Recovering from failure by asking for help

Citation: Knepper, Ross A., Stefanie Tellex, Adrian Li, Nicholas Roy, and Daniela Rus. "Recovering from Failure by Asking for Help." Auton Robot 39, no. 3 (August 6, 2015): 347–362.

As Published: http://dx.doi.org/10.1007/s10514-015-9460-1

Publisher: Springer US

Persistent URL: http://hdl.handle.net/1721.1/105525

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Auton Robot (2015) 39:347–362
DOI 10.1007/s10514-015-9460-1

Recovering from failure by asking for help

Ross A. Knepper[1] · **Stefanie Tellex**[2] · **Adrian Li**[3] · **Nicholas Roy**[4] · **Daniela Rus**[4]

Received: 6 December 2014 / Accepted: 2 July 2015 / Published online: 6 August 2015

Abstract Robots inevitably fail, often without the ability to recover autonomously. We demonstrate an approach for enabling a robot to recover from failures by communicating its need for specific help to a human partner using natural language. Our approach automatically detects failures, then generates targeted spoken-language requests for help such as "Please give me the white table leg that is on the black table." Once the human partner has repaired the failure condition, the system resumes full autonomy. We present a novel *inverse semantics* algorithm for generating effective help requests. In contrast to forward semantic models that interpret natural language in terms of robot actions and perception, our inverse semantics algorithm generates requests by emulating the human's ability to interpret a request using the Generalized Grounding Graph (G^3) framework. To assess the effectiveness of our approach, we present a corpus-based online evaluation, as well as an end-to-end user study, demonstrating that our approach increases the effectiveness of human interventions compared to static requests for help.

Ross A. Knepper and Stefanie Tellex have contributed equally to this paper.

This is one of several papers published in *Autonomous Robots* comprising the "Special Issue on Robotics Science and Systems".

✉ Ross A. Knepper
rak@cs.cornell.edu

[1] Department of Computer Science, Cornell University, Ithaca, USA

[2] Computer Science Department, Brown University, Providence, USA

[3] Department of Engineering, University of Cambridge, Cambridge, UK

[4] Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, USA

Keywords Natural language generation · Failure detection · Failure handling · Assembly · Human–robot interaction

List of Symbols

$\lambda \in \Lambda$	Set of language variables (words or short phrases)
$\gamma \in \Gamma$	Set of grounding variables (concepts in the real world)
$\phi \in \Phi$	Set of correspondence variables
M	Environmental context model
a	Target symbolic action
γ_a^*	Target action grounding variable

1 Introduction

Robotic capabilities such as robust manipulation, accurate perception, and fast planning algorithms have led to recent successes such as robots that can fold laundry (Maitin-Shepard et al. 2010), cook dinner (Bollini et al. 2012), and assemble furniture (Knepper et al. 2013). However, when robots execute these tasks autonomously, failures often occur, for example failing to pick up an object due to perceptual ambiguity or an inaccurate grasp. A key aim of current research is reducing the incidence of these types of failures, but eliminating them completely remains an elusive goal.

When failures occur, a human can often intervene to help a robot recover. If the human is familiar with the robot, its task, and its common failure modes, then they can provide this help without an explicit request from the robot. However, if a person is unfamiliar with the robotic system, they might not know how to help the robot recover from a failure. This situation will occur frequently when robots interact with untrained users in the home. Moreover, even trained users

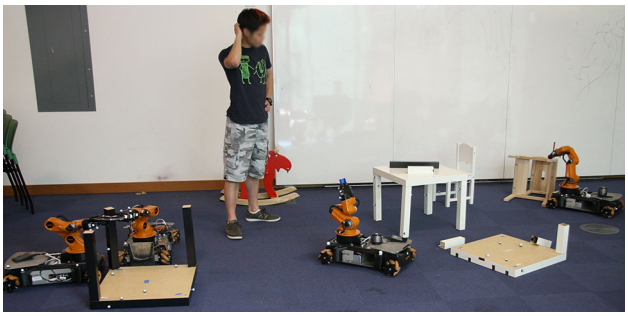


Fig. 1 A robot engaged in assembling an IKEA Lack table requests help using natural language. A vague request such as "Help me" is challenging for a person to understand. Instead, this paper presents an approach for generating targeted requests such as "Please hand me the black table leg that is on the white table"

who are deeply familiar with the robot's capabilities may experience problems during times of high cognitive load, such as a human supervising a large team of robots on a factory floor.

We propose an alternative approach to recovering from the inevitable failures which occur when robots execute complex tasks in real-world environments: when the robot encounters failure, it verbally requests help from a human partner. After receiving help, it resumes autonomous task execution. The contribution of our paper is a family of algorithms for formulating a pithy natural language request so that a human without situational awareness can render appropriate aid.

As a test domain, we focus on a human–robot team assembling IKEA furniture. Consider the example shown in Fig. 1. The robot is unable to reach the final leg it needs to assemble the black table, and it turns to a human for assistance. A carefully-worded help request by the robot can make it easy for the human to give the appropriate aid. The robot chooses not to say "help me," even though it is pithy and correct, because that request has many possible interpretations. Similarly, "hand me the black table leg" permits four possible interpretations corresponding to the four black legs. The robot issues the shortest unambiguous request, "hand me the black table leg that is on the white table." Interpreting this request places a minimal cognitive load on the person, who is able to quickly render the necessary aid.

Generating natural language requests requires a robot to map from perceptual aspects of the environment to words, such that a person will infer the same percepts. Human interpretation of such a request is highly contextual, and the robot must solve this problem in arbitrary, novel contexts that are not known at training time. Hard-coded or template-based methods for generating requests do not take into account the ability of a person to understand the request, while existing work in referring expression generation assumes access to a symbolic representation of the environment, including ambiguous spatial relations such as "near" or "under" which may not be directly computed from the robot's perceptual system (Krahmer and Deemter 2012).

We propose an algorithm that addresses this problem by searching for an utterance that maximizes the probability of a correspondence between the words in the language and the action the robot desires the human to perform. It takes as input a symbolic action that the robot would like to perform. The core of this algorithm is the Generalized Grounding Graph (G^3) (Tellex et al. 2011), a probabilistic model that the robot employs as a stand-in for a person's language understanding faculty. When understanding language, the robot maps from linguistic symbols to low-level motor actions and perceptual features that the robot encounters in the environment.

In this paper, we invert that model, mapping from a desired low-level motor action that the robot would like the human to execute to a linguistic description. Inverting the model requires developing a novel algorithm for generating help requests and adapting the request to the specific environment. By modeling the probability of a human misinterpreting the request, the robot is able to generate targeted requests that humans follow more quickly and accurately compared to baselines involving either generic requests (e.g. "Help me") or template-based non-context-specific requests (e.g. "Hand me the $\underline{< \text{part} >}$ ").

We evaluate our approach using a corpus-based experiment with Amazon Mechanical Turk as well as a real-world user study. The corpus-based approach allows us to efficiently test the performance of different algorithms. The user study assesses whether we have met our engineering goals in the context of an end-to-end system. Our evaluation demonstrates that the inverse semantics language generation system improves the speed and accuracy of a human's intervention when a human–robot team is engaged in a furniture assembly task and also improves the human's subjective perception of their robotic teammates.

This paper contributes the following:

- a framework for language generation by inverse semantics,
- an algorithm implementing inverse semantics that minimizes ambiguity by inverting a black-box model of language understanding, and
- an autonomous assembly system in which robots automatically detect failures and trigger requests for human help.

This paper builds on a previous version (Tellex et al. 2014) by adding a more detailed description of the framework for language inference along with a description of our planning framework for furniture assembly.

