Efficient Generation of Motion Plans from Attribute-Based Natural Language Instructions Using Dynamic Constraint Mapping

Jae Sung Park, Biao Jia, Mohit Bansal, and Dinesh Manocha
http://gamma.cs.unc.edu/SafeMP/NLP/ (full video)

***Abstract*— We present an algorithm for combining natural language processing (NLP) and fast robot motion planning to automatically generate robot movements. Our formulation uses a novel concept called Dynamic Constraint Mapping to transform complex, attribute-based natural language instructions into appropriate cost functions and parametric constraints for optimization-based motion planning. We generate a factor graph from natural language instructions called the Dynamic Grounding Graph (DGG), which takes latent parameters into account. The coefficients of this factor graph are learned based on conditional random fields (CRFs) and are used to dynamically generate the constraints for motion planning. We map the cost function directly to the motion parameters of the planner and compute smooth trajectories in dynamic scenes. We highlight the performance of our approach in a simulated environment and via a human interacting with a 7-DOF Fetch robot using intricate language commands including negation, orientation specification, and distance constraints.**

I. Introduction

In the field of human-robot interaction (HRI), natural language has been used as an interface to communicate a human's intent to a robot [1], [2], [3], [4]. Much of the work in this area is related to specifying simple tasks or commands for robot manipulation, such as picking up and placing objects. As robots are increasingly used in complex scenarios and applications, it is important to develop a new generation of motion planning and robot movement techniques that can respond appropriately to diverse, attribute-based NLP instructions for HRI, e.g., instructions containing negation based phrases or references to position, velocity, and distance constraints. Furthermore, we need efficient techniques to automatically map the NLP instructions to such motion planners.

Humans frequently issue commands that include sentences with orientation-based or negation constraints such as "put a bottle on the table and keep it upright" or "move the knife but don't point it towards people," or sentences with velocity-based constraints such as "move slowly when you get close to a human." To generate robot actions and movements in response to such complex natural language instructions, we need to address two kinds of challenges:

1. The accurate interpretation of attribute-based natural language instructions and their grounded linguistic semantics, especially considering the environment and the context.

2. The motion planner needs to generate appropriate trajectories based on these complex natural language instructions. This includes appropriately setting up the motion planning problem based on different motion constraints (e.g., orientation, velocity, smoothness, and avoidance) and computing smooth and collision-free paths.

(a) (b) (c)

Fig. 1. The Fetch robot is moving a soda can on a table based on NLP instructions. Initially the user gives the "pick and place" command. However, when the robot gets closer to the book, the person says *"don't put it there"* (i.e. negation) and the robot uses our dynamic constraint mapping functions and optimization-based planning to avoid the book. Our approach can generate appropriate motion plans for such attributes.

At a high level, natural language instructions can be decomposed into task description and attributes. Task descriptions are usually verb or noun phrases that describe the underlying task performed by a robot. The attributes include various adjectives, adverbs, or prepositional phrases that are used to specify additional conditions the robot must (or must not) satisfy. For example, these conditions may specify some information related to the movement speed, the orientation, the physical space characteristics, or the distances. Therefore, it is important to design motion planners that consider these robotic task descriptions and robot motion constraints.

Main Results: We present an algorithm for generating parameterized constraints for optimization-based motion planning from complex, attribute-based natural language instructions. We use *Dynamic Grounding Graphs* (DGG) to parse and interpret the commands and to generate the constraints. Our formulation includes the latent parameters in the grounding process, allowing us to model many continuous variables in our grounding graph. Furthermore, we present a new dynamic constraint mapping that takes DGG as the input and computes different constraints and parameters for the motion planner. The appropriate motion parameters are speed, orientation, position, smoothness, repulsion, and avoidance. The final trajectory of the robot is computed using a constraint optimization solver. Compared to prior techniques, our overall approach offers the following benefits:

- The inclusion of latent parameters in the grounding graph allows us to model continuous variables that are used by our mapping algorithm. Our formulation computes the dynamic grounding graph based on conditional random fields.
- We present a novel dynamic constraint mapping used to compute different parametric constraints for

optimization-based motion planning.

- Our grounding graphs can handle more complex, attribute-based natural language instructions and our mapping algorithm uses appropriate cost functions as parameters over the continuous space. Compared to prior methods, our approach is much faster and better able to handle more complex and attribute-based natural language instructions.

We highlight the performance of our algorithms in a simulated environment and on a 7-DOF Fetch robot operating next to a human. Our approach can handle a rich set of natural language commands and can generate appropriate paths. These include complex commands such as picking (e.g., *"pick up a red object near you"*), correcting the motion (e.g., *"don't pick up that one"*), and negation (e.g., *"don't put it on the book"*).

