NATURAL LANGUAGE BASED CONTROL AND PROGRAMMING OF ROBOTIC BEHAVIORS

By

Yu Cheng

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Electrical Engineering – Doctor of Philosophy

2019

ABSTRACT

NATURAL LANGUAGE BASED CONTROL AND PROGRAMMING OF ROBOTIC BEHAVIORS

By

Yu Cheng

Robots have been transforming our daily lives by moving from controlled industrial lines to unstructured and dynamic environments such as home, offices, or outdoors working closely with human co-workers. Accordingly, there is an emerging and urgent need for human users to communicate with robots through natural language (NL) due to its convenience and expressibility, especially for the technically untrained people. Nevertheless, two fundamental problems remain unsolved for robots to working in such environments. On one hand, how to control robot behaviors in dynamic environments due to presence of people is still a daunting task. On the other hand, robot skills are usually preprogrammed while an application scenario may require a robot to perform new tasks. How to program a new skill to robots using NL on the fly also requires tremendous efforts. This dissertation tries to tackle these two problems in the framework of supervisory control.

On the control aspect, it will be shown ideas drawn from dynamic discrete event systems can be used to model environmental dynamics and guarantee safety and stability of robot behaviors. Specifically, the procedures to build robot behavioral model and the criteria for model property checking will be presented. As there are enormous utterances in language with different abstraction level, a hierarchical framework is proposed to handle tasks lying in different logic depth. Behavior consistency and stability under hierarchy are discussed.

On the programming aspect, a novel online programming via NL approach that formulate the problem in state space is presented. This method can be implemented on the fly without terminating the robot implementation. The advantage of such a method is that there is no need to laboriously labeling data for skill training, which is required by traditional offline training methods. In addition, integrated with the developed control framework, the newly programmed skills can also be applied to dynamic environments.

In addition to the developed robot control approach that translates language instructions into symbolic representations to guide robot behaviors, a novel approach to transform NL instructions into scene representation is presented for robot behaviors guidance, such as robotic drawing, painting, etc. Instead of using a local object library or direct text-to-pixel mappings, the proposed approach utilizes knowledge retrieved from Internet image search engines, which helps to generate diverse and creative scenes. The proposed approach allows interactive tuning of the synthesized scene via NL. This helps to generate more complex and semantically meaningful scenes, and to correct training errors or bias.

The success of robot behavior control and programming relies on correct estimation of task implementation status, which is comprised of robotic status and environmental status. Besides vision information to estimate environmental status, tactile information is heavily used to estimate robotic status. In this dissertation, correlation based approaches have been developed to detect slippage occurrence and slipping velocity, which provide grasp status to the high symbolic level and are used to control grasp force at lower continuous level. The proposed approaches can be used with different sensor signal type and are not limited to customized designs.

The proposed NL based robot control and programming approaches in this dissertation can be applied to other robotic applications, and help to pave the way for flexible and safe human-robot collaboration.

ACKNOWLEDGEMENTS

First of all, I want to express my grateful acknowledgment to my advisor Dr. Ning Xi and Dr. Lixin Dong, for their encouragement, guidance, kindness, and support. Their insightful vision and high standard make me a qualified researcher.

I would like to thank my committee members: Dr. Joyce Chai, Dr. Fathi Salem, and Dr. Xiaobo Tan. I highly appreciate their valuable feedback and discussions throughout my study at MSU.

Many thanks go to my lab members: Dr. Yunyi Jia, Dr. Jianguo Zhao, Dr. Liangliang Chen, Dr. Zhiyong Sun, Dr. Bo Song, Dr. Hongzhi Chen, Dr. Erick Nieves, Dr. Yongliang Yang, Xiao Zeng, Lai Wei, etc., for their help and the happiness we have enjoyed together. I also want to thank Dr. Zhenxue Chen, Dr. Daoxiong Gong, Dr. Zhanxin Zhou, Dr. Jiatong Bao, Dr. Zhihui Deng, Dr. Haichu Chen, Dr. Sheng Bi, etc., for their support to my study and life.

I would like to thank my friends at MSU: Yan Shi, Dr. Zhe Wang, Yuan Liang, Dr. Guangwei Sun, Dr. Jiankun Liu, Dr. Yiqun Yang, Qianwei Jiang, Dr. Jie Li, Biyi Fang, Dr. Mingquan Yuan, Dezhi Feng, etc., for all the help they gave me and the joy we had together.

I also would like to thank Dr. Qinghu Meng and Dr. Jun Zhang for their continuous support and encouragement.

Last but not least, I want to thank my parents for their unconditional support and showing me what love truly means.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

Robots have been widely used in controlled environments to replace human workers in 3D (dangerous, dull, and dirty) jobs. Robots are known for their higher accuracy and efficiency, increasing product/service quality and thus profitability, and longer working hours. On the other hand, there are still some limitations in terms of the type of tasks they can perform, which is mainly due to limited knowledge and cognitive capability. To make use of the strength of robots in more applications, there has been an emerging and increasingly urgent need for robots and people to collaborate together, since human can compensate for inadequate knowledge and intelligence.

Successful collaboration between robots and human users require effective and informative interaction means for information exchange and behavior control. To take full advantage of human skills, it is important that intuitive user interfaces are properly designed, so that human users can easily program and interact with robots. There is a variety of methods been used such as teach pendant, guiding, programming language, graphical interfaces, body language, etc. Among them, using NL to control and program robot is very attractive because it has several advantages compared with other approaches. First, due to the expressiveness of NL, it is capable to represent domain-general tasks with concepts of different logic depth. Second, NL is intuitive and friendly, which makes it convenient to use for technically untrained users. Third, using NL frees users' hands such that the users can focus on the task itself.

With the advantages of NL based robots, they can be useful for many applications. For industrial or domestic applications where robots and human have to collaborate closely in a common area, robots may not behavior correctly due to limited sensing ability or insufficient domain knowledge. Human users can tell their robot co-workers the missing information via NL for correct decision making, or guide the robots behavior directly through language instructions. This helps to improve

robots adaptivity and safety coexisting with people in dynamic and complex environment, and enhance the robots cognitive ability. In addition, users can teach a robot new skills/concepts based on its already learned knowledge through NL without terminating the robot and reprogramming the robot by hand-coding. This helps to reduce robot development cost and facilitate wider applications. Furthermore, with NL understanding and interaction ability, the robots can access the Internet for information query. This allows robots to learn knowledge from this multi-disciplinary knowledge base intended for human use when a human instructor is not available. It helps a robot to become a self-learning assistant.

1.2 Challenges and Objectives

NL based robots discussed in previous section will work alongside people and communicate through NL. Since the environments are complex and dynamic due to presence of people, and robots cannot understand NL directly, these robot should satisfy three basic requirements in order for successful task execution in such environments. First, robots are executing tasks specified in NL, they should be able to transfer linguistic input into behavioral output. Second, the environments in practical are unstructured, the robots should be able to deliver the task goal under dynamic and uncertain situations. Third, the robots may encounter new task requirements that hasn't been programmed in advance. In these cases, it is expected that robots can learn new knowledge through language interaction, similar to the way in which an adult teaches his kids.

To address these three requirements, three challenges exist. First, NL instructions intended for human use may seem to be vague and underspecified to robots. It is nontrivial to transfer ambiguous language input into deterministic behavior output. Second, the robots are working in dynamic environments and thus may encounter different situations. It is not practical for human users to specify all the possible specifications to accomplish the assigned task. It is difficult to tackle environmental uncertainties with underspecified instructions. Third, the physical structure and cognitive capability of robots and human are completely different. It is difficult to teach a robot new skills through NL in a similar way as to a human.

The objectives for the research presented in this dissertation are to investigate how to address the previous three challenges using discrete event system (DES) principles, i.e., how DES theory can be employed for the control and programing of robots so that they can perform tasks assigned in NL successfully (especially in dynamic environments) and be programmed via NL instructions. The specific focuses on the control and programming are briefly described in the following.

On the control side, we first focus on how robots can accomplish tasks in dynamic environments, especially when uncertainties of environments affect task execution.Towards this goal, the robot should take environmental dynamics into consideration during modeling. In addition, a correct-by-design mechanism is desired to ensure behavioral safety and stability of robots in order for practical deployment. The second focus on the control side is to deal with tasks of different logic depth. Although a robot that satisfies the previous objective can perform in dynamic scenarios, it is difficult and unrealizable to process all the possible actions within a single control layer. This can lead to the state explosion problem [2] as the number of actions and states goes up. It is critical that the robot can process commands in a hierarchical manner since some concepts can build up from others. The third focus on the control side is to transform linguistic input into a scene representation for behavior control. Existing approaches of NL based robot control transfer NL instructions into symbolic representations to guide robots' behavior. Compared with symbolic representation, representing the task goal as a scene can contain more details accomplish tasks that are not suitable for symbolic representations.

On the programming side, we focus on programming new skills to robots in a hierarchical way since complex movement can be decomposed into an ordered sequence of primitive actions. For example, a *stack* action is comprised of an ordered sequence of *move*, *pick up*, and *drop* actions, and can be taught using the three basic actions [3].

In addition, successful robot control and programming rely on correct estimation of current robot status in order to properly the enable action of next step. Besides encoders used for estimate robot positional status, we use tactile information to estimate the gripper status, which can aid correct decision making in the following action selection. For instance, if an object is grasped

successfully, the robot can continue to the next move. Otherwise, the robot has to replan a grasp. We propose correlation based slippage detection, which is able to detect slippage occurrence and slippage velocity, providing perceptive feedback for both the high level planning and lower level control.

1.3 Literature Review

1.3.1 Natural Language based Robot Control

Starting with SHRDLU [4], using NL to control robotic behavior has received increasingly attention from robotic communities. A variety of approaches has been proposed to achieve this goal. We can classify these methods by their assumption on the robot working environments.

In first category of the approaches, robots are assumed to work under static environments where no dynamics considered. Based on how the language commands are processed into executable action plans, these approaches can be divided into two sub-categories: logic-based statistics-based. An action means a preprogrammed or pre-trained action schema.

Logic-based methods translate linguistic commands into executable action plans based on a set of rules extracted from available prior knowledge. The earlies efforts to NL based robot control, SHRDLU, developed at MIT by Terry Winograd, uses a rule-based semantic parser to translate NL commands into actions to manipulate blocks in a simulated world [4]. A similar work is also presented in [5]. Lauria et al. use hand-code grammars to map navigational commands into action templates [6][7]. MacMahon et al. define a set of rules to process navigational commands into predicate-argument structure [8]. Brenner et al. use rules to map linguistic components to an action and its precondition and effect [9]. Dzifcak et al. use a heuristic-based parser that supports several combinatorial rules to translate NL commands into temporal logic expressions, which specifies task goals for action planning based on first-order dynamic logic [10]. Bollini et al. translate plain text recipe into states in a state-action space, and search for an action sequence which maximizes the reward function as the cooking plan [11].

Statistics-based methods employ data-driven techniques to learn the implicit mapping rules

from data instead of using hand-designed explicit ones. They differ in their probabilistic models (e.g. naive Bayes [12], support vector machine (SVM) [13], hidden Markov model (HMM) [14], conditional random field (CRF) [15], etc.), formal representations (e.g., predicate-argument structure [16], $\lambda-$calculus [17], graphical representation [18], customized templates [19], etc.) and features employed for model training. Huang et al. train a semantic parser based on naive Bayes to map NL commands into spatial description clause consisting of a figure, a verb, a landmark, and a spatial relation for navigational guidance [12][19]. Chen and Mooney train SVM classifiers to generate formal navigational plans according to observations of current states and language input [13]. Takano et al. use HMM to model robot body movements with relevant words [14]. Misra et al. build a CRF model which is able to infer implicit steps that are not specified explicitly in NL [15]. She et al. employ a general-purpose CCG semantic parser to process NL and generate action frames with predicate-argument structure for manipulation tasks [16]. Matuszek et al. train a probabilistic CCG parser to translate NL navigational commands into a subset of $\lambda-$calculus representation which can include control loop structures similar to that in generic programming language. Kollar et al. propose Generalized Grounding Graphs for object grounding and action selection according to the next language instruction [18].

In addition, there is a tendency to use hybrid approach for NL based control. Tenorth et al. employs Stanford parser [20] for syntactic parsing of NL commands retrieved from the Internet and predefined rules to map parsed results into customized action frames [21]. Lisca et al. use Markov logic network to capture the relation between language and predefined action schemas, which combines first-order logic and probability theory together in a formalism [22][23].

The second category of approaches assume robots working in a structured environment, where constrained dynamics is considered and status evolution is totally predictable. Kress-Gazit et al. translate structured English commands into linear temporal logic formulas and later synthesize a nondeterministic automata to control robot behaviors [24][25][26]. In their following work, the constraint on structured English is relaxed to NL [27]. The proposed approach is able to recognize failures in controller synthesis due to incomplete or conflicting information from NL

commands [28][29]. We propose a framework to translate NL instructions into target state set, which corresponds to a subset of states in a finite state machine built in advance. This framework can ensure safety and stability of robot behaviors [30]. However, the above presented approaches are task-dependent. The control models have to be rebuilt when applied to a new environment or task.

1.3.2 Natural Language based Robot Programming

Natural language programming has been the subject of much discussion and conjecture for a long time [31][32]. Some researchers asserted that NL is the most desirable programming language because of its versatility and ease of use for humans, no professional training requirements on users, and the possibility of eventually using speech recognizers for input. We classify the NL based robot programming into offline and online programming.

On the offline programming side, the work done by Naval Post-graduate School could be looked as the earliest attempts for NL programming [33][34][35]. The goal of this project was to develop a system that would generate a simulation program after a simple queuing English dialogue with a user about a simple queuing problem. IBM's Thomas Watson Research Center developed NL automatic programming system for business applications with dialogues [36]. Later in 1977, Lieberman and Wesley from IBM Thomas Watson Research center developed a system which uses structured language as input to program mechanical assembly tasks [37]. In 1981, Miller from IBM Thomas Watson research center, did a research work about the possibilities of natural language programming based on the temporal state of the art technologies [38]. He summarized there were three major obstacles for the natural language programming. The first one is the vast difference between the NL and programming languages. NL is much richer in grammar and vocabulary than that of programming languages, which makes it harder to transfer NL into programming language representations. The second is language grounding, i.e., to decide the meaning of words in NL instructions, especially to what extent should the word meaning determined based on immediate and previous linguistic input. The third one is that the conversion from

NL to programming language is heavily relied on shared experience and world knowledge. At the end, he provided two possible solutions: one is to implementing a NL interface subject to several constraints, the other is to modify programming languages to include more NL features. These early works focused on translating language commands into robot understandable form. They use structured language with limited vocabulary.

In recent years, the constraint on structured language is relaxed. Vogel and Jurafsky use reinforcement learning to train a policy for navigational action enablement in accordance with language input and the robot's current state [39]. Branavan et al. present a similar work [40]. Shimizu and Haas develop a Markov model for navigational guidance with NL instructions [41]. Branavan et al. train a CRF model trying to learn the preconditions and postconditions of actions. Matuszek et al. treat the robot programming as a machine translation problem and learn a log-linear model to translate NL commands into a formal representation named Robot Control Language [17]. Artzi and Zettlemoyer train a weighted linear CCG to transform linguistic input into typed $\lambda-$calculus [42]. Stenmark and Nugues translate NL instructions into a sequence of predicate-argument frames and encode the frame sequence as a new skill [43][44]. Quirk et al. use log-linear models to translate If-This-Then-That recipes into executable code [45]. Campagna and Ramesh train long short-term memory (LSTM) recurrent neural network for trigger-action programming.

Online programming allows the users to program new skills/tasks to a robot without terminating it. Compared with offline robot programming using NL, less work has been proposed for online robot programming. Lauria et al. propose to represent online programmed behaviors as combinations of learned actions [7]. Rbyski et al. encode new skills through recognizing the cause and effect of a navigational task and mapping them into a action template [46][47][48]. Cantrell et al. develop a similar approach [49]. Kollar et al. propose to use dialogues and demonstration to learn new landmark knowledge on the fly [50].

1.4 Contributions

The contributions for this dissertation can be summarized into two aspects: control and programming. For the control aspect, the developed control approach can also be applied to other DES based systems. On the programming aspect, the online programming approach provides a friendly interface and can contribute a lifelong learning.

Based on the DES inspirations, the proposed control approach can achieve the following two merits, which distinguishes it from other existing NL based robot control methods. First, it is a correct-by-design approach and takes environmental dynamics into consideration, which ensure safety and stability of robot behaviors under dynamics environments. Second, the hierarchical structure alleviates state explosion problem for large and complex scenarios and can deal with requests of different levels of abstraction. We derive the conditions of stability in the sense of Lyapunov for hierarchical structure. Experimental results show that the robot is able to accomplish assigned tasks under uncertainties from its environment.

In addition, we propose to generate scene representations from NL instructions to control robot behaviors, which can be applied to scenarios where symbolic representation may fail, such as NL based robotic drawing, painting, polishing, etc. The robot uses knowledge retrieved from the Internet to generate scenes conditioned on NL, and is capable to deal with possible uncertainties that can cause failures of scene synthesis, such as contaminated data from web or detection error. This is achieved through the proposed scene generation framework that integrates with the control approach developed in previous sections, and experimental results show it helps to increase success rate of scene generation and accelerate the process. Human evaluation show that the generated scenes have higher recognizability and better alignment with the input language descriptions over the state-of-the-art approaches.

The success of robot application in dynamic environments relies on correct robot state estimation. Gripper status is an important component of robot states. We propose correlation based approach for slippage detection to estimate and monitor grasp status. The proposed approach is able to detect slippage velocity without using customized sensor designs or limit to specific sensor

signal type.

On the programming side, this dissertation presents a hierarchical programming approach that allows a robot to learn new skills using acquired skills through dialogues and demonstrations. This approach mimics the teach and learning process in which an adult teaches his/her children, and thus is intuitive and friendly to technically untrained users. Moreover, it only requires one-shot training, which is time-efficient and laborless.

1.5 Outline of This Dissertation

The dissertation is divided into three parts: natural language based control, and natural language based programming, and experimental study for slippage detection. Chapter 2 and Chapter 4 present NL based robot control approach, Chapter 3 discusses NL based robot programming method, and Chapter 5 shows the data correlation based approach for slippage detection. Chapter 6 concludes the dissertation and outline future research work. The specific contents for each chapter are discussed as follows.

Chapter 2 presents the NL based control approach. The mathematical model of the control model will be first discussed. After that, property analysis for the model will be elaborated. Then a hierarchical control framework is presented to process commands of different logic depth. With the hierarchical controller, the stability analysis is performed based on behavior consistency. Finally, we present the implementation details and experiment results.

Chapter 3 introduces the hierarchical online programming approach. First of all, comparison of existing programming approaches is reviewed. After that, the interactive programming process is discussed. Finally, we present the testing results on an industrial application.

Chapter 4 discusses the proposed control framework that uses scene representation to guide robot behaviors. First, the overall framework is presented. Then, the modeling to deal with possible uncertainties and the problem is formulated in DES form. Finally, we present experimental results on system performance comparison and generated as well as drawn scene evaluation using standard metrics and human studies.

Chapter 5 introduces the data correlation based slippage detection method. First of all, data correlation approach is introduced. Then the experiment implementation details are presented. Finally, experimental results under static and dynamic environment are presented to validate the proposed approach.

Chapter 6 summarizes the dissertation and outlines future works.

CHAPTER 2

NATURAL LANGUAGE BASED ROBOTIC BEHAVIOR CONTROL

2.1 Introduction

Using natural language (NL) to control robot behaviors makes it easier for people to employ robots, especially for novice users. One of the key challenges is to make robots following NL instructions successfully. To pave the way for the goal, research has been focusing on translating instructions in natural language into formal representations [17], controller synthesis [26], task planning [51], requesting information or help from human partner [52][53], NL programming [54], and knowledge representation [55], etc. However, existing works assume a static working environment. While in practical applications, robots instructed by natural language work in a dynamic shared common area with people. Modeling environmental dynamics into NL-based robot control has received few attention.

Figure 2.1 depicts a robot working in highly dynamic environments due to presence of people. In addition, NL instructions are usually vague and underspecified, in that only the intent of the instructor is provided, and details of implementation are ignored. To successfully accomplish the assigned task in a dynamic scenario, a robot must possess the capability to deal with the unforeseen events from its surroundings.

in this paper, we propose a method to model environment dynamics into high-level robot behavior model. A robot is modeled as a discrete event dynamic system (DEDS), where the system behaves in response to external events usually happened at possibly unknown and irregular time. Both NL instructions and sensory information are projected into the same state space for the formulation of an action plan that can lead the robot to reach desired target state set. The environment dynamics is considered as the unforeseen events occurred in the working environment that cause abrupt changes in the task execution status. The synthesized robot behavior model captures both the robot dynamics and environment dynamics. In addition, three metrics are used to evaluate and

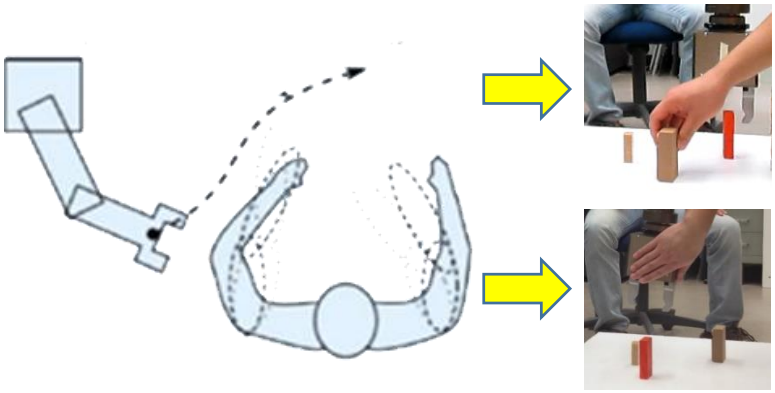


Figure 2.1: Illustration examples of possible environment dynamics when robots coexist with human co-workers in a shared working area. The upper right subfigure shows an example where the position of the brown block (target) is changed as the robot is approaching it. The lower right subfigure shows an example in which the robot is blocked by a human partner.

refine robot behaviors to guarantee the assigned tasks can finally be accomplished.

The main contributions of this chapter are as follows:

- We propose a novel approach to capture environmental dynamics into robot model controlled by natural language descriptions. The modeling method improves system adaptivity to dynamic environments. It allows robots to cope with unexpected events that cause abrupt changes on the task progress.

- Our approach can complement the underspecification in natural language instructions. It allows the robot to deliver the task goal without requiring the instructor to enumerate all the possible scenarios and details in language commands.

- The proposed approach guarantees the generated behaviors can achieve desired task goal finally. Our method synthesizes an analyzable robot model satisfying controllability, nonblockingness, and stability to guarantee its dynamic performance. Controllability makes the robot be able to process unforeseen environmental events; nonblockingness guarantees the task goals are accessible; stability guarantees convergence to the final task goals in presence of finite unforeseen dynamics.

- The hierarchical structure of the approach helps to process concepts of different logic depth and alleviates state explosion problem.

The rest of this chapter is organized as follows. Section 2.2 presents the basics of supervisory control theory. Section 2.3 illustrates the overall pipeline of developed control framework. Section 2.4 elaborates the design for both the plant and controller. Section 2.5 provides experimental configurations and results.

2.2 Preliminary on Supervisory Control

Different from the continuous variable dynamic systems (CVDS), where the system behaviors are governed by physical laws and modeled by differential equations,the robotic system is modeled as a discrete event system (DES), where the system behaviors evolve in accordance with rules of operation or algorithms. Supervisory control is one of the modeling theory of DEDS, which focuses on maintaining the system behavior described by formal language [56].

In the context of supervisory control, a DES is modeled using a five-tuple deterministic finite automaton:

$$G = \{Q, \Sigma, \delta, Q_0, Q_m\}$$

G denotes the plant to be controlled.

Q is a finite nonempty set of states abstracted from the robotic system, denoting the status of the plant.

Σ represents the set of events, in which the elements drive the system to evolve from one state to another. The events of Σ can be classified into two parts: Σ_c in which events can either be enabled or disabled, and Σ_{uc} where events are set to be always enabled by default. By continuously enabling actions from Σ, actions are implemented one by one and thus a state trajectory as well as an action sequence will be produced. Since the language generated by an automaton is comprised of elements from event set Σ, it is also called an alphabet.

$\delta : \Sigma \times Q \rightarrow Q$ is the transition function that captures conditional state changes.

Q_0 : elements of Q_0 denote the initial states from where a language or a system starts.

$Q_m \subset Q$: a subset of the state set Q, called the set of marker states. Usually, Q_m are used to represent the successfully completed tasks.

The behaviors of an automaton G can be described by the set of the output event trajectories, also called strings or languages. G is called a language generator.

Let Σ^* represents the set of all the finite strings s comprised of elements from Σ, including the empty string ε. The language $L(G)$ is the set of all event trajectories that are physically possible for the plant

$$L(G) = \{s : s \in \Sigma^*,\ \delta(q_0, s) \textit{ is defined, } q_0 \in Q_0\}.$$

The marker language $L_m(G)$, describes all the event sequences that can reach the marker states

$$L_m(G) = \{s : s \in \Sigma^*,\ \delta(q_0, s) \in Q_m,\ q_0 \in Q_0\}.$$

Supervisory control scheme separates the open loop dynamics (plant) from the feedback control. This separation is reminiscent of the feedback control scheme adopted in CVDS. The controller is modeled as a pair:

$$\mathfrak{R} = (S, \psi)$$

where

$$S = \{X, \Sigma_s, \xi, x_0, X_m\}$$

is a deterministic automaton with state set X, event set Σ_s, transition function ξ, initial state x_0 and marker state $X_m \subset X$. The ψ is a total function that maps supervisor states x into control patterns, which is defined as:

$$\phi(x)(\sigma) = \begin{cases} 1, \text{ for each } \sigma \in \Sigma_{uc} \text{ or } \sigma \text{ is allowed at x} \\ 0, \text{ for } \sigma \text{ not allowed at x} \end{cases}$$

The supervisor is synthesized by regulating the plant behaviors with physical constrains and task specifications. The sets of its behaviors, represented by formal language, are denoted as $L(S/G)$ and $L_m(S/G)$, respectively.

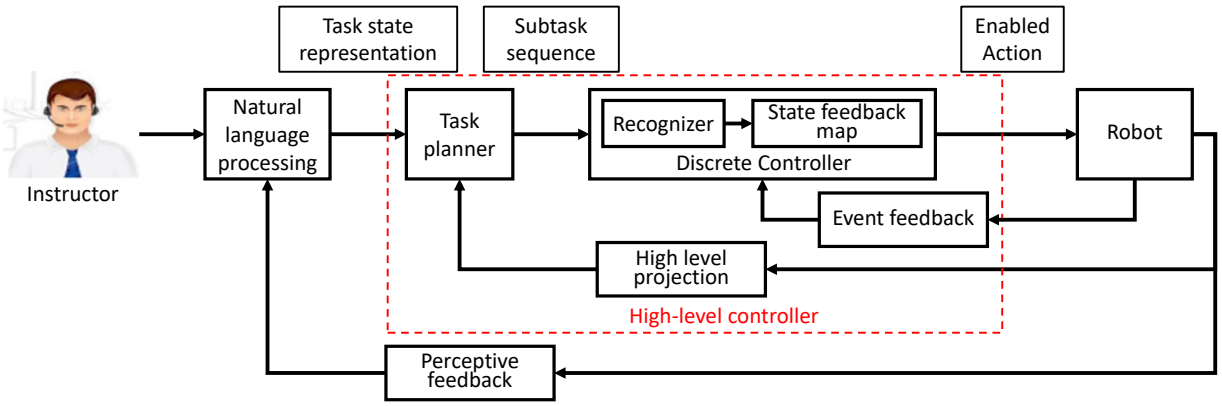


Figure 2.2: Overall pipeline of the proposed approach.

2.3 Proposed Framework

The proposed framework is shown in Figure 2.2. Given NL commands, the robot generates behaviors to accomplish the assigned task through the sequence of following modules:

- **Natural language processing** (NLP) module processes input NL instructions into grounded semantic representations, which carry necessary task specifications. At first, NL are parsed into formal representations. The linguistic components are then mapped to physical items using graph-based grounding with sensory information dispatched through perceptive feedback.

- **Task planner** takes grounded goal configuration as input and generates an ordered sequence of subtasks. In this work, operations on each object is considered as a subtask. Manipulation priority of each subtask is determined by their dependency relations [51].

- **Discrete controller** takes each subgoal configuration as input and generates action sequences in accordance with the current task status. The introduced event feedback allows the robot to react to environmental uncertainties during task implementation.

The system framework has three loops. The innermost loop is the action level, which monitors and controls the behaviors of a robot and responds to unforeseen events. The intermediate loop is the task level, the status of the system is projected into the state space of task planner through high

level projection. Completion of current subtask will stimulate the task planner to issue the next subtask. The outermost loop is perceptive feedback, through which the NL processing module grounds linguistic input with physical instances.

2.4 Natural Language Control Approach

2.4.1 Natural Language Processing

To convert the NL instructions into a robot understandable specification, the robot must identify the underlying linguistic structure of the commands and convert it into a formal representation. In this work, the robot first parses the user's instructions through a combinatory categorial grammar (CCG) parser [57]. Semantic representations including action and their roles (*Theme* and *Destination*) are generated.

Then a graph-based grounding algorithm is used to map the roles of actions to objects in the robot working environment [58]. A dialogue graph can be generated from the user's NL instructions, where nodes represent objects and edges denote spatial relations between objects. Likewise, a similar graph can be generated from vision data. By comparing the dialogue graph and the vision graph, the reference resolution becomes a graph matching problem to find a one-to-one mapping between the nodes in the two graphs. Both the dialogue graph and the visual graph can have errors due to insufficiencies of the NLP and vision recognition module. An inexact graph matching algorithm is employed to find the best match [59]. This grounding process is in essence to ground action parameters. Finally, the action and its grounded parameters are transformed into a state representation as the grounded semantic representation. This state represents the goal status of the robot and its environment when completing the task.

2.4.2 Discrete Controller

In practical application scenarios, a robot may fail due to uncertainties which can be caused by sensor noises, model errors, and environmental dynamics. This section focuses on solving environmental uncertainties that haven't been considered in existing NL based control approaches.

To tackle dynamics from the environment, it is necessary but nontrivial to model the environmental dynamics into the robot behavior model. The robot should be able to respond to language commands and environmental uncertainties, which can both be viewed as discrete events. The basic behaviors of the robot are modeled as automata as introduced in Section 2.2 with domain knowledge. For the rest of this section, We use an example in robotic manipulation for illustration of the discrete controller modeling.

Consider a robotic arm performing grasp-move-place tasks. The arm is comprised of two functional modules, the arm body and the gripper. The arm moves the gripper to the target position with appropriate orientation, and the gripper performs open and close operations to grip and release objects. Various combinations of these primitive actions form diverse manipulations. Figure 2.3 shows the model for the arm and the gripper represented as deterministic automata, respectively.