2 Related work

Traditional methods for generating language rely on a dedicated language-generation system that is not integrated with a language-understanding framework (Jurafsky and Martin 2008; Reiter and Dale 2000). These approaches typically consist of a sentence planner which computes a symbolic description of what to say combined with a surface realizer which converts the symbolic representation into words, but contain no principled model of how an instruction-follower would comprehend the instruction (Striegnitz et al. 2011; Garoufi and Koller 2011; Chen and Mooney 2011; Roy 2002). These models assume access to a formal symbolic representation of the object to be referenced and its properties and relations to other options (Krahmer and Deemter 2012). Our approach differs in that it generates language by inverting a module for language understanding. This inversion is required to generate physically grounded language about a particular environment that maps to the robot's perceptual data and communicates it to a human partner. Furthermore, our approach generates references to objects, as well as actions, unlike much previous work which focuses on objects and sets of objects.

Some previous work has approached the generation problem by inverting a semantics model. Golland et al. (2010) use a game-theoretic approach combined with a semantics model to generate referring expressions. Our approach, in contrast, uses probabilistic grounded semantics, biasing the algorithm toward shorter sentences unless a longer, more descriptive utterance is unambiguous. Goodman and Stuhlmüller (2013) describe a rational speech-act theory of language understanding, where the speaker chooses actions that maximize expected global utility. Similarly, recent work has used Dec-POMDPs to model implicatures and pragmatics in language-using agents (Vogel et al. 2013a, b) but without focusing on grounded, situated language as in this paper, and without implementing an end-to-end robotic system. Goeddel and Olson (2012) present a model for generating directions by inverting a model of human wayfinding and using a particle filter to efficiently evaluate trajectories; a similar approach could enable our direction inference to run faster, but they do not apply it to manipulating objects. There is a deep connection between our models and the notion of legibility and predictability for grasping and pointing, as defined by Dragan and Srinivasa (2013), pointing toward a unified framework for grounded communication using language and gesture.

Our approach views the language generation problem as inverse language understanding; a large body of work focuses on language understanding for robots (Fasola and Mataric 2013; MacMahon et al. 2006; Dzifcak et al. 2009; Kollar et al. 2010; Matuszek et al. 2012). Of these previous approaches, we chose to invert the G^3 framework because it is a probabilistic framework which explicitly models the mapping between words in language and sensed aspects of the external world, so metrics based on entropy may be used to assess the quality of generated utterances.

Cooperative human–robot activities, including assembly, have been broadly studied (Wilson 1995; Simmons et al. 2007; Dorais et al. 1998; Fong et al. 2003). These architectures permit various granularities of human intervention through a sliding autonomy framework. A failure triggers the replay of video of the events preceding failure, from which the human must obtain situational awareness. By contrast, in our approach the robot diagnoses the failure and leverages natural language to convey to the user exactly how the problem should be resolved.

3 Assembling furniture

Our assembly system, IkeaBot, comprises a team of KUKA youBots that collaborate to assemble IKEA furniture, originally described in Knepper et al. (2013). The robots receive assembly instructions encoded in a STRIPS-style planning language called ABPL. A centralized executive takes as input the symbolic plan and executes each plan step in sequence. Each symbolic action corresponds to a manipulation or perception action to be performed by one or two robots. Assembling an IKEA Lack table requires executing a 48-step plan. If the instructions are not provided, the IkeaBot can synthesize a plan derived from only the CAD files of the parts. All of the steps are autonomously executable under the right conditions, and the team can assemble the table in approximately 10 min when no failures occur. Since perception is not a focus of this paper, we employ a Vicon motion capture system to track the location of each participating robot, human and furniture part during the assembly process. In our experiments, failures occurred at a rate of roughly one every 2 min, mostly due to mechanical problems such as grasping failures, perceptual issues such as a part not being visible on Vicon, and planning failures such as a robot failing to find a path to reach its goal due to obstacles. When the robots detect a failure, one of the robots requests help using one of the approaches described in Sect. 5. Figure 2 shows the algorithm used to control the robots and request help.

3.1 Assembly planning

Symbolic planning problems are specified using a recently-designed planning language. ABPL (Knepper et al. 2013) is an object-oriented symbolic planning specification language, exemplified in Fig. 3. Conceptually similar to OPL (Hertle (2011)), ABPL describes planning problem data in a manner which respects the logical and intuitive features of the

```
function conditions_satisfied(ℓ – list of conditions)
 1: q ← World state
 2: for all c ∈ ℓ do
 3:     if c not satisfied in q then
 4:         a ← generate_remedy_action(c)        ▷ See Section 3.5
 5:         generate_help_request(a)             ▷ See Section 5
 6:         while c not satisfied do
 7:             if time ≥ 60 then                ▷ wait up to 60 seconds
 8:                 return false
 9: return true
```

```
function executive(g – goal state)
 1: repeat
 2:     p ← symbolic_plan(g)                     ▷ p – list of actions
 3:     f ← true                                 ▷ f – are we finished?
 4:     while p ≠ ∅ do
 5:         s ← p[0]                             ▷ first plan step
 6:         if conditions_satisfied(s.preconditions) then
 7:             s.execute()
 8:             if not conditions_satisfied(s.postconditions) then
 9:                 f ← false
10:         else
11:             f ← false
12:         p.retire(s)                          ▷ s succeeded; remove it from p
13: until f                                      ▷ no actions failed
```

Fig. 2 An executive algorithm generates robot actions and help requests

physical environment as a planning space. ABPL enables an intuitive statement of the problem by logically organizing concepts using various object-oriented programming (OOP) paradigms.

ABPL aims to overcome the necessary complexity of expressing object-oriented relations within first-order logical systems such as PDDL, the Planning Domain Definition Language (McDermott et al. 1998). PDDL relies entirely on a flat, unstructured representation of the objects in the environment and the relations between them. The burden of creating structure, such as regions of symbolic symmetry or commonalities between multiple objects, falls entirely on the user. While such systems are capable of describing OOP structures, requiring the user to express each element of those structures as a set of propositional statements would be time-consuming and burdensome to the user. ABPL, in contrast, allows the user to provide data in a more conventional object-oriented format. An ABPL problem specification can be about one-quarter the size of the equivalent PDDL specification. This simplicity improves readability and ease of ABPL problem creation, whether manual or automatic.

3.2 Specification language design and structure

ABPL is based on the object-oriented approach to data structure design, allowing the user to hierarchically organize objects within the environment. Objects can be assigned properties that evaluate to either Boolean values or references to other objects. These values can be changed by the effects

```
type Robot {
    object arm {
        property(Object) holding = None;
    }
}
type AttachmentSpot {
    property(Object) attached_to = None;
}
type TableTop {
    group hole(AttachmentSpot)[4]{}
    property(bool) upside_down = True;
}
type Leg {
    object hole(AttachmentSpot){}
}

object table_top(TableTop){}
group leg(Leg)[4]{}
group robot(Robot)[2]{}

action pick_up_leg(robot(Robot), leg(Leg)) {
    pre {
        robot.arm.holding == None;
        leg.hole.attached_to == None;
    }
    post {
        robot.arm.holding = leg;
    }
}
action attach_leg_to_top(robot(Robot), leg(Leg), table_top(TableTop)) {
    pre {
        robot.arm.holding == leg;
        table_top.hole[0].attached_to == None;
    }
    post {
        robot.arm.holding = None;
        table_top.hole[0].attached_to = leg.hole;
        leg.hole.attached_to = table_top.hole[0];
    }
}
action flip_table(robot(Robot)[2], leg(Leg)[4], table_top(TableTop)) {
    pre {
        leg[2].hole.attached_to == table_top.hole[1];
        leg[3].hole.attached_to == table_top.hole[0];
        leg[0].hole.attached_to == table_top.hole[2];
        leg[1].hole.attached_to == table_top.hole[3];
        table_top.upside_down == True;
    }
    post {
        table_top.upside_down = False;
    }
}

goal assembly(table_top(TableTop)) {
    table_top.upside_down == False;
}
```

Fig. 3 Excerpt of input file for the symbolic planner, which builds an upright IKEA Lack table. The full input is 220 lines

of symbolic actions to reflect those actions' consequent alterations of the environment. Objects themselves can be defined either at the top level in the global space, or as sub-elements of other objects. These sub-object relations can be syntactically referenced in the same way as object-reference properties, but are semantically different in that their values cannot be changed after declaration. This fact is meant to convey the distinction between an object's sub-objects, which represent fixed components of that object, and its object-typed properties, which represent mutable inter-object relations. ABPL also allows the user to define "types," which fulfill the role of object classes. A type can be assigned properties and sub-objects, which are then replicated in each object that is declared to be of that type. Types can also be used as action and goal predicates, for example to restrict a symbolic action variable to values of a specific type.