II. Related Work

Most algorithms used to map natural language instruction to robot actions tend to separate the problem into two parts: parsing and motion planning computation. In this section, we give a brief overview of prior work in these areas.

A. Natural Language Processing

Nyga et al. [6], [7], [8], [9] have developed Probabilistic Robot Action Cores through which robots understand natural language instructions from a knowledge base. The natural language understanding system creates a probabilistic model for converting taxonomies to semantics and find the combination of semantics that gives the highest probability. Duvallet et al. [10] use a probabilistic graphical learning model called Generalized Grounding Graphs (G^3) on a ground vehicle for a navigation problem given natural language commands. Branavan et al. [3], [11] use reinforcement learning to learn a mapping from natural language instructions and then apply it to sequences of executable actions. Matuszek et al. [4] use a statistical machine translation model to map natural language instructions to path description language, which allows a robot to navigate while following directions. Duvallet et al. [12] use imitation learning to train the model through demonstrations of humans following directions. Paul et al. [13] propose the Adaptive Distributed Correspondence Graph (ADCG). Arkin et al. [14] further extend DCG, proposing the Hierarchical Distributed Correspondence Graph (HDCG), which defines constraints as discrete inequalities and grounds word phrases to corresponding inequalities. Chung et al. [15] use HDCG on ground vehicles to implement navigation commands and demonstrate performance improvements over G^3 in terms of running time, factor evaluations, and correctness. We chose the DCG model as our base framework, because it has a flexibility for many applications and extensions based on the parse tree from the input natural language instructions.

B. Robot Motion Planning in Dynamic Environments

Many replanning algorithms have been suggested to generate collision-free motion plans in dynamic environments. Fox et al. [16] propose the dynamic window approach to compute optimal velocity in a short time window. Optimization-based motion planners [17], [18], [19] solve a constrained optimization problem to generate smooth and collision-free robot paths. We present an automatic scheme that generates the motion planning problem from NLP instructions.

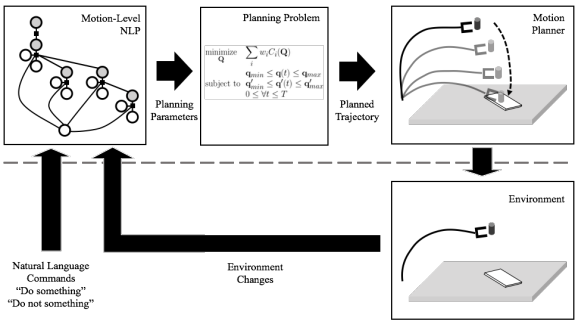


Fig. 2. Overall pipeline of our approach highlighting the NLP parsing module and the motion planner. Above the dashed line (from left to right): Dynamic Grounding Graphs (DGG) with latent parameters that are used to parse and interpret the natural language commands, generation of optimization-based planning formulation with appropriate constraints and parameters using our mapping algorithm. We highlight the high-level interface below the dashed line.

There is some work on integrating optimization-based motion planning with NLP in 2D workspaces. Silver et al. [20] develop an algorithm for learning navigation cost functions from demonstrations. Howard et al. [2] use a probabilistic graphical model to generate motion planning constraints for a 2D navigation problem. Compared to these methods, our approach can handle 3D workspaces and high-dimensional configuration spaces to generate robot motions corresponding to complex NLP instructions. Other techniques focus on efficiency in human-robot collaborative tasks. Markov Decision Processes (MDP) are widely used to compute the best robot action policies [21], [22]. These techniques are complementary to our approach.

III. Dynamic Grounding Graphs

Fig. 2 shows the basic pipeline of our approach. When natural language commands are given as input, the NLP module (upper left) creates an optimization problem for a motion planning module (upper middle). The robot motion trajectory is then computed from the motion planning module (upper right). As the planned trajectory is executed (bottom right), the result is fed back to the NLP module. In this section, we present the algorithms used in the NLP module.

We extend the ideas of the Generalized Grounding Graphs (G^3) model and the Distributed Correspondence Graph (DCG) model [2] by including the latent variables in the grounding graph and using them to compute the constraints for motion planning. The input to our algorithm is the natural language instruction. We do not account for any errors due to voice recognition. From a natural language command input, we construct a factor graph, as shown in Fig. 3(a), which is based on the parsing of the command. For each node of the parse tree, we generate three types of nodes: word phrase node λ, grounding node γ, and correspondence node ϕ.

The input sentence Λ is parsed using the NLTK library [23]. The word phrase of each node in the parse tree is denoted as λ_i for $i = 1, 2, \cdots$. Children of λ_i are λ_{i1}, $\cdots$, λ_{im}. The root node of the parse tree is λ_1. For example, in Fig. 3(a), the input sentence is *"Put the cup on the table."* The parse tree has the root word phrase λ_1 ="*Put*". Its noun λ_2 ="*the cup*" and the preposition λ_3 ="on," which are the children nodes of the root node. The noun phrase λ_4 ="*the table*" is the child node of λ_3.