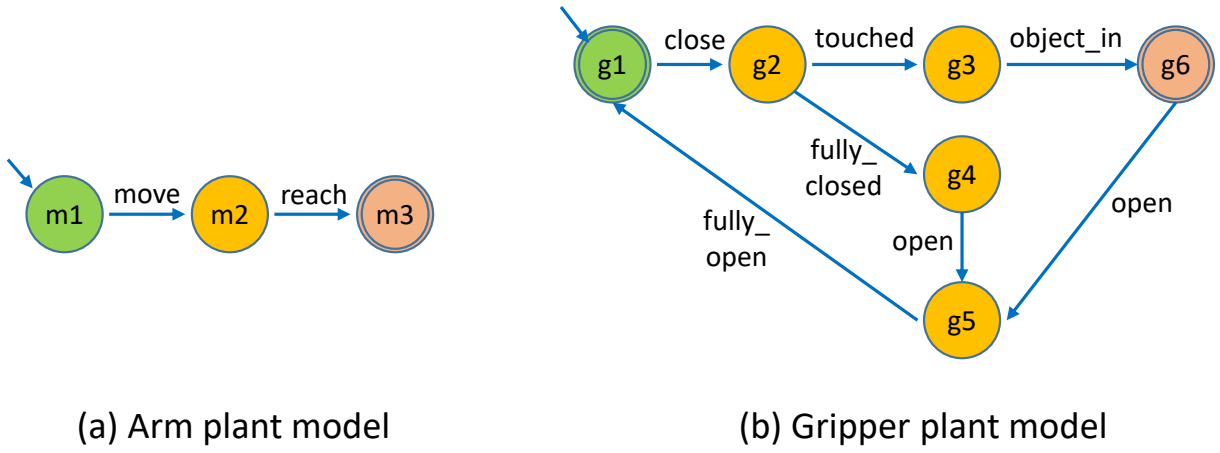


Figure 2.3: The deterministic automata models corresponding to each functional module of the arm. (a) model of the arm body to control position and orientation (G_{arm}). (b) model of the gripper to control gripper open-close status ($G_{gripper}$). The states in green with an entry arrow denote initial states. The states with double circles are marker states. Yellow states represent intermediate states. *m* represents *move*, and *g* represents *gripper*.

Shuffle the two automata using parallel composition, i.e.,

$$G_{robot} = G_{arm} \| G_{gripper}$$

the shuffled product captures nominal behavior of the robot, which can work under static environment without environmental uncertainties involved. In a dynamic environment, a robot has to cope

with events that cause abrupt changes of the task progress state. This abrupt change can be captured using an uncontrollable event *exception* attached to each pair of state transitions. This event represents the unforeseen and uncontrollable dynamics of the environment due to other partners' activities that alter the task execution status and are usually unpredictable. However, the effect of these events can be characterized: they cause abrupt state jump of the task execution, and this change can be represented using robot state change. Table 2.1 shows a complete list of primitive actions from the event set.

Table 2.1: List of primitive events.

Primitive Events	**Controllability**	**Description**
move (pos, ort)	controllable	move the arm to the specified pose
reach	uncontrollable	Target pose has achieved
open	controllable	Open the gripper
close	controllable	Close the gripper
touched	uncontrollable	The inner sides of gripper fingers have touched something
object_in (obj)	uncontrollable	Object grasped in gripper
fully_closed	uncontrollable	Gripper fully closed
fully_open	uncontrollable	Gripper fully open
exception	uncontrollable	Denote unforeseen events

Actions are parameterized according to the grounding of object and their attributes.

After adding the event *exception* to the plant model, we denote the model as G'_{robot}, which has 18 states and 561 transitions. It represents unsupervised behaviors of the robot and some of which are unlikely or not expected to happen. To guarantee reasonable and legal behaviors of the robot, a discrete controller (e.g., supervisor) should be synthesized to remove the impossible or illegal state transitions in accordance with physical constraints and domain application knowledge. To this end, we propose the following specifications that should be satisfied by the robot:

1. The gripper's open and close status can only be changed by environmental uncertainties.

2. The movement of the arm and the gripper are asynchronous, i.e., the grippers status remains unchanged when the arm is moving, and vice versa.

3. To guarantee that the target state set can finally be achieved, the generated supervisor should be refined to be controllable, nonblocking, and asymptotically stable. The checking criteria for the three properties will be illustrated in the next following.

Controllability characterizes a system's capability to satisfy specified requirements under uncontrollable events (e.g., the position of a workpiece changed when the arm is approaching it; the arm is blocked by a person walking by, etc.). The following gives the definition of controllability in terms of formal language [56].

Controllability: Let K and $L = \overline{L}$ be two arbitrary languages over event set Σ. The overbar of L denotes a language set of all the prefix of L, including L itself. K is said to be controllable with respect to L and Σ_{uc} if $\overline{K}\Sigma_{uc} \cap L \subseteq \overline{K}$.

If a supervisor is controllable, it is able to drive the system to its target state even encountering unforeseen events from the environment. Algorithms to synthesize a controllable supervisor can be found in [60].

Nonblocking characterizes the property of a system to avoid either being stuck at a state (deadlock) or being trapped in a subset of states (livelock). The following gives the definition and criteria of nonblocking property [56].

Nonblocking: Let $L(S/G)$ and $L_m(S/G)$ represent the controlled behavior and the marked behavior by the supervisor, respectively. A supervisor satisfying $\overline{L_m(S/G)} = L(S/G)$ is said to be nonblocking.

Nonblockingness of a supervisor guarantees the task state set is accessible. If a supervisor fails to be nonblocking, the criteria can help the designer to identify the blockingness and remove them in advance.

Stability. Nonblockingness only guarantees the accessibility of the state set. To check whether the task state can finally be achieved or not, we further analyze the stability of the supervisor, which captures the ability of a system to converge to a steady state. The work by Passino et al. [61] provides the foundation for analyzing the stability as well as asymptotical stability in the sense of

Lyapunov for system modelled by automata. In this chapter, we choose a Lyapunov function

$$V(q) = \rho(q, q_m) \tag{2.1}$$

where

$$\rho(q, q_m) = \inf n : \xi(s, q) = q_m, s = \sigma_1\sigma_2...\sigma_n, \sigma_i \in \Sigma \tag{2.2}$$

and $q \in Q$, $q_m \in Q_m$ for system stability analysis.

A discrete event controller can be synthesized by applying the specifications to refine the behaviors of the plant model, which results in a deterministic automata with 15 states and 27 transitions (as shown in Figure 2.5). It can be seen that the synthesized supervisor satisfies controllability and nonblockingness in the presence of uncontrollable events. At each state of the supervisor, it allows all the permissive legal behaviors to happen. In addition, it is straightforward to prove Lyapunov asymptotical stability of the synthesized controller under finite occurrence of event $exception$. Operations on automata are implemented using DESUMA [62].

Each state transition is a triplet $q_i\sigma_{ij}q_j$, where $q_i, q_j \in Q$, and $\sigma_{ij} \in \Sigma$. The state q_i is the initial state in which the action σ_{ij} can take place. It is the precondition for action σ_{ij}. The state q_j is the final state, resulting fo the action σ_{ij} applied to the initial state. For a sequence of actions to be realizable, the final state of one action must be compatible with the precondition of the next one. Backward search algorithm is employed to search for an action sequence to drive the system from its initial state to the desired target state with the least number of actions [63]. When the task execution is interrupted by an unexpected events that causes an abrupt state change, the robot will update its internal state and replan a new action sequence.

Figure 2.4 summarizes the modeling pipeline presented in the above.

2.4.3 Task Planner

The task planner residues at a higher level to deal with tasks presented in more abstract utterance. It is modeled as a Moore automata [64] that is a finite state machine whose outputs depend on

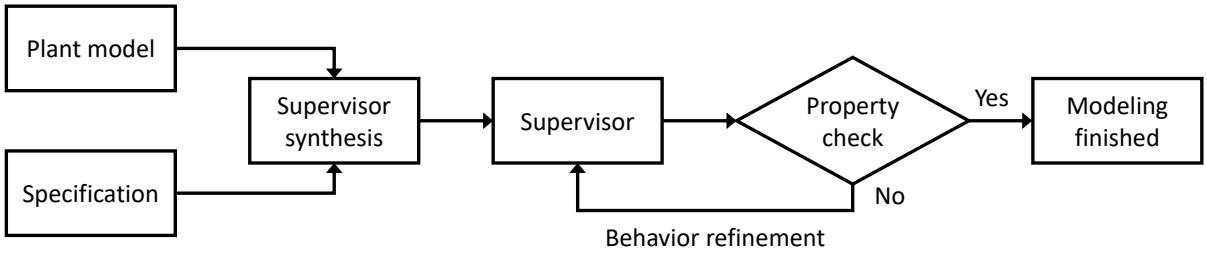


Figure 2.4: Discrete controller modeling pipeline.

only the present state. A Moore machine can be describe by a seven-tuple $(Q, \Sigma, \delta, q_0, O, X, Q_m)$, where O is a finite set of symbols denoting output events, X is the output transition function where $X : Q \rightarrow O$. The rest of the components have the same explanation as presented in preliminary of supervisory control.

Rather than denoting a single robot state, each state in the task planner represent one task configuration, which trying to capture the whole picture related with the task. Each state is represented as a $n \times n$ matrix. Each element in the matrix captures the dependency relation between two items or events. n is the number of items/events appeared in the natural language commands. The elements of the matrix are defined based on logical dependency, i.e., spatial dependency or temporal dependency. The matrix is named as dependency relation matrix (DRM). The value of the relation reflect relative priority in manipulation or execution. For a DRM based on spatial dependency, the element at the intersection of the i_{th} row and j_{th} column of the DRM can be defined as:

$$[DRM]_{ij} = \begin{cases} 0, \text{relation with the object itself} \\ 1, i^{th} \text{ object and } j^{th} \text{ object are not directly dependent} \\ 2(-2), i^{th} \text{ object is directly on (under) } j^{th} \text{ object} \\ 3(-3), i^{th} \text{ object is directly in (surrounding) } j^{th} \text{ object} \\ ..., \text{other relations.} \end{cases} \tag{2.3}$$

While, for a DRM based on temporal dependency, the element at the intersection of the i_{th} row

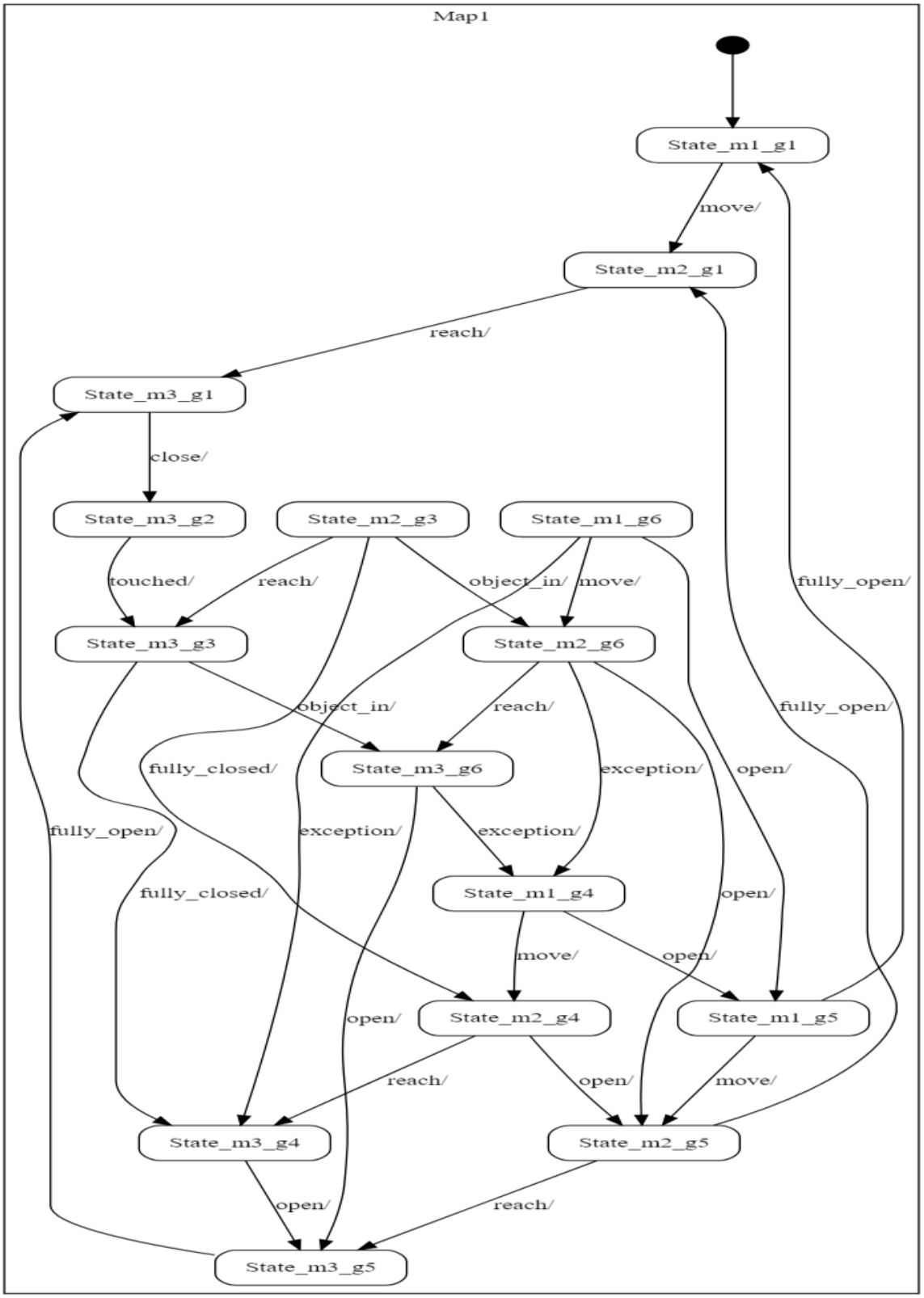


Figure 2.5: Synthesized discrete controller.

and j_{th} column of the *DRM* can be defined as:

$$[DRM]_{ij} = \begin{cases} 0, \text{relation with the event itself,} \\ 1, i_{th} \text{ event and } j_{th} \text{ event do not have direct logical preference} \\ 2(-2), i_{th} \text{ event is directly logically behind (ahead of) } j_{th} \text{ event} \\ ..., \text{other relations.} \end{cases} \tag{2.4}$$

Each object/event is viewed as a subtask. In this work, the NLP module translates NL instructions into a set of states corresponding to the desired final states and organizes them in a matrix form. This target state set only captures the relations that are explicitly stated in the language commands. *DRM* plays two important roles. First, considering the fact that the instructions may be issued in a mixed order, and simply following the order can cause failures of tasks. To figure out a correct order of subtasks, we derive a complete goal configuration represented in *DRM* (denoted as DRM_{goal}) that captures all the dependency relations between each pair of items/events using following transitivity rules:

$$\begin{aligned} &[DRM]_{ij} = relation \\ &\Rightarrow \begin{cases} [DRM]_{ji} = -relation \quad \text{if } relation \neq 1, \\ [DRM]_{ji} = relation \quad \text{if } relation = 1 \text{and } [DRM]_{ji} = 1 \\ [DRM]_{ij} = -[DRM]_{ji} \quad \text{if } relation = 1 \text{and } [DRM]_{ji} \neq 1 \end{cases} \end{aligned} \tag{2.5}$$

where $relation \in R$, $R = \{\ldots -2, 1, 0, 1, 2, \ldots\}$ depends on the relation definition.

$$\begin{cases} [DRM]_{ij} = relation \\ [DRM]_{jk} = relation \end{cases} \Rightarrow [DRM]_{ik} = relation \tag{2.6}$$

In the meanwhile, we form the current configuration in *DRM* form according to the sensory information, denoted as denoted as DRM_{temp}. It keeps updating itself with the sensory feedback.

The DRM_{temp} plays two important roles. Subtract DRM_{goal} by the initial DRM_{temp}, we get a difference between the target and initial configurations, denoted as DRM_{error}.

$$DRM_{error} = DRM_{goal} - DRM_{temp} \tag{2.7}$$

The smaller of the relation value, the higher priority of the corresponding subtask. Subtasks with higher priorities should be implemented earlier than others that have lower priorities. By summing up all the relation values for each item/event against other items/events, as represented by Equ. 4.4, we can obtain a vector and its values are used as criteria for subtask planning. The order of the subtasks is obtained through sorting the elements of the vector in an ascending manner.

$$f(DRM_{error}) = \left[\sum_{j=1}^{n}(a_{1j}), \quad \sum_{j=1}^{n}(a_{2j}), \quad \cdots, \quad \sum_{j=1}^{n}(a_{nj})\right]^T \tag{2.8}$$

The second role of DRM is to monitor the task execution. After completion of each subtask, the DRM_{temp} should converge to the DRM_{goal}. We use matrix 2-norm to measure the similarity between the DRM_{goal} and DRM_{temp}, as shown by Equ. 2.9.

$$V(k) = ||DRM_{goal} - DRM_{temp}||_2^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}|g_{ij} - t_{ij}|^2 \tag{2.9}$$

where $g_{ij} = [DRM_{goal}]_{ij}$, $t_{ij} = [DRM_{temp}]_{ij}$, and $k \in N^+$. If some abrupt events occurred during the task implementation that cause $\Delta V = V(k) - V(k-1) > 0$, the system will stop for task replanning or for further diagnosis.

Each event from the planner output event set Σ_{output} has the form $operation(object, state)$, where *object* denotes the item to manipulate, *state* represents the target state. This event passes the goal state information to the discrete controller of the next level. The environmental dynamics modeling and behavior refining can be implemented in a similar way as presented in Section 2.4.2

2.4.4 Lyapunov Stability Analysis of Hierarchical DES

The task planner and the discrete controller form a hierarchical structure of DES. Even though each module is guaranteed controllable, nonblocking, and stable at design phase, this doesn't ensure

the stability of the entire behavior. In this section, we explore the condition to guarantee the stability of the robot's behaviors. Firstly, the hierarchical structure and some concepts relating to hierarchical behavior will be introduced. Then, we derive the conditions to ensure Lyapunov stability of hierarchical DESs.

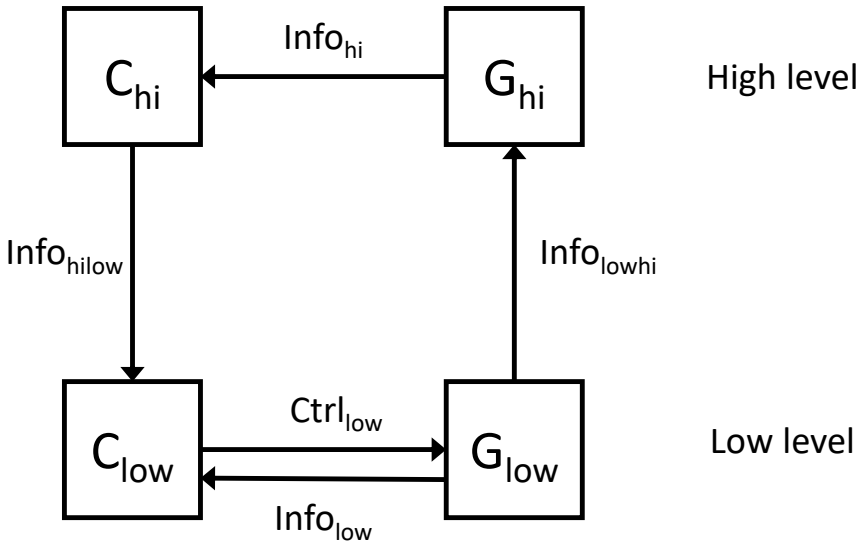


Figure 2.6: A two-level hierarchical structure.

Figure 2.6 shows a two-level hierarchical system structure [2]. Plant G_{low} and controller C_{low}, along with a high-level plant G_{hi} and controller C_{hi}. C_{low} is the actual plant to be controlled by C_{low}, while G_{hi} is an abstract, simplified model of G_{low} that is employed for decision-making in an ideal world of C_{hi}. The model G_{hi} is updated every so often via the information channel labeled $Info_{lowhi}$ (information-low-to-high) to G_{hi} from G_{lo}. Alternatively, one can interpret $Info_{lowhi}$ as carrying information sent up by G_{low} to G_{hi}. Another information channel, $Info_{low}$ (low-level information), provides conventional feedback from $G_{low}$to its controller C_{low}, which in turn applies conventional control to G_{low} via the control channel labeled $Ctrl_{low}$ (low-level control). Returning to the high level, we consider that G_{hi} is endowed with control structure, according to which it makes sense for C_{hi} to attempt to exercise control over the behavior of G_{hi} via the control channel $Ctrl_{hi}$ (high-level control), on the basis of feedback received from G_{hi} via the information channel $Info_{hi}$ (high-level information). In actuality, the control exercised by C_{hi} in this way is

only virtual, in that the behavior of G_{hi} is determined entirely by the behavior of G_{low}, through the updating process mediated by $Info_{lowhi}$. The structure is completed by the command channel $Info_{hilow}$ linking C_{hi} to C_{lo}. The function of $Info_{hilow}$ is to convey C_{hi}'s high level control signals as commands to the C_{lo}, which must translate these commands into corresponding low-level control signals which will actuate G_{low} via $Ctrl_{low}$. State changes in G_{low} will eventually be conveyed in summary form to G_{hi} via $Info_{lowhi}$. G_{hi} is updated accordingly, and then provides appropriate feedback to C_{hi} via $Info_{hi}$. In this way the hierarchical loop is closed. The forward path sequence $Info_{hilow} \rightarrow Ctrl_{low}$ is conventionally designated 'command and control', while the feedback path sequence $Info_{lowhi} \rightarrow Info_{hi}$ could be interpreted as 'report'.

The high level and low level represent robot behavior models with different abstraction level. The behavior consistency of hierarchical structure system plays a critical role in successful task execution and system stability. Next we will introduce a few concepts relating behavior consistency and derive the system stability in the sense of Lyapunov.

Output-control-consistenct (OCC): a generator G is a Moore automaton. It is said to be output-control-consistent if there is no ambiguity in the controllability of its output event.

Strict-output-control-consistent (SOCC): a generator G is said to be strict-output-control-consistent if it is output-control-consistent, and it can control the enablement and disablement of its output independently.

Hierarchical consistency: a pair (G_{low},G_{hi}) is said to possess hierarchical consistency if G_{low} is SOCC, and $E_{hi} \subseteq L(G_{hi})$ be nonempty, closed, and controllable.

When hierarchical consistency holds for the pair (G_{low},G_{hi}), every high-level task (represented by a choice of E_{hi}) will be successfully decomposed and executed in the hierarchical control loop.

Based upon work on the discerte Lyapunov stability [61], we achieve the following results.

Proposition 2.4.1 *If the pair of* (G_{lo}, G_{hi}) *possesses hierarchical consistency, and the higher level system* G_{hi} *is stable in the sense of Lyapunov, then the lower level system* G_{low} *is stable in the sense of Lyapunov.*

Proof: To prove the proposition, firstly, we will prove the existence of invariant set in G_{low}, denoted as X_{m_lo}.

For $x_{0\_hi} \in X_{m\_hi}$, since G_{hi} is stable $\Rightarrow$

$$X(x_{0\_hi}, E_k, k) \in X_{m\_hi}$$

for all E_k such that $E_k E \in E_v(x_{0\_hi})$ and $k \in N$ where E is an infinite event sequence.

For $\forall s \in E_v(x_{0\_hi})$, according to hierarchical consistency,

$$\exists\ s_{low} \subseteq L_{low}$$

$$s_{lo} = \{t \in L_{low} | \theta(t) = s\}$$

$\Rightarrow$

$$\exists\ X_{0\_low\_s} = \{x_{0\_low\_s} | \exists\ t \text{ such that } \delta(x_0, t)!, t \in s_{low}, x_0 \in X_{low}\}$$

$\Rightarrow$

$$X_{m\_low} = \bigcup_{i=1}^{i=n} X_{m\_low\_i}$$

$\Rightarrow$

$\exists$ invariant set X_{m_low}, for $x_0 \in X_{m\_low}$, there is $X(x_{0\_low}, E_{k\_low}, k) \in X_{m\_low}$, where E_{k_low} such that $E_{k\_low} E_{lo} \in E_{k\_low}(x_{0\_low})$.

Next is to prove the existence of $V(x)$, $S(X_{m\_low}, r_{low})$ and its nonincreasing property.

Select $V(x) = \rho_d(x, X)$, which denotes the distance between the state x and the set X.

For higher level controller G_{hi}, $S(X_{m\_hi}, r_{hi})$, $\exists\ c_1 > 0$, $c_2 > 0$ such that

$$V(x) > c_2 \Rightarrow \rho_d(x, X_{m\_hi}) > c_1 \text{ (in this case } c_1 = c_2)$$

Similarly, $\exists\ c_3$ and c_4 (in this case $c_3 = c_4$) such that

$$V(x) < c_4 \Rightarrow \rho_d(x, X_{m\_hi}) < c_3$$

where $V(X(x_0, E_k, k))$ is nonincreasing, for $x_0 \in S(X_{m\_hi}, r_{hi})$, $0 < r_{hi} < \rho_{hi}$.

For $x_{0_h}i \in X_{0_h}i$, $\exists\, \xi(x_0, s) \in X_{m_h}i$.

Assume

$$s = s's'',\ s' \in E_{v\_hi}(x_0),\ s'' \in T^*$$

such that

$$\xi(x_0, s') \notin X_{m\_hi}$$

$$\xi(x_0, s') \in S\,(X_{m\_hi}, r_{hi})$$

Since G_{hi} and G_{low} are hierarchical consistent,

$\Rightarrow$

$$\exists\, E_{t'} \subseteq L_{low}, E_{t'} = \{t' | \delta(x_0, t')!, \theta(t') = s'\}$$

$\Rightarrow$

$$\exists\, r_{low} \text{ such that } 0 < r_{low} < \rho_{low}$$

Set

$$r_{low} = min\{max\{\rho_d(x_{01}, \delta(x_{01}, t'_1))...\}, max\{\rho_d(x_{02}, \delta(x_{02}, t'_2))...\}...max\{\rho_d(x_{0m}, \delta(x_{0m}, t'_m))...\}\}$$

$\Rightarrow$

There exists a neighborhood $S\,(X_{m\_low}, r_{low})$ of the invariant set X_{m_low} such that we can find $V(x)$ satisfies the Lyapunov stability criteria, i.e., G_{low} is stable in the sense of Lyapunov.

Finally, it should be noted that even though the analysis is on a hierarchy of two levels DES, but this restriction is inessential.

Similarly, the following proposition also holds

Proposition 2.4.2 *If the pair of* (G_{low}, G_{hi}) *possesses hierarchical consistency, and the higher level system* G_{hi} *is asymptotically stable in the sense of Lyapunov, then the lower level system* G_{low} *is asymptotically stable in the sense of Lyapunov.*

2.5 Experiment Validation

2.5.1 Experimental setup

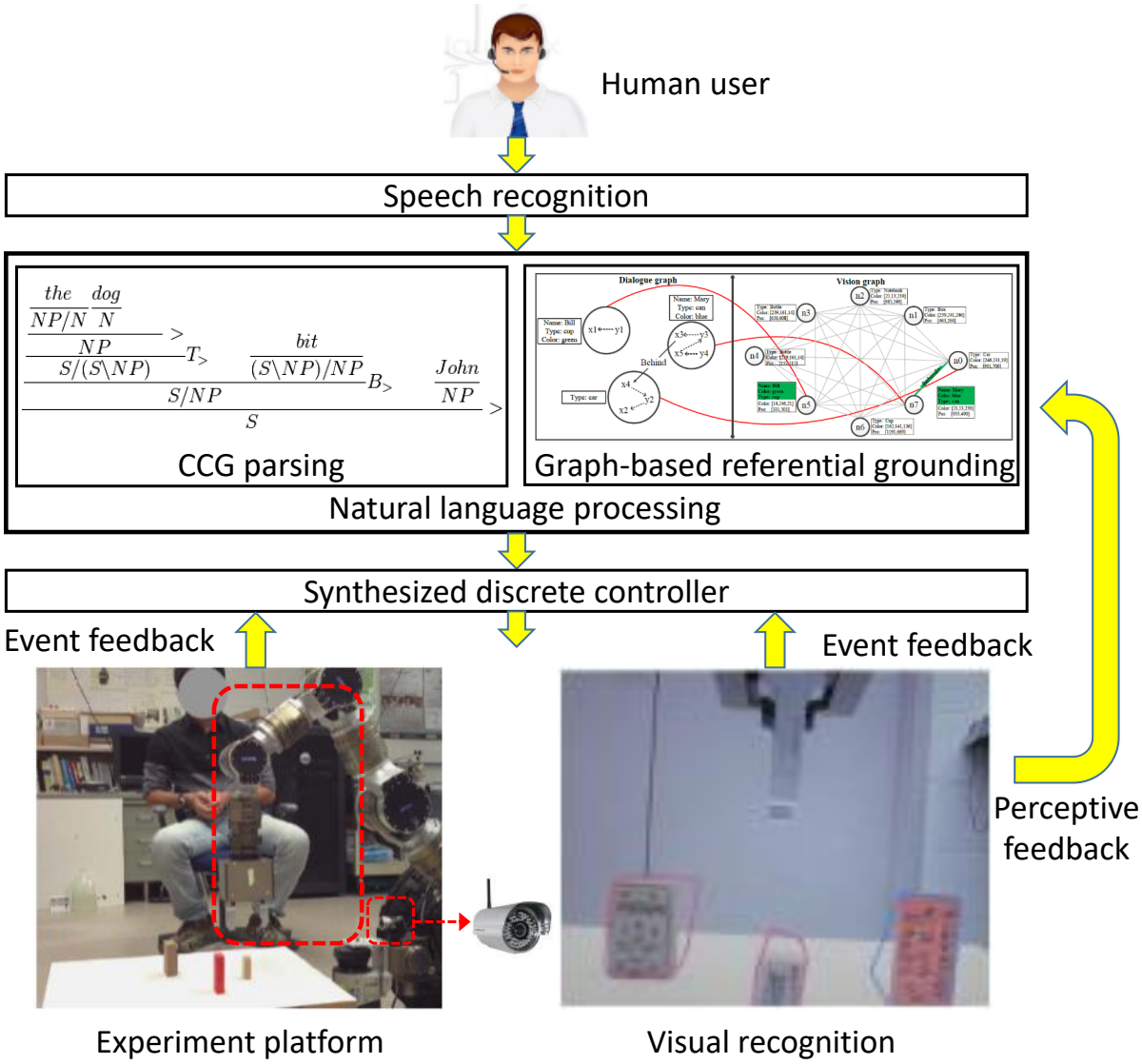


Figure 2.7: Experimental setup.

The experiment setup is configured as shown in Figure 2.7. The speech recognition is done by Dragon Speech Recognition software. A flat table is placed in front of the mobile base on which a 7 DOF robot arm is mounted, and a camera is mounted right under the arm to perform object recognition. We use MOPED for object recognition [65].

2.5.2 Experimental Results

2.5.2.1 Manipulation in Dynamics Environments

This section presents two examples that demonstrate the capability of dealing with unforeseen events from its working environment. In these two scenarios, a robot acts as a co-worker to manipulate parts. In the first scenario, the robot receives instruction "*pick up the big brown block on the table*", which is translated into target state representation

$$In(block(big, brown), gripper) \wedge At(block(big, brown), gripper)$$

The goal specification is used to search for a shortest event path driving the robot from the initial state to the specified target state. During approaching the big brown block, the position of the orange one and the target block are switched that causes failure of current action plan. The robot detects the state change of task progress through sensory information. The controller then replans a shortest action path from the current state to the target state. The snapshots in Figure 2.8 shows the implementation process.

In the second scenario, the robot is tasked to grasp the orange block. During the task execution, the arm body is blocked by a human co-worker. This situation may happen quite often if robots and people are collaborating or living together in a shared common area. The robot detects that its body is blocked and is forced back to the stationary state. After the co-worker leaves, the robot resumes to complete the task. Figure 2.9 shows the snapshots of the scenario.