An important distinction from other object-oriented symbolic planners is the inclusion of groups. Groups represent the existence of multiple identical instances of a given object.

```
step locate_part(robot[1], leg[2])
step locate_part(robot[1], leg[1])
step locate_part(robot[1], leg[0])
step locate_part(robot[1], leg[3])
step locate_part(robot[1], table_top)
step navigate_to_part(robot[1], leg[1])
step pick_up_leg(robot[1], leg[1])
step navigate_to_attach_spot(robot[1], table_top, table_top.hole[3])
step place_table_leg(robot[1], leg[1], table_top, table_top.hole[3])
step navigate_to_attach_spot(robot[0], table_top, table_top.hole[3])
step receive_table_leg(robot[0], leg[1], table_top)
step hand_off_table_leg([robot[1], robot[0]], leg[1], table_top)
step attach_table_leg(robot[0], leg[1], table_top, table_top.hole[3])
step release_attached_table_leg(robot[0], leg[1], table_top)
step navigate_to_part(robot[1], leg[0])
step pick_up_leg(robot[1], leg[0])
step navigate_to_attach_spot(robot[1], table_top, table_top.hole[2])
step place_table_leg(robot[1], leg[0], table_top, table_top.hole[2])
step navigate_to_attach_spot(robot[0], table_top, table_top.hole[2])
step receive_table_leg(robot[0], leg[0], table_top)
step hand_off_table_leg([robot[1], robot[0]], leg[0], table_top)
step attach_table_leg(robot[0], leg[0], table_top, table_top.hole[2])
step release_attached_table_leg(robot[0], leg[0], table_top)
step navigate_to_part(robot[1], leg[3])
step pick_up_leg(robot[1], leg[3])
step navigate_to_attach_spot(robot[1], table_top, table_top.hole[0])
step place_table_leg(robot[1], leg[3], table_top, table_top.hole[0])
step navigate_to_attach_spot(robot[0], table_top, table_top.hole[0])
step receive_table_leg(robot[0], leg[3], table_top)
step hand_off_table_leg([robot[1], robot[0]], leg[3], table_top)
step attach_table_leg(robot[0], leg[3], table_top, table_top.hole[0])
step release_attached_table_leg(robot[0], leg[3], table_top)
step navigate_to_part(robot[1], leg[2])
step navigate_to_attach_spot(robot[0], table_top, table_top.hole[1])
step pick_up_leg(robot[1], leg[2])
step navigate_to_attach_spot(robot[1], table_top, table_top.hole[1])
step place_table_leg(robot[1], leg[2], table_top, table_top.hole[1])
step receive_table_leg(robot[0], leg[2], table_top)
step hand_off_table_leg([robot[1], robot[0]], leg[2], table_top)
step navigate_to_part(robot[1], table_top)
step attach_table_leg(robot[0], leg[2], table_top, table_top.hole[1])
step release_attached_table_leg(robot[0], leg[2], table_top)
step navigate_home(robot[0])
step put_away_screwing_device(robot[0])
step navigate_to_part(robot[0], table_top)
step flip_table([robot[1], robot[0]], [leg[1], leg[3], leg[2], leg[0]], table_top)
```

Fig. 4 Complete symbolic plan to build the IKEA Lack table

For example, Lack tables have four legs, declared simply as `group leg(Leg)[4].`

Groups enable the planner to reason efficiently about one type of symmetry. Because the four legs are identical, they are interchangeable. Thus any leg can be attached to any corner of the table, and they can be installed in any sequence. When referencing a member of the group, it is sufficient to reference the first member of that group. The planner knows that this rule extrapolates to all other group members as well. Subsequently, one may reference the second group member to indicate any table leg except the one already referenced.

3.3 ABPL implementation for assembly

IkeaBot's planning module uses the ABPL specification for its raw input and output data. Figure 4 lists a complete symbolic plan beginning from a completely unassembled start state. Note that the plan must be computed online because the system must plan from any arbitrary state of partial assembly. The resulting plan comes with a guarantee that *correctly* executing each specified action in the sequence will result in a complete and correct assembly.

3.4 Detecting failures

To detect failures, the system compares the expected state of the world before and after executing each action to the actual state, as sensed by the perceptual system (line 6 of the `executive` function). We represent the state, q, as a vector of values for logical predicates. Elements of the state for the IKEA Lack table include whether the robot is holding each table leg, whether the table is face-up or face-down, and whether each leg is attached to the table. In the furniture assembly domain, we compute the state using the tracked pose of every rigid body known to the Vicon system, including each furniture part, each robot chassis and hand, and each human. The system recomputes q frequently, since it may change independently of any deliberate robot action, such as by human intervention or from an unintended side-effect.

Prior to executing each action, the assembly executive verifies the action's preconditions against q. Likewise, following each action, the postconditions are verified. Any unsatisfied condition indicates a failure and triggers the assembly executive to pause the assembly process and initiate error recovery. For example, the robot must be grasping a table leg before screwing it into the hole. If it tries and fails to pick up a leg, then the post-condition for the "pick up" action will not be satisfied in q, which indicates a failure.

Table 1 Summary of common failures and the symbolic groundings used to form a help request

Failed symbolic condition	Symbolic request
Part is not visible to the robot	locate_part(*robot, part*)
Robot is not holding the part	give_part(*robot, part*)
Leg is not aligned with the hole	align_with_hole(*leg, top, hole*)
Leg is not attached to the hole	screw_in_leg(*leg, top, hole*)
Table top is not upside down	flip(*top*)
Legacy software is in infinite loop	<not detectable>
Risk of hardware damage	<not detectable>

3.5 Recovery strategy

When a failure occurs, its description takes the form of an unsatisfied condition. The system then asks the human for help to address the problem. The robot first computes actions that, if performed by the human, would resolve the failure and enable the robotic team to continue assembling the piece autonomously. The system computes these actions using a pre-specified model of physical actions a person could take to rectify failed preconditions. Remedy requests are expressed in a simple symbolic language, shown in Table 1. This symbolic request, a, specifies the action that the robot would like the person to take to help it recover from failures. However these symbolic forms are not appropriate for speaking to an untrained user. In Sect. 5, we explore a series of approaches that take as input the symbolic request for help and generate a language expression asking a human for assistance. Before doing so, we provide some background material for performing semantic inference.

Input: Parsed natural language input, $\Lambda = \lambda_1 \dots \lambda_M$, with root $\lambda_{root} \in \Lambda$
1: $\Phi \leftarrow \phi_1 \dots \phi_M$
2: $\Gamma \leftarrow \gamma_1 \dots \gamma_M$
3: $F \leftarrow []$
4: **for** $\lambda_i \in \Lambda$ **do**
5: $\quad \Gamma_{children} \leftarrow []$
6: $\quad$ **for** $\lambda_{child} \in children(\lambda_i)$ **do**
7: $\quad\quad$ Add γ_{child} to $\Gamma_{children}$
8: $\quad$ Add $(\phi_i, \lambda_i, \gamma_i, \Gamma_{children})$ to F
Output: Λ, Φ, Γ, F

Fig. 5 Generating a grounding graph from natural language input. The output is a CRF structured to resemble the parse tree. The structure incorporates a set of correspondence variables, Φ, that cause conditional independence of each term, so that inference may be performed independently on each

4 Generalized grounding graphs

Our approach to language generation involves the inversion of a model of language understanding. Although any understanding facility could be used here, we employ Generalized Grounding Graphs, which were described in previous work (Tellex et al. 2011). We offer a primer on G^3 here for background.