Our goal is to compute a mapping from a natural language sentence Λ to the cost function parameters H given the robotic environment E where the robot is operating. E is a representation of the environment, which is composed of obstacle positions, orientations, and the robot's configuration. Feature vectors are constructed in the factor graph from the description of the environment. H is a real-valued vector that contains all cost function parameters and weights used in the optimization-based motion planner. It also includes the weights of different types of cost functions used in the optimization formulation.

We first compute the groundings γ_i of each word phrase λ_i. The grounding of each word phrase is the mapping from the word phrase to its meaning in the real world. Groundings can be objects, locations, motions, tasks, or constraints. In our model, the grounding γ_i depends on its word phrase λ_i and its children grounding nodes γ_{i1}, $\cdots$, γ_{im}, where the tree structure of the grounding nodes follows the parse tree. Correspondence node ϕ_i indicates the correct matching between the word phrase λ_i and the grounding γ_i. It is a binary variable; ϕ_i is $true$ if the word phrase and the grounding match and $false$ if they do not.

A. Latent Parameters

A key novel component of our approach is the inclusion of latent variables in the grounding graph. Our primary goal is to compute the best cost function parameters H to be used directly for optimization-based motion planning. We denote $H \in \mathbb{R}^h$, a real vector of size h, as a collection of cost function parameters. In this case, the size h and the number of cost function parameters depend on the types of cost functions that are used. [1] From the predicted groundings γ_i, the cost function parameters in the motion planning formulation (Fig. 3(b)) are inferred through the latent variable H. H contains all the cost function parameters (e.g., weights of cost functions, locations, and orientations). Details about the cost functions are described in Section IV.

In Fig. 3(b), the resulting constraint-based motion planning problems are shown. We use the collision avoidance cost function as the default smoothness cost function and the target location cost function, though weights can vary. The target location, whose 3D coordinates are the cost function parameters, is set on the surface of the table. The cost function parameter node H contains the weights of the parameters and the 3D coordinates of the target location. In the bottom of Fig. 3(b), where a new *"Don't"* command is given, a repulsion cost function is added. Thus, the cost function weight and the location of the repulsion source (below the robots end-effector position) are added to H.

[1] In this paper, we set maximum $h = 22$ to fully specify the smoothness, the end-effector position, the end-effector orientation, the end-effector speed, and the repulsion cost functions. It is a sum of three terms: 5 for weights, 16 for positions and orientations, and 1 for an exponential constant.

B. Probabilistic Model

We present a new probabilistic model to compute H, Λ and E. We assume the conditional independence of the probabilities so that we can construct a factor graph (see Fig. 3(a)). With the independence assumptions, a single factor is connected to a word phrase node and its children grounding nodes, which contain information about the subcomponents. This graphical representation corresponds to the following equation:

$$p(H|\lambda_1, \cdots, \lambda_n, E) = p(H|\gamma_1, E) \prod_i p(\gamma_i|\lambda_i, \phi_i, \gamma_{i1}, \cdots, \gamma_{im}, E). \tag{1}$$

For the root factor connecting H, γ_1 and E, we formulate the continuous domain of H. We compute the Gaussian Mixture Model (GMM) on the probability distribution $p(H|\gamma_1, E)$ and model our probability with non-root factors as follows:

$$p(\gamma_i|\lambda_i, \phi_i, \gamma_{i1}, \cdots, \gamma_{im}, E) \tag{2}$$

$$= \frac{1}{Z} \exp(-\theta_i^T f(\gamma_i, \lambda_i, \phi_i, \gamma_{i1}, \cdots, \gamma_{im}, E)), \tag{3}$$

where Z is the normalization factor, and θ_i and f are the log-linearizations of the feature function. The function f generates a feature vector given a grounding γ_i, a word phrase λ_i, a correspondence ϕ_i, children groundings γ_{ij}, and the environment E. The information from the robotic environment is used in the feature function f and in the log-linearized feature function f. The attributes of objects in the robotic world such as shapes and colors are encoded as multidimensional binary vectors, which indicate whether the object has a given attribute.

Word phrases. The feature vector includes binary-valued vectors for the word and phrase occurrences, and Part of Speech (PoS) tags. There is a list of words that could be encountered in the training dataset such as {*put, pick, cup, up, there,* $\cdots$ }. The word list is constructed from H2SL dataset [13] that contains 1672 word phrases. If the word phrase contains the word *"put,"* then the occurrence vector at the first index is set to 1 and the others are set to 0. This list also includes real-valued word similarities between the word and the pre-defined seed words. The seed words are the pre-defined words that the users expect to encounter in the natural language instructions. We used Glove word2vec [24] to measure cosine-similarity (i.e. the inner product of two vectors divided by the lengths of the vectors) between the words.