2.5.2.2 DRM Guided Assembly

In this experiment we test the proposed *DRM* framework on a simulated assembly task. Instead of using MOPED for object recognition, the robotic system uses a RGBD sensor (Kinect) mounted on its left shoulder to identifies objects in the working environment. Compared with MOPED that uses a single camera for object recognition, using a RGBD sensor relaxes the constraint that object dimensions have to be known as a priori. The visual recognition system can detect object

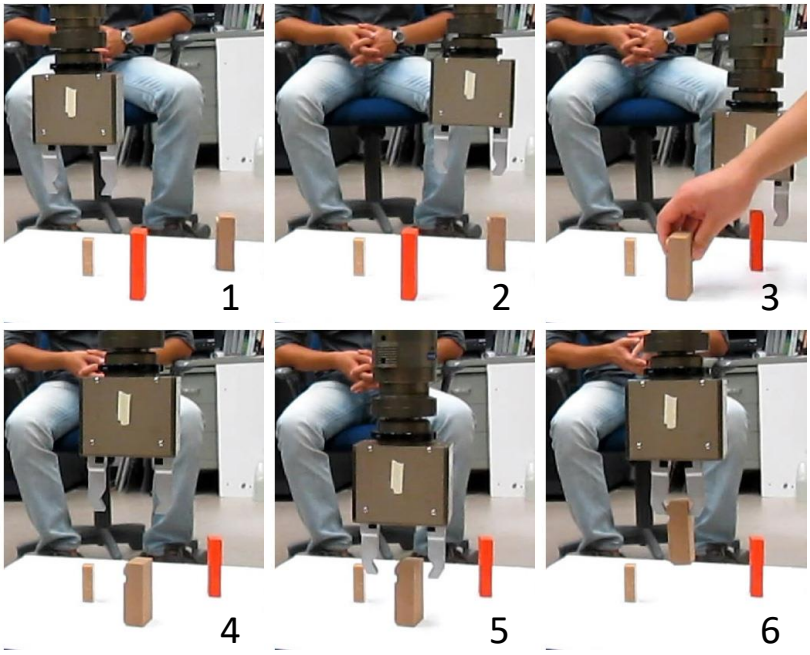


Figure 2.8: Snapshots of setup and implementation on scenario 1 "*pick up the big brown block on the table*". During robot approaching target big brown block, its position and that of the orange block are switched (as shown in subfigure 3).

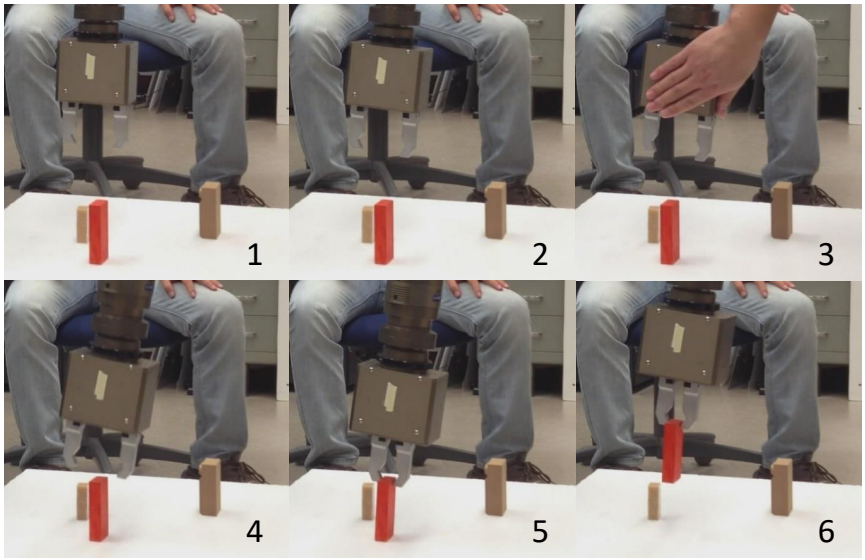


Figure 2.9: Snapshots of setup and implementation on scenario 2 "*Grasp the orange block*". During the approaching of the target block, the arm body is blocked by its co-worker (as shown in subfigure 3).

features such as color, 3-D position, and dimensions (height, width, and length), as shown in Figure 2.11 [66].

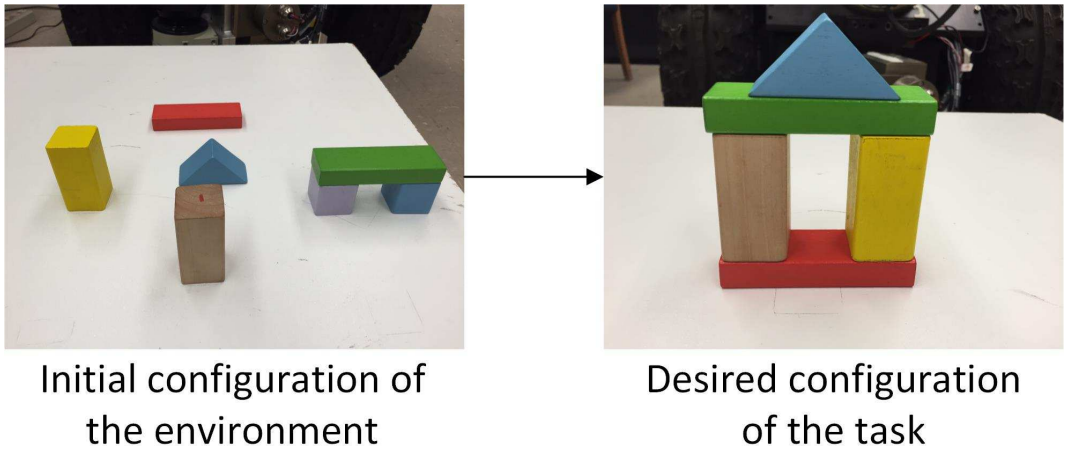


Figure 2.10: Initial setup and desired configuration of the task.

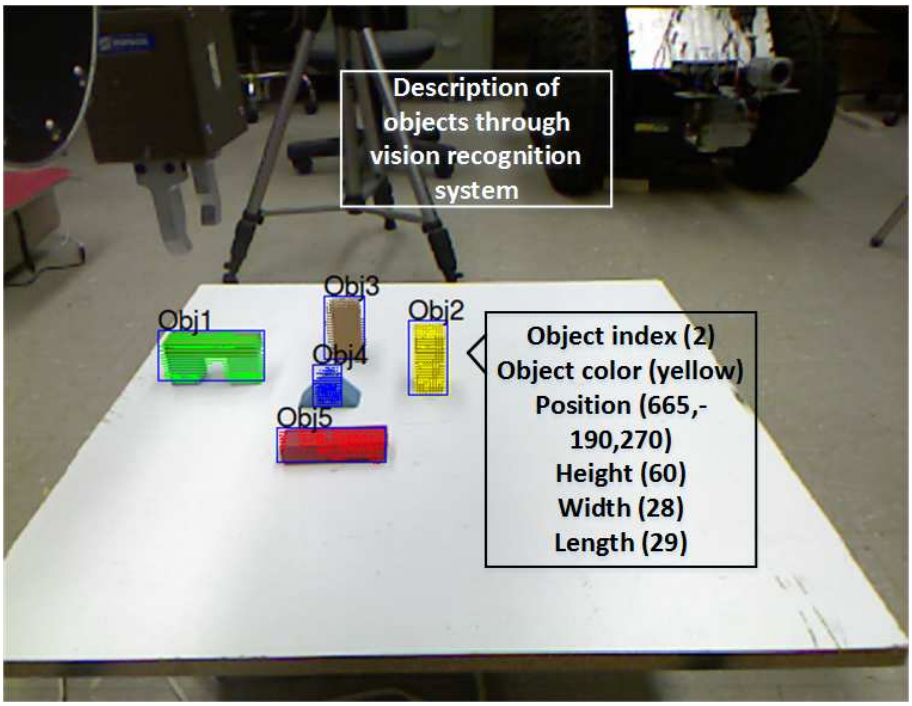


Figure 2.11: Description of perceived workpieces through visual recognition system.

Initially, five parts of a product are placed on the worktable without overlaps. The robot is tasked to assemble a block castle as shown in the right subfigure of Figure 2.10. The list of states

Table 2.2: List of states that used to represent the task and robot status.

Environmental State	Description
In(a,b)	Object a is in object b
On(a,b)	Object a is on object b
Around(a, b)	Object a is surrounding object b
Under(a, b)	Object a is under object b
InAir(a)	Object a is held in air and not
Adjacent(a,b)	Objects a and b are not at the above states
Robotic State	Description
In(a, gripper)	The end effector is gripping object a
At(a, gripper)	The end effector is at position of object a
Idle	Nothing in gripper and the gripper is not at any object position

are shown in Table 2.2, denoting the robot status and environment status for high level planning and control. One example of natural language task descriptions from an untrained user is shown as follows:

A blue workpiece is on top of the green one. The grey and yellow parts are on the red workpiece. The green workpiece is above the grey and yellow parts.

Instead of giving out step by step guidance, the utterance describes the desired configuration, which is more abstract. In addition, the language instructions are issued in a mixed order. It is impossible for most existing NL based control approaches to finish the task which simply following the given order. By adding the layer of task organization, it is possible for the system to figure out a reasonable subtask plan based on information from language instructions with incorrect order.

As shown in the left subfigure of Fig. 2.10, the state representation of initial setup is

$$\{Adjacent(W_i, W_j) | i \in I, j \in I, i \neq j\}$$

where W_i denotes the i_{th} workpiece, $I = \{1, 2, 3, 4, 5\}$. Following the DRM element definition and

rules presented in Equ. 4.1, Equ. 2.5, and Equ. 2.6, the initial DRM for this task is obtained:

$$DRM_{init} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

where from the top to the bottom are the five objects denoted as from W_1 to W_5 by following the object index assigned by visual recognition system. Each row shows one object's dependency relations with others.

From the natural language instructions, a partial set of goal state can be obtained:

$$On(W_4, W_1) \wedge On(W_3, W_5) \wedge On(W_2, W_5)$$
$$\wedge On(W_1, W_3) \wedge On(W_1, W_2)$$

Here, "partial" means direct translation of natural language can only lead to a subset of the DRM_{goal} representation.

The corresponding DRM_{goal} representation is:

$$DRM_{goal} = \begin{bmatrix} 0 & 2 & 2 & -2 & 2 \\ -2 & 0 & 1 & -2 & 2 \\ -2 & 1 & 0 & -2 & 2 \\ 2 & 2 & 2 & 0 & 2 \\ -2 & -2 & -2 & -2 & 0 \end{bmatrix}$$

Subtracting DRM_{goal} by DRM_{init}, we have the DRM_{error}:

$$DRM_{error} = DRM_{goal} - DRM_{init}$$
$$= \begin{bmatrix} 0 & 1 & 1 & -3 & 1 \\ -3 & 0 & 0 & -3 & 1 \\ -3 & 0 & 0 & -3 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ -3 & -3 & -3 & -3 & 0 \end{bmatrix}$$

Applying the function $f(X)$ defined in Equ. 4.4 to the DRM_{error}, we get the vector that contains quantified priorities of subtasks:

$$f(DRM_{error}) = \begin{matrix} W_1 \\ W_2 \\ W_3 \\ W_4 \\ W_5 \end{matrix} \begin{bmatrix} 0 \\ -5 \\ -5 \\ 4 \\ -12 \end{bmatrix}$$

Sorting the elements in an ascending manner, we will have the manipulation order of the blocks:

$$W_5 \Rightarrow W_2 \Rightarrow W_3 \Rightarrow W_1 \Rightarrow W_4$$

Then the assembly task is implemented with the parts manipulation order. Fig. 2.12 shows the snapshots of the assembly execution.

The task execution is monitored by using the matrix two norm of DRM_{error}. Figure 2.13 shows the plot of $\|DRM_{error}\|$ during task execution.

We also notice that the workpieces W_2 and W_3 have the same priority. This means the order of operating the two blocks won't impact the completion of the task. Since only one arm is equipped, we randomly select one block to manipulate first.

This experiment assumes initially all the parts are scattered without overlaps. Otherwise, extra operations are required. If initially overlaps exist, the whole task will be set to two phases. Phase I

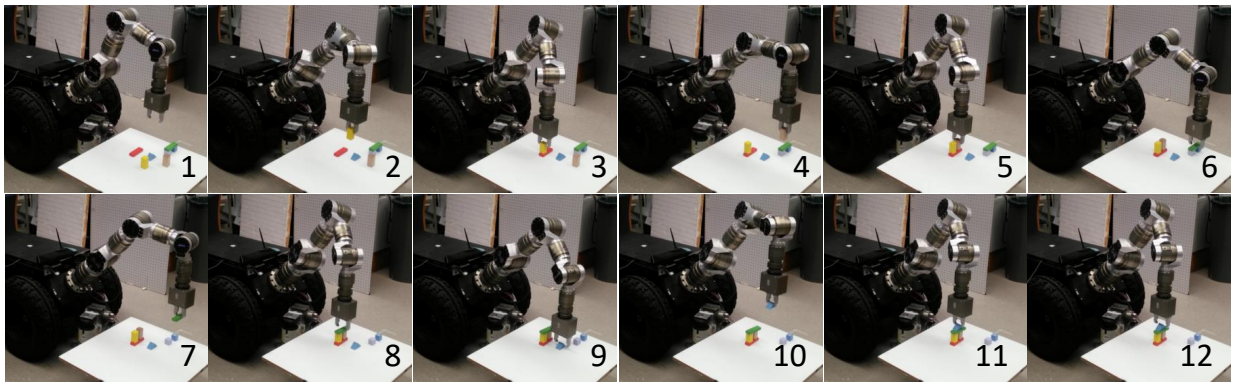


Figure 2.12: Snapshots of robot implementation on assembly task.

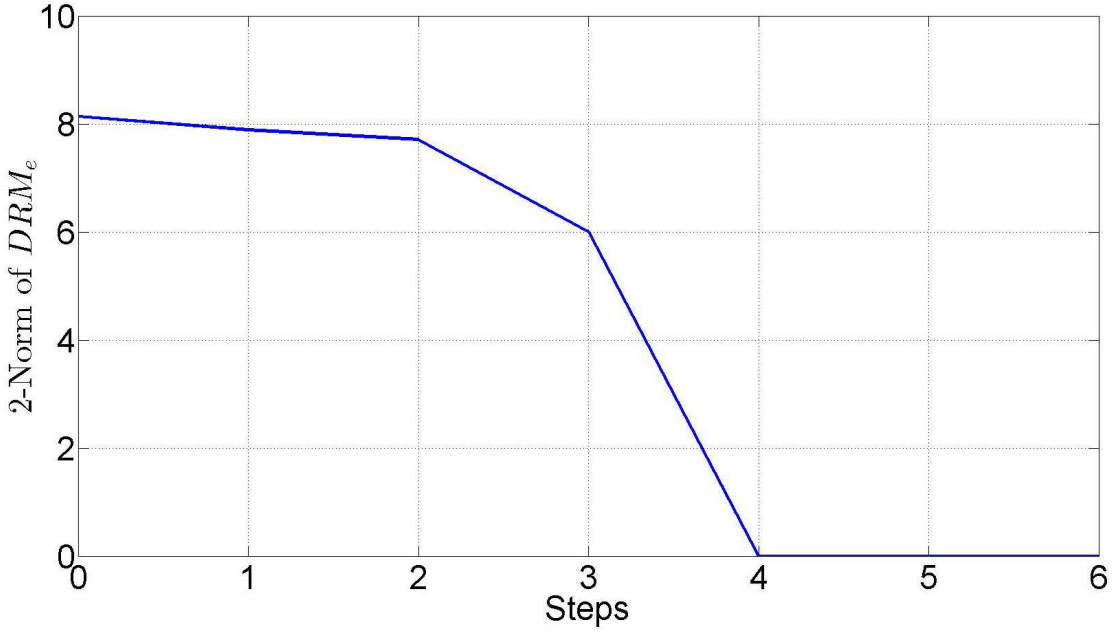


Figure 2.13: $||DRM_e||_2$ during the task execution.

is to remove the overlaps and phase II is to generate task plan and implement the job as illustrated above. This task planner framework can also be integrated with other NL based robot control frameworks to figure out a correct subtask order. In addition, *DRM* is hierarchically compositional. It enables to represent complex tasks from simple ones, which helps to alleviate the state and action explosion as the tasks become more complicated and to organize task components in a consistent way.

2.6 Summary

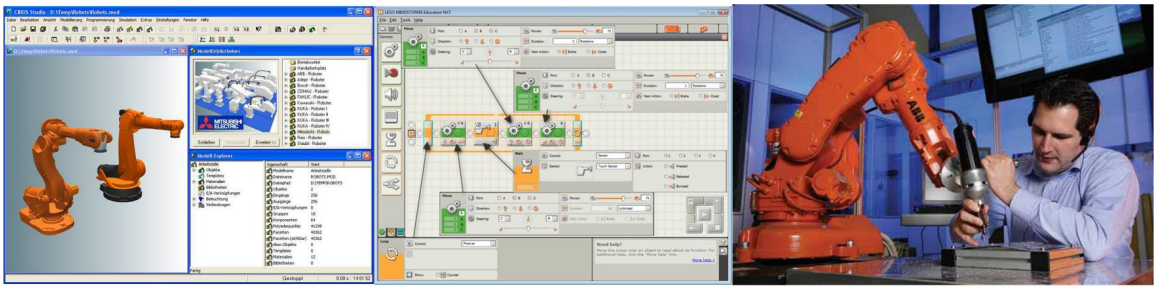
To facilitate the robustness to environmental uncertainties of robots, this chapter presents a correct-by-design control framework taking NL as input. Different from existing NL based control approaches, the robot can work under dynamic environments. Moreover, hierarchical control structure is performed to deal with commands of different logic levels and compact the state space. Experimental results show that the robot has a good performance under uncertainties. Furthermore, the control approach presented in this chapter may also be applied to other DESs.

CHAPTER 3

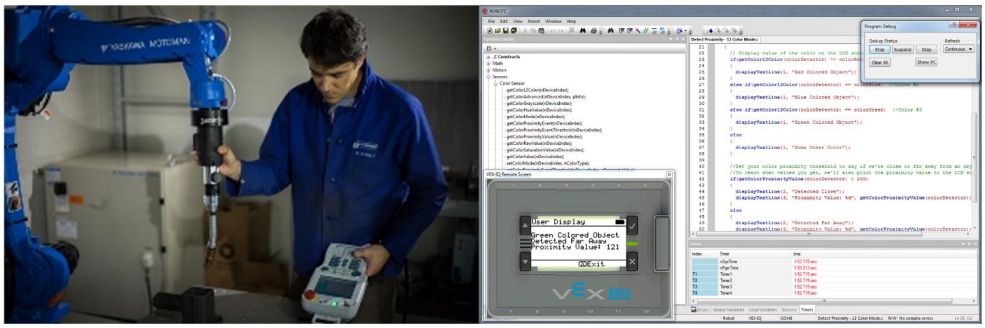
NATURAL LANGUAGE BASED ROBOTIC BEHAVIOR PROGRAMMING

3.1 Introduction

The goal of robot programming is to pass human knowledge from human to the robot, such that the robot is able to execute tasks that were not able to be done by it before. In essence, robots are reconfigurable and multifunctional machines. To deploy robots and harness their power, multiple approaches have been proposed to ease the process of programming robots, such as guiding [67],controller specific language [68], generic procedural languages [69], graphical programming [70], programming by Demonstration (PbD) [71]. Figure 3.1 shows existing robot programming approaches. Table 3.1 briefly summarizes the pros and cons of existing approaches.



(a) Specific language (b) Graphical language (c) Programming by Demonstration



(d) Guiding (e) Generic language

Figure 3.1: Existing robot programming approaches.

Guiding is the earliest and most widespread method of programming robots. It involves manually moving the robot to each desired position, and recording the internal joint coordinates cor-

Table 3.1: Pros and cons of existing robot programming approaches.

Robot Programming	Advantages	Disadvantages
Guiding	Immediacy; simple to implement	Not support complex control structures (e.g., loops, conditions)
Controller specific language	Simple to use; support loops and react to sensory input	Lack of universal standard; not suitable for novice users
Generic language	Flexible and powerful	Specific to the robot; not suitable for novice users
Graphical programming	Easy to use	Not as flexible as text-based language; not suitable for complicated programming
PbD	Allows encoding of very different types of signals	Not allow to reproduce complicated high-level skills

responding to that position. In addition, operations such as closing the gripper or activating a welding gun are specified at some of these positions. The resulting "program" is a sequence of vectors of joint coordinates plus activation signals for external equipment. Such a program is executed by moving the robot through the specified sequence of joint coordinates and issuing the indicated signals. Robot guiding is simple to use and to implement. Because guiding can be implemented without a general-purpose computer, it was in widespread use for many years before it was cost-effective to incorporate computers into industrial robots. Programming by guiding has some important limitations, particularly regarding the use of sensors. During guiding, the programming specifies a single execution sequence for the robot; there are no loops, conditionals, or computations. This is adequate for some applications, such as spot welding, painting, and simple materials handling. In other applications, however, such as mechanical assembly and inspection, one needs to specify the desired action of the robot in response to sensory input, data retrieval, or computation. In these cases, robot programming requires the capabilities of a general-purpose computer programming language.

Controller-specific languages were also one of the original method of controlling industrial robots. Until now, they are still the most common approach today. Every robot controller has some form of machine languages, and there is usually a programming language to go with it which can

be used to create programs for the robot. This type of programming languages are usually very simple, with a BASIC-like syntax and simple commands for controlling the robot and program flow, such as languages used by KUKA [72] and ABB [73]. Controller-specific languages have some drawbacks [69]. The biggest problem is the lack of a universal standard between languages from different robot manufacturers. If a factory employs robots from different manufacturers, then they have to either train their programmers for each type of robot, or pay the manufacturer to develop the desired programs. Either way increases significantly the time and cost for developing new programs.

Generic languages provide an alternative to controller-specific languages for programming robots. Generic means a high-level multi-purpose language, such as C++ and Java. These languages have been extended in some way to provide robot-specific functionality, which is particularly common in research environments. The most common extension to a multi-purpose language is a robot abstraction, which is a set of classes, methods, or similar constructs that provides access to common robot functions in a simple way. This removes the need to handle low-level functionality such as setting output ports high to turn on motors or translating raw sensor data. However, these abstractions suffer from the same weakness as the controller-specific languages for industrial robots. They are still specific for the robots they are designed for.

Graphical programming (or visual programming) systems provide an alternative to text-based methods for manual programming [74][75]. They are one small step closer to the automatic programming, as they provide a simplified graphical medium for rapid prototyping and implementation of certain systems. They require manual input to specify actions and program flow. Graphical systems typically use a graph, flow-chart or diagram view of the robot system. Graphical programming are also known to ease the process of providing end-user programming, where the user of an application is able to modify the behavior of the application in some way. This programming approach has been successfully adopted both by experienced programmers and non-technical computer users. One advantage of graphical systems is their ease to use, which is achieved at the cost of text-based programming's flexibility.

At the beginning of the 1980s, PbD started attracting attention in the field of manufacturing robotics. PbD appeared as a promising route to automate the tedious manual programming of robots and as way to reduce the costs involved in the development and maintenance of robots in a factory. A large body of work uses a symbolic representation of both the learning and the encoding of skills and tasks. This symbolic way of encoding skills may take several forms. One common way is to segment and encode the task according to sequences of predefined actions, described symbolically. Encoding and regenerating the sequences of these actions can, however, be done using classical machine learning techniques, such as HMM [76]. Often, these actions are encoded in a hierarchical manner. In [77], the authors employ a hierarchical and incremental approach to encode various household tasks. There, learning consists in extracting symbolic rules that manages the way each object must be handled. The main advantage of symbolic approaches is that high-level skills can be learned efficiently through an interactive process. However, because of the symbolic nature of their encoding, the methods rely on a large amount of prior knowledge to predefine the important cues and to segment those efficiently. For learning and encoding a skill at trajectory-level, choosing the variables well to encode a particular movement is crucial, as it already gives part of the solution to the problem of defining what is important to imitate. Work in PbD encodes human movements in either joint space, task space or torque space. Encoding often encompasses the use of dimensionality reduction techniques that project the recorded signals into a latent space of motion of reduced dimensionality. These techniques may either perform locally linear transformations [78] or exploit global non-linear methods [79]. The advantage of learning at trajectory-level is it allows encoding of very different types of signals/gestures. However, it does not allow to reproduce complicated high-level skills.

The above presented robot programming approaches usually require expertise on robot and coding, which are not suitable for novice users. In addition, robot programming usually happens earlier than robot execution. However, there is an increasing demands for online programming to cope with uncertainties in tasks and working environments. Humans not only need to program the robots before task execution, but also, in many instances, work cooperatively with the systems

to online program the robots in order to accomplish tasks. For example, in automated assembly and material handling operations, multiple robots have to function coordinately or synchronously. A local disturbance on an individual robot can easily affect the overall task synchronization and coordination, resulting in a global catastrophe since the current state of the art approaches follow the general design method of fixed automation. Typically, a robotic system is only programmed to deal with a small subset of geometric dynamic and time duration uncertainties which can be reliably pre-estimated. As anything that falls outside the pre-estimated local uncertainty bound will cause a global disturbance. This will require a halt of overall operation and a reprogramming of the operation.

If a robot could be programmed using NL, it could not only make the robot programming easier and more convenient, but also significantly improve the scalability and safety of the robot operations, especially for unstructured tasks and dynamic environments.

Natural language programming has been researched along either logic rule based methodology or probability based methodology [80]. Although many promising results on natural language based robot programming have been achieved, the trained system models suffer from the following problems. First, the training process is usually offline. When the robot encounters a new and unseen scenario, the performance of the robot becomes worse as the similarity between the test data and the training data decreases. Secondly, a natural language controlled robot is supposed to successfully complete the given task in a dynamic environment. The correctness of the programmed behaviors and the stability of the system are unknown to both the robot itself and programmer. This creates major vulnerability of programmed robot operations.

In this chapter, we propose a programming approach that allows non-technically trained users to teach a robot new skills through NL on the fly. New skills can be taught hierarchically using already learn skills and represented as a combination of states. The proposed programming approach can integrate with the developed control approach presented in last chapter, which allows the robot to learn a skill under static environment, but can still perform it under dynamic environment.

The remainder of this chapter is organized as follows. Section3.2 introduces the overall pipeline

of the proposed framework. Section 3.3 illustrates the proposed programming approach. Finally, Section 3.4 presents the experimental results on an industrial scenario.

3.2 Overview of System Framework

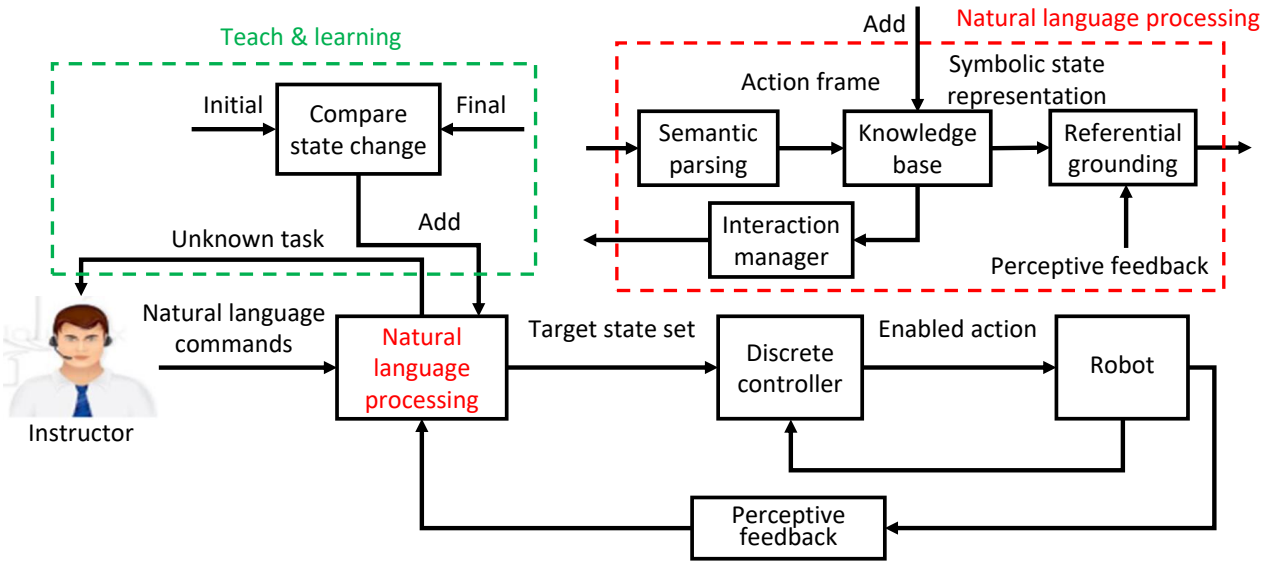


Figure 3.2: Overview of the proposed NL based robot programming framework.

Figure 4.2 presents the proposed interactive robot programming framework. The NL instructions given by a human user are processed through a semantic parser into action frames with predicate-argument structure [3]. The knowledge base uses action frames as query items and search for the compatible goal state representation. If there exists a goal state set with respect to the input commands, this symbolic state representation is then dispatched to the referential grounding module. The discrete controller takes the grounded state as input and performs the task as illustrated in last chapter.

If there doesn't exist a goal state representation corresponding to the input command, the knowledge base notifies the interaction manager to send the human instructor a warning of undefined mapping and wait for either the activation of teaching and learning procedure to encode the new command or another command. Details about the proposed interactive robot programming will be illustrated in Section 3.3. After learning the new command, the learned *command-state*

mapping will be added to the knowledge base.

3.3 New Skill Acquisition

When the robot receives an instruction for which its knowledge base fails to generate a goal state, the interaction manager inquires the human instructor for advice on new skill programming. If the response is positive, the interaction manager switches to teaching and learning mode. Table 3.2 presents a full list of working mode of interaction manager.

Table 3.2: Working Mode of Interaction Manager

Working Mode	Description
Action execution	System is performing tasks by NL
Teaching and learning	System is under programming using NL
Query	State transformation fails

In teaching and learning mode, the instructor decomposes the skill into a sequence of skills that the robot already learned, and verbally guides the robot to perform the skill step by step. Before the robot implements the guiding instructions, its own state and environmental state are recorded and denoted as initial configuration. After the robot finishes all the guiding instructions, its own state and environmental states are recorded as the final configuration. Compare the two configurations and eliminate the common states, it is the new skill that is considered to cause the difference between the initial and final configurations. The remained two sets of states are considered as preconditions and postconditions of this new skill, respectively. The new skill is represented using the following template and added to the knowledge base:

Precondition : *initial_status*

Action : *new command*

Postcondition : *final_status*

where the *initial_status* as well as the $final\_status$ is a conjunction of states. Also, attributes of objects within the same predicate-argument state will be compared, such as color, shape, etc., which can be extracted from sensory information. Attributes are considered as part of the states.

Because attributes of object play no role in system model analysis, they will not be represented explicitly. However, it should be noted that even the same state representation with different attributes are considered as different states. The state definition is dependent on the robot type and application domain knowledge. In this work, we use the state definition as presented in Table 2.2.

A complex behavior can be decomposed into a combination of several atomic actions implemented in a specific order. During the teaching phase, the instructor can utilize already learned high level skill instead of only the primitive actions. This eases the teaching process and makes it time-efficient and user-friendly. A learned skill is represented as state transitions rather than an ordered sequence of actions demonstrated by the instructor, as the work presented in [7][46]. The state-based representation has two advantages over the action-based representation. First, it is easier to generalize. State-based representation focuses on the results caused by the behavior. The realization process can be flexible and compliant with different environment setup. While actions-based representation can only be applied to scenarios with similar environment setup, or the behavior may fail to deliver the task. Second, the state-based representation can be integrated with our developed control framework. This allows to program a behavior under static environment but work in a dynamic environment. The instructor doesn't have to enumerate all the possible situations to the robot in order to improve the behavior robustness to uncertainties. Figure 3.3 presents skill learning paradigm.