To understand natural language, a robot must be able to map between the linguistic elements of a command, such as "Pick up the white leg," and the corresponding aspects of the external world. Each constituent phrase in the command refers to a particular object, place, or action that the robot should execute in the environment. We refer to the object, place, path, or event as the *grounding* for a linguistic constituent. The aim of the G^3 framework is to find the most probable groundings $\gamma_1 \dots \gamma_N$ given a parsed natural language command Λ and the robot's model of the environment, M:

$$\underset{\gamma_1 \dots \gamma_N}{\text{argmax}}\ p(\gamma_1 \dots \gamma_N | \Lambda, M). \tag{1}$$

The environment model M consists of the robot's location along with the locations, structure and appearance of objects and places in the external world. It defines a space of possible values for the grounding variables $\gamma_1 \dots \gamma_N$. A robot computes the environment model using sensor input. An object such as a table leg is represented in the map as a three-dimensional geometric shape, along with a set of symbolic tags that might be produced by an object classifier, such as "white leg" or "table." A place grounding represents a particular location in the map; it has a zero-length trajectory and no tags. A path grounding consists of a trajectory of points over time. An event or action grounding consists of the robot's trajectory over time as it performs some action (e.g., as it picks up a leg). Formally, each γ_i is a tuple, (g, t, p), where:

- g is a three-dimensional shape, such as a room, or the vicinity of an object. It is expressed as a set of points that define a polygon $(x_1, y_1), \dots, (x_N, y_N)$ together with a height z (e.g., as a vertical prism).
- $p \in \mathbb{R}^{T \times 7}$ is a sequence of T points. Each point is a pose for the region, g. It consists of a tuple $(\tau, x, y, z, roll, pitch, yaw)$ representing the location and orientation of g at time τ (with location interpolated linearly across time). The combination of the shape, g, and the trajectory, T, define a three-dimensional region in the environment corresponding to the object at each time step.
- t is a set of pre-defined textual tags $\{tag_1, \dots, tag_M\}$ that are the output of perceptual classifiers, such as object recognizers or scene classifiers.

The system infers the type of each grounding variable from the associated syntactic constituent using a rule-based algorithm: noun phrases map to objects; prepositional phrases map to places or paths; verb phrases map to events. We assume that the robot has access to the environment model whenever it infers groundings for a natural language command. For brevity, we omit M from future equations in this section.

Since learning the joint distribution over language and groundings is intractable, we must factor Eq. 1 into simpler components. One standard approach is to factor based on certain independence assumptions, then train models for each factor. Natural language has a well-known compositional, hierarchical argument structure (Jackendoff 1983), dividing a sentence into a parse tree where each node is a *linguistic constituent* (Akmajian 2010). A promising approach is to exploit this structure in order to factor the model. The G^3 framework takes as input a natural language command and uses the parse structure of the command to define the random variables and factors in a graphical model (Fig. 5). Specifically, the system creates random variables and factors for each constituent in the parse tree:

$$p(\gamma_1 \dots \gamma_N | \Lambda, M) = \frac{1}{Z} \prod_m \psi_m(\lambda_m, \gamma_{m_1} \dots \gamma_{m_k}). \tag{2}$$

The individual factors ψ_m quantify the correlation between words in the linguistic constituent λ_m and the groundings $\gamma_{m_1} \dots \gamma_{m_k}$. The factors should have large values where words correspond well to groundings and small values otherwise. For example, Fig. 6 shows the graphical model for the phrase "pick up the table leg." There are two grounding variables, one for the verb phrase "pick up" and one for the noun phrase "the table leg." Values for these variables range over events and objects in the environment. There is a sepa-

rate factor ψ_m for each linguistic constituent, and we make independence assumptions so that each factor ψ_m depends on a subset of the language and grounding variables. For example, the factor for "on" depends only on the values of the variables γ_1 and γ_2 and not on other words in the command. These types of independence assumptions are key to our approach in that they enable efficient inference and learning. Without using the structure of language to provide these independence assumptions, we would need to learn a single joint model in which the full joint of the grounding variables would depend on the values of all the language variables. We will describe how to formally derive the structural independencies from the parse tree in Algorithm 5.

The normalization constant Z in Eq. 2 can be ignored during language understanding inference because we do not need the full distribution; we are optimizing only over $\gamma_1 \dots \gamma_N$. However, if models for the functions are learned from labeled data, then the proper distribution (and hence the normalizing constant) will be needed during training. Moreover, when doing language generation (described in the following sections), this term is not constant with respect to the command. If we define a directed model over these variables (e.g., $p(\gamma_1 \dots \gamma_N | \Lambda)$), we can learn each component in order, each with a separate constant, but we must assume a possibly arbitrary order to the conditional γ_i factors. For example, for a phrase such as "the white leg near the table," we could factor in either of the following ways:

$$p(\gamma_{\text{leg}}, \gamma_{\text{table}} | \Lambda) = p(\gamma_{\text{table}} | \gamma_{\text{leg}}, \Lambda) \times p(\gamma_{\text{leg}} | \Lambda) \quad (3)$$

$$p(\gamma_{\text{leg}}, \gamma_{\text{table}} | \Lambda) = p(\gamma_{\text{leg}} | \gamma_{\text{table}}, \Lambda) \times p(\gamma_{\text{table}} | \Lambda). \quad (4)$$

Depending on the order of factorization, we will need different conditional probability tables that correspond to the meanings of words in the language. To resolve this issue, another approach is to use Bayes' Rule to estimate $p(\Lambda | \gamma_1 \dots \gamma_N)$, but learning with this approach would require normalizing over all possible words in the language Λ. Another alternative is to use an undirected model, such as a CRF over the γ_i, but this approach is intractable because it requires normalizing over all possible values of all γ_i variables in the model, including continuous attributes such as location and size.

To address these problems and to make training tractable, the G^3 framework introduces a correspondence vector Φ to capture the dependency between $\gamma_1 \dots \gamma_N$ and Λ. Each entry $\phi_i \in \Phi$ captures whether a particular linguistic constituent $\lambda_i \in \Lambda$ corresponds to the values of the grounding variables, $\gamma_1 \dots \gamma_N$ which are are associated with that constituent by the parse structure of the language. Thus there is one entry in Φ for each constituent in the natural language command. For example, the correspondence variable would be $True$ for the phrase "the white leg" and a grounding of an actual white table leg, and $False$ if the grounding was a different object, such as a black leg or a table. We assume that $\gamma_1 \dots \gamma_N$ are independent of Λ unless Φ is known. Introducing Φ enables factorization according to the structure of language, and simplifies the normalization for each factor to two values for ϕ_i. These locally normalized factors can simply be multiplied together during training without the need to compute a global normalization constant. This representation also allows us to express word meanings as probabilistic predicates, following extensive literature in semantics (Heim and Kratzer 1998). Using the correspondence variable, we can write:

$$\underset{\gamma_1 \dots \gamma_N}{\text{argmax}} \; p(\gamma_1 \dots \gamma_N | \Phi, \Lambda). \quad (5)$$

To perform inference, we assume the correspondence variable Φ to be $True$ and thereby search for the groundings that correspond to the words in the natural language command. When the correspondence vector takes on the value of $True$, this maximization is identical to Eq. 1, except that Φ is listed as a given variable. Listing Φ as a given variable makes explicit the assumption that all linguistic constituents correspond to their associated groundings. This inference is equivalent to maximizing the joint distribution of all groundings $\gamma_1 \dots \gamma_N$, Φ and Λ:

$$\underset{\gamma_1 \dots \gamma_N}{\text{argmax}} \; p(\gamma_1 \dots \gamma_N, \Phi, \Lambda). \quad (6)$$