Robot states. From the robot state, we collect the robot joint angles, the velocities, the end-effector position, the end-effector velocity, etc. This information can affect the cost function parameters even while processing the same natural language commands. We also store information about the objects that are close to the robot.

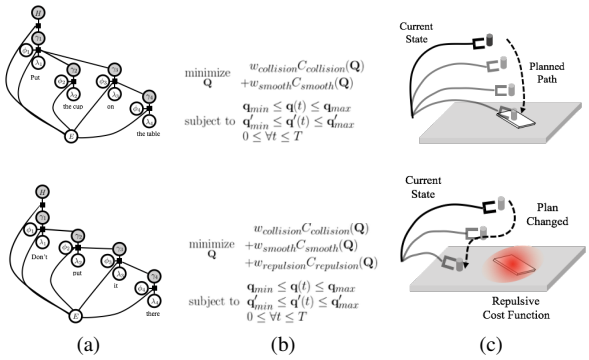


Fig. 3. **Factor graphs for different commands:** In the environment in the right-hand column, there is a table with a thin rectangular object on it. A robot arm is moving a cup onto the table, but we want it to avoid moving over the book when given NLP instructions. (a) The command *"Put the cup on the table"* is given and the factor graph is constructed (left). Appropriate cost functions for the task are assigned to the motion planning algorithm (middle) and used to compute the robot motion (right). (b) As the robot gets close to the book, another command *"Don't put it there"* is given with a new factor graph and cost functions.

C. Factor Graph using Conditional Random Fields

We represent our dynamic grounding graph as a factor graph. We build a factor graph based on the probabilistic model described in Section III-B and use it to train and to infer the meaning of given commands. In particular, we use Conditional Random Fields (CRF) [25] as a learning model for factor graphs because CRFs are a good fit for applying machine learning to our probabilistic graph model with conditional probabilities.

At the inference step, we used the trained CRF factor graph models to find the best groundings Γ and the cost function parameters H by solving the CRF maximization problem

$$\underset{H,\gamma_1,\cdots,\gamma_n}{\text{maximize}} p(H|\gamma_1,E)\prod_i \frac{1}{Z}\exp(\theta_i^T f(\gamma_i,\lambda_i,\phi_i,\gamma_{i1},\cdots,\gamma_{im},E)). \tag{4}$$

When the nodes H, $\gamma_1,\cdots,\gamma_n$ are optimized, they create a tree structure in the factor graph, meaning that we can solve the optimization problem efficiently using dynamic programming. Each factor depends on its parent and children varying variables and other fixed variables connected to it. This implies that we can solve the sub-problems in a bottom-up manner and combine the results to solve the bigger problem corresponding to the root node.

During the training step of CRF, we solve the optimization problem of maximizing the probability of the samples in the training dataset over the feature coefficients θ_i for every parse tree structure. The optimization problem is the same as Eq. (4), except that the problem is solved over θ instead of H and Γ. More details are given in [26].

IV. Dynamic Constraint Mapping With NLP Input

We use an optimization-based algorithm [27] to solve the cost minimization problem. The function and constraints of this cost minimization problem come from DGG, as explained in Sec. III. In this section, we present our mapping algorithm, Dynamic Constraint Mapping, which maps the word phrase groundings to proper cost function parameters corresponding to natural language instructions [26].

A. Robot Configurations and Motion Plans

We denote a single configuration of the robot as a vector $\mathbf{q}$, which consists of joint-angles or other degrees-of-freedom. A configuration at time t, where $t \in \mathbb{R}$, is denoted as $\mathbf{q}(t)$. We assume $\mathbf{q}(t)$ is twice differentiable, and its derivatives are denoted as $\mathbf{q}'(t)$ and $\mathbf{q}''(t)$. The n-dimensional space of configuration $\mathbf{q}$ is the configuration space $\mathscr{C}$. We represent bounding boxes of each link of the robot as B_i. The bounding boxes at a configuration $\mathbf{q}$ are denoted as $B_i(\mathbf{q})$.

