stop moving, stand up and and let his head and arms go limp. The human then moves them to show NAO what to do. Finally, NAO makes the same moves by himself. However, Nerva does not know when to quit the guide process.

5.2. Practical user-defined tasks

Nerva is friendly to users without programming knowledge. Nerva can synthesize other practical applications for user-defined tasks which are not in the application store.

Task Description	**LOC**	**T (ms)**
Walk to the door and open it.	32	304
Open your right hand.	26	125
Say "hello" to my friends.	26	149
Play "Little Star" for me.	28	211
When the bell rings, go to the door and open it.	56	782
Move forward. Stop and turn left when you encounter obstacles.	60	822
When someone comes in, say "Hi, nice to meet you".	54	725

Table 3. Examples of User-Defined Tasks.

Table 3 shows some examples. The first column gives the descriptions of tasks, the later two columns show line of code for the final NAOqi modules and the time used. User-defined tasks are not too complex but ever-changing. Based on NLP techniques, Nerva can split them into simple actionss and put them together in the order guided by the structure of the NL descriptions. Besides, Nerva can be used without significant restrictions to vocabulary or grammar. “Walk to the door” or “go to the door” is the same to Nerva. For Example 1, the description “Keep a watch on my baby. When you can’t see her, send a message ‘baby lost’ to my phone 123-4567” would work too.

6. Limitations and future works

Although successfully synthesized 78.9% applications in the NAO applications store, Nerva still have the following limitations.

- Nerva heavily relies on the result of NL parser. If the result of NL parser is incorrect, Nerva will failed either. In this paper, we choose the state-of-art Stanford NLP tool as our parser, but it is still very hard to handle typo or grammar errors or ambiguity of natural language in the task description. We will keep attention on the development of NLP techniques and update our parser to improve the accuracy.
- To support different robotic systems, Nerva defines an intermediate language. We need to implement a set of actions and events as well as an event mechanism on each target robot, which brings a lot of manual work. We could simplify this process by semi-automatically generating with robot specification files in the future.
- Nerva only focuses on NP, VP and SBAR nodes of the syntax tree. Other kind of nodes, like prepositional phrase or adjective phrase, may also have useful information. The ability of Nerva is limited without that information. we will try to analyze other kinds of nodes and support more complex structures like event nesting and action loops.

7. Related works

7.1. Visual programming systems for robots

Some approaches focus on designing a visual programming system for robots. There are two common categories of visual programming systems. One is developer focused and the other is end-user focused.

LabVIEW [19], Simulink [3], SCADE System [11] and ControlShell [17] belong to the former category. With specialized add-ons [2], these products provide powerful tools for rapid robot application development for experienced developers. However, they require a level of sophistication and knowledge unlikely to be possessed by many end-users.

In the latter category, LEGO's RoboLab [14] aims to control educational LEGO robotics, Choregraphe [15] provides a graphical environment for programming Aldebaran Robotics robots, and Microsoft's Robotics Developers Studio (MRDS) [6] facilitates the mapping between decoupled software modules and hardware components. Those systems are more friendly to end-users. However, they often focus on specific types of robots or suffer from limited capabilities when used to address complex problems.

7.2. Robot instruction system by natural language

Many works have proposed robot instruction systems based on natural language (NL). For example, Kollar [8] extracts a sequence of spatial description clauses from NL input and uses a probabilistic model to connect them together to find a path for a robot. Bugmann [1] implemented an instruction-based learning (IBL) system with 15 primitive functions (`go until`, `turn`, `exit from`, etc.), each of which has a fixed parameter list. MacMahon [12] infers a set of predefined actions from knowledge of both linguistic conditional phrases and local configurations. Many of these focus on navigating the robot, but a comprehensive instruction system should handle more general tasks. Rybski [16] proposed a system that can learn simple action scripts from NL, but the instructions must follow a predefined grammar.

8. Conclusion

Nerva brings three enhancements to previous human-robot interaction systems.

Firstly, Nerva is more intelligent in "understanding" end-user's tasks. By combining natural language processing and program synthesis techniques, Nerva synthesizes code blocks from sentence clauses, by mapping text chunks to tokens in NIL at the level of phrases: events, actions, and values.