Note that even though Eq. 6 uses the joint distribution, Λ and Φ are fixed and we optimize over $\gamma_1 \dots \gamma_N$. Next, we rewrite it as a conditional distribution on Φ multiplied by a prior:

$$\underset{\gamma_1 \dots \gamma_N}{\text{argmax}} \; p(\Phi | \Lambda, \gamma_1 \dots \gamma_N) p(\Lambda, \gamma_1 \dots \gamma_N). \quad (7)$$

When Φ is not known, we assume that Λ and $\gamma_1 \dots \gamma_N$ are independent, as in the graphical model shown in Fig. 6, yielding:

$$\underset{\gamma_1 \dots \gamma_N}{\text{argmax}} \; p(\Phi | \Lambda, \gamma_1 \dots \gamma_N) p(\Lambda) p(\gamma_1 \dots \gamma_N). \quad (8)$$

This independence assumption is justified because if we do not know whether $\gamma_1 \dots \gamma_N$ correspond to Λ, then the language does not tell us anything about the groundings.

Finally, for simplicity, we assume that any object in the environment is equally likely to be referenced by the language, which amounts to a constant prior on $\gamma_1 \dots \gamma_N$. We ignore $p(\Lambda)$ since it does not depend on $\gamma_1 \dots \gamma_N$, leading to:

$$\underset{\gamma_1 \dots \gamma_N}{\text{argmax}} \; p(\Phi | \Lambda, \gamma_1 \dots \gamma_N). \quad (9)$$

We factor the model according to the hierarchical, compositional linguistic structure of the command:

$$p(\Phi|\Lambda, \gamma_1 \dots \gamma_N) = \prod_m p(\phi_m|\lambda_m, \gamma_{m_1} \dots \gamma_{m_k}). \quad (10)$$

This factorization defines a probabilistic graphical model that constitutes the grounding graph. It is equivalent to Eq. 2, except for a constant, but assumes that the factors take a specific, locally normalized form. Introducing the correspondence variable allows us to define our model as a conditional random field over Φ, where the random variables and factors are defined systematically from the parse structure of a natural language command (Sutton and McCallum 2012). Each factor $p(\phi_m|\lambda_m, \gamma_{m_1} \dots \gamma_{m_k})$ can be normalized appropriately—only over the domain of the Φ vector for each component—during learning, substantially improving our ability to learn the model. Moreover, each factor corresponds to a probabilistic predicate, consistent with logical representations for word meanings used in the literature on semantics (Heim and Kratzer 1998). As we describe below, the random variables and dependencies are automatically extracted from the parse tree and constituent structure of the natural language command. These factors capture the meanings of words in λ_m. We describe the specific parametric form for the factors in the following sections. The factorization corresponds to the compositional structure of the natural language parse. The parse tree, which provides the structure for creating the graphical model, can be extracted automatically, for example with the Stanford Parser (de Marneffe et al. 2006) or annotated using ground-truth parses.

Formally, a grounding graph consists of random variables for the language $\Lambda = [\lambda_1 \dots \lambda_M]$, the groundings $\Gamma = [\gamma_1 \dots \gamma_M]$, and the correspondence variables $\Phi = [\phi_1 \dots \phi_M]$, along with a set of factors F. Each factor $f \in F$ consists of $\phi_m, \lambda_m, \gamma_{m_1} \dots \gamma_{m_k}$. Algorithm 5 specifies how random variables and factors are automatically created from the parse structure of a natural language command.

Figure 6 shows the grounding graph for the command, "Pick up the table leg on the table." The random variable ϕ_3 is associated with the constituent "the table leg" and the grounding variable γ_2. The random variable ϕ_1 is associated with the entire phrase, "Pick up the table leg on the table" and depends on both the grounding variables γ_1 (the action that the robot or human takes) and the arguments γ_2 (the object being manipulated). The λ_i variables correspond to the text associated with each constituent in the parse tree.

Having described the mechanism for performing forward semantics inference using G^3 for language understanding, we now turn to the inverse semantics problem. In the following section, we discuss methods for a robot to automatically generate help requests directed at a human.

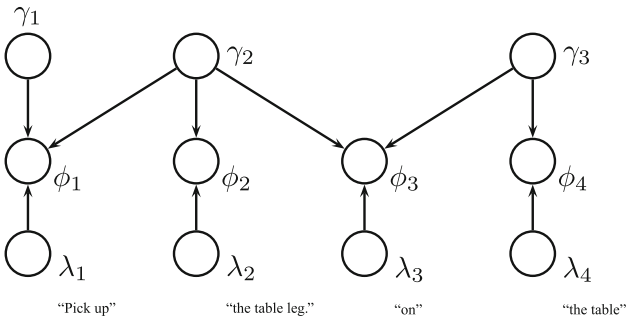


Fig. 6 Grounding graph for the request, "Pick up the table leg on the table." Random variables and edges are created in the graphical model for each constituent in the parse tree. The λ variables correspond to language; the γ variables correspond to groundings in the external world. Edges in the graph are created according to the parse structure of the sentence.

5 Asking for help from a human partner

Once the system computes a symbolic representation of the desired action, a, it searches for words, Λ, which effectively communicate this action to a person in the particular environmental context, M, on line 5 of the `conditions_satisfied` function. This section describes various approaches to the `generate_help_request` function which carries out this inference. We define four increasingly complex approaches for finding Λ, which lead to more targeted natural language requests for help by modeling the ability of the listener to understand it. The contribution of this paper is an approach to finding Λ using *inverse semantics*. Forward semantics is the problem of mapping from words in language to aspects of the external world; the canonical problem is enabling a robot to follow a person's natural language commands (MacMahon et al. 2006; Kollar et al. 2010; Tellex et al. 2011; Matuszek et al. 2012). Inverse semantics is the reverse: mapping from specific aspects of the external world (in this case, an action that the robot would like the human to take) to words in language. To apply this approach we use the G^3 model of natural language semantics described in the previous section.

More concretely, the conventional G^3 model infers the best action to take given the known language. In contrast, in our problem, we *know* what action we want the human to take but want to infer the best language to elicit that action.

Sections 5.1 and 5.2 give simplistic baseline solutions to this problem. Then, for Sects. 5.3 and 5.4, we formally define a function h to score candidate sentences:

$$\underset{\Lambda}{\text{argmax}}\ h(\Lambda, a, M) \quad (11)$$

The specific function h used in Eq. 11 will greatly affect the results. This inference process is therefore a search over possible sentences given the known action. We define a space

$$
\begin{aligned}
S &\rightarrow VB\ NP \\
S &\rightarrow VB\ NP\ PP \\
PP &\rightarrow TO\ NP \\
VB &\rightarrow \text{flip|give|pick up|place} \\
NP &\rightarrow \text{the white leg|the black leg|me} \\
&\quad\ \text{the white table|the black table} \\
TO &\rightarrow \text{above|by|near|under|with}
\end{aligned}
$$

Fig. 7 Part of the context-free grammar defining the linguistic search space

of sentences using a context-free grammar (CFG), shown in Fig. 7, and the inference procedure creates a grounding graph for each candidate sentence using the parse structure derived from the CFG and then scores it according to the function h, but with the hypothesized language and the known action groundings. This search space is quite large, and we use greedy search to expand promising nodes first.

5.1 Speaking by reflex

The simplest approach from the assembly executive's perspective is to delegate diagnosis and solution of the problem to the human with the simple fixed request, $\Lambda =$ "Help me." This algorithm takes into account neither the environment or the listener when choosing what to say. We refer to this algorithm as S_0.