For a planning task with a given start configuration $\mathbf{q}_0$ and derivative $\mathbf{q}'_0$, the robot's trajectory is represented by a matrix $\mathbf{Q}$, whose elements correspond to the waypoints [17], [18], [27]:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{q}_0 & \mathbf{q}_1 & \cdots & \mathbf{q}_{n-1} & \mathbf{q}_n \\ \mathbf{q}'_0 & \mathbf{q}'_1 & \cdots & \mathbf{q}'_{n-1} & \mathbf{q}'_n \\ t_0 = 0 & t_1 & & t_{n-1} & t_n = T \end{bmatrix}. \tag{5}$$

The robot trajectory passes through $n+1$ waypoints $q_0,\cdots,q_n$, which will be optimized by an objective function under constraints in the motion planning formulation. Robot configuration at time t is interpolated from two waypoints. Formally, for j such that $t_j \le t \le t_{j+1}$, the configuration $\mathbf{q}(t)$ and derivative $\mathbf{q}'(t)$ are cubically interpolated using $\mathbf{q}_j$, $\mathbf{q}'_j$, $\mathbf{q}_{j+1}$, and $\mathbf{q}'_{j+1}$.

The i-th cost functions of the motion planner are $C_i(\mathbf{Q})$. Our motion planner solves an optimization problem with non-linear cost functions and linear joint limit constraints to generate robot trajectories for time interval $[0,T]$,

$$\begin{aligned} &\underset{\mathbf{Q}}{\text{minimize}} && \sum_i w_i C_i(\mathbf{Q}) \\ &\text{subject to} && \mathbf{q}_{min} \le \mathbf{q}(t) \le \mathbf{q}_{max}, \quad 0 \le \forall t \le T. \\ & && \mathbf{q}'_{min} \le \mathbf{q}'(t) \le \mathbf{q}'_{max} \end{aligned} \tag{6}$$

In the optimization formulation, C_i is the i-th cost function and w_i is the weight of the cost function.

B. Parameterized Constraints

The overall optimization formulation is given in Eqn. 6. To formulate the constraints, we use the following cost functions, which are designed to account for various attributes in the NLP instructions, including collision avoidance. In our formulation, we use many types of cost functions such as collision avoidance, robot smoothness, robot end-effector speed, target positions, and target orientations. These cost functions are used to handle many attributes of natural language instructions. Each cost function has its weight and may also have other cost function parameters, if necessary.

To handle various attributes, we use the following parameterized constraints in our optimization formulation.

Collision avoidance:

$$C_{collision}(\mathbf{Q}) = \int_0^T \sum_i \sum_j dist(B_i(t), O_j)^2 dt, \tag{7}$$

where $dist(B_i(t), O_j)$ is the penetration depth between a robot bounding box $B_i(t)$ and an obstacle O_j.

Smoothness: This cost function corresponds to the integral of the first derivative of joint angles over the trajectory duration.

End-effector position:

$$C_{position}(\mathbf{Q}) = \int_0^T ||\mathbf{p}_{ee}(t) - \mathbf{p}_{target}||^2 dt,, \tag{8}$$

where $\mathbf{p}_{ee}(t)$ is the robot end-effector position at time t and $\mathbf{p}_{target}$ is the target position. The target position $\mathbf{p}_{target}$ is considered as a cost function parameter.

End-effector orientation:

$$C_{orientation}(\mathbf{Q}) = \int_0^T angledist(\mathbf{q}_{ee}(t), \mathbf{q}_{target})^2 dt, \tag{9}$$

$$C_{upvector}(\mathbf{Q}) = \int_0^T angledist(\mathbf{n}_{up}(t), \mathbf{n}_{target})^2 dt, \tag{10}$$

where $\mathbf{q}_{ee}(t)$ is the quaternion representation of the robot end-effector's orientation at time t, $\mathbf{q}_{target}$ is the end-effector orientation that we want the robot to maintain, $\mathbf{n}_{up}$ is the normal up-vector of the robot's end-effector, and $\mathbf{n}_{target}$ is the target up-vector. As with the *end-effector position* cost, the target orientation $\mathbf{q}_{target}$ is the cost function parameter.

End-effector speed:

$$C_{speed}(\mathbf{Q}) = \int_0^T ||\mathbf{v}_{ee}(t) - \mathbf{v}_{target}||^2 dt, \tag{11}$$

where $\mathbf{v}_{ee}(t)$ is the robot's end-effector speed at time t and $\mathbf{v}_{target}$ is the target speed. The parameters of this cost function correspond to $\mathbf{v}_{target}$.

Repulsion:

$$C_{repulsion}(\mathbf{Q}) = \int_0^T \exp\left(-c||\mathbf{p}_{ee}(t) - \mathbf{p}_r||\right) dt, \tag{12}$$

where $\mathbf{p}_r$ is the position to which we don't want the robot to move. The coefficient $c > 0$ suggests how much the cost is affected by $||\mathbf{p}_{ee}(t) - \mathbf{p}_{repulsive}||$, the distance between the end-effector position and the repulsion source.