For practical applications, usually, the scenarios will not be as simple as the one used for teaching and learning. The environmental setup can be more complex, or there are more workpieces to manipulate with the same command. The first difficulty can be overcome by integrating with the proposed control framework presented in last chapter. To solve the second problem, the learned skill is represented as:

$$\text{Precondition} : \wedge_{i=1}^{n}(initial\_status_i)$$

$$\text{Action} : new\ command$$

$$\text{Postcondition} : \wedge_{i=1}^{n}(final\_status_i)$$

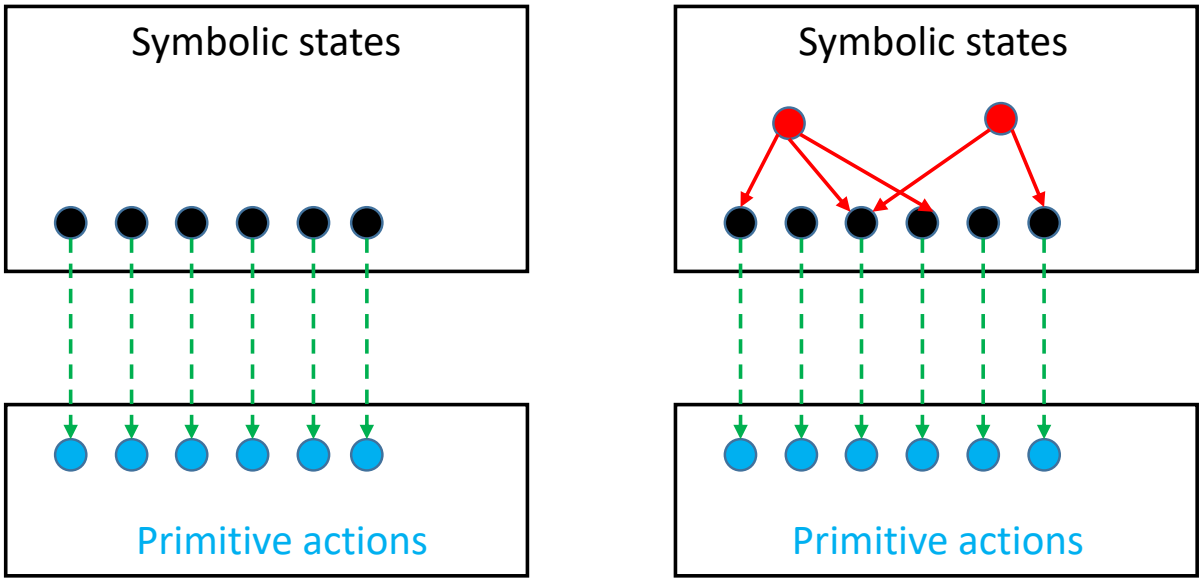


Figure 3.3: New skill learning. (left) is a schematic representation of initial robot knowledge base that comprises pre-programmed primitive actions and their cause and effect. (right) the user has programmed two new skills with already learned knowledge.

where n denotes the total number of workpieces to be manipulated. It can be determined from natural language instructions or from vision system, or by combining information of both. Considering the fact that the robot may work in an industrial environment, and the workpieces can be delivered by a conveyor belt such that its number keeps changing. Accordingly, The total workpiece number n should keeps updating.

A behavior causes change from initial configuration to the final configuration. Both configurations have the same number of states. Two states with the same parameters (one from the initial configuration and one from the final configuration) are grouped together as a pair:

$$(initial\_status_i,\ final\_status_i)$$

Each pair of states is looked as a subtask.

3.4 Experiment Evaluation

3.4.1 Experiment Setup

Natural language processing. NL commands are processed by a CCG semantic parser [57] and translated into action frames with predicate-argument structure. Object grounding is performed by a graph-based matching algorithm [58]. In the experiments we use typed commands instead of speech and skip the voice capture and speech recognition modules. The NLP modules are run on a laptop and communicates with the robot wirelessly.

Robotic platform. We use a 7 Degree-of-Freedom (DOF) SCHUNK robot arm to test the proposed programming approach. It is mounted on a four wheeled Segway RMP 440 mobile base to provide mobility when necessary, as shown in Figure 5.3.

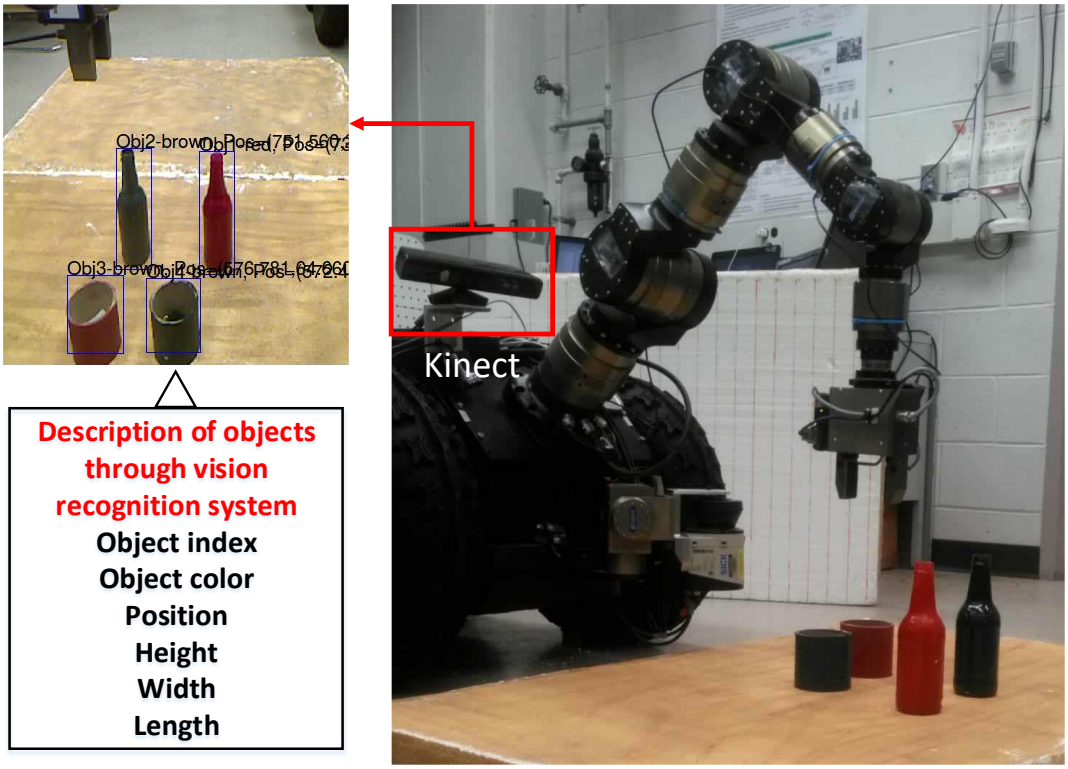


Figure 3.4: Experiment platform and setup.

Vision recognition. A RGBD sensor (Kinect) is mounted on the left shoulder of the robotic platform. It performs object recognition and identifies the object type and object attributes (color,

shape, dimensions) [66]. The vision recognition module assigns each detected object an index and frame the object information into an object schema as shown in lower left subfigure of Figure 5.3.

3.4.2 New Behavior Programming

In this experiment, we assume an industrial robot that is capable of executing open-move-close manipulations. Firstly, an instructor uses NL to program *pick up* and *put* behaviors to the robot. As for the *pick up*, initially the gripper is at the position of the object to be manipulated, the instructor verbally guides the robot to close the gripper until the bottle is firmly grasped, then the robot *pick up* lifts the arm. The learned pick up is represented as:

$$\text{Precondition} : On(bottle, table) \wedge At(bottle, gripper))$$

$$\text{Action} : pick\ up$$

$$\text{Postcondition} : InAir(bottle) \wedge In(bottle, gripper))$$

Similarly, the behavior *put* can be taught from having a bottle grasped in gripper and ending at releasing the bottle on table and lift the arm back to Idle, which can be represented as:

$$\text{Precondition} : InAir(bottle) \wedge In(bottle, gripper)$$

$$\text{Action} : put$$

$$\text{Postcondition} : On(bottle, table) \wedge Idle$$

Complex behavior can be verbally taught by referring to previously programmed behaviors. Assume a sorting task is expected to performed by the robot. The instructor firstly tasks the robot to sort the beer bottles by color, in which each beer bottle is expected to be packaged with a box with the same color. The robot provides a negative response showing that it doesn't know how to implement the behavior. Then the instructor gives step by step instructions to the robot employing newly learned *pick up* and *put* behavior. The initial condition before the teaching is shown in the right subfigure of Figure 5.3: there are two bottles and two cylinder boxes, with colors red and green for each bottle and cylinder box. After the teaching phase, the robot compares the final

configuration with the initial configuration. The common states of the two state sets are eliminated and the rest two sets of states are assumed to be caused by the behavior *sort by color*. Figure 3.5 shows the programming procedure. Accordingly, the general form of the programmed skill sort by color is represented as:

$$\text{Precondition} : \wedge_{i=1}^{n}(Adjacent(a_i, b_i) \wedge Idle)$$

$$\text{Action} : sort\ by\ color$$

$$\text{Postcondition} : \wedge_{i=1}^{n}(In(a_i, b_i) \wedge Idle)$$

where n denotes the number of items to-be-sorted and will be instantiated conditioned on the working environments. In addition, attribute of color of objects that form the states are ignored in representation.

3.4.3 New Behavior Testing under Exception

In this experiment, we test the learned sorting behavior under exception. The robot is tasked to perform a bottle sorting task in a new experimental setup, as shown in the subfigure 1 of Figure 3.6. Two beer bottles with different colors placed in the working environment. Three cylinder boxes of different colors are placed in the working environment as well. After receiving the instruction "*Sort the beer bottles by color*", the robot generate the goal state representation:

$$\wedge_{i=1}^{2}(In(bottle\ i, box\ i) \wedge Idle)$$

The pairing between bottles and cylinders is based on their color attributes extracted from sensory information through perceptive feedback. Each pair of initial states and goal states is viewed as a subtask. Because there is no compatible yellow bottle to pair the yellow cylinder box, it is not included in the goal states. When the robot just finishes packaging the red beer bottle, one yellow bottle and one red bottle are added into the working environment. This abrupt change of working environment is updated via perceptive feedback to the NLP module. In the meanwhile, the red bottle and red cylinder have already been pair together. As a result, the goal state representation

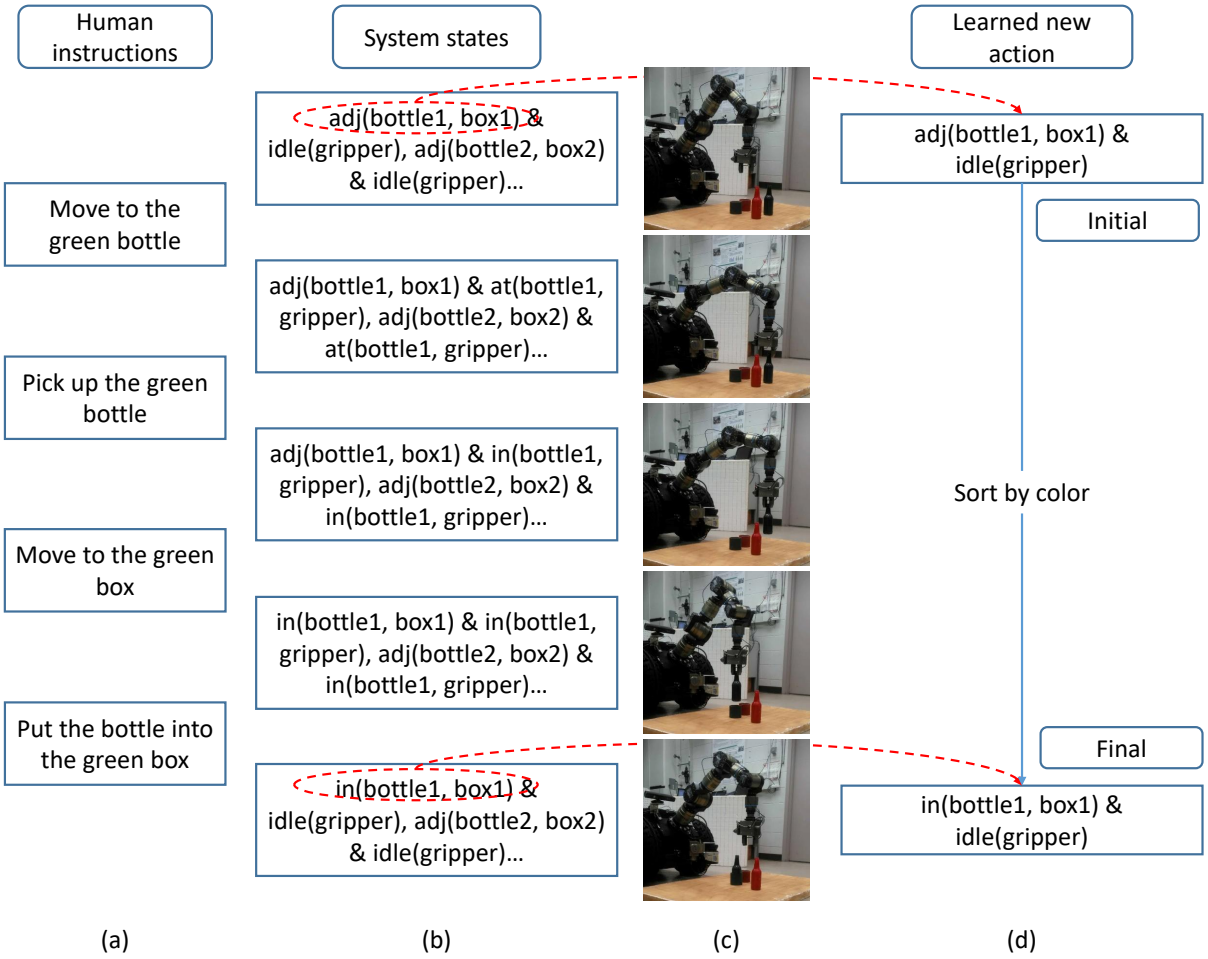


Figure 3.5: Programming the skill of "*Sorting by color*" with step by step instructions. (a) Natural language commands given by the instructor. (b) Initial system state and temporal states after each implementation. (c) Snapshots of robot execution corresponding to system state in column (b). (d) The representation of learned new behavior.

changes to:

$$\wedge_{2=1}^{3}(In(bottle\ i, box\ i) \wedge Idle)$$

Because there is no cylinder box matching the newly added red bottle in color, no goal state is generated for the new red bottle. The sorting task is considered as completed when there is no more color-consistent but unsorted pairs of bottles and boxes. Figure 3.6 shows a few snapshots of the task implementation.

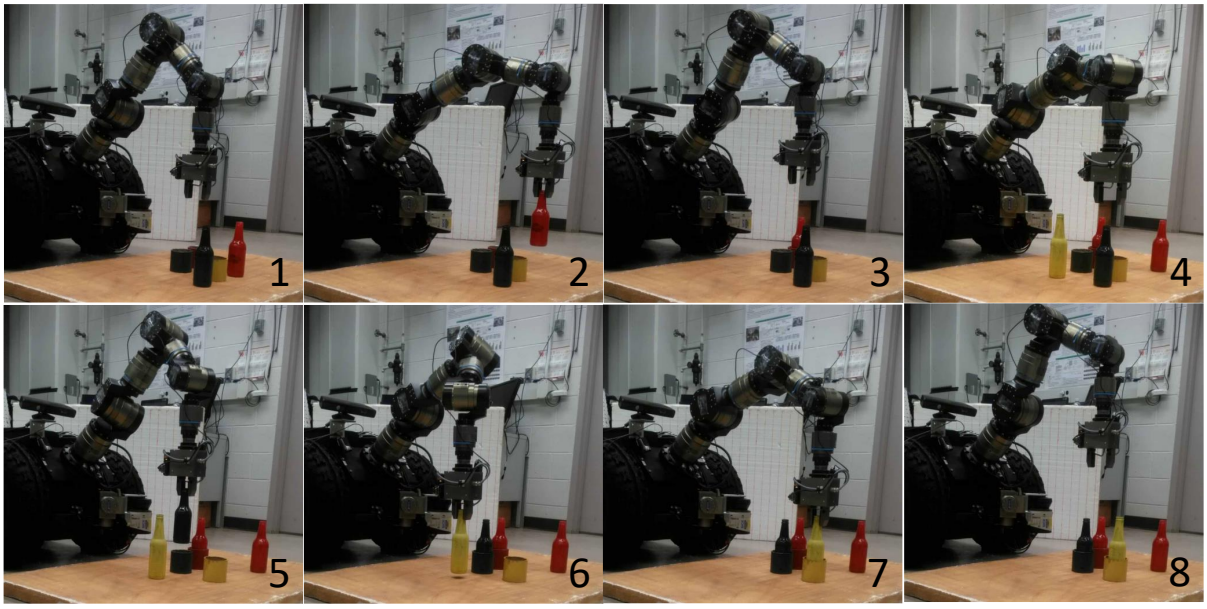


Figure 3.6: Test of the learned skill "*Sort by color*" in a more complex scenario. After the robot put the red bottle into the box (subfigure 3), two more bottles were added into the working environment (subfigure 4).

3.5 Summary

In this chapter, a novel approach to program robotic behaviors via NL instructions is presented. It allows human users to program the robots on the fly without terminating the robots, which is beneficial and convenient for human-robot collaboration. Users can teach verbally teach the robots in a hierarchical manner by referring to previously learned skills. This is time-efficient and allows to teach using concepts with different abstraction level. The data collected during training process can be used to improve the capability of natural language processing. In addition, this online programming framework can be integrated with the proposed control framework, which helps to improve the robustness of the learned skills under dynamic environments.

CHAPTER 4

ROBOTIC DRAWING CONDITIONED ON NATURAL LANGUAGE DESCRIPTIONS

4.1 Introduction

Controlling robotic behaviors via NL utilizes the rich expressibility of NL to deliver task requirements, complementary knowledge and information from a human instructor, which can benefit both the robots and human users. Existing approaches of NL based robotic control focus on modeling the mapping relationship between language and primitive robotic actions [15][81]. Symbolic representations are generated to represent the task goal and specifications, and then are parameterized for the reasoning of action selection. The success of existing approaches relies on well-trained language-action mapping and correct language grounding using environmental information from sensors and knowledge from an on-board local knowledge base. However, if the tasks are not dependent on environmental information, or there doesn't exist a well-built knowledge base for the robot to use, it is difficult for the robot to perform the task, such as instructing a robot with NL to draw a scene on paper. As illustrated in Figure 4.1, a robotic painter is tasked to draw a scene with NL description "*A lovely dog sits in front of a sofa in a room. A chair is at the left side of the dog*". The robot has to not only infer the context (*room*), object type (*sofa*, *dog*, *chair*), object property (*lovely*), the spatial layout among objects (*in*, *left*, *front*), but also generate motion plans to draw the scene on paper.

Scene generation and drawing conditioned on NL has many applications. One important application is literacy development. For children who are learning to read and for second language learners, seeing scenes together with language can enhance learning [82]. Another application is as a reading helper for people with learning disabilities or brain damage. The robot can convert textual menus, signs, and operating instructions into graphical representations. In addition, robotic drawing conditioned on NL can extend to similar application scenarios, such as painting and polishing.

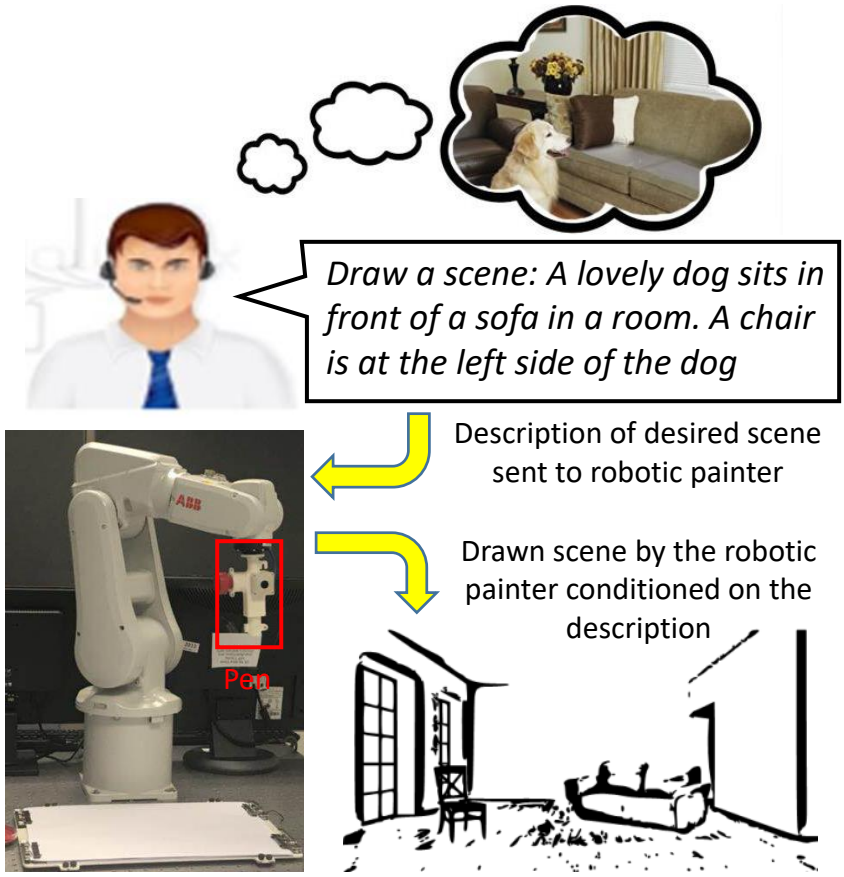


Figure 4.1: Robotic painter setting where an instructor describes a scene in NL. The painter infers a scene that matching the description and draws the scene on a paper using a marker pen mounted on the robot's end-effector.

There can be a huge number of choices for object types and object properties (e.g., color, shape, texture, etc.), and it is nontrivial to construct a local knowledge base or train direct text-to-pixel mappings for the robot to use. The success of existing scene generation approaches is limited to clipart scenes [83][84] or single objects [85]. Generating realistic and complex scenes is still challenging. Instead of generating scenes via a local knowledge base or training mappings from data, we propose to directly use information from the Internet for complex scene generation. The Internet can be viewed as a multi-discipline knowledge base collected from diverse distributed sources. However, the Internet is "dirty" because it often contains erroneous or corrupted data [86], which makes it difficult for the robot to use information directly retrieved from the Internet. By introducing a mechanism of dealing with unexpected events during scene generation using the

retrieved information, the proposed method can generate complex and semantically meaningful scenes.

The proposed framework for hierarchical scene generation and drawing has three parts: a spatial layout generator, a scene generator, and a motion planner. The spatial layout generator infers spatial configuration among objects. The scene generator firstly retrieves images through query with object types and object properties from an online image source. Then the object instances are segmented from the retrieved images and assembled together in accordance with the spatial layout. In addition, the scene generator deals with unexpected events in image retrieval and scene generation, and allows users to modify scenes by replacing objects, changing location and size of objects via NL before drawing the scene on paper. The motion planner takes generated scene as input and plans trajectories of end-effector for scene drawing.

The main contributions of this chapter are as follows:

- We propose a novel approach for robots to generate and draw scenes conditioned on NL instructions. Instead of maintaining a knowledge base in limited on-board memory, the proposed method leverages the power of the Internet, which helps to increase the diversity and creativity of generated scenes, and to reduce burden on knowledge learning and storage.

- A mechanism is developed to tackle unexpected events to guarantee successful knowledge retrieval and scene generation.

- The proposed approach offers an interactive framework and interface to control scene generation process: users are allowed to modify the synthesized scene by replacing objects, changing size and location of objects through NL. This helps to correct possible training errors or bias using human experience. As a result, the generated scenes can be more recognizable and well-aligned with the descriptions.

- We conduct quantitative evaluations on MS-COCO dataset and qualitative evaluations as well, and demonstrate improvement in quality of both the scene generation and scene drawing over baseline works.

The rest of the this section is organized as follows. Section 4.2 briefly reviews related work. Section 4.3 provides an overview of the developed framework and Section 4.4 illustrates the proposed approach in detail. Section 4.5 discusses the experimental results. Finally, Section 4.6 summarizes this chapter.

4.2 Related Work

Text to Scene. Scene generation conditioned on text has been researched along two directions: object level and pixel level. Object level scene generation synthesizes scenes by placing objects in accordance with the estimated spatial layout. While pixel level approaches map text to most likely object pixel distributions trained from data.

For object level scene synthesis, Zhu et al. [87] use nouns appeared in the text as keywords to search for images of objects and then place the images together to represent the meaning of the input text. Only object type information is employed, visual attributes of objects and spatial configuration are ignored. Zitnick et al. [83] train a conditional random field model to generate clipart scenes using clipart characters. Chang et al. [84] employ Bayesian probability to generate 3D office scenes using objects from a 3D model database. These two works utilize spatial relations between objects. The visual attributes of objects are still not considered in scene generation. The proposed approach takes into consideration of both the object properties and the specified spatial configuration in scene generation, which helps to generate more diverse and complex scenes.

More recent works focus on using generative models for pixel level scene generation. Yan et al. [88] use variational auto-encoder to generate scenes conditioned on visual attributes. Reed et al. [89][90] train generative adversarial networks (GANs) for scene generation. These approaches are limited to single objects, such as birds [85] and flowers [91]. Mansimov et al. [92] synthesize scenes on more complicated scene descriptions (MS-COCO) using a variational recurrent autoencoder with attention mechanism. Han et al. [93][94] propose stacked GANs structure to generate and refine scenes for better recognizability than single GANs. Dong et al. [95] use a cascaded convolutional neural network and recurrent neural network with GAN to augment the visual features

of objects in the scene. The major focus of pixel level scene generation is on visual features, while spatial mappings are either ignored or modeled implicitly. In comparison, the proposed approach is able to generate complex scenes use both information.

In addition, existing scene generation approaches do not allow modification on synthesized scenes. The proposed approach allows users to change objects, modify the position and the size of objects in synthesized scenes through NL in an interactive manner, which helps to correct possible training errors or bias using human experience, and the data can be recorded for later training augmentation. Furthermore, the proposed approach uses information from the Internet, which helps to increase the diversity of generated scenes.

Natural Language based Robot Control. Using NL to control robot behaviors makes it easier for people to employ robots, especially for novice users. One key step of NL based robot control is to translate the NL commands into formal and unambiguous representations such that robots are able to understand and implement the instructions.

Based on how the language commands are processed into executable action plans, existing approaches can be divided into two categories: logic-based and statistics-based. Logic-based approaches translate linguistic commands into executable action plans based on a set of rules designed from human experience, such as using keywords [48], hand-coded grammar [6], user-defined templates [39], target state set [30], and formal language representations [17][81].

Statistics-based methods use data-driven algorithms to learn the implicit mapping rules or action selection policies from data instead of hand-designed explicit rules. These methods differ in their probabilistic models (e.g., linear model [41], hidden Markov model [14], Markov logic network [23], conditional random field [15], etc.), formal representations (e.g., predicate-argument structure [16], lambda calculus [17], graphical representation [18], etc.), and features used for training the models.

In comparison to the existing approaches focusing on forming a symbolic representation for robotic behavior generation, the proposed approach generates a recognizable scene as the intermediate representation of the instructor's intent to bridge the language and robot movements. By this

way, it can indicate more details of the assigned tasks than pure symbolic representations.

4.3 Overall Pipeline

The overall pipeline of the proposed framework is illustrated in Figure 4.2. Given a language description of a scene, the robotic painter draws the generated scene on paper through the sequence of following modules:

- **Natural language processing (NLP)** module takes language descriptions as input, and generates relation frames (as shown in Figure 4.3). This process is realized through two subsequent modules: firstly, semantic roles are labeled to identify verbs and their arguments; secondly, syntactic structure of each argument is parsed to identify properties of objects and spatial relations (Section 4.4.1).

- **Spatial layout generator** infers a spatial layout of objects using relation frames generated from the prior NLP module. A spatial layout is an organized position distribution of objects. An object's position is assumed to depend on its surrounding objects. The position of each object is calculated based on the positional and dimensional information of its dependent objects (Section 4.4.2).

- **Scene generator** synthesizes a scene according to the estimated spatial layout. The scene generator retrieves images from Internet image search engines using information of object type and property, segments objects from images and merges each object into a proper context. The robot uses images from diverse cloud resource and an imperfect object detector. Unexpected events may happen and cause failure of scene generation. The scene generator deals with these unexpected events and ensures completion of scene generation. (Section 4.4.3).

- **Motion planner** takes the generated scene as input, and outputs trajectories of the robotic painter to draw the scene on paper. The motion planner first extracts visual features that are semantically meaningful, then plans the trajectory to draw the feature areas (Section 4.4.4).

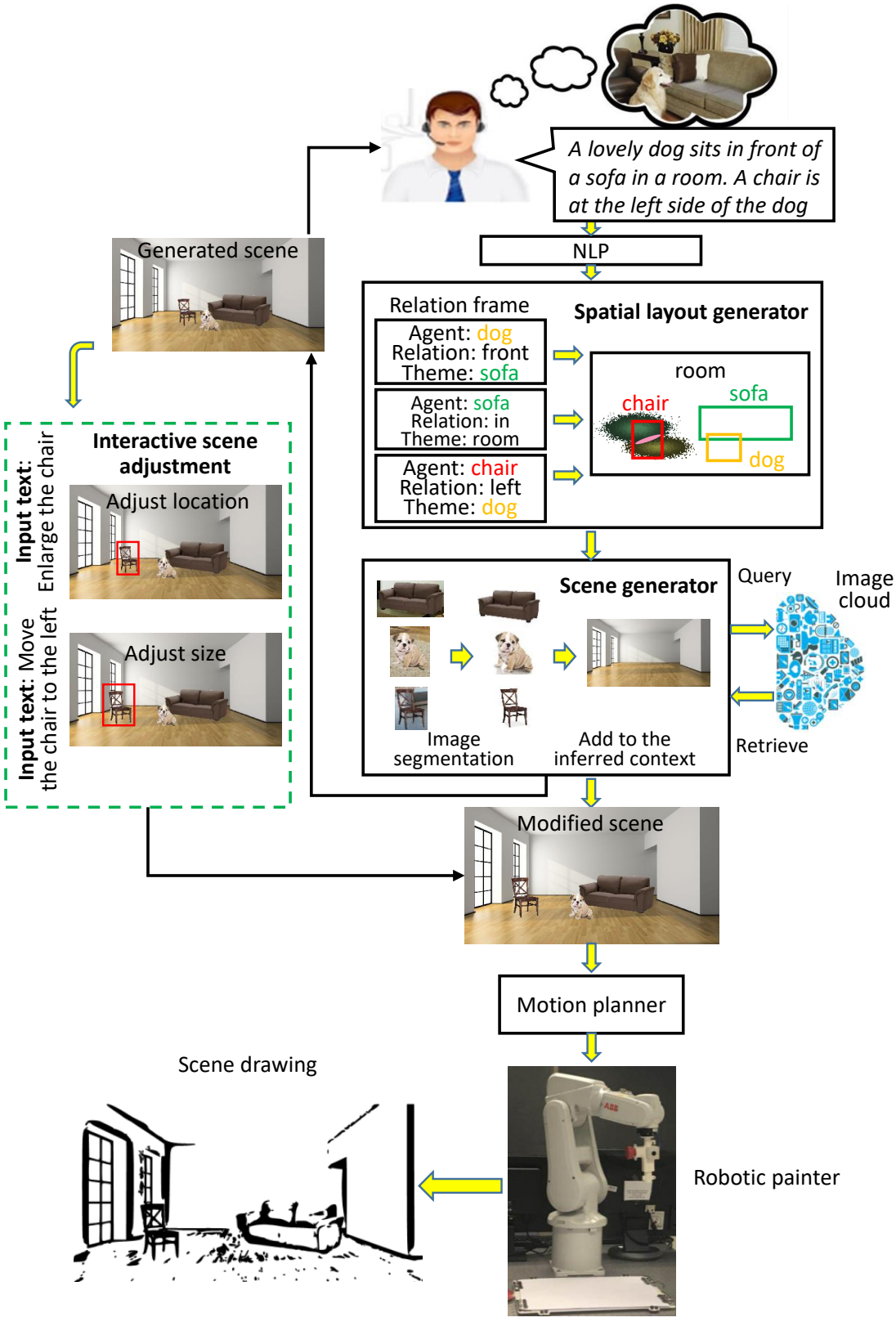


Figure 4.2: The overall pipeline of the proposed approach. The relations frames shown in this figure ignore information of *Verb* and *Property* for simplicity purposes.