5.2 Speaking by template

As a second baseline, we implemented a template-based algorithm, following traditional approaches to generating language (Fong et al. 2003; Reiter and Dale 2000). This approach uses a lookup table to map symbolic help conditions to natural language requests. These generic requests take the following form:

- "Place part 2 where I can see it."
- "Hand me part 2."
- "Attach part 2 at location 1 on part 5." (i.e. screw in a table leg)

Note that the use of first person in these expressions refers to the robot. Since Vicon does not possess any semantic qualities of the parts, they are referred to generically by part identifier numbers. Such templates can be effective in simple situations, where the human can infer the part from the context, but do not model how words map to the environment, and thus do not reflect the mapping between words and perceptual data. In constrained interaction scenarios, the programmer could hard-code object names for each part, but this approach becomes impractical as the scope of interaction increases, especially for referring expressions that depend on the specific spatial context, such as "the part on the table."

5.3 Speaking by modeling the environment

Next, we describe a more complex model for speaking, that takes into account a model of the environment, but not a model of the listener. To speak using a model of the environment, the robot searches for language that best matches the action that the robot would like the human to take. We compute this model using the G^3 framework. The system converts the symbolic action request a to a value for the action grounding variable, $\gamma_a \in \Gamma$. This variable, γ_a, corresponds to the entire sentence; we refer to the desired value of γ_a as γ_a^*. It then searches for the most likely sentence Λ according to the semantics model. Equation 11 becomes:

$$\underset{\Lambda}{\operatorname{argmax}}\ h(\Lambda, \gamma_a^*, M) \tag{12}$$

It does not consider other actions or groundings in any way when making this determination. Formally:

$$h(\Lambda, \gamma_a^*, M) = \max_{\Gamma|\gamma_a=\gamma_a^*} p(\Lambda|\Gamma, M) \tag{13}$$

With the correspondence variable, this function is equivalent to:

$$h(\Lambda, \gamma_a^*, M) = \max_{\Gamma|\gamma_a=\gamma_a^*} p(\Phi|\Lambda, \Gamma, M) \tag{14}$$

We refer to this metric as S_1. Note that while the optimization in Eq. 14 does assess the quality of the generated sentences, it does not actually model the listener's understanding of those sentences, because it only considers those interpretations where the overall action γ_a matches the desired action γ_a^*. The speaker considers possible sentences and evaluates them using the G^3 framework, selecting the language that best matches the desired action γ_a^* in the environment.

This optimization considers sentences such as "Pick up the black table leg on the white table" even though the grounding for "the white table" is not specified in the symbolic description of the request, because it is added as a disambiguating expression. We handle this by searching over both sentences and paired groundings using beam search.

5.4 Speaking by modeling the listener and the environment

The previous S_1 metric scores shorter, ambiguous phrases more highly than longer, more descriptive phrases. For example, "the white leg" will always have a higher score than

"the white leg on the black table" because the corresponding grounding graph for the longer sentence is identical to the shorter one except for an additional factor, which causes the overall probability for the more complex graph to be lower (or at most equal).

However, suppose the robot team needs a specific leg; for example, in Fig. 10, the robots might need specifically the leg that is on the black table, because that leg is out of reach. In this case, a phrase produced under the S_1 metric such as "Hand me the white leg" will be ambiguous: the person will not know which leg to give to the robot because there are several legs in the environment. If the robot instead said, "Please hand me the white leg that is on the black table," then the person will know exactly which leg to give to the robot.

To address this problem, we augment our robot with a model of the listener's ability to understand a request in the particular environment. Rather than optimizing how well the language in the request fits the desired action, we minimize the uncertainty a listener would experience when using the G^3 model to interpret the request. This metric, which we refer to as S_2, explicitly measures the probability that the listener will correctly understand the requested action γ_a^*:

$$h(\Lambda, \gamma_a^*, M) = p(\gamma_a = \gamma_a^* | \Phi, \Lambda, M) \tag{15}$$

To compute this metric, we marginalize over values of Γ, where $\gamma_a = \gamma_a^*$:

$$h(\Lambda, \gamma_a^*, M) = \sum_{\Gamma | \gamma_a = \gamma_a^*} p(\Gamma | \Phi, \Lambda, M) \tag{16}$$

We factor the model with Bayes' rule:

$$h(\Lambda, \gamma_a^*, M) = \sum_{\Gamma | \gamma_a = \gamma_a^*} \frac{p(\Phi | \Gamma, \Lambda, M) p(\Gamma | \Lambda, M)}{p(\Phi | \Lambda, M)} \tag{17}$$

We rewrite the denominator as a marginalization and conditional distribution on Γ':

$$h(\Lambda, \gamma_a^*, M) = \sum_{\Gamma | \gamma_a = \gamma_a^*} \frac{p(\Phi | \Gamma, \Lambda, M) p(\Gamma | \Lambda, M)}{\sum_{\Gamma'} p(\Phi | \Gamma', \Lambda, M) p(\Gamma' | \Lambda, M)} \tag{18}$$

The denominator is constant so we can move the summation to the numerator:

$$h(\Lambda, \gamma_a^*, M) = \frac{\sum_{\Gamma | \gamma_a = \gamma_a^*} p(\Phi | \Gamma, \Lambda, M) p(\Gamma | \Lambda, M)}{\sum_{\Gamma'} p(\Phi | \Gamma', \Lambda, M) p(\Gamma' | \Lambda, M)} \tag{19}$$

Next we assume that $p(\Gamma | \Lambda, M)$ is a constant, K, for all Γ, so it can move outside the summation. This term is constant because Γ and Λ are independent when we do not know Φ:

$$h(\Lambda, \gamma_a^*, M) = \frac{K \times \sum_{\Gamma | \gamma_a = \gamma_a^*} p(\Phi | \Gamma, \Lambda, M)}{K \times \sum_{\Gamma'} p(\Phi | \Gamma', \Lambda, M)} \tag{20}$$

The constant K cancels, yielding:

$$h(\Lambda, \gamma_a^*, M) = \frac{\sum_{\Gamma | \gamma_a = \gamma_a^*} p(\Phi | \Gamma, \Lambda, M)}{\sum_{\Gamma'} p(\Phi | \Gamma', \Lambda, M)} \tag{21}$$

This equation expresses the S_2 metric. Note that unlike the S_1 metric, the S_2 metric actually uses the G^3 framework to understand the sentences that it generates: computing the denominator in Eq. 21 is equivalent to the problem of understanding the language in the particular environment because the system must assess the mapping between the language Λ and the groundings Γ' for all possible values for the groundings (compare with Eq. 9).

In practice, evaluating the S_2 metric is expensive because of the language understanding computation in the denominator. We therefore consider only the best k sentences produced by the S_1 metric, and then re-evaluate them using the S_2 metric. This optimization may remove candidate sentences that the S_1 sentence scores with a low score, but also removes many obviously incorrect sentences and significantly increases overall inference speed.

5.5 Approach to search

The problem of grounded help request generation requires a sophisticated search over sentence structures, grounding targets, and word choice. We organize search into levels. The S_1 metric employs a single-level approach akin to G^3 understanding, whereas the S_2 metric performs generation using two distinct levels.

The S_1 metric involves a search of a single level (Fig. 8). Given the desired action a, a single grounding graph structure is generated, which contains an appropriate set of grounding variables Γ, subject only to the action grounding variable γ_a corresponding to the desired action grounding variable γ_a^*. The language variables Λ are then populated by appropriate words via search. The result of the S_1 metric is then the sentence in level **I** that corresponds to the maximum value of $(\Phi | \Lambda, \Gamma, M)$.