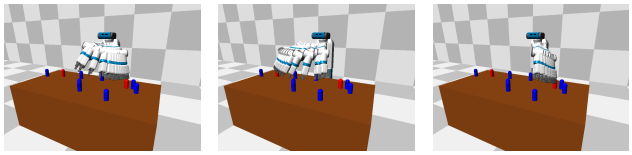
V. Implementation and Results

We have implemented our algorithm and evaluated its performance in a simulated environment and on a 7-DOF Fetch robot. All the timings are generated on a multi-core PC with Intel i7-4790 8-core 3.60GHz CPU and a 16GB RAM. We use multiple cores for fast evaluation and parallel trajectory search to compute a good solution to the constrained optimization problem [27].

A. Training DGGs for Demonstrations

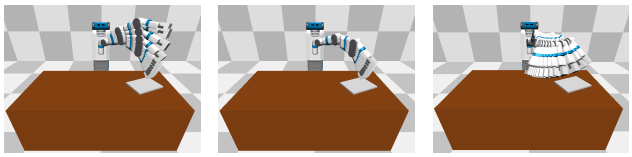
We describe how the training dataset for our DGG model were generated. The training dataset for DGGs requires three components: a natural language sentence, a robotic environment, and the cost function parameters for optimization-based motion planners.

For each demonstration, we write tens of different sentences that specify the take goals the constraints for the motion plans with different nouns, pronouns, adjectives, verbs, adverbs, preposition, etc. For each sentence, we generate a random robotic environment and an initial state for the robot.



(a) (b) (c)

Fig. 4. The simulated Fetch robot arm reaches towards one of the two red objects. (a) When a command *"pick up one of the red objects"* is issued, the robot moves to the red object on the right because of the DGG algorithm. (b) If the user doesn't want the robot to pick up the object on the right, he/she uses a command *"don't pick up that one."* Our DGG algorithm dynamically changes the cost function parameters. (c) The robot approaches the object on the right and stops.



(a) (b) (c)

Fig. 5. In this simulated environment, the human instructs the robot to *"put the cube on table"* (a). As it approaches the laptop (b), the human uses a negation NLP command *"don't put it there,"* so the robot places it at a different location (c).

In addition, the robot joint values and joint velocities are randomly set as initial states. We collect tens or hundreds of random robotic environments. For a natural language sentence, a random robotic environment, and a random initial state for the robot, the cost function parameters are assigned manually or synthesized from other examples. Hundreds of multiple data samples are generated from generated data samples by switching the correspondence variable in the DGG model from 1 (true) to 0 (false) and changing the grounding variables to the wrong ones to match the false correspondence variable. The training dataset is created with up to $100,000$ samples in our experiments. When the cost function parameters are determined, the optimization-based motion planner is used to compute a feasible robot trajectory. In the optimization-based motion planning algorithm, there are some waypoints through which the robot trajectory should pass. For the robot's safety, we check if the robot trajectory with given cost function parameters is in- collision and appropriately set a higher value of the coefficient of the collision cost and compute a new trajectory. This process is repeated until the trajectory is collision-free.

We use different training data for each scenario. For the scenarios shown in Fig. 4, the initial pose of the robot in front of the table and the positions of the blue and red objects on the table are randomly set. For *"Pick up"* commands, appropriate cost function parameters are computed so that the robot picks up a blue or red object depending on the given command. Similarly, in Fig. 5, the position and orientation of the laptop is initialized randomly. Given the *"Put"* command, we create an end-effector position cost function so that the robot places the object on the table; and a repulsive cost function to avoid the laptop position.

TABLE I

PLANNING PERFORMANCES WITH VARYING SIZES OF TRAINING DATA FOR THE SCENARIO IN FIG. 4 WITH 21 DIFFERENT NLP INSTRUCTIONS.

# Training Data	Success Rate	Duration	Smoothness Cost
1,000	5/10	23.46s (5.86s)	8.72 (5.56)
3,000	9/10	16.02s (3.28s)	2.56 (0.64)
10,000	10/10	13.16s (1.24s)	1.21 (0.32)
30,000	10/10	12.81s (0.99s)	0.78 (0.12)
100,000	10/10	12.57s (0.97s)	0.72 (0.10)

TABLE II

RUNNING TIME OF OUR DGG AND MOTION PLANNING MODULES.

Scenarios	Instructions	$\|H\|$	DGG Time	Planning Time
Pick up an object (Fig. 4)	10	12	32ms	93ms
Don't put on laptop (Fig. 5)	20	13	16ms	98ms
Move around obstacle	45	9	16ms	95ms
Static Instructions (video)	20	18	73ms	482ms
Dynamic Instructions (Fig. 1)	21	18	58ms	427ms

B. Simulations and Real Robot Demonstrations

We evaluate the performance on optimization problems that occur in complex environments composed of multiple objects. Based on the NLP commands, the robot decides to pick an appropriate object or is steered towards the goal position in a complex scene. In particular, the user gives NLP commands such as *"move right," "move up," "move left,"* or *"move down"* to guide the robot. For each such command, we compute the appropriate cost functions.