Secondly, Nerva supports triggered tasks, which allows human-like responses to the external world.

At last, Nerva provides a robot-independent intermediate language so that we can easily port it to different robotic systems.

References

[1] G. Bugmann, E. Klein, S. Lauria, and T. Kyriacou, *Corpus-based robotics: A route instruction example*, in: Proceedings of Intelligent Autonomous Systems, 96–103, Citeseer (2004).

[2] P. Corke, *Robotics, vision and control: fundamental algorithms in MATLAB*, Vol. 73, Springer (2011).

[3] J. B. Dabney and T. L. Harman, *Mastering simulink*, Pearson/Prentice Hall (2004).

[4] C. D. M. Danqi Chen, *A Fast and accurate dependency parser using neural networks*, in Proceedings of EMNLP (2014).

[5] D. Ingalls, S. Wallace, Y.-Y. Chow, F. Ludolph, and K. Doyle, *Fabrik: a visual programming environment*, in: ACM SIGPLAN Notices, Vol. 23, 176–190, ACM (1988).

[6] J. Jackson, *Microsoft robotics studio: A technical introduction*, IEEE Robotics & Automation Magazine **14** (2007), no. 4, 82–87.

[7] S. H. Kim and J. W. Jeon, *Programming LEGO Mindstorms NXT with visual programming*, in: Control, Automation and Systems, 2007. ICCAS'07. International Conference on, 2468–2472, IEEE (2007).

[8] T. Kollar, S. Tellex, D. Roy, and N. Roy, *Toward understanding natural language directions*, in: 2010 5th ACM/IEEE International Conference on Human-Robot Interaction (HRI), 259–266, IEEE (2010).

[9] S. Lauria, G. Bugmann, T. Kyriacou, J. Bos, and E. Klein, *Personal robot training via natural-language instructions*, IEEE Intelligent systems **16** (2001), no. 3, 38–45.

[10] S. Lauria, G. Bugmann, T. Kyriacou, and E. Klein, *Mobile robot programming using natural language*, Robotics and Autonomous Systems **38** (2002), no. 3, 171–181.

[11] T. Le Sergent, *SCADE: A comprehensive framework for critical system and software engineering*, in: International SDL Forum, 2–3, Springer (2011).

[12] M. MacMahon, B. Stankiewicz, and B. Kuipers, *Walk the talk: Connecting language, knowledge, and action in route instructions*, Def **2** (2006), no. 6, 4.

[13] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox, *Learning to parse natural language commands to a robot control system*, in: Experimental Robotics, 403–415, Springer (2013).

[14] M. Portsmore, *ROBOLAB: Intuitive robotic programming software to support life long learning*, APPLE Learning Technology Review, Spring/Summer (1999).

[15] E. Pot, J. Monceaux, R. Gelin, and B. Maisonnier, *Choregraphe: a graphical tool for humanoid robot programming*, in: RO-MAN 2009-The 18th IEEE International Symposium on Robot and Human Interactive Communication, 46–51, IEEE (2009).

[16] 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 **1** (2008), no. 2, 159–167.

[17] S. A. Schneider, V. W. Chen, G. Pardo-Castellote, and H. H. Wang, *Controlshell: A software architecture for complex electromechanical systems*, The International Journal of Robotics Research **17** (1998), no. 4, 360–380.

[18] N. C. Shu, *Visual programming*, Van Nostrand Reinhold New York (1988).

[19] L. K. Wells and J. Travis, *LabVIEW for everyone: graphical programming made even easier*, Prentice-Hall, Inc. (1996).

Tsinghua National Laboratory for Information Science and Technology(TNList), Department of Computer Science and Technology
Tsinghua University, Beijing, China
E-mail address: roselonelh@gmail.com

Tsinghua National Laboratory for Information Science and Technology(TNList), Department of Computer Science and Technology
Tsinghua University, Beijing, China
E-mail address: wyp@tsinghua.edu.cn

Tsinghua National Laboratory for Information Science and Technology(TNList), Department of Computer Science and Technology
Tsinghua University, Beijing, China
E-mail address: dejungle@mail.tsinghua.edu.cn