The S_2 metric performs a deeper search (Fig. 9). Level **I** uses a CFG to elaborate a space of requests, simultaneously forming a grounding graph structure and populating the language variables, Λ. For each $\lambda \in \Lambda$, level **II** infers the grounding variables, γ_a using G^3 as a model of the listener's understanding faculty. Note that the action grounding variable γ_a of these newly inferred sets of grounding variables is not necessarily the same as the desired action grounding variable γ_a^* of the original set of grounding variables: these

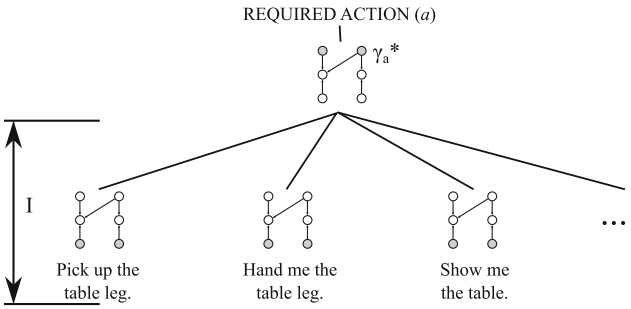


Fig. 8 Approach S_1 performs language generation as a one-level search. A set of desired groundings, a are provided as structured input in the required action. In this approach, a parse tree structure γ_a^* is hard-coded for each action based on the arguments in the request. Approach S_1 then searches over possible language variables, λ, for the highest-confidence correspondence. This technique for language generation resembles the approach to language understanding using G^3 as detailed in Sect. 4

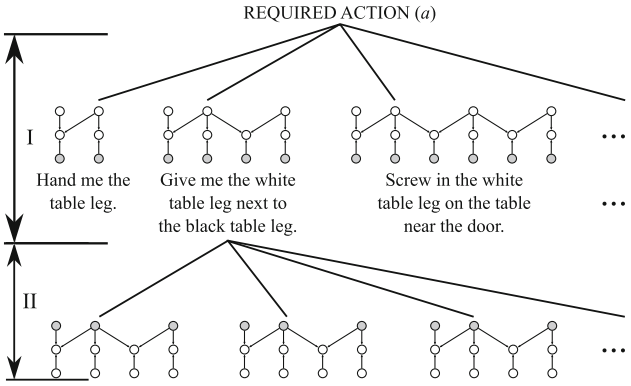


Fig. 9 Approach S_2 performs language generation via a two-level search. A set of desired groundings, a are provided as structured input in the required action. Level **I** elaborates a context-free grammar to search across possible sentence structures and language variables, Λ. This search level resembles approach S_1. Level **II** infers a set of groundings, γ_i, from λ via a model of human comprehension such as G^3. The algorithm terminates with success when it finds a phrase that gives back the desired γ_a^* with probability above a threshold value

are the ambiguous interpretations of each sentence. For each candidate sentence, the grounding graphs in level **II** correspond to a term in the denominator of Eq. 21; those grounding graphs where the action grounding variable γ_a matches the desired action grounding variable γ_a^* correspond to a term in the numerator.

5.6 Search complexity

The S_1 metric searches in the space of all possible CFG parses, and as a result scales with the size of the CFG. Adding a richer vocabulary to the system increases the size of the search space, in worse than polynomial in the number of rules and size of the vocabulary. As a result a limitation of our approach is the ability to scale to larger grammars. We are exploring adding a probabilistic CFG model to our approach to bias the search in the space of words to promising parts of the state space.

The S_2 metric additionally must consider all possible groundings, a challenging problem as the size of the environment increases. When there are many parts, for each unbounded noun phrase the S_2 metric must evaluate the likelihood that that part will be grounded. We use beam search to make this inference tractable.

5.7 Training

We trained the model for understanding language following the same procedure as (Tellex et al. 2011). We collected a new dataset of natural language requests given by a human to another human in the furniture assembly domain. We created twenty-one videos of a person executing a task involved in assembling a piece of furniture. For example, one video shows a person screwing a table leg into a table, and another shows a person handing a table leg to a second person. Each video has an associated context consisting of the locations, geometries, and trajectories of the people and objects, produced with Vicon. We asked annotators on Amazon Mechanical Turk to view the videos and write a natural language request they would give to ask one of the people to carry out the action depicted in the video. Then we annotated requests in the video with associated groundings in the Vicon data. The corpus contains 326 requests with a total of 3279 words. In addition we generated additional positive and negative examples for the specific words in our context-free grammar.

6 Evaluation

The goal of our evaluation was to assess whether our algorithms increase the effectiveness of a person's help, or in other words, to enable them to more quickly and accurately provide help to the robot. To evaluate whether our algorithms enable a human to accurately provide help compared to baselines, we use an online corpus-based evaluation. We conducted a real-world user study to assess whether our leading algorithm improves the speed and accuracy of a person's help to a team of autonomous robots engaged in a real-world assembly task.

6.1 Corpus-based evaluation

Our online evaluation used Amazon Mechanical Turk (AMT) to measure whether people could use generated help requests to infer the action that the robot was asking them to perform. We presented a worker on AMT with a picture of a scene, showing a robot, a person, and various pieces of furniture, together with the text of the robot's request for help. Fig-

"Help me" (S_0)	"Help me."
Templates	"Please hand me part 2."
G^3 S_1	"Give me the white leg."
G^3 S_2	"Give me the white leg that is on the black table."
Hand-written	"Take the table leg that is on the table and place it in the robot's hand."

Fig. 10 Scene from our dataset and the requests generated by each approach

ure 10 shows an example initial scene, with several different requests for help generated by different algorithms, all asking the human to carry out the same action. Next, we showed the worker five videos of a human taking various actions in the scene in response to the requests. We asked them to choose the video that best matched the request for help. We chose actions to film based on actions that would recover from typical failures that the robots might encounter. A trial consists of a worker viewing an initial scene paired with a request for help and then choosing a corresponding video.

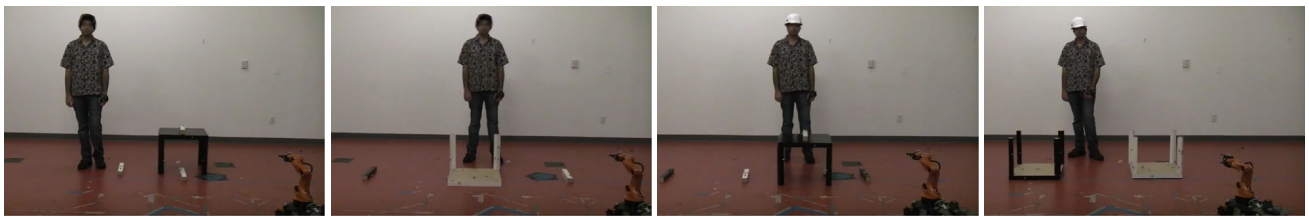
We created a dataset consisting of twenty trials by constructing four different initial scenes and filming an actor taking five different actions in each scene. We present results for the four automatic methods described in Sect. 5, as well as a baseline consisting of hand-written requests which we created to be clear and unambiguous. Figure 11 shows the four initial scenes paired with handwritten help requests. For the "help me" and hand-written baselines, we issued each of the twenty generated requests to five subjects, for a total of 100 trials. We issued each request in the template and G^3 approaches to fifteen users for a total of 300 trials. We assumed the robot had accurate perceptual access to the objects in the environment and their locations using the Vicon system. Results appear in Table 2.

Table 2 Fraction of correctly followed requests

Metric	% Success	95 % Confidence
Chance	20.0	
"Help me" baseline (S_0)	21.0	±8.0
Template baseline	47.0	±5.7
G^3 inverse semantics with S_1	52.3	±5.7
G^3 inverse semantics with S_2	64.3	±5.4
Hand-written requests	94.0	±4.7

Our results show that the "Help me" baseline performs at chance, whereas the template baseline and the G^3 inverse semantics model both improved performance significantly. The S_1 model may have improved performance over the template baseline, but these results do not rise to the level of statistical significance. The S_2 model, however, realizes a significant improvement, $p = 0.002$ by Student's t-test, due to its more specific requests, which model the uncertainty of the listener. These results demonstrate that our model successfully generates help requests for many conditions.