In terms of the 7-DOF Fetch robot, we test the performance of our approach on different tasks corresponding to: (1) moving a soda can on the table from one position to an other; (2) not moving the soda can over the book [26].

C. Analysis

We evaluate the performance based on the following:

Success Rate: The ratio of successful task completion among all trials. Failure includes colliding with the obstacles due to an incorrect mapping of cost function parameters, violating constraints specified by natural language commands, and not completing the task due to some other reason.

Trajectory Duration: The time between the giving of the first NLP command and the robot's successful completion of the task after trajectory computation. A shorter duration implies a higher performance.

Trajectory Smoothness Cost: A cost based on evaluating the trajectory smoothness according to standard metrics and dividing it by the trajectory duration. A lower cost implies a smoother and more stable robot trajectory.

Table I shows the results on our benchmarks with varying numbers of training data samples in the simulation environment shown in Fig. 4. When the number of training data samples increases, the success rate also increases while the trajectory duration and the trajectory smoothness cost decrease. Table II shows the running time of our algorithm and the distances from the obstacle on the table in the real-world scenarios.

VI. BENEFITS AND COMPARISONS

Most prior methods that combine NLP and motion planning have focused on understanding natural language instructions to compute robot motion for simple environments and constraints. Most of these methods are limited to navigation applications [28], [15], [10] or simple settings [11], or they are not evaluated on real robots [14]. Nyga et al. [6], [7], [8], [9] use probabilistic relation models based on knowledge bases to understand natural language commands that describe visual attributes of objects. This is complimentary to our work. Broad et al. [29] extend DCG for a robot manipulator so that it will handle natural language correction for robot motion in realtime. In our approach, the goal is to generate appropriate high-DOF motion trajectories in response to attribute-based natural language instructions like negation, distance or orientation constraints, etc. Unlike prior methods, the output of our NLP parsing algorithm is directly coupled with the specification of the motion planning problem as a constrained optimization method.

It may be possible to extend prior methods [1], [2] to handle attribute-based NLP instructions. For example, distance attributes require a number of constraints in the motion planning formulation. In natural language instructions such as *"Pick up the blue block and put it 20cm to the left of the red block"* or *"Pick up one of the two blocks on the rightmost, and place it 10 inches away from the block on the leftmost,"* the exact distance specifications are the distance attributes. Prior methods that use G^3, DCG, or the Hybrid G^3-DCG models have only been evaluated with a small number of attributes (distance, orientation and contact) to solve constrained motion planning problems. These prior techniques use discretized constraints [2], each of which can be active (i.e. $f(x) > 0$)), inverted ($f(x) < 0$), or ignored (i.e. not included). Therefore, it is not possible to represent an explicit constraint corresponding to the value of the continuous variable *distance* in their formulation.

VII. LIMITATIONS, CONCLUSIONS AND FUTURE WORK

We present an motion planning algorithm that computes appropriate motion trajectories for a robot based on complex NLP instructions. Our formulation is based on two novel concepts: dynamic grounding graphs and dynamic constraint mapping. We highlight the performance in simulated and real-world scenes with a 7-DOF manipulator operating next to humans. We use a high dimensional optimization algorithm and the solver may get stuck in local minima, though we use multiple initializations to solve this problem.

As future work, we would like to overcome these limitations and evaluate the approach in challenging scenarios with moving obstacles while performing complex robot tasks. More work is needed to handle the full diversity of a natural language, especially for rare words, complicated grammar styles, and hidden intentions or emotions in human speech. We plan to incorporate stronger natural language processing and machine learning methods such as those based on semantic parsing, neural sequence-to-sequence models, etc.

ACKNOWLEDGMENT

This research is supported in part by ARO grant W911NF19-1-0069, ARO-YIP Award W911NF-18-1-0336, and Intel.

REFERENCES

[1] T. Kollar, S. Tellex, M. R. Walter, A. Huang, A. Bachrach, S. Hemachandra, E. Brunskill, A. Banerjee, D. Roy, S. Teller, *et al.*, "Generalized grounding graphs: A probabilistic framework for understanding grounded language," *JAIR*, 2013.

[2] T. M. Howard, S. Tellex, and N. Roy, "A natural language planner interface for mobile manipulators," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 6652–6659.

[3] S. R. Branavan, H. Chen, L. S. Zettlemoyer, and R. Barzilay, "Reinforcement learning for mapping instructions to actions," in *ACL-IJCNLP*. Association for Computational Linguistics, 2009.