4.4 Robotic Drawing Approach

4.4.1 Natural Language Processing

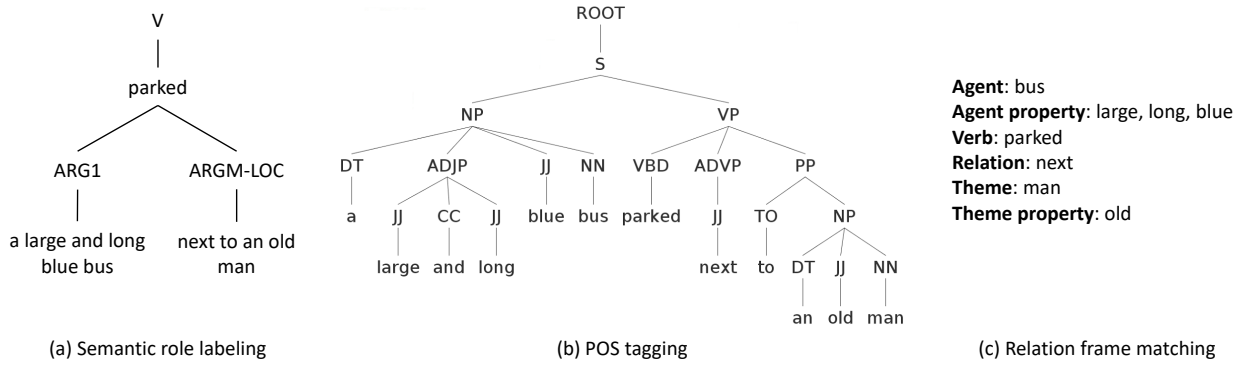


Figure 4.3: Conversion of the sentence *a large and long blue bus parked next to an old man* into relation frame through parsing, tagging, and semantic interpretation.

To convert the users' instructions into a formal specification, the system must identify the underlying linguistic structure of the description and convert it into a logical representation. This section describes the process of this conversion and its implementation. The users' instructions are processed through a pipeline of natural language components which identify the semantic structure of sentences and the syntactic roles of them, and create formal representations to be used in spatial layout inference and image query. Different from many previous natural language systems for robot control have relied on per-scenario grammars that combine semantic and syntactic information [10], this work uses a combination of robust, general-purpose components for parsing and tagging the input. An advantage of this approach compared to per-scenario grammars is that the core language models need not be modified across scenarios. This reduces the role of the fragile process of grammar engineering and minimizes the cost of adapting the system to handle commands in new domains.

The NLP module uses a pipeline of domain-general natural language processing components. The input is parsed using AllenNLP semantic role labeling (SRL) parser to identify verbs and arguments [96]. Each argument is classified into one type of PropBank modifiers which indicates the argument's semantic function in the sentence [97]. For example, as shown in Figure 4.3, the

chunk *next to an old man* is identified as modifier *ARGM-LOC* to indicate the location of the bus. Then each argument is tagged using Stanford Log-linear POS Tagger to identify the subject and corresponding properties as well as possible relations with neighboring items appeared in the sentence [20]. The identified information is matched to a relation frame

$$(Agent,\ Agent\ property,\ Verb,\ Relation,\ Theme,\ Theme\ property)$$

The expected relation (*Relation*) between the two items (*Agent* and *Theme*), the specification of the items (*Agent property* and *Theme property*), and pairwise position dependency (*Agent* depends on *Theme*) are identified using their tags and syntactic positions.

4.4.2 Spatial Layout Generator

Spatial configuration is required in order to generate scenes using objects segmented from real images. Pairwise spatial relations acquired through the first step of language processing contain partial information of the layout. The right subfigure of spatial layout generator in Figure 4.2 shows an example of the inferred spatial layout corresponding to the input description *A lovely dog sits in front of a sofa in a room. A chair is at the left side of the dog*. The position of each object is determined by its surrounding objects on which it is dependent as described in the language. For example, the chair's position is directly dependent on the dog, and the dog's position depends on the sofa. As a result, only after the placement of the sofa and the dog, the chair's position can be figured out using positional and dimensional information of the dog and the sofa. In this work we use dependency relation matrix (*DRM*) to determine dependencies among objects [51]. An agent is considered to be dependent on its theme.

Using the matched relation frames from language parsing, a *DRM* can be built as follows:

$$\begin{array}{c} \\ O_1 \\ O_2 \\ \vdots \\ O_n \end{array} \begin{array}{c} \begin{array}{cccc} O_1 & O_2 & \cdots & O_n \end{array} \\ \begin{bmatrix} [DRM]_{11} & [DRM]_{12} & \cdots & [DRM]_{1n} \\ [DRM]_{21} & [DRM]_{22} & \cdots & [DRM]_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ [DRM]_{n1} & [DRM]_{n2} & \cdots & [DRM]_{nn} \end{bmatrix} \end{array}$$

It is a $n \times n$ matrix, where n is the number of objects to appear in the scene. O denotes object. The column index of *DRM* denotes the object indices from the top to the bottom. The row index represents object indices with the same order from the left to the right. Each element in the *DRM* captures the dependency relation between the column object over the row object. The value of an element in the *DRM* reflects the relative dependency between two objects. The element at the intersection of the i_{th} row and j_{th} column of the matrix can be defined as:

$$[DRM]_{ij} = \begin{cases} 0, \text{relation with the object itself,} \\ 1, i^{th} \text{ object and } j^{th} \text{ object are notdependent,} \\ 2(-2), i^{th} \text{ object supports (is dependent on) } j^{th} \text{ object.} \end{cases} \tag{4.1}$$

Initially, all elements in *DRM* are set to be 0. The relation frames only indicate pairwise dependency relations. To find the dependencies of each object on others, after parameterizing the *DRM* using relation frames, we apply the transitivity rule as represented in Equation 4.2 and 4.3 repetitively on the *DRM* until element values stop changing:

$$\begin{aligned} &[DRM]_{ij} = relation \\ &\Rightarrow \begin{cases} [DRM]_{ji} = -relation \quad \text{if } relation \neq 1, \\ [DRM]_{ji} = relation \quad \text{if } relation = 1 \text{and } [DRM]_{ji} = 1, \\ [DRM]_{ij} = -[DRM]_{ji} \quad \text{if } relation = 1 \text{and } [DRM]_{ji} \neq 1, \end{cases} \end{aligned} \tag{4.2}$$

$$\begin{cases} [DRM]_{ij} = relation \\ [DRM]_{jk} = relation \end{cases} \Rightarrow [DRM]_{ik} = relation \tag{4.3}$$

where $relation \in R$, $R = \{-2, 1, 0, 1, 2\}$ in accordance with the relation definition in Equation 4.1.

The more items on which an object's position is dependent, the more positive numbers in the corresponding row of DRM. If we sum up each row elements of the DRM as shown in Equation 4.4, we get a vector. A larger value in the vector means more object instances the corresponding object is dependent on. If objects are manipulated in a sequential order, a larger value indicates lower priority in manipulation.

$$f(DRM) = \left[\sum_{j=1}^{n}(a_{1j}), \quad \sum_{j=1}^{n}(a_{2j}), \quad \cdots, \quad \sum_{j=1}^{n}(a_{nj})\right]^T \tag{4.4}$$

Usually, the first object in the sequence is the context because all the other objects are dependent on it. If the context is not explicitly specified in the language, it will be inferred using the objects appeared in the scene description:

$$P(context|object_{1:m}) = \prod_{i=1}^{m} P(context|object_i) \tag{4.5}$$

the context type that maximize the probability will be selected. In this work we consider five context types: indoor, road, city plaza, rural field, and sea shore (if a context is explicitly specified in the description, the scene generator can retrieve candidate context images from cloud resources; otherwise, the context will be inferred from the five categories). The priors for object occurrence in different context types are trained using 1708 scenes from MS-COCO training dataset (the object detector we used in this work can recognize 20 category of objects, so we filtered the dataset to get scenes comprised of objects lay in the category set):

$$P(context|object_i) = \frac{count(object_i \text{ in } context)}{count(context)} \tag{4.6}$$

After determining the context, each time when adding an object into the scene, its position is calculated based on other objects on which it is dependent. The spatial knowledge is trained using

relative location data from [83], which consists of 10020 scenes created using 58 clipart objects. The centroid position of an object is randomly chosen from the intersection area of its relative locations in each spatial relation with a dependent object (theme). As shown in the subfigure of Figure 4.2 circled by the red dashed rectangle, the dark yellow cloud shaped area represents "left" locations in relation to the dog, and the dark green cloud shaped area denotes "left" locations in relation to the sofa. The centroid position of the chair is randomly selected from the intersection area (marked in pink). The dimensional ratio between each two object types is set as a priori.

4.4.3 Scene Generator

In this step, the robot retrieves images containing specified objects from an image cloud (in this work, the images are retrieved using Google image search engine. Similarly, other image search engines also work) and synthesize a scene using segmented objects from retrieved images. The robot uses object instances from a "dirty" image cloud that is intended for human use rather than manually labeled datasets [83][84]. Unexpected events may happen and cause failures of scene generation, such as detection failure of objects in the retrieved images due to contaminated data. In addition, it is helpful to generate semantically meaningful scenes if a human instructor can tune the synthesized scene through NL. To this end, a dynamic discrete event model is developed to tackle unexpected events and modify generated scenes interactively. First, the modeling process using supervisory control theory is illustrated in detail. Then, property analysis of the developed model is conducted for behavior refinement to guarantee successful scene generation.

Receiving matched frames and estimated spatial layout, the scene generator retrieves images that probably contain desired objects from image cloud, segment the objects from their original images, and assembles them together in accordance with the specified spatial layout. In interactive scene tuning phase, the scene generator replaces objects, modifies position and size of objects following the instructor's commands. Figure 4.4 shows the decentralized plant models of the robotic painter. Table 4.1 presents the event set and controllability of each event. The plant model

is a shuffle product of the plant models:

$$G = \|_{i=1}^{8} G_i \tag{4.7}$$

where $\|$ represent parallel composition operation. The shuffled plant model has 896 states and 4096 transitions in total.

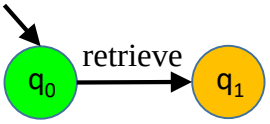
Table 4.1: List of basic actions for scene generation.

Primitive Actions	**Controllability**	**Description**
(α) retrieve_img (obj)	controllable	Retrieve image from local image library or remote image cloud
(β) resize (obj)	controllable	Adjust the size of the object instance
(γ) detect (obj)	controllable	Detect object category
(δ) compare (obj1, obj2)	controllable	Compare category labels between objects
(ϑ) segment (obj)	controllable	Extract object instances from images
(ζ) merge (obj, context)	controllable	Add object into current scene
(η) affirmative	uncontrollable	Confirm the previous operation
(θ) negative	uncontrollable	Deny the previous operation
(κ) change_img (obj)	controllable	Change another image from the image base
(λ) zoom_in (obj)	controllable	Enlarge the size of an object
(μ) zoom_out (obj)	controllable	Reduce the size of an object
(ν) move (obj)	controllable	Change the position of an object
(ρ) save (obj, img)	controllable	Save the image that has the object instance
(ξ) stop	controllable	Stop current being implemented action

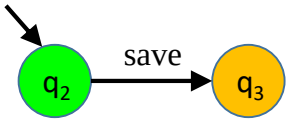
Each action is parameterized by the required objects or images. The Greek letters are used to encode the primitive actions for simplicity of representation in planner property analysis (Section IV, Part C).

When the robotic painter receives language descriptions of a scene, it retrieves images that ranked high in the query results using keyword combinations of (*Agent* + *Agent Property*) or (*Theme* + *Theme Property*). Then the robot uses an object detector to detect desired object instances from retrieved images. If an image contains the object that meets the specification, then the object is segmented from the image and resized to fit into the new scene. Otherwise, the robot switches to the next image in the rank and repeat the process until all the required objects have been found and arranged to the specified spatial layout.

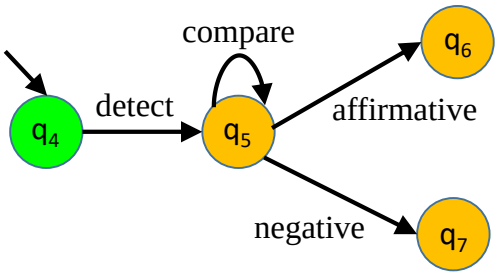
The plant model captures all the physically possible behaviors, including desired behaviors (legal behaviors, i.e., behaviors lead to the marker states) and behaviors that should be forbidden (illegal behaviors, i.e., behaviors that cause system failure to reach the marker states). To guarantee



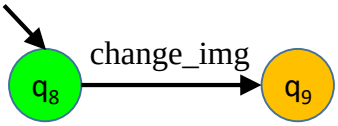
(a) G_1: Image retrieve



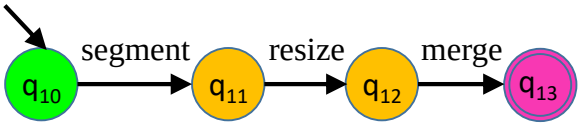
(b) G_2: Image save



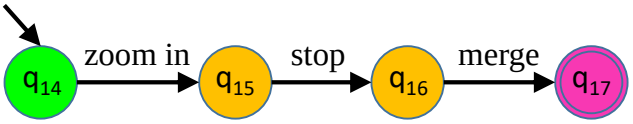
(c) G_3: Detect objects from images



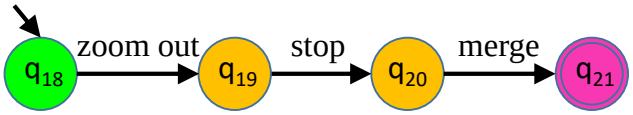
(d) G_4: Change to next image



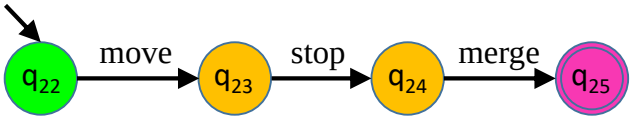
(e) G_5: Scene synthesis



(f) G_6: Zoom in an object



(g) G_7: Zoom out an object



(h) G_8: Change the position of an object

Figure 4.4: Decentralized plant model of scene generation. Subfigure (a) to (e) are the models of scene generation using objects extracted from retrieved images. Subfigures (f) to (h) are the models for scene tuning. States in green with an entry arrow represent initial states. Yellow colored states mean intermediate states. The states in pink with double circles are marked states and also represent the target state of each behavior.

the legal behaviors, a supervisor is required to regulate the plant's behaviors. In addition to following the scene synthesis procedure, following properties should also be held by the supervisor:

1. **Controllability**: Controllability characterizes the capability of a system to accomplish the assigned task (reach target state) under abrupt and unexpected occurrence of uncontrollable events.

   Definition of Controllability [56]: Let K and $L = \overline{L}$ be two arbitrary languages over event set Σ. K is said to be controllable with respect to L and Σ_{uc} if

   $$\overline{K}\Sigma_{uc} \cap L \subseteq \overline{K}.$$

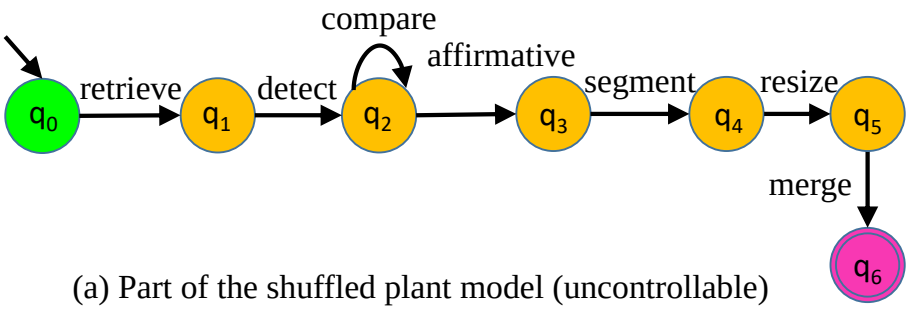
   If a supervisor is controllable, it is able to drive the system to its target state even encountering uncontrollable events during the implementation. As shown in Figure 4.5 (a), the behavior is a desired one. However, this behavior is uncontrollable: at state q_2, an uncontrollable event *negative* can happen and cause the failure of image generation, which falls out of the legal range. To ensure successful image generation, uncontrollable behaviors should be removed from the supervisor (i.e. image generator).

   Controllability guarantees the marker states are achievable. However, sometimes the robot may be trapped by a state or a subset of state such that it takes longer time to accomplish the task. To avoid this issue and make the robotic implementation more smoothly, the second property to analyze behaviors is introduced next.

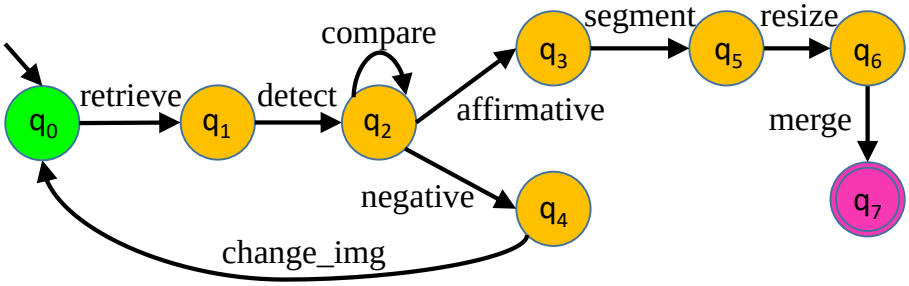
2. **Nonblocking**: Nonblocking characterizes the property of a system to avoid being stuck at a state or trapped in a subset of states. The following gives the definition and criteria of nonblocking property.

   Definition of Nonblocking [56]: Let $L(G)$ and $L_m(G)$ represent the legal behaviors and the marked behaviors (in this context, marked behaviors represent successful scene generations) of the supervisor, respectively. A set of formal language satisfying

   $$\overline{L_m(G)} = L(G)$$



(a) Part of the shuffled plant model (uncontrollable)



(b) Part of the shuffled plant model (blocking)

Figure 4.5: Partial models of the shuffled plant model. For simplicity, the two models all start with q_0. In fact, the states in the two partial plant models are different states.

is said to be nonblocking. An overline of a formal language represent all the prefixes of the formal language, including the language itself. The supervisor refers to the image generator in the proposed framework.

The robot retrieves images from a noisy image cloud and may encounter problems that hamper the scene generation. For example, the retrieved images that ranked high in the results may not contain the desired objects. Then the robot has to keep changing images until the target object instance has been detected (as shown in Figure 4.5 (b)). This is time-consuming and may trap the robot in a livelock. Perform nonblocking check on the synthesized supervisor helps to detect and get rid of these blocking situations and ensure the accessibility of the marker states.

If we use formal language to represent the behavior of the partial plant model shown in Figure 4.5 (b), and use the Greek letters as indicated in Table 4.1 for simplicity of representation,

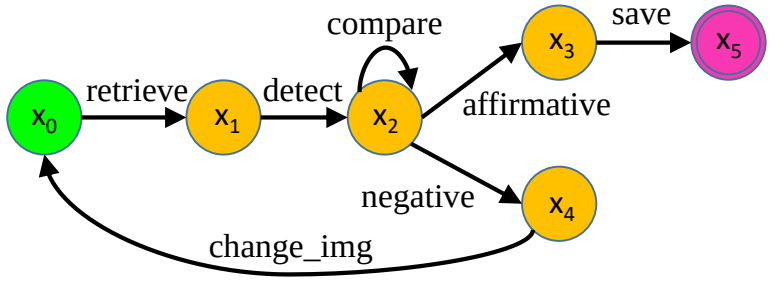
it's behavior can be represented as:

$$L(G_p) = (\alpha\gamma\delta(\theta\kappa)^*)^*(1 + \eta\vartheta\zeta)$$

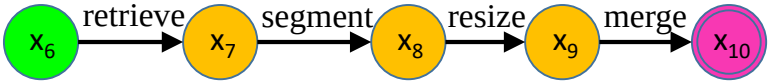
where G_p denotes "partial plant". While the marked behavior is:

$$L_m(G_p) = (\alpha\gamma\delta(\theta\kappa)^*)^*\eta\vartheta\zeta$$

from which it can be seen that $\overline{L_m(G_p)} \subset L(G_p)$. This means there is a possibility that



(a) S_1: retrieved image evaluation



(b) S_2: scene synthesis

Figure 4.6: Partial supervisors for (a) image retrieve and (b) scene synthesis.

the system will be trapped in an infinite loop to search for images that contain target objects. To avoid the blocking, the retrieved image evaluation procedure and scene generation procedure are separated as two independent processes, as shown in Figure 4.6 (a) and (b), respectively. The retrieved images will be evaluated first to detect desired objects and only the qualified images will be saved to a local image library (Figure 4.6 (a)). Then the robot randomly retrieves an image from the local image library and extracts the object instance for scene synthesis (Figure 4.6 (b)). The behaviors generated by these two refined models are nonblocking and controllable. They serve as partial supervisors for scene generation. In addition, the plant models G_6 to G_8 are both controllable and nonblocking. The supervisors

for interactive scene tuning are designed with the same event and state sets, respectively. The operations on automata (plant model synthesis, property check, supervisory synthesis) are implemented using DESUMA [62].

4.4.4 Motion Planner

In this work, we employ the feature extraction and path planning algorithm developed in [98]. Figure 4.7 shows the pipeline of its framework. Local binarization and global binarization are performed to extract visual features under different illumination conditions. Local binarization binarizes the input scene using adaptive thresholds for each pixel. While global binarization uses one threshold to binarize the scene, and this parameter varies by scene and can be determined automatically. The two feature graphs are then added up for robot path planning with minimum drawing time.

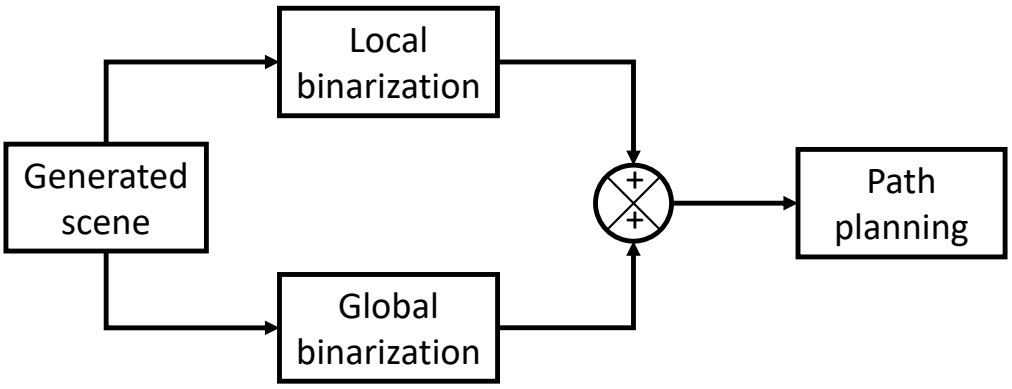


Figure 4.7: Overview of the framework of feature extraction and motion planning.

4.5 Experiments Evaluation

We have evaluated the proposed approach on two tasks: scene generation and scene drawing.

4.5.1 Experimental Setup

Robotic painter. As shown in Figure 4.1, the robotic painter is a six-axis ABB IRB 120 robot with a special designed container mounted at the end-effector to fix a marker pen. It is equipped with a spring inside the chamber to make the fixed marker pen more adaptive when drawing on paper.

Object detection. The object detection performs twofold roles: detection and segmentation. The object detector developed in [99] is used this work, which is able to detect 20 categories of objects. Object detection is implemented as pixel-level labelling tasks. The output is a 2D matrix that has the same dimension as the input image. Each value of the matrix denotes the object category of the pixel at the same location as the input image.

Dataset. We use the MS-COCO evaluation dataset [100] to evaluate our approach. Since the object detection algorithm we used can recognize 20 categories of objects, we filtered the MS-COCO dateset to find text descriptions that only contain objects lay in the range of processable object categories. In total we test our approach on 128 descriptions for scene generation and drawing.

Evaluation metrics. We evaluated the proposed approach using caption generation and human evaluation.

Baseline works. We compare our approach with the following baseline works.

- **Ground Truth**: The ground truth uses original scenes from the MS-COCO evaluation dataset paired with the scene description.
- **Random**: A random scene is selected from the ground truth.
- **Reed et al.** [89]: A GAN developed by Reed et al. to generate images according to the input text.
- **AttnGAN** [94]: Attentional GAN with attention-driven and multi-stage refinement for text to scene generation.

- **Ours**: the scenes are generated using the proposed method without further tuning by a human instructor.
- **Ours-Human**: the generated scenes have been tuned by human instructors using natural language.

4.5.2 Scene Generation

Caption generation: We evaluate text-conditional scene generation performance using caption generation. We generate sentences from the synthesized scenes and measure the similarity between input text and predicted descriptions. The underlying intuition is that if the generated image is relevant to input text and its contents are recognizable, one should be able to guess the original text from the synthesized scene. We use an image caption generator [101] trained on MS-COCO to generate sentences, where one sentence is generated per scene. We report four standard language similarity on BLEU [102], METEOR [103], CIDEr [104], and Rouge-L [105].

Table 4.2: Quantitative evaluation results

	Caption Generation						
Method	BLEU-1	BLEU-2	BLEU-3	BLEU-4	CIDEr	ROUGH_L	METEOR
Random	0.272	0.105	0.039	0.019	0.121	0.256	0.072
Reed et al. [89]	0.277	0.109	0.044	0.023	0.201	0.269	0.092
AttnGAN [94]	0.307	0.136	0.068	0.037	0.220	0.296	0.097
Ours	**0.352**	**0.174**	**0.091**	**0.053**	**0.416**	**0.335**	**0.134**
Ours-Human	**0.378**	**0.219**	**0.140**	**0.094**	**0.654**	**0.362**	**0.151**
Ground Truth	0.502	0.339	0.240	0.180	1.298	0.476	0.221

BLEU, CIDEr, ROUGH_L, and METEOR metrics based on caption generation are presented. The last row presents the caption generation performance on MSCOCO dataset images. Higher is better in all columns.

Table 4.2 summarized the quantitative evaluation results based on caption generation performance. The Random baseline shows the worst performance. This demonstrates that random scenes in the ground truth rarely convey the same semantic meaning. It can be seen that the proposed method significantly outperforms the baseline approaches. Tuning from human instructors also

helps to increase the semantics of the generated scenes dramatically. Caption generation performance shows that captions generated from our synthesized scenes are more correlated with the input text than the baselines, which means that the scenes synthesized by our method are better aligned with the input descriptions and are easier to recognize semantic contents.

Human evaluation: Using caption generator for evaluation is beneficial for large scale data. However, it may introduce unintended bias of the caption generator. To validate the caption generation evaluation, we have also conducted human evaluation on Amazon Mechanical Turk. For each text description from the evaluation dataset, six scenes generated by different approaches are presented to the Turkers. They are asked to score how well the scenes match the description on a scale of 1 (very poorly) to 5 (very well). The results shown in Figure 4.12 and 4.8 demonstrates consistency with the caption evaluation performance. Figure 4.9 shows some randomly chosen qualitative results. In addition, the proposed approach is able to generate scenes that are not likely to happen in the real world, as shown in Figure 4.13.

Besides comparison between the proposed approach and other baseline works, we also have conducted comparison experiments of our full model against models without controllability guarantee (None-Cont) and nonblocking guarantee (None-Nonb), respectively. The three models generate scenes conditioned on the 128 descriptions, respectively. The results are shown in Figure 4.10. Without guaranteeing controllability, the model has lower success rate in scene generation due to desired object detection failure from retrieved images. This can be caused by either misalignment between the image and its description on the Internet or the object detector error. Without guaranteeing nonblocking, averagely it costs longer time for the robot to generate a scene.

4.5.3 Scene Drawing

We perform feature extraction and path planning on all the generated scenes. The parameters used for feature extraction and motion planning are the same as presented in [98] for all the scenes produced with each baseline work.

Human evaluation: Similar to the human evaluation of the generated scenes, we also conduct

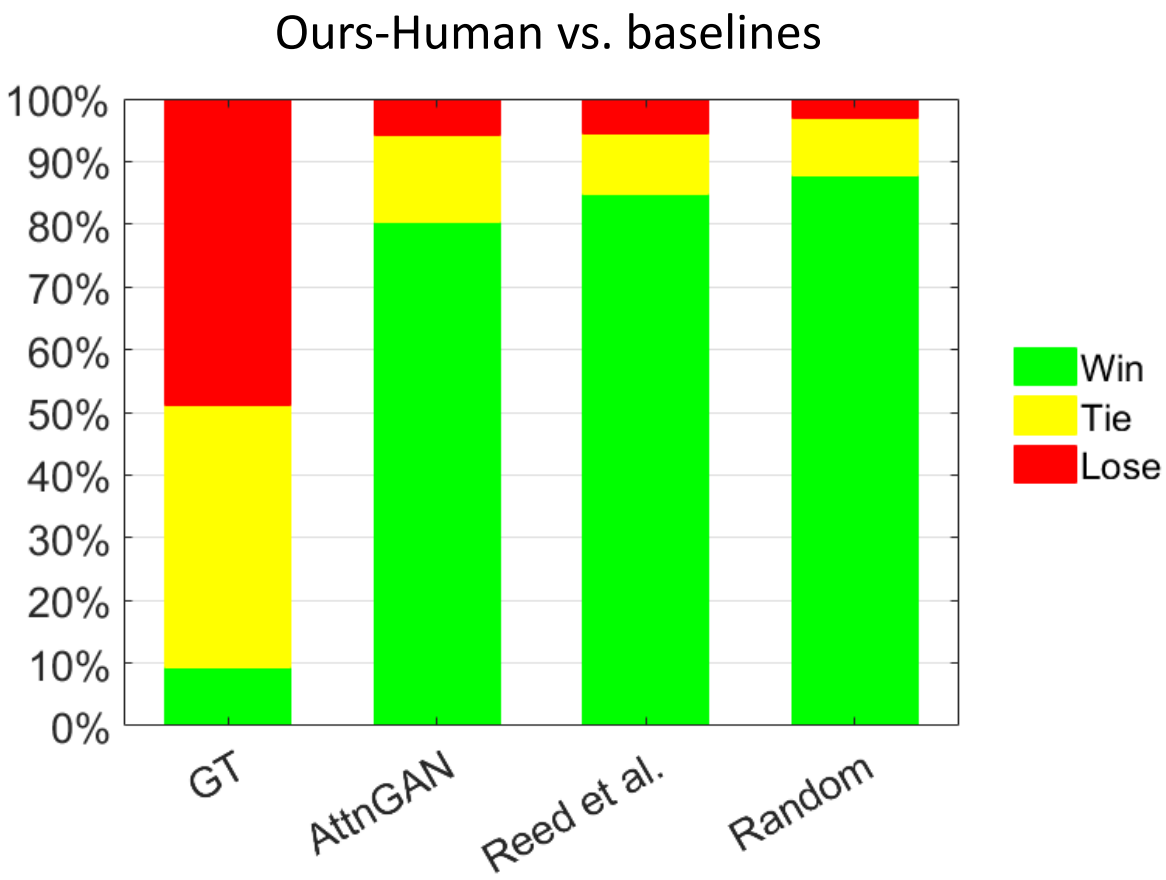


Figure 4.8: Turkers are asked to score how well the scenes match the description on a scale of 1 (very poorly) to 5 (very well). "GT" denotes "Ground Truth". The subjects find our scenes better represent the input sentences than other baseline approaches. In fact, our approach wins over or ties with the ground truth scenes frequently.

human studies on scene drawings using Amazon Mechanical Turk. Six scene drawings are presented to the turkers with a text description, and they are asked to score how well the drawings match the description on a scale of 1 (very poorly) to 5 (very well). Figure 4.12 shows the results. The proposed approaches substantially outperforms other baseline works (Random, Reed et al., and AttnGAN) and close the the performance of Ground Truth. Figure 4.11 shows scene drawings of the randomly chosen scenes in Figure 4.9.