Most failures occurred due to ambiguity in the language, even in sentences generated by the S_2 model. For example, many people confused "the white leg that is near the black table" with "the white leg that is under the black table."



Take the table leg that is on the table and place it in the robot's hand.	Screw the white table leg into the hole in the table top.	Take the white table leg that is next to the table and put it in front of the robot.	Take the white table, flip it over, and set it down in place.
Take the table leg that is under the table and place it in the robot's hand.	Screw the black table leg into the hole in the table top.	Take the black table leg that is next to the table and put it in front of the robot.	Take the black table, flip it over, and set it down in place.
Take the table leg that is next to the table and place it in the robot's hand.	Take the white table leg and insert it in the hole, but do not screw it in.	Take the black table leg that is far away from the table and put it in front of the robot.	Take the white table and move it near the robot, keeping it upside-down.
Pick up the table leg that is on the table and hold it.	Move the white table leg over near the table top.	Take the white table leg that is on top of the table and place it in the robot's hand.	Pick up the white table and hold it.
Take the table leg that is on the table and place it on the floor in front of the robot.	Take the table top and place it near the white table leg on the floor.	Pick up the white table leg next to the table and hold it.	Take the white table, flip it over, and put it in the robot's hand.

Fig. 11 The four initial scenes from the evaluation dataset, together with the hand-written help requests used in our evaluation

Adding more prepositions, such as "next to" would address this issue by enabling the algorithm to generate more specific referring expressions that more accurately match people's expectations.

6.2 User study

In our experiment, humans and robots collaborated to assemble IKEA furniture. The study split participants into two conditions using a between-subjects design, with 8 subjects in each condition. In the baseline condition, robots requested help with the S_0 approach, using only the words "Please help me." In the test condition, robots requested help using the S_2 inverse semantics metric. The robots autonomously planned and executed the assembly on two real robots, and all detected failures were real. Our goal was to assess the effect of the choice of help request, made to a user with limited situational awareness, within an end-to-end system. We chose approach S_0 as a baseline to evaluate the magnitude of this effect. The accompanying video is online at http://youtu.be/2Ts0W4SiOfs.

We measure effectiveness by a combination of objective and subjective measures. We report two objective measures: *efficiency*—the elapsed time per help request, and *accuracy*—the number of error-free user interventions. Taken together, these measures show how effectively the human's time is being used by the robots. We also report three subjective measures derived from a post-trial survey, as well as their own written feedback about the system, to gain an understanding of their view of the strengths and weaknesses of our approach.

6.2.1 Procedure

Subjects in each condition were gender-balanced and had no significant difference in experience with robots or furniture assembly. To familiarize users with the robot's capabilities, we gave them a list of actions that might help the robots. During preliminary trials, subjects had problems when handing parts to the robot (called a hand-off), so we demonstrated this task and gave each user the opportunity to practice. The entire instruction period lasted less than 5 min, including the demonstration. During the experiment, we instructed users to focus on a different assembly task and only help the robots when requested.

For each subject, the robot team started from the same initial conditions, shown in Fig. 12. Some failures were inevitable given the initial conditions (e.g., a table top turned upside down; a part on a table out of the robots' reach.) Other failures happened naturally (e.g., a table leg that slipped out of a robot's gripper.) When a failure occurred during assembly, the failing robot addressed the person by saying, "Excuse me," and generated and spoke a request for help through an

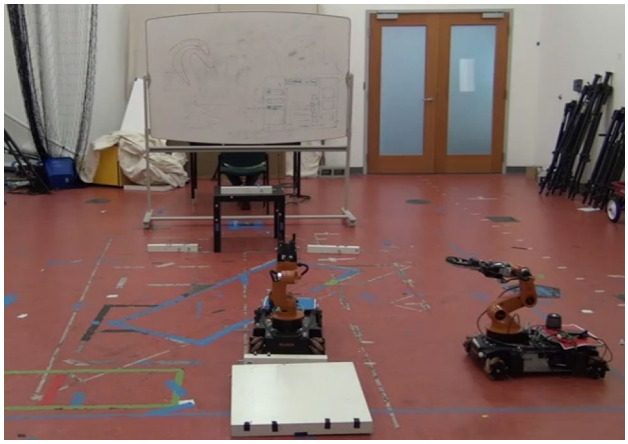


Fig. 12 Initial configuration for the user study. The user is seated behind the whiteboard in the background

on-board speaker, distinguishing itself by color if necessary. We projected all dialogue on a large screen to remove dependence on understanding synthesized speech. The human then intervened in the way they felt was appropriate.

After communicating a help request, the robots waited up to 60 sec for the user to provide help. If the environment changed in a way that satisfied the request, the robot said "Thank you, I'll take it from here," and we counted the person's intervention as successful. If the allotted time elapsed, the robot instead said "Never mind, I'll take it from here," and moved on to a different part of the assembly process. These instances were recorded as failed interventions. For each intervention, we recorded the time elapsed and number of actions the human took in attempting to solve the problem.

Each trial ran for 15 min. Although we tried to limit experimenter intervention, there were several problems with the robotic assembly system that required expert assistance. Experimenters intervened when either of two situations arose: potential damage to the hardware (19 times), or an infinite loop in legacy software (15 times). In addition, software running on the robots crashed and needed to be restarted 5 times. In the future, we plan to address these issues using methods for directing requests to the person most likely to satisfy them, rather than only targeting requests at untrained users.

6.2.2 Results and discussion

Over the course of the study, the robots made 102 help requests, of which 76 were satisfied successfully within the 60-sec time limit. The most common request type was the hand-off, comprising 50 requests.

Table 3 gives results from all metrics. Compared to the baseline, S_2 found a decrease in response time for the human to complete non-hand-off interventions and an increased rate

Table 3 End-to-end user study results

Objective metrics	Interventions	S_0	S_2	p (t test)
Intervention time (sec)	Non-hand-offs	33.3	25.1	0.092
Intervention time (sec)	All	33.7	31.6	0.499
Error-free interventions (%)	All	57.1	77.6	0.039
Successful interventions (%)	All	70.3	80	0.174
Subjective metrics	**p (Kruskal–Wallis)**			
Robot is effective at communicating its needs	0.001			
Prefer working in parallel with the robot	0.056			
Easy to switch between and robots' task and user's	0.388			

of accuracy at performing those interventions. Mean intervention time on all tasks, including handoffs, shows a less-clear improvement. After observing the trials, we noticed that subjects found it difficult to successfully hand a part to the robot, despite our initial training. The change in overall success rate was not statistically significant, probably because users were permitted to try multiple actions within the 60-sec window when the request was ambiguous. Such retries counted against the accuracy and intervention time metrics.

Qualitatively, subjects preferred the language generation system; Table 3 shows subjective metrics on a five-point Likert scale. Two subjective metrics showed statistically significant results. Users in the S_2 condition reported that the robot was more effective at communicating its needs. They were also more likely to record a preference for assembling two kits in parallel as opposed to assembling one kit at a time together with the robots. However, users in the S_2 condition did not find it significantly easier to switch between helping the robot and working on their own task. We attribute this finding to the facts that interruptions were sometimes frequent, and that hand-offs, which made up a majority of requests, remained difficult even with clear requests.

Figure 13 shows comments from participants in the study in each condition. Overall, users enjoyed the experience more in the S_2 condition. Even when users successfully helped the robots in the baseline condition, they frequently complained that they did not know what the robot was asking for.

Despite these promising successes, important limitations remain. First, hand-offs remained difficult for users even after training. Second, the system required frequent intervention by the experimenters to deal with unexpected failures. Both of these conditions might be modified by a more nuanced model of what help a human teammate could provide. For example, if the robots could predict that handoffs are challenging for people to successfully complete, they might ask for a different action, such as to place the part on the ground near the robot. Similarly, if the robots were able to model the ability of different people to provide targeted help