[4] C. Matuszek, D. Fox, and K. Koscher, "Following directions using statistical machine translation," in *HRI*. IEEE, 2010.

[5] A. D. Dragan, K. C. Lee, and S. S. Srinivasa, "Legibility and predictability of robot motion," in *HRI*. IEEE, 2013.

[6] D. Nyga and M. Beetz, "Everything robots always wanted to know about housework (but were afraid to ask)," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 243–250.

[7] ——, "Cloud-based probabilistic knowledge services for instruction interpretation," in *Robotics Research*. Springer, 2018, pp. 649–664.

[8] D. Nyga, M. Picklum, S. Koralewski, and M. Beetz, "Instruction completion through instance-based learning and semantic analogical reasoning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4270–4277.

[9] D. Nyga, M. Picklum, and M. Beetz, "What no robot has seen beforeprobabilistic interpretation of natural-language object descriptions," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4278–4285.

[10] F. Duvallet, M. R. Walter, T. Howard, S. Hemachandra, J. Oh, S. Teller, N. Roy, and A. Stentz, "Inferring maps and behaviors from natural language instructions," in *Experimental Robotics*. Springer, 2016, pp. 373–388.

[11] S. Branavan, N. Kushman, T. Lei, and R. Barzilay, "Learning high-level planning from text," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*. Association for Computational Linguistics, 2012, pp. 126–135.

[12] F. Duvallet, T. Kollar, and A. Stentz, "Imitation learning for natural language direction following through unknown environments," in *ICRA*. IEEE, 2013, pp. 1047–1053.

[13] R. Paul, J. Arkin, N. Roy, and T. M. Howard, "Efficient grounding of abstract spatial concepts for natural language interaction with robot manipulators." in *Robotics: Science and Systems*, 2016.

[14] J. Arkin and T. M. Howard, "Towards learning efficient models for natural language understanding of quantifiable spatial relationships," in *RSS 2015 Workshop on Model Learning for Human-Robot Communication*, 2015.

[15] I. Chung, O. Propp, M. R. Walter, and T. M. Howard, "On the performance of hierarchical distributed correspondence graphs for efficient symbol grounding of robot instructions," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 5247–5252.

[16] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[17] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2011, pp. 4569–4574.

[18] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "Chomp: Covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[19] C. Park, J. Pan, and D. Manocha, "Real-time optimization-based planning in dynamic environments using GPUs," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2013.

[20] D. Silver, J. A. Bagnell, and A. Stentz, "Learning autonomous driving styles and maneuvers from expert demonstration," in *Experimental Robotics*. Springer, 2013, pp. 371–386.

[21] S. Nikolaidis, P. Lasota, G. Rossano, C. Martinez, T. Fuhlbrigge, and J. Shah, "Human-robot collaboration in manufacturing: Quantitative evaluation of predictable, convergent joint action," in *Robotics (ISR), 2013 44th International Symposium on*. IEEE, 2013, pp. 1–6.

[22] H. S. Koppula, A. Jain, and A. Saxena, "Anticipatory planning for

human-robot teams," in *Experimental Robotics*. Springer, 2016, pp. 453–470.

[23] S. Bird, "Nltk: the natural language toolkit," in *Proceedings of the COLING/ACL on Interactive presentation sessions*. Association for Computational Linguistics, 2006, pp. 69–72.

[24] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *EMNLP*, 2014, pp. 1532–1543.

[25] C. Sutton, A. McCallum, *et al.*, "An introduction to conditional random fields," *Foundations and Trends® in Machine Learning*, vol. 4, no. 4, pp. 267–373, 2012.

[26] J. S. Park, B. Jia, M. Bansal, and D. Manocha, "Generating realtime motion plans from complex natural language commands using dynamic grounding graphs," *CoRR*, vol. abs/1707.02387, 2017. [Online]. Available: http://arxiv.org/abs/1707.02387

[27] C. Park, J. Pan, and D. Manocha, "ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments," in *Proceedings of International Conference on Automated Planning and Scheduling*, 2012.

[28] J. Oh, T. M. Howard, M. R. Walter, D. Barber, M. Zhu, S. Park, A. Suppe, L. Navarro-Serment, F. Duvallet, A. Boularias, *et al.*, "Integrated intelligence for human-robot teams," in *International Symposium on Experimental Robotics*. Springer, 2016, pp. 309–322.

[29] A. Broad, J. Arkin, N. Ratliff, T. Howard, and B. Argall, "Real-time natural language corrections for assistive robotic manipulators," *The International Journal of Robotics Research*, vol. 36, no. 5-7, pp. 684–698, 2017.