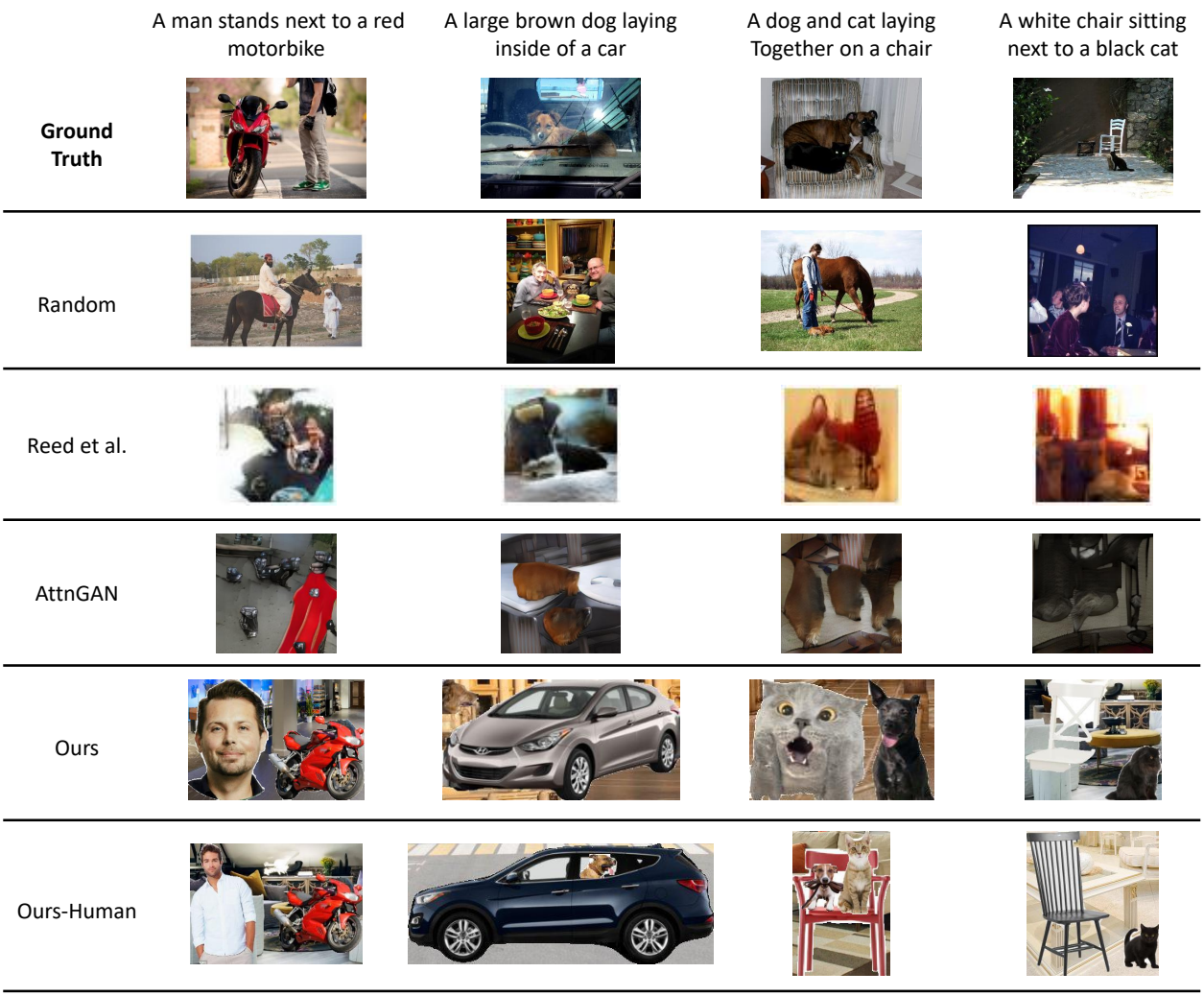


Figure 4.9: Qualitative examples of generated scenes conditioned on text descriptions from the MS-COCO dataset, using the proposed approach and baseline methods. The input description and ground truth scenes are shown in the first row.

4.5.4 Discussion

One crucial aspect for generated scenes to be well-aligned with language descriptions is the correct grounding of spatial requirements. Sometimes, the spatial descriptions can be ambiguous. For instance, "A man sitting in a chair" and "A man sitting on a chair" express the same meaning, while the robot may initialize the location of "a man" close to the centroid of the chair or on top of the chair back, respectively. Similarly, "A boy standing over a bed" means the boy standing next to a bed and leaning over it rather than standing on the bed. Failures to solve the ambiguity may due

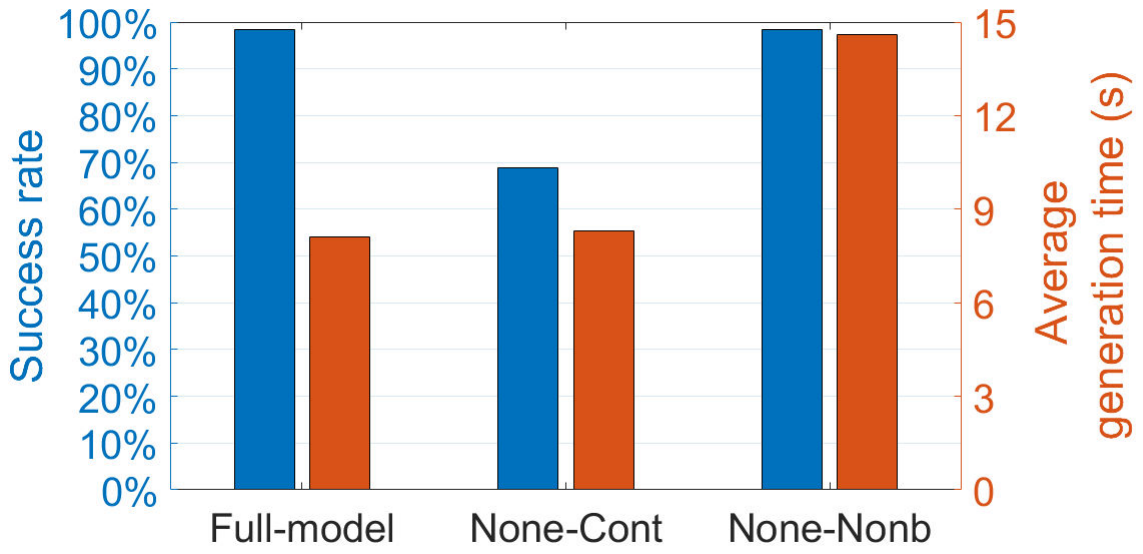


Figure 4.10: The results of performance comparison of our full model against models without controllability guarantee and nonblocking guarantee, respectively. Blue bars represent scene generate success rate, and brown bars represent average scene generation time.

to that the dataset we used for spatial priors training doesn't consider this spatial ambiguity. One of our future work is to use the recorded data from scene adjustment to augment the spatial priors.

Another aspect to generate recognizable realistic scenes is the completeness of the segmented objects. The object detector used in this work sometimes fails to segment complete objects, which can cause difficulty in recognizing them in generated scenes. This may also be one of the reasons that decease the performance of our approach without human adjustment. In our future work, we plan to tackle this issue from two perspectives. The first one is to improve the performance of the object detector in combination with other visual processing techniques. The second is to develop an object completeness detection algorithm and apply it in the image retrieval process.

4.6 Summary

In conclusion, we propose a framework for robotic drawing conditioned on NL descriptions. The proposed approach decomposes the scene generation conditioned on NL into several manageable steps. Instead of learning a direct text-to-pixel mapping or retrieving from a local constrained image material library, the proposed approach utilizes knowledge retrieved from the Internet for

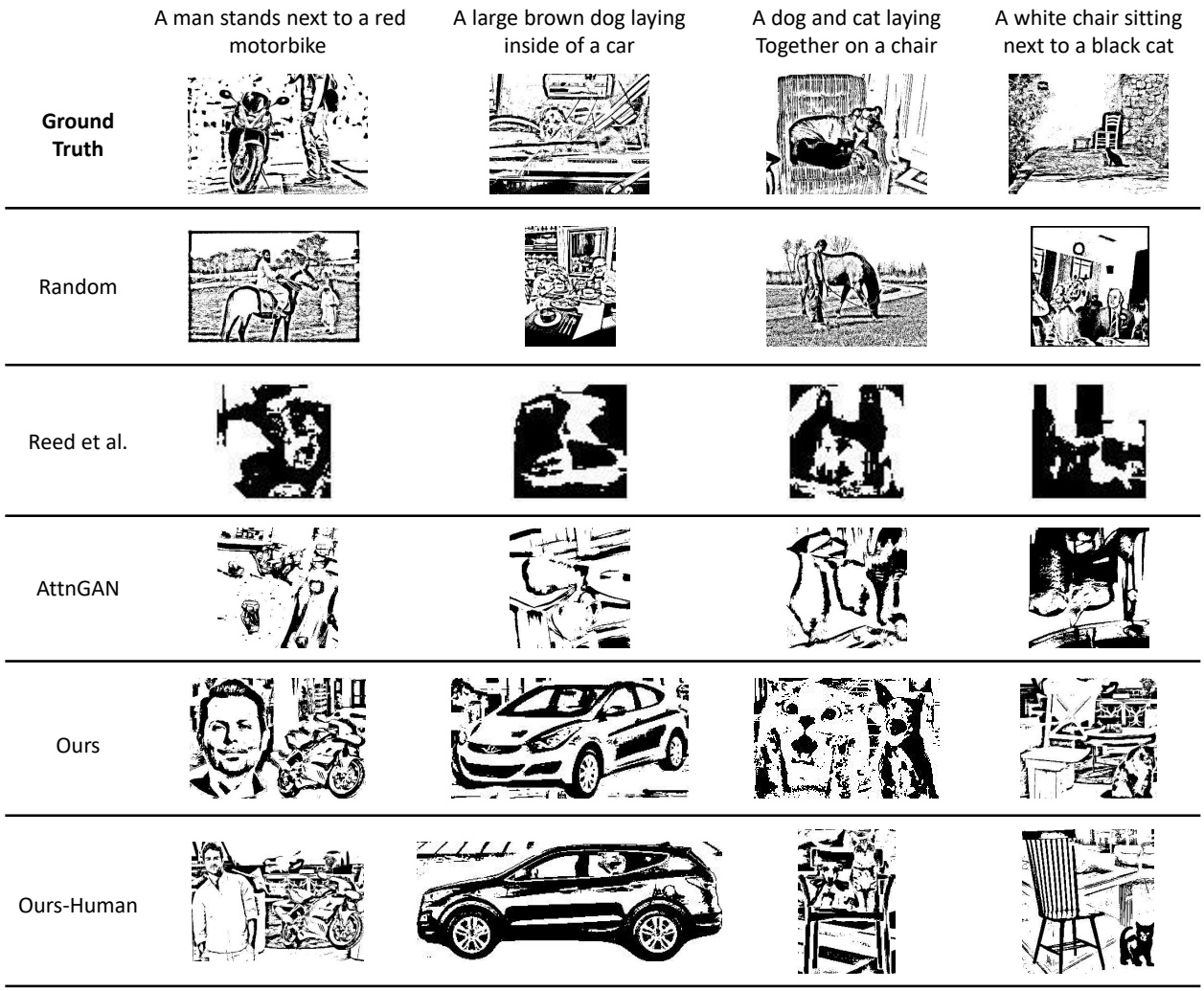


Figure 4.11: Qualitative examples of simulated robotic drawings of generated scenes with the same parameters.

more diverse and creative scene generation and drawing. The developed mechanism to handle unexpected events makes the scene generation more stable and reliable. The interactive modification of scenes helps better alignment with scene descriptions and training error correction.

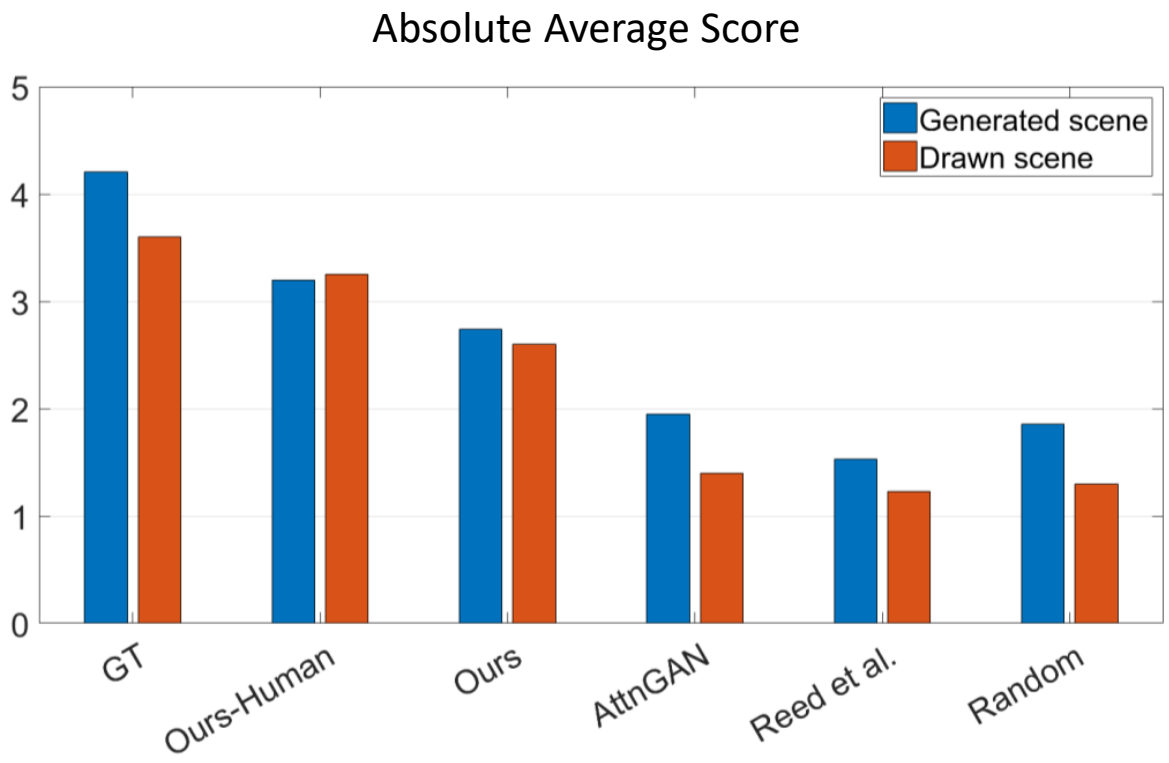


Figure 4.12: Turkers are asked to score how well the scenes match the description on a scale of 1 (very poorly) to 5 (very well). "GT" denotes "Ground Truth". We achieve absolute scores slightly worse than the ground truth, but better than the baselines.

Figure 4.13: Generation results on scenes that are not likely to happen in the real world. The first column shows the unmodified synthesized scenes. The second column scenes are modified by the instructor with language instructions on the first column scenes. The third column shows scene drawings by the robotic painter.

CHAPTER 5

TACTILE FEEDBACK FOR NL BASED BEHAVIOR CONTROL AND PROGRAMMING

5.1 Introduction

The success of robot control and programming relies heavily on the correct state estimation. Besides vision and encoder information, tactile information plays a critical role in robot interaction with its surroundings.

Tactile sensing is an essential survival skill for living creatures and has been a component of robots almost as long as vision. It is extremely useful and widely applied in three types of tasks: manipulation, exploration and response [71]. Slippage plays an important role in completion of the above mentioned tasks. Applications of which are grasp force control [106][107], frictional properties estimation [108], surface texture perception [109], wearable devices [110], prosthetics [111], etc. All of which motivate the desire for stable and general slippage detection methods.

An early slippage detection approach was based on detecting displacements of a moving item in the gripper surface [112]. Various approaches have since been developed for slippage detection. On signal type aspect, researchers have used vibrations [108], acoustic emissions [113], variation of normal force, shear force [107], etc.,which are generated during the slippage to analyze and determine the slippage condition. Approaches to classification and decision on slippage occurrence have employed neural networks [114], frequency analysis [115][116], the Coulomb friction model [117], principle component analysis [118], etc. However, these methods suffer several deficiencies:

1. They are dependent on either custom hardware or specific sensor signal type.

2. In real applications, slippage can occur in both static and dynamic environments. The above methods only demonstrate their ability to detect slippage in static environments.

3. They can only detect occurrence of slippage rather than the sliding velocity. The main reason for this issue is that these methods are unable to detect or estimate sliding displacements of slippage. Recent research has been conducted into developing a method to detect sliding distance [111][1]. The method proposed in [111], however, heavily relies on custom hardware. While in [1], the authors design a slip vector to measure the sliding distance based on optical flow, but it is less accurate compared with our proposed approach.

To provide partial remedies for the above issues, we propose two correlation based approaches for slippage detection using a tactile sensor array, which is one of the oldest and most common tactile sensor types. The first approach is able to distinguish between slip and nonslip, independent of the sensor signal type, slip type (translational or rotational) and sliding direction. In addition, it is proved to work well in both static and dynamic environments. The second approach emphasizes its ability to detect slippage velocity with tactile sensor array information.

The chapter is organized as follows. Section 5.2 illustrates the proposed methods in detail. Section 5.3 provides an overview of the experimental setup and implementation details. While in Section 5.4 we validated our approaches with experimental data. Section 5.5 summarizes this chapter.

5.2 Data Correlation Approach

Generally, the common setup for testing and evaluation of a slippage detection algorithm is to use either electric motor or gravity as driving force to create a sliding motion between the object and the tactile sensor array. Assuming the tactile sensor array has the dimensions of $M \times N$, the output of the sensor usually has the same dimensions as the sensor array. The output signal can be normal pressure, shear force, voltage, etc. In the following sections, we illustrate in detail two data correlation approaches that are independent of sensor signal types, only taking sensor signal spatial distribution and values into consideration. Figure 5.1 briefly demonstrates the proposed data correlation approaches.

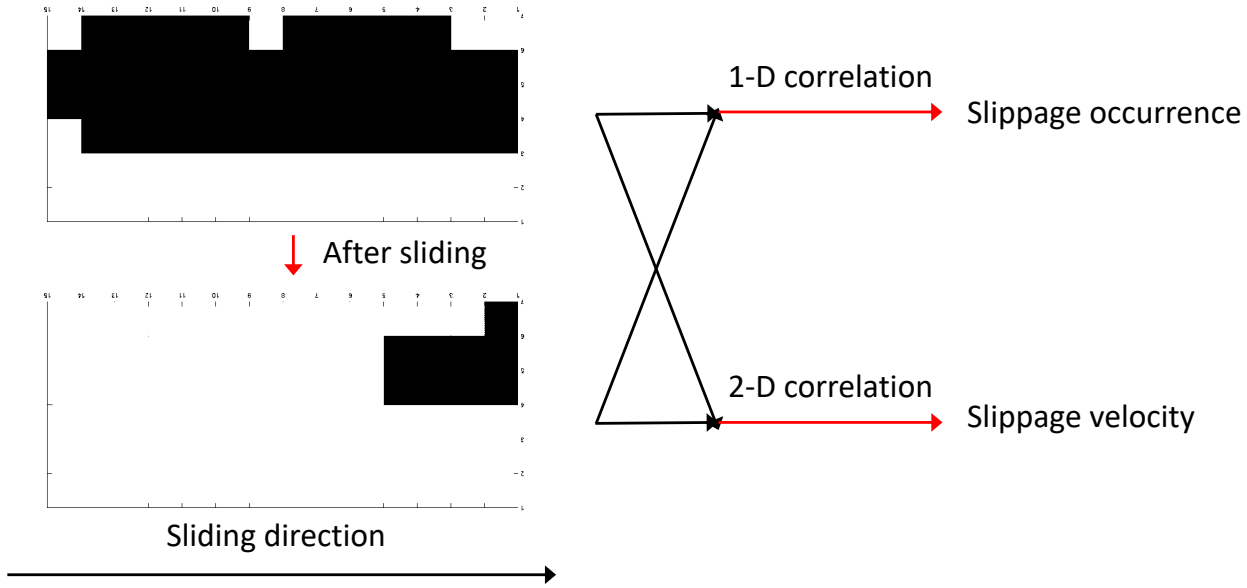


Figure 5.1: Illustration of data correlation based approaches. The left subfigures are two tactile images (black for taxels in contact with grasped object, white for no contact).

5.2.1 1-D Rank Correlation for Slippage Detection

During a slippage, the larger the displacement, the less similarity between the initial tactile sensor array data and the current tactile sensor array data. So the similarity between sensor array data sampled at different instants in time can be used as a criterion to determine the occurrence of slippage. To this end, we employ 1-D rank correlation to characterize the similarity and determine the slip condition. The correlation coefficient between two sensor array data is calculated as follows:

$$r_{XY} = \frac{(X-\overline{X})(Y-\overline{Y})^T}{\sqrt{(X-\overline{X})(X-\overline{X})^T}\sqrt{(Y-\overline{Y})(Y-\overline{Y})^T}} \tag{5.1}$$

both X and Y represent $M \times N$ tactile sensor array data rearranged as $1 \times MN$ (or $MN \times 1$) vectors. X is the initial tactile sensor array data that is not equal to zero, while Y denotes tactile sensor readings sampled at later instants. $\overline{X}$ and $\overline{Y}$ are mean value vectors of X and Y, respectively. The result of Equation 5.1 is a correlation coefficient that reflects the similarity between the two vector-form sensor data X and Y.

Then, we implement Fast Fourier Transform (FFT) over every K calculated correlation coefficients, and compare the amplitude of the first frequency component ($F[1]$) with an experimentally-

tuned threshold ε. If the amplitude is greater than the threshold, the grasped object is considered to happen. Otherwise, the object is considered to be static.

$$F[n] = \sum_{i=0}^{K-1} f[i] e^{-j\frac{2\pi}{K}ni} \qquad n = 0, 1, 2, ..., K-1 \tag{5.2}$$

5.2.2 2-D Cross Correlation for Slippage Velocity Detection

For any $M \times N$ tactile sensor array, the instantaneous values constitute a tactile image of that array. The shift of object on tactile image reflects its displacements on the contact surface between the gripper finger and the object, which can be calculated via 2-D cross correlation.

The 2-D cross correlation of an $M \times N$ matrix X and a $P \times Q$ matrix Y is a $(M+P-1) \times (N+Q-1)$ matrix C given by

$$C(k,l) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X(m,n) \overline{Y}(m-k, n-l) \tag{5.3}$$

$$-(P-1) \leq k \leq (M-1), \ -(Q-1) \leq l \leq (N-1),$$

where $\overline{Y}$ denotes complex conjugation of Y.

Following is an example to illustrate the working principle of the 2-D correlation. As shown in Figure 5.2, O_1 and O_2 represent positions of a grasped object before and after shift, respectively. Without losing generality, we assume the object has higher pixel value than the background. As shown in Equation 5.3, each element of the output matrix, $C(k,l)$, is calculated by shifting Y with certain displacements. $C(k,l)$ has both negative and positive row and column indices. A negative row index denotes an upward shift of the rows of Y, and a negative column index represents a leftward shift of the columns of Y. Similarly, positive row and column indices denote the shift of opposite directions. The position of the element with maximum values means the original object image and shifted object image completely overlap with each other after certain displacements along the horizontal and vertical directions. If the background has higher values than the object, then the position of minimum element indicates the shifts along both horizontal and vertical

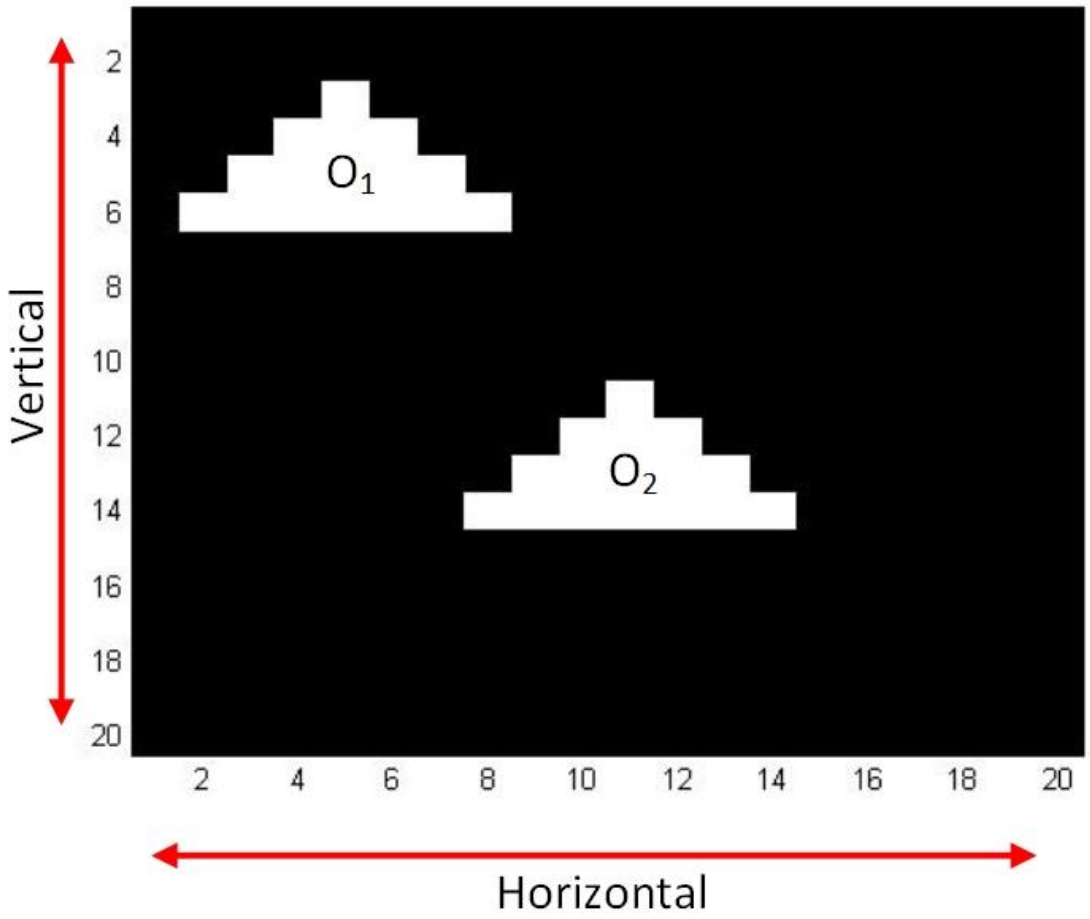


Figure 5.2: Illustration of 2-D correlation. O_1 is the initial object position and O_2 for the object position after shift.

directions. Assume the maximum value element is $C(k_{max}, l_{max})$, with the tactile sensor spatial resolution d_{taxel} as a priori, two dimensional displacements can be obtained:

$$\begin{cases} d_v = k_{max} \cdot d_{taxel} \\ d_h = l_{max} \cdot d_{taxel} \end{cases} \tag{5.4}$$

where the subscript v denotes vertical and h for horizontal.

With the number of samples N_{sample} between X and Y, and the sampling rate f_{sample}, the time interval of the shift can be calculated as:

$$\triangle t = \frac{N_{sample}}{f_{sample}} \tag{5.5}$$

Then the vertical and horizontal slippage velocity is:

$$\begin{cases} s_v = \frac{d_v}{\triangle t} \\ s_h = \frac{d_h}{\triangle t} \end{cases} \tag{5.6}$$

Here, by vertical we mean the direction along the long side of the tactile sensor pad and correspondingly the horizontal represents the direction along the short side.

5.3 Robot Experimental System

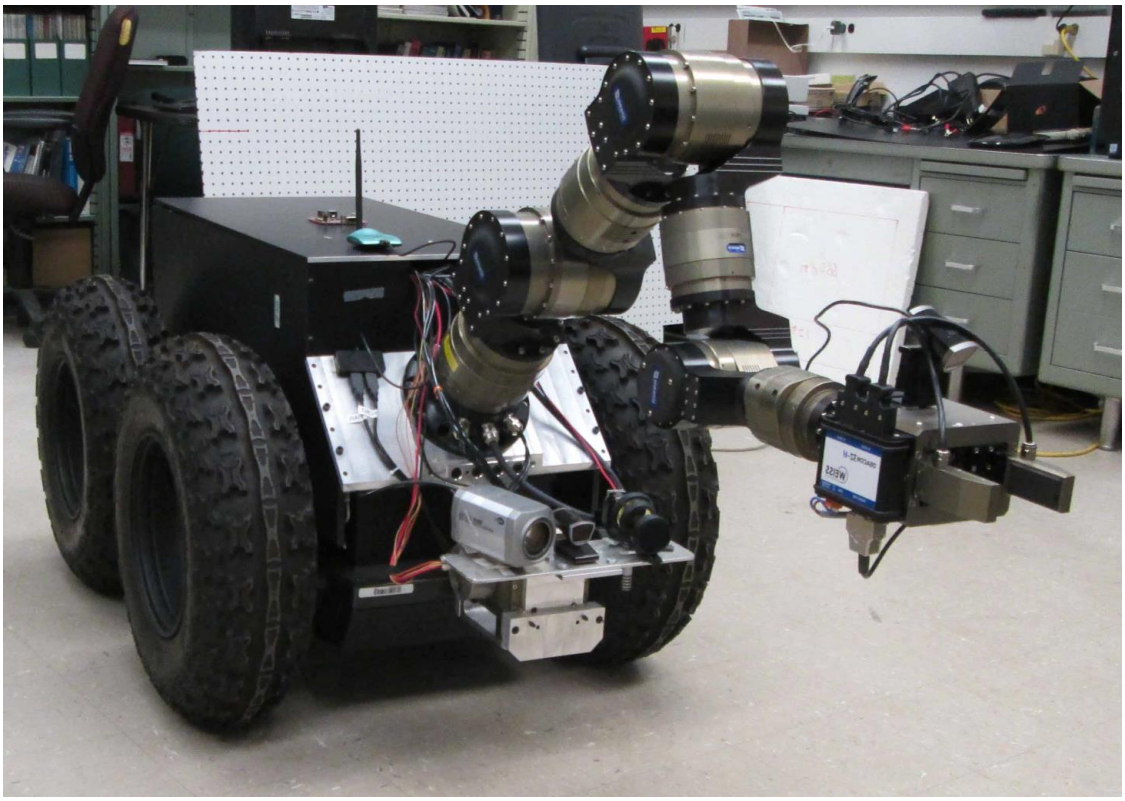


Figure 5.3: Mobile manipulator used as experimental platform.

Experiments were conducted using the robot experimental platform as shown in Figure 5.3. It has a four wheeled base, a seven-degree-of-freedom (DOF) arm and a one-DOF parallel-jaw gripper. Figure 5.4 shows the parallel-jaw gripper mounted on the arm. Its actuator is a brushless DC servo-motor with gear mechanism, a spindle and an encoder. Each finger of the gripper is equipped with a tactile pressure sensor array with dimensions of $24.4 \times 51.4 \times 5.4$ *mm*. Each sensor

array consists of 14×6 sensor cells with a spatial resolution of 3.4 *mm*. These resistive sensors (manufactured by Weiss Robotics GmbH & Co. KG, Germany) measure the normal pressure which is applied in each sensed region [119]. As shown in Figure 5.4, the entire sensor pad surface is covered by a silicone rubber for protection and enhance compliance for successful grasp.

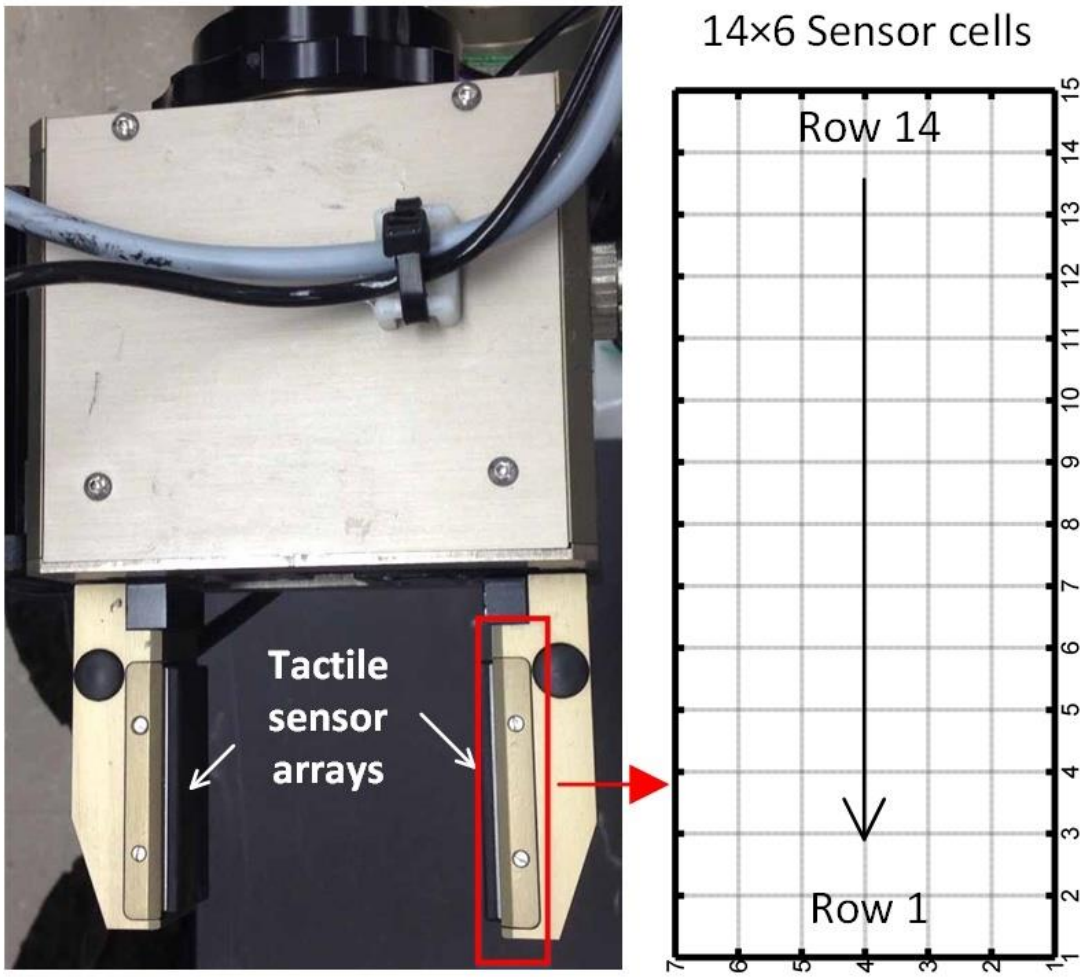


Figure 5.4: The robot gripper. The pressure sensor arrays are attached to the gripper's fingers under the silicone rubber.

In order to evaluate the performance of the second approach, another setup has been designed, as shown in Figure 5.5. A high-speed camera (CASIO EXILIM EX-H5) is used here to record the sliding motion of the grasped object, with a frame rate of 480 frames per second (fps). Each frame has the dimensions of 160×224 pixel. The start and end time point of the slip, the displacement at any instant and corresponding slippage velocity can be recorded and calculated from the frames

accurately. For the purpose of tracking the sliding movement of the grasped object, white markers have been stuck to the object surface.

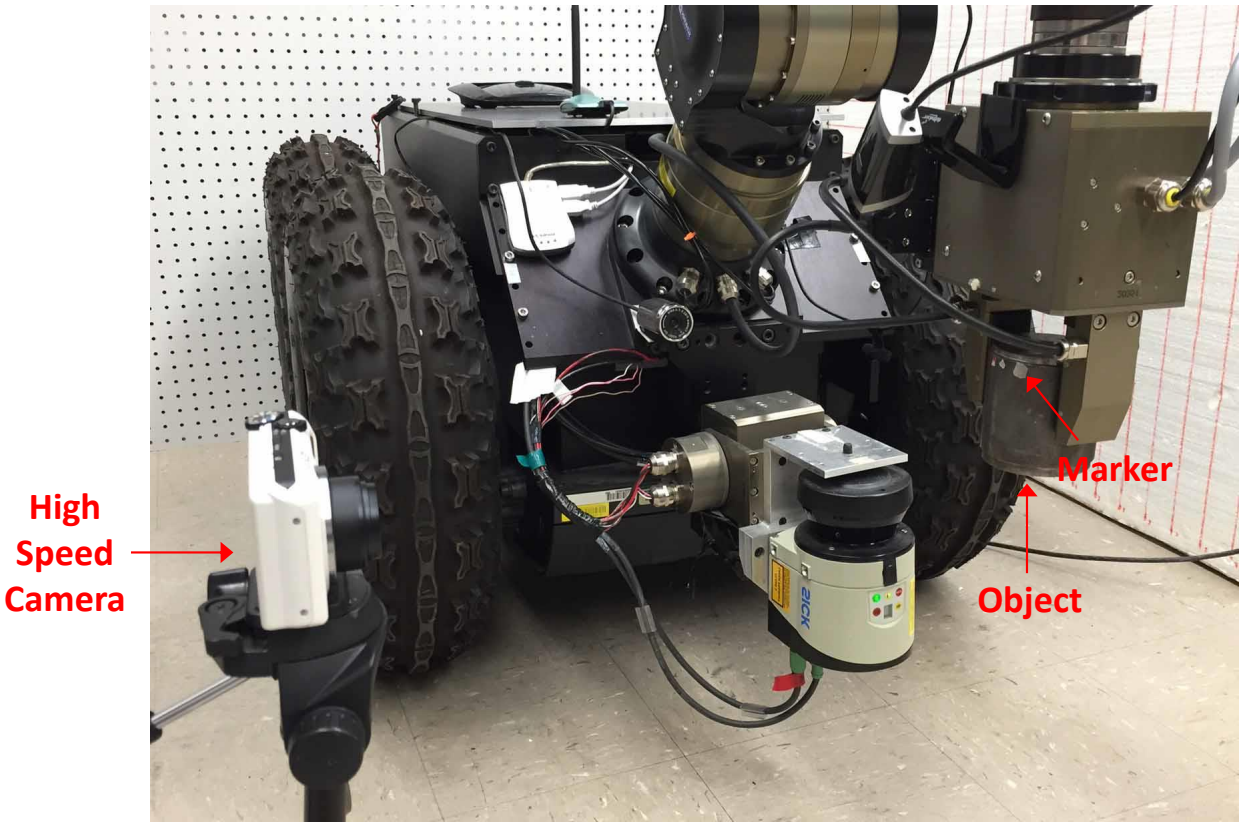


Figure 5.5: This system is used to evaluate the performance of the second approach. White markers are stuck to the cylinder's surface and facing the camera lens.

5.4 Experimental Results

This section displays experimental results of the proposed data correlation approaches. For the first method, we show its ability to detect slippage, both translational and rotational, in static as well as dynamic environments. For the second approach, the ability to detect displacements and velocity of the sliding motion has been presented. Also, we compare the velocities calculated by the proposed approach, the method proposed in [1] and from the video clip recorded by a high-speed camera.

5.4.1 Translational and Rotational Slippage Detection

As described in Equation 5.1, similarity between two tactile sensor array data is calculated as correlation coefficient. During either translational or rotational slippage, variations in both the value and distribution of the normal pressure cause changes of calculated correlation coefficient sequence. All sensor cells were sampled simultaneously at a rate of 20 Hz in this setup.

Figure 5.6 shows variations of the amplitude of the first frequency component, correlation coefficient and grasp force in a static configuration. The gripper is gripping a metal cylinder with a grasp force large enough to hold it. The cylinder weighs 2 kg. The gripper posture is perpendicular to the ground. The grasped force fluctuates slightly due to mechanical vibrations. By FFT on the correlation coefficient sequence, all the amplitudes of first frequency component were less than the threshold ($\varepsilon = 5 \times 10^{-3}$ when $K = 4$ in our setup). The amplitudes that are greater than the threshold have been marked red, otherwise, they are marked as blue.

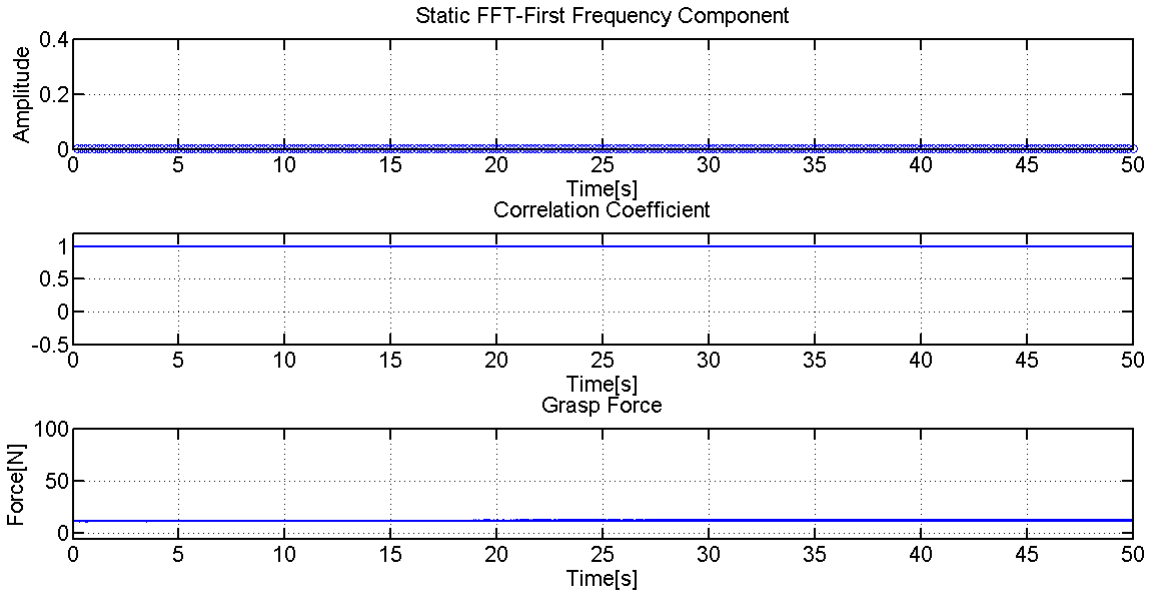


Figure 5.6: Experimental results of the static case.

Figures 5.7 through Figure 5.10 show results of slippage detection under different conditions. Figure 5.7 is about translational slippage. The initial configuration is the same as the static case. Then the gripper releases slowly to a smaller force which is not able to hold the metal cylinder. The

releasing velocity of the gripper fingers was 1 mm/s. The cylinder began to slip downward. As soon as the grasped object begins to slip, correlation coefficient starts to decrease and the amplitude of the first frequency component goes up.

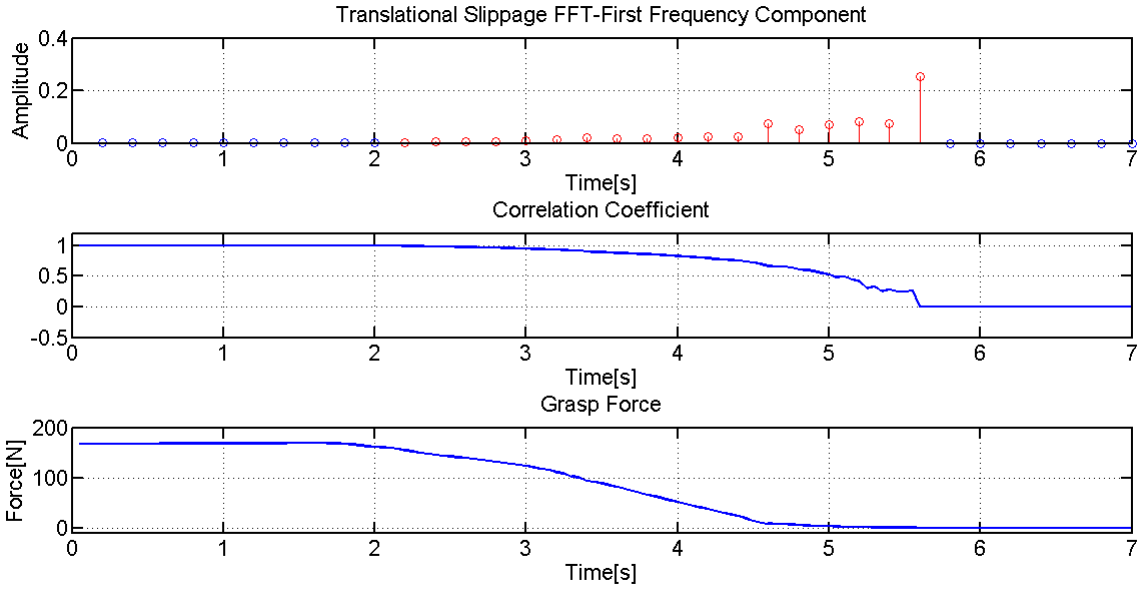


Figure 5.7: Experimental results of translational slippage case.

In Figure 5.8, the difference in experimental setup compared with the translational one is that the grasped object is not released gradually, but kept in the gripper and made to rotate manually. In the middle plot of rotational slippage we can observe that correlation coefficient could still return to a relatively high value. This is due to the symmetry of the cylinder used in the experiments.

In Figure 5.9, both translational and rotational slippage occurred simultaneously. Similarly, at first the cylinder is gripped with a large force. The initial posture is neither perpendicular nor parallel to the ground, but has an angle with the horizontal plane. During the release process, the cylinder slides and rotates by gravity at the same time.

In above setups, the gripper posture is perpendicular to the ground. While in Figure 5.10, the gripper is set to be horizontal. All other keep the same as the translational slippage setup. During a slippage the force distribution and value do not change significantly, so is the correlation coefficient. This is because the cylinder is longer than the short side of the gripper. However, the fluctuation of correlation coefficient during a slippage is still larger than that in static case.

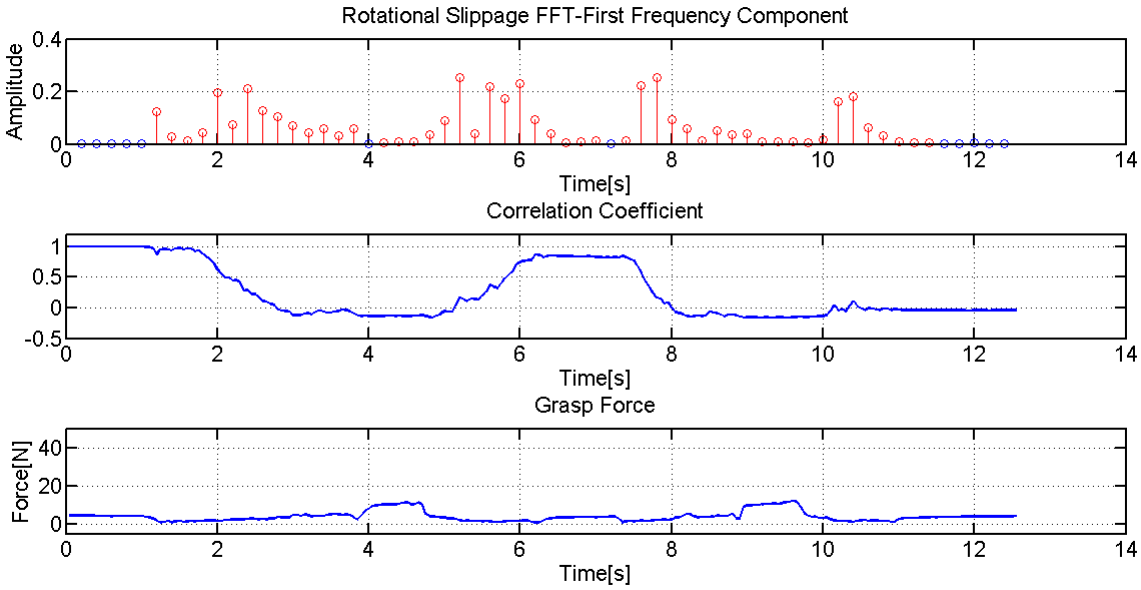


Figure 5.8: Experimental results of rotational slippage case.

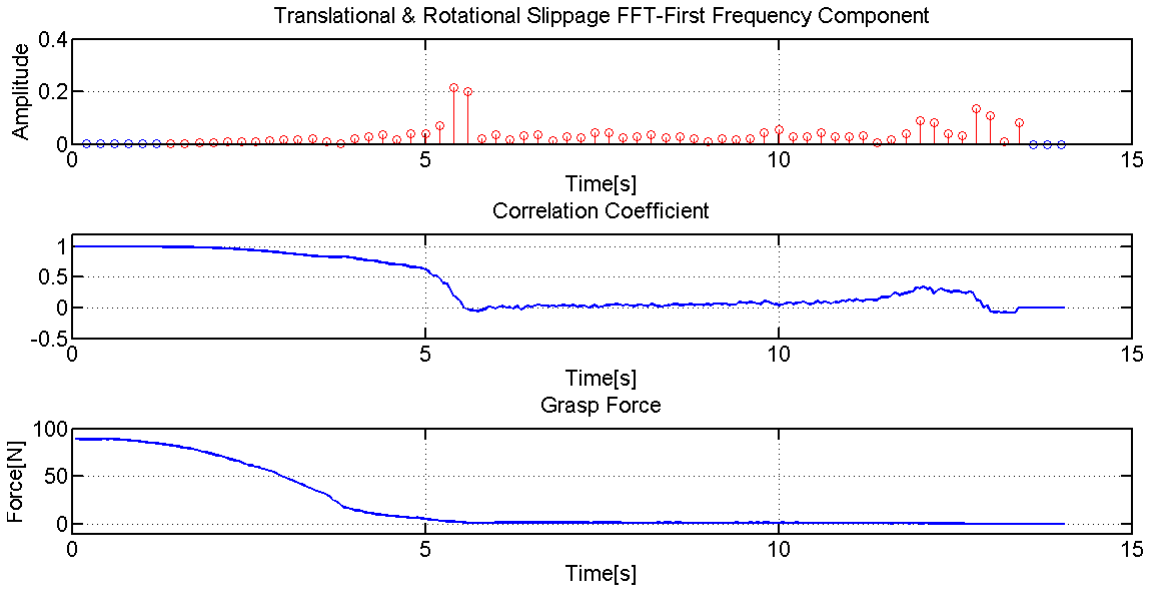


Figure 5.9: Experimental results of combined slippage case.

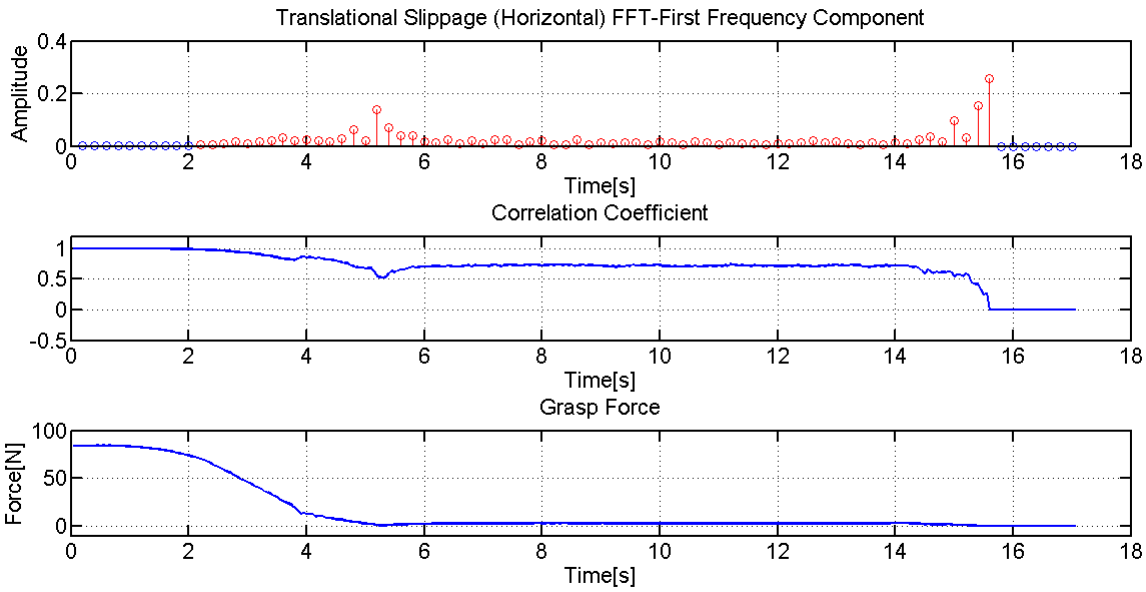


Figure 5.10: Experimental results of translational slippage case, the gripper was set horizontally.

1-D correlation based method employs sensor signal distribution and values. During slippage process, variations of these two features cause heavier fluctuation in correlation coefficient sequence than in the static case. By the static case we mean both the arm and gripper do not move, while in real applications it is not always the case. A slippage usually occurs when the arm/gripper is moving/rotating. In the following section we investigate this issue.

5.4.2 Slippage Detection In Dynamic Environments

In order to evaluate the performance of the proposed method under dynamic situations, we implemented experiments in which the robotic arm is moving while the cylinder keeps relatively static to the gripper. Four common movements are considered here, which can constitute most of the complex behaviors, as shown in Figure 5.11. Figure 5.12 shows 1-D correlation based slippage detection approach works normally against the impact of gripper rotation. Similar results are also observed under other three conditions.

Figure 5.13 through Figure 5.16 display results of slippage detection under four common movements. In Figure 5.13, a slip occurs when releasing the gripper. It demonstrates that slippage under this condition can still be detected.

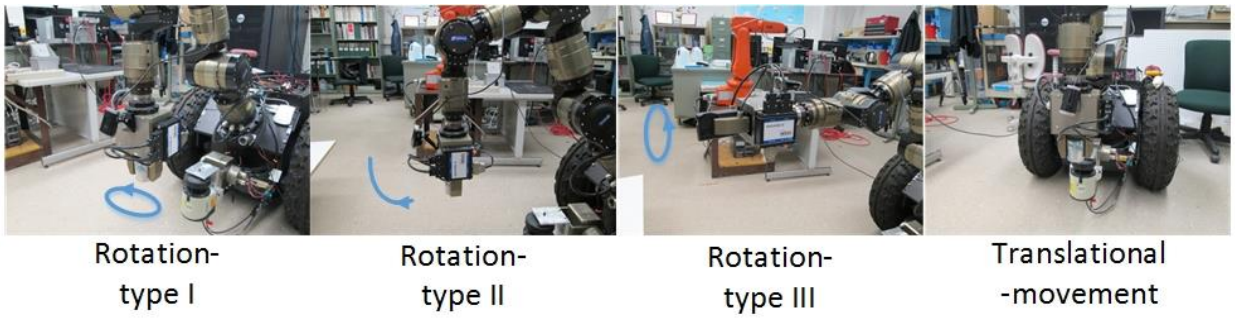


Figure 5.11: Four basic motions of the robotic arm. Rotation type I is to rotate the gripper when it is perpendicular to the ground. Rotation type II means to rotate the arm rather than the gripper. Rotation type III is almost the same as type I, but to place the gripper horizontally. The fourth motion is only a translational movement of the arm without any rotation

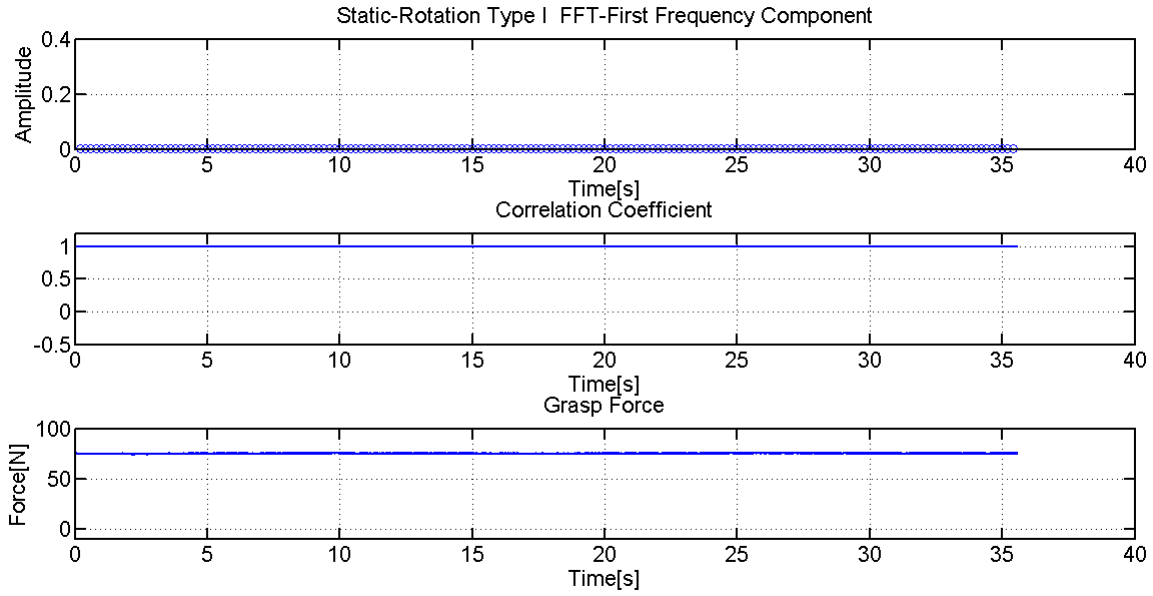


Figure 5.12: Experimental results of rotation type I in static case.

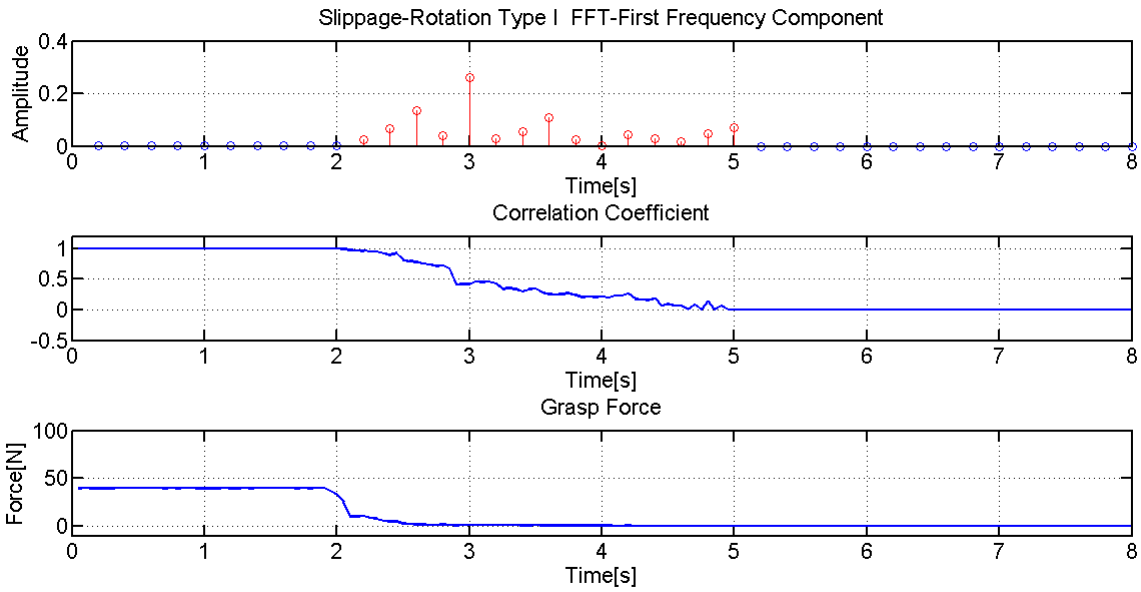


Figure 5.13: Experimental results of rotation type I in slippage case.

Figure 5.14 and 5.15 show results in which the cylinder slips when the arm is in type II and type III rotation mode, respectively. The grasped item is grasped in gripper with a smaller grasp force that is insufficient to hold the object.

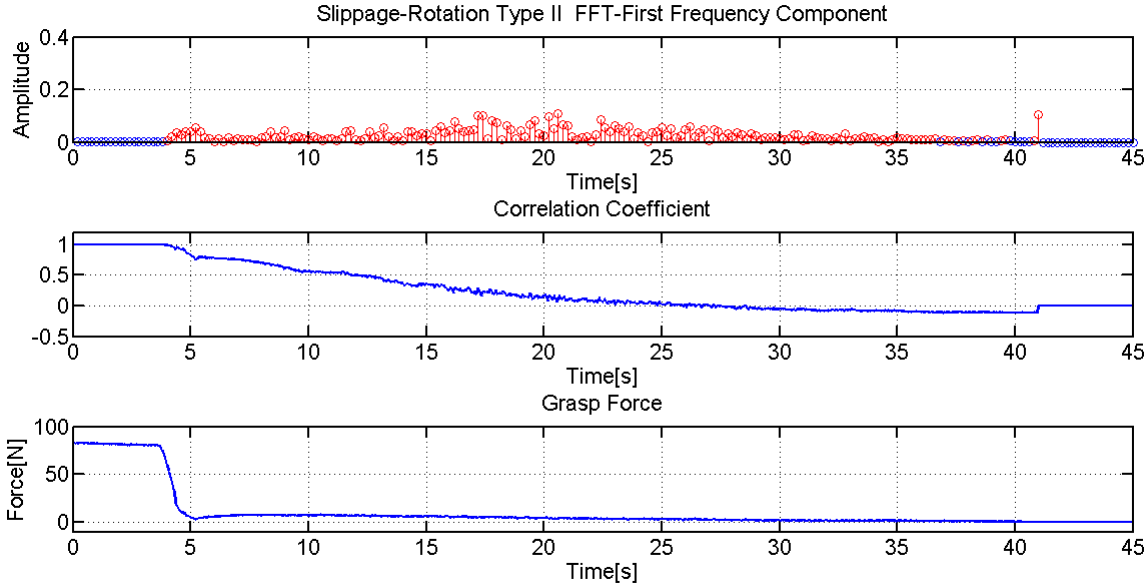


Figure 5.14: Experimental results of rotation type II in slippage case.

Figure 5.16 describes the slip process during a translational motion of the robotic arm. In this

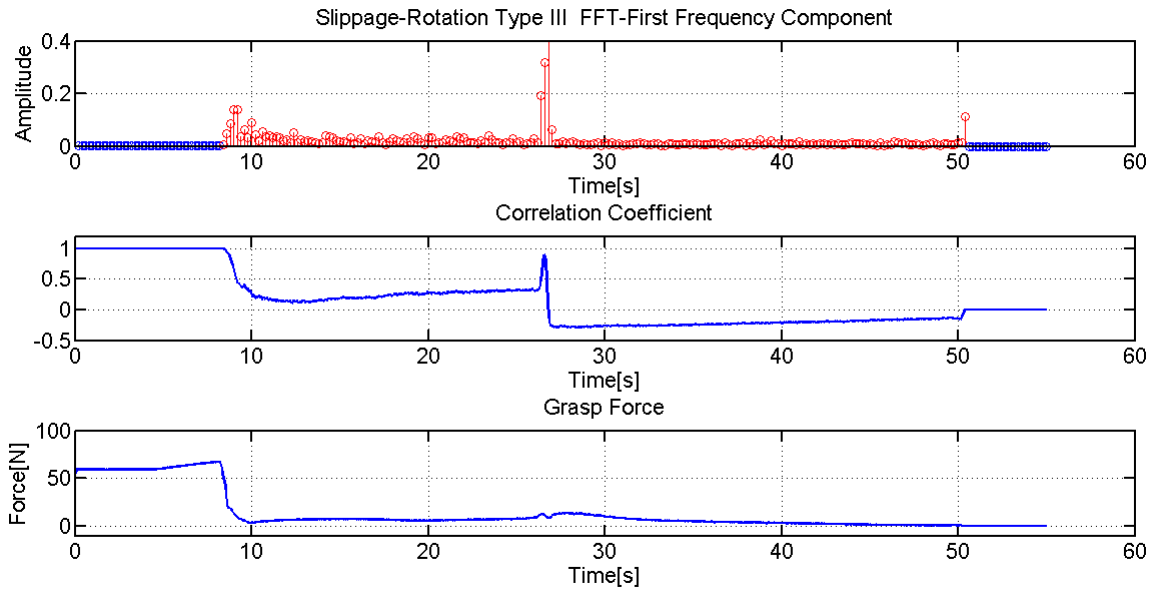


Figure 5.15: Experimental results of rotation type III in slippage case.

experiment, the arm moves from a point of lower left corner to another point of top right corner in its working space. At the same time, the gripper slowly releases the its fingers to allow a slippage to happen.

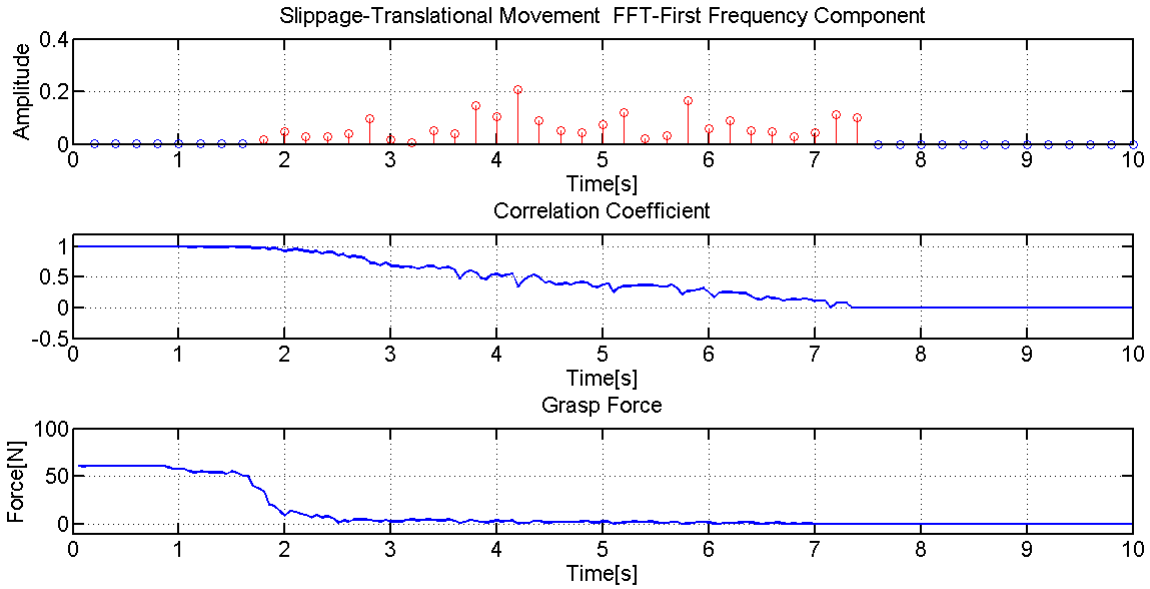


Figure 5.16: Experimental results of translational movement in slippage case.

The above results show the proposed 1-D correlation based approach is able to detect slippage

under multiple conditions, especially in dynamic environments with mechanical vibrations caused by arm/gripper movements.

5.4.3 Slippage Velocity Detection

In this section, we evaluate the performance of the proposed slippage velocity detection approach with tactile data obtained from experimental evaluation setup as shown in Figure 5.5. The cylinder is gripped firmly at first and then the gripper releases at a rate of 1 mm/s to allow a slippage to occur. All sensor cells are sampled simultaneously at a rate of 47 Hz. And the sliding motion is recorded by both the tactile sensor array and the high-speed camera. In this experiment, only the sliding motion along the slippage direction is considered. The spacial resolution of the video clip is 1.23 $mm/pixel$. The results are shown in Figure 5.17.

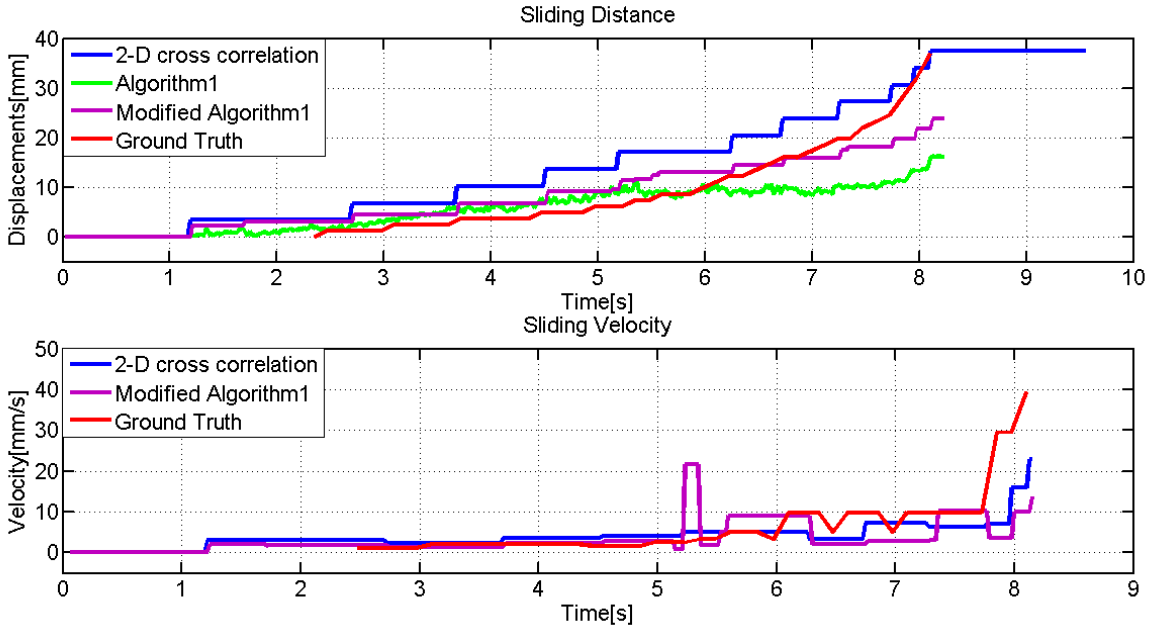


Figure 5.17: Comparison of experimental results. The top plots represent sliding distance calculated from 2-D cross correlation, algorithm1 [1], modified algorithm1, and ground truth. The bottom plots are the sliding velocities computed from sliding distance of 2-D cross correlation, modified algorithm1 and ground truth.

The top plot of Figure 5.17 shows the sliding distances detected by the 2-D correlation approach, the method in [1] and measured from video clip. The actual sliding distance is 36.9 mm,

while the detected distance is 37.4 *mm*. The difference comes from the limited spacial resolution of tactile sensor array since the detected displacements can only be integral multiple of the spacial resolution. Also, this difference between the detection and ground truth is dependent on the initial start position. The largest difference should be no greater than the spacial resolution. This also explains the earlier detection of slippage by the proposed algorithm than results from the video clip.

The green plot represents the displacement calculated by algorithm 1 [1]. Due to the vibration caused by slippage and the nonideal object surface, the calculated sliding displacement fluctuates. To smooth the plot, first we assume the later displacements should be no less than former displacements. Additionally, we assume higher row taxels in tactile image have larger values. With these two assumptions the algorithm 1 [1] is modified and the results seem to be more reasonable. But overall, the method presented in this paper has a better accuracy.

The trends of the slippage displacements and velocity detected by the 2-D correlation based algorithm are consistent with the results from the high-speed camera video clip. The differences between these two results are due to the limited spacial resolution of tactile sensor array and sampling rate. The performance could be improved by increasing either the spacial resolution or the sampling rate, or both.

Compared with the slippage velocity detection method presented in [111], which depends on self designed tactile sensor array to detect slippage velocity. The proposed approach has a wider application. Theoretically, it is suitable for any tactile sensor arrays.

5.5 Summary

In this chapter, two correlation based approaches are proposed for slippage detection. The proposed data correlation based slippage detection approaches are able to detect not only the occurrence of a slippage but also the slippage velocity as well. The estimated status is later sent back to the high level controller for task planning and supervised implementation. In addition, the slippage status information can be used for grasp status estimation and grasp force control. The

proposed methods are independent of the sensor signal type, which can be applied to other tactile arrays.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In this dissertation, a DES inspired approach for NL based robot control and programming is investigated. The aim of such an approach is to achieve reliable robotic behavior management through NL under unstructured environments. But with the underspecification of language intended for human use and complexity of environments, three challenges exist. First, it is nontrivial to transfer underspecified language input into deterministic behavior output. Second, it is difficult to achieve stable movement under structured environments, especially with underspecified instructions. Third, t is difficult to teach a robot new skills through NL since the physical structure and cognitive capability of robots and human are completely different. This dissertation tries to address these three challenges by using DES inspired control and programming methods.

For the control part, a DES control framework is proposed that can achieve reliable behaviors under dynamic environments with underspecified language instructions. The reliability of the behavior model is achieved by behavior refinement using three properties: controllability, nonblocking, and stability. It is a correct-by-design approach that detects design flaws in advance and avoid costly redesign-rework that occurs late in development cycle. NL can have concepts/commands of different level of abstraction. Accordingly, a hierarchical control approach is proposed to tackle commands of different logic depth, which helps to alleviate state and action explosion. Behavior consistency and stability of hierarchical system is proved. Experimental results on prevalent manipulation scenarios clearly show the developed control framework can achieve satisfactory performance under dynamic environment.

Existing NL based control approaches transform linguistic input into symbolic representation to guide robot movement, which may fail for some applications such as robotic drawing, painting, polishing, etc. controlled by NL. We proposed an approach to generate scene representation in

accordance with language input and to draw the scene on paper. Scene representation contains more details than symbolic representation. In addition, this approach employs knowledge retrieved from web resources instead of laboriously labeled data. Results of standard metrics and human studies show that the generated scenes and drawn scenes have higher recognizability and better alignment with the input language description.

The success of robot application in dynamic environments relies on correct robot state estimation. Gripper status is an important component of robot states. We propose correlation based approach for slippage detection to estimate and monitor grasp status. The proposed approach is able to detect slippage occurrence and velocity without using customized sensor designs or limit to specific sensor signal types.

For the programming part, this dissertation presents a hierarchical programming approach that allows a robot to learn new skills using acquired skills through dialogues and demonstrations. This approach mimics the teach and learning process in human society, and thus is intuitive and friendly to technically untrained users. Moreover, it only requires one-shot training, which is time-efficient and laborless.

6.2 Future Research Work

Based on the research presented in this dissertation, there can be many future research directions, among which only a few are outline in the following.

For using scene representation to control robot behaviors, currently it is only able to synthesize complex scenes using existing resource rather than creating new single instances. For better creation and alignment with input language descriptions, we should design approaches have advantages from the generative approach (such as GAN based methods) and compositional approach using web resources. In addition, the proposed approach has only been applied to robotic drawing and similar applications. Since scene representation can include more details, we should extend this approach to other robotic applications, such as navigation, exploration, house chores, etc.

For robot programming through NL, the proposed approach depends on the assumption that

new behaviors can be represented using atomic states in the state set. What if the a behavior cannot be represented using existing states? This presents a knowledge representation challenge. We need to explore knowledge representation that relates to both high discrete level and low continuous level. In addition, the proposed approach assumes the robot is capable to detect the entire environmental configuration correctly. What if the there are detection errors or part of the environment is unobservale? This challenge may be solve through human-robot interaction via NL. First, this helps to the user to know the true understanding of the robot on the environment, thus to identify the problems that the robot is encountering. Second, the user can employ NL to share information with the robot to compensate for the incapability of robots.

For tactile sensing that provides grasp status to the robot, in this dissertation it is used for robot state estimation in the discrete event level control and grasp force regulation in the continuous level control. Environmental information, such object layout and object attributes, are detected by visual sensors. Tactile features have not been used for object grounding yet, such as textures, softness, roughness, etc. Recognizing these features has helps the robots to have a richer lexicon and better cognition. As a result, it improves object grounding performance by combination of visual and tactile attributes of objects.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Javier Alcazar, Leandro G Barajas, et al. Estimating object grasp sliding via pressure array sensing. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1740–1746, 2012.

[2] Hao Zhong and W Murray Wonham. On the consistency of hierarchical supervision in discrete-event systems. *IEEE Transactions on automatic Control*, 35(10):1125–1134, 1990.

[3] Lanbo She, Shaohua Yang, Yu Cheng, Yunyi Jia, Joyce Y Chai, and Ning 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*, volume 89, 2014.

[4] Terry Winograd. Understanding natural language. *Cognitive psychology*, 3(1):1–191, 1972.

[5] Hossein Motallebipour and August Bering. A spoken dialogue system to control robots. 2002.

[6] Stanislao Lauria, Guido Bugmann, Theocharis Kyriacou, Johan Bos, and Ewan Klein. Personal robot training via natural-language instructions. *IEEE Intelligent systems*, 16(3):38–45, 2001.

[7] Stanislao Lauria, Guido Bugmann, Theocharis Kyriacou, and Ewan Klein. Mobile robot programming using natural language. *Robotics and Autonomous Systems*, 38(3):171–181, 2002.

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

[9] Michael Brenner, Nick Hawes, John D Kelleher, and Jeremy L Wyatt. Mediating between qualitative and quantitative representations for task-orientated human-robot interaction. In *IJCAI*, pages 2072–2077, 2007.

[10] Juraj Dzifcak, Matthias Scheutz, Chitta Baral, and Paul Schermerhorn. What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, pages 4163–4168, 2009.

[11] Mario Bollini, Stefanie Tellex, Tyler Thompson, Nicholas Roy, and Daniela Rus. Interpreting and executing recipes with a cooking robot. In *Experimental Robotics*, pages 481–495. Springer, 2013.

[12] Albert S Huang, Stefanie Tellex, Abraham Bachrach, Thomas Kollar, Deb Roy, and Nicholas Roy. Natural language command of an autonomous micro-air vehicle. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2663–2669, 2010.

[13] David L Chen and Raymond J Mooney. Learning to interpret natural language navigation instructions from observations. *San Francisco, CA*, pages 859–865, 2011.

[14] Wataru Takano, Ikuo Kusajima, and Yoshihiko Nakamura. Generating action descriptions from statistically integrated representations of human motions and sentences. *Neural Networks*, 80:1–8, 2016.

[15] Dipendra K Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research*, 35:281–300, 2015.

[16] Lanbo She, Yu Cheng, Joyce Y Chai, Yunyi Jia, Shaohua Yang, and Ning Xi. Teaching robots new actions through natural language instructions. In *Proc. of IEEE International Symposium on Robot and Human Interactive Communication*, pages 868–873, 2014.

[17] Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, pages 403–415. Springer, 2013.

[18] Thomas Kollar, Stefanie Tellex, Matthew R Walter, Albert Huang, Abraham Bachrach, Sachi Hemachandra, Emma Brunskill, Ashis Banerjee, Deb Roy, Seth Teller, et al. Generalized grounding graphs: A probabilistic framework for understanding grounded language. *Journal of Artificial Intelligence Research*, 2013.

[19] Thomas Kollar, Stefanie Tellex, Deb Roy, and Nicholas Roy. Toward understanding natural language directions. In *Proc. of ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 259–266, 2010.

[20] Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proc. of Annual Meeting on Association for Computational Linguistics*, pages 423–430. Association for Computational Linguistics, 2003.

[21] Moritz Tenorth, Daniel Nyga, and Michael Beetz. Understanding and executing instructions for everyday manipulation tasks from the world wide web. In *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, pages 1486–1491. IEEE, 2010.

[22] Daniel Nyga and Michael Beetz. Everything robots always wanted to know about housework (but were afraid to ask). In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 243–250. IEEE, 2012.

[23] Gheorghe Lisca, Daniel Nyga, Ferenc Bálint-Benczédi, Hagen Langer, and Michael Beetz. Towards robots conducting chemical experiments. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5202–5208, 2015.

[24] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Where's waldo? sensor-based temporal logic motion planning. In *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, pages 3116–3121. IEEE, 2007.

[25] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Translating structured english to robot controllers. *Advanced Robotics*, 22(12):1343–1359, 2008.

[26] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[27] Daniel J Brooks, Constantine Lignos, Cameron Finucane, Mikhail S Medvedev, Ian Perera, Vasumathi Raman, Hadas Kress-Gazit, Mitch Marcus, and Holly A Yanco. Make it so: Continuous, flexible natural language interaction with an autonomous robot. In *Grounding language for physical systems workshop at the AAAI conference on artificial intelligence*, 2012.

[28] Vasumathi Raman, Constantine Lignos, Cameron Finucane, Kenton C Lee, Mitch Marcus, and Hadas Kress-Gazit. Sorry dave, i'm afraid i can't do that: Explaining unachievable robot tasks using natural language. Technical report, University of Pennsylvania Philadelphia United States, 2013.

[29] Vasumathi Raman and Hadas Kress-Gazit. Explaining impossible high-level robot behaviors. *IEEE Transactions on Robotics*, 29(1):94–104, 2013.

[30] Yu Cheng, Yunyi Jia, Rui Fang, Lanbo She, Ning Xi, and Joyce Chai. Modelling and analysis of natural language controlled robotic systems. *IFAC Proceedings Volumes*, 47(3):11767–11772, 2014.

[31] Jean E Sammet. The use of english as a programming language. *Communications of the ACM*, 9(3):228–230, 1966.

[32] Alan W Biermann, Bruce W Ballard, and Anne H Sigmon. An experimental study of natural language programming. *International journal of man-machine studies*, 18(1):71–87, 1983.

[33] George E Heidorn. Natural language inputs to a simulation programming system: An introduction. Technical report, Monterey, California. Naval Postgraduate School, 1971.

[34] George E Heidorn. *Simulation programming through natural language dialogue*. IBM Thomas J. Watson Research Division, 1973.

[35] George E Heidorn. English as a very high level language for simulation programming. *ACM SIGPLAN Notices*, 9(4):91–100, 1974.

[36] George E Heidorn. Automatic programming through natural language dialogue: A survey. *IBM Journal of research and development*, 20(4):302–313, 1976.

[37] Lawrence I. Lieberman and Michael A. Wesley. Autopass: an automatic programming system for computer controlled mechanical assembly. *IBM Journal of Research and Development*, 21(4):321–333, 1977.

[38] Lance A Miller. Natural language programming: Styles, strategies, and contrasts. *IBM Systems Journal*, 20(2):184–215, 1981.

[39] Adam Vogel and Dan Jurafsky. Learning to follow navigational directions. In *Proc. of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 806–814. Association for Computational Linguistics, 2010.

[40] Satchuthananthavale RK Branavan, Harr Chen, Luke S Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *Proc. of Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 82–90, 2009.

[41] Nobuyuki Shimizu and Andrew R Haas. Learning to follow navigational route instructions. In *IJCAI*, volume 9, pages 1488–1493, 2009.

[42] Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62, 2013.

[43] Maj Stenmark and Pierre Nugues. Natural language programming of industrial robots. In *International Symposium on Robotics (ISR)*, pages 1–5. IEEE, 2013.

[44] Maj Stenmark and Jacek Malec. Describing constraint-based assembly tasks in unstructured natural language. *IFAC Proceedings Volumes*, 47(3):3056–3061, 2014.

[45] Chris Quirk, Raymond Mooney, and Michel Galley. Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL-15)*, pages 878–888, 2015.

[46] Paul E Rybski, Kevin Yoon, Jeremy Stolarz, and Manuela M Veloso. Interactive robot task training through dialog and demonstration. In *Proc. of ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 49–56. IEEE, 2007.

[47] Kevin Yoon and Paul E Rybski. Teaching procedural flow through dialog and demonstration. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 807–814, 2007.

[48] Paul E Rybski, Jeremy Stolarz, Kevin Yoon, and Manuela Veloso. Using dialog and human observations to dictate tasks to a learning robot assistant. *Intelligent Service Robotics*, 1(2):159–167, 2008.

[49] Rehj Cantrell, Kartik Talamadupula, Paul Schermerhorn, J Benton, Subbarao Kambhampati, and Matthias Scheutz. Tell me when and why to do it! run-time planner model updates via natural language instruction. In *Proc. of ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 471–478, 2012.

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

[51] Yu Cheng, Jiatong Bao, Yunyi Jia, Zhuhui Deng, Zhiyong Sun, Sheng Bi, Congjian Li, and Ning Xi. Modeling robotic operations controlled by natural language. *Control Theory and Technology*, 15(4):258–266, 2017.

[52] Maya Cakmak and Andrea L Thomaz. Designing robot learners that ask good questions. In *Proc. of ACM/IEEE international conference on Human-Robot Interaction*, pages 17–24, 2012.

[53] Ross A Knepper, Stefanie Tellex, Adrian Li, Nicholas Roy, and Daniela Rus. Recovering from failure by asking for help. *Autonomous Robots*, 39(3):347–362, 2015.

[54] Lanbo She and Joyce Y Chai. Incremental acquisition of verb hypothesis space towards physical world interaction. In *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (ACL), Berlin, Germany*, 2016.

[55] Michael Beetz, Daniel Beßler, Andrei Haidu, Mihai Pomarlan, Asil Kaan Bozcuoglu, and Georg Bartels. Knowrob 2.0? 2nd generation knowledge processing framework for cognition-enabled robotic agents. 2018.

[56] Peter JG Ramadge and W Murray Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[57] The opennlp ccg library. `http://openccg.sourceforge.net/`. Accessed: 2013.

[58] Changsong Liu, Rui Fang, Lanbo She, and Joyce Chai. Modeling collaborative referring for situated referential grounding. In *Proc. of the SIGDIAL Conference*, pages 78–86, 2013.

[59] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, 18(03):265–298, 2004.

[60] RD Brandt, V Garg, R Kumar, F Lin, SI Marcus, and WM Wonham. Formulas for calculating supremal controllable and normal sublanguages. *Systems & Control Letters*, 15(2):111–117, 1990.

[61] Kevin M Passino, Anthony N Michel, and Panos J Antsaklis. Lyapunov stability of a class of discrete event systems. *IEEE Transactions on Automatic Control*, 39(2):269–279, 1994.

[62] Laurie Ricker, Stephane Lafortune, and S Genc. Desuma: A tool integrating giddes and umdes. In *International Workshop on Discrete Event Systems*, pages 392–393, 2006.

[63] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning: theory & practice*. Elsevier, 2004.

[64] Samuel Eilenberg. *Automata, languages, and machines*. 1974.

[65] Alvaro Collet, Manuel Martinez, and Siddhartha S Srinivasa. The moped framework: Object recognition and pose estimation for manipulation. *The International Journal of Robotics Research*, 30(10):1284–1306, 2011.

[66] Jiatong Bao, Yunyi Jia, Yu Cheng, and Ning Xi. Saliency-guided detection of unknown objects in rgb-d indoor scenes. *Sensors*, 15(9):21054–21074, 2015.

[67] David D Grossman. *Programmimg a Computer Controlled Manipulator by Guiding Through the Motions*. IBM T. J. Watson Res. Cen., Res. Rep. RC6393, 1977.

[68] Tomas Lozano-Perez. Robot programming. *Proceedings of the IEEE*, 71(7):821–841, 1983.

[69] Geoffrey Biggs and Bruce MacDonald. A survey of robot programming systems. In *Proc. Australasian Conference on Robotics and Automation*, pages 1–3, 2003.

[70] Wemi Dai and Markus Kampker. User oriented integration of sensor operations in a offline programming system for welding robots. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1563–1567, 2000.

[71] Mark R. Cutkosky, Robert D. Howe, and William R. Provancher. *Springer Handbook of Robotics*. Springer-Verlag, 2008.

[72] The kuka robot programming language. `https://drstienecker.com/tech-332/11-the-kuka-robot-programming-language/`. Accessed: 2016-04-25.

[73] Test based robot programming made easy. `http://www.abb.com/blog/gad00540/1DDE6.aspx?tag=RAPID%20programming`. Accessed: 2016-04-25.

[74] Wesley M Johnston, JR Hanna, and Richard J Millar. Advances in dataflow programming languages. *ACM Computing Surveys*, 36(1):1–34, 2004.

[75] Tiago Boldt Sousa. Dataflow programming concept, languages and applications. In *Doctoral Symposium on Informatics Engineering*, volume 7, page 13, 2012.

[76] Geir E Hovland, Pavan Sikka, and Brenan J McCarragher. Skill acquisition from human demonstration using a hidden markov model. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2706–2711, 1996.

[77] Michael Pardowitz, Steffen Knoop, Ruediger Dillmann, and Raould D Zollner. Incremental learning of tasks from user demonstrations, past experiences, and vocal comments. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(2):322–332, 2007.

[78] Sethu Vijayakumar and Stefan Schaal. Locally weighted projection regression: An o (n) algorithm for incremental real time learning in high dimensional space. In *Proc. International Conference on Machine Learning*, 2000.

[79] Aaron P Shon, Keith Grochow, and Rajesh PN Rao. Robotic imitation from human motion capture using gaussian processes. In *Proc. IEEE-RAS International Conference on Humanoid Robots*, pages 129–134, 2005.

[80] Rui Liu and Xiaoli Zhang. Methodologies for realizing natural-language-facilitated human-robot cooperation: A review. *arXiv preprint arXiv:1701.08756*, 2017.

[81] Constantine Lignos, Vasumathi Raman, Cameron Finucane, Mitchell Marcus, and Hadas Kress-Gazit. Provably correct reactive control from natural language. *Autonomous Robots*, 38(1):89–105, 2015.

[82] Richard E Mayer. Multimedia learning. In *Psychology of learning and motivation*, volume 41, pages 85–139. Elsevier, 2002.

[83] C Lawrence Zitnick, Devi Parikh, and Lucy Vanderwende. Learning the visual interpretation of sentences. In *Proc. of IEEE International Conference on Computer Vision (ICCV)*, pages 1681–1688, 2013.

[84] Angel X Chang, Manolis Savva, and Christopher D Manning. Learning spatial knowledge for text to 3d scene generation. In *EMNLP*, pages 2028–2038, 2014.

[85] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[86] Ben Kehoe, Sachin Patil, Pieter Abbeel, and Ken Goldberg. A survey of research on cloud robotics and automation. *IEEE Trans. Automation Science and Engineering*, 12(2):398–409, 2015.

[87] Xiaojin Zhu, Andrew B Goldberg, Mohamed Eldawy, Charles R Dyer, and Bradley Strock. A text-to-picture synthesis system for augmenting communication. In *AAAI*, volume 7, pages 1590–1595, 2007.

[88] Xinchen Yan, Jimei Yang, Kihyuk Sohn, and Honglak Lee. Attribute2image: Conditional image generation from visual attributes. In *European Conference on Computer Vision*, pages 776–791. Springer, 2016.

[89] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.

[90] Scott E Reed, Zeynep Akata, Santosh Mohan, Samuel Tenka, Bernt Schiele, and Honglak Lee. Learning what and where to draw. In *Proc. of Advances in Neural Information Processing Systems (NIPS)*, pages 217–225, 2016.

[91] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Proc. Indian Conference on Computer Vision, Graphics and Image Processing*, pages 722–729, 2008.

[92] Elman Mansimov, Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Generating images from captions with attention. *arXiv preprint arXiv:1511.02793*, 2015.

[93] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaolei Huang, Xiaogang Wang, and Dimitris Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *arXiv preprint arXiv:1612.03242*, 2016.

[94] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. *arXiv preprint*, 2017.

[95] Hao Dong, Jingqing Zhang, Douglas McIlwraith, and Yike Guo. I2t2i: Learning text to image synthesis with textual data augmentation. In *Proc. of IEEE International Conference on Image Processing*, pages 2015–2019. IEEE, 2017.

[96] Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. Deep semantic role labeling: What works and what is next. In *Proc. of Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 473–483, 2017.

[97] Olga Babko-Malaya. Propbank annotation guidelines. *URL: http://verbs. colorado. edu*, 2005.

[98] Xinlong Huang, Sheng Bi, Min Dong, Heping Chen, Siwen Fang, and Ning Xi. Automatic feature extraction and optimal path planning for robotic drawing. In *IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems*, pages 19–24, 2016.

[99] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *Proc. of IEEE International Conference on Computer Vision*, pages 1529–1537, 2015.

[100] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[101] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3156–3164, 2015.

[102] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proc. of Annual Meeting on Association for Computational Linguistics*, pages 311–318, 2002.

[103] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proc. of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, 2005.

[104] Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. Cider: Consensus-based image description evaluation. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, pages 4566–4575, 2015.

[105] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*, 2004.

[106] Joseph M Romano, Kaijen Hsiao, Günter Niemeyer, Sachin Chitta, and Katherine J Kuchenbecker. Human-inspired robotic grasp control with tactile sensing. *IEEE Transactions on Robotics*, 27(6):1067–1079, 2011.

[107] Takashi Maeno, Shinichi Hiromitsu, and Takashi Kawai. Control of grasping force by detecting stick/slip distribution at the curved surface of an elastic finger. In *Proc. IEEE International Conference on Robotics and Automation*, volume 4, pages 3895–3900, 2000.

[108] Marc R Tremblay and Mark R Cutkosky. Estimating friction using incipient slip sensing during a manipulation task. In *Proc. IEEE International Conference on Robotics and Automation*, pages 429–434, 1993.

[109] Robert D Howe and Mark R Cutkosky. Sensing skin acceleration for slip and texture perception. In *Proc. IEEE International Conference on Robotics and Automation*, pages 145–150, 1989.

[110] Joseph M Romano, Steven R Gray, Nathan T Jacobs, and Katherine J Kuchenbecker. Toward tactilely transparent gloves: Collocated slip sensing and vibrotactile actuation. In *Proc. Third Joint EuroHaptics conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 279–284, 2009.

[111] Dana D Damian, Harold Martinez, Konstantinos Dermitzakis, Alejandro Hernandez-Arieta, and Rolf Pfeifer. Artificial ridged skin for slippage speed detection in prosthetic hand applications. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 904–909, 2010.

[112] Minoru Ueda, Kazuhide Iwata, and Hiroyashu Shingu. Tactile sensors for an industrial robot to detect a slip. In *Proc. of the 2nd International Symposium on Industrial Robots*, pages 63–70, 1972.

[113] David Dornfeld and Christopher Handy. Slip detection using acoustic emission signal analysis. In *Proc. IEEE International Conference on Robotics and Automation*, volume 4, pages 1868–1875, 1987.

[114] Gaetano Canepa, Matteo Campanella, and Danilo De Rossi. Slip detection by a tactile neural network. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 224–231, 1994.

[115] EGM Holweg, H Hoeve, W Jongkind, Lorenzo Marconi, Claudio Melchiorri, and Claudio Bonivento. Slip detection by tactile sensors: Algorithms and experimental results. In *Proc. IEEE International Conference on Robotics and Automation*, volume 4, pages 3234–3239, 1996.

[116] Morteza Vatani, Erik D Engeberg, and Jae-Won Choi. Force and slip detection with direct-write compliant tactile sensors using multi-walled carbon nanotube/polymer composites. *Sensors and Actuators A: physical*, 195:90–97, 2013.

[117] Claudio Melchiorri. Slip detection and control using tactile and force sensors. *IEEE/ASME Transactions on Mechatronics*, 5(3):235–243, 2000.

[118] Dirk Goeger, Nico Ecker, and Heinz Woern. Tactile sensor and algorithm to detect slip in robot grasping processes. In *Proc. IEEE International Conference on Robotics and Biomimetics*, pages 1480–1485, 2009.

[119] Karsten Weiß and Heinz Worn. The working principle of resistive tactile sensor cells. In *Proc. IEEE International Conference Mechatronics and Automation*, volume 1, pages 471–476, 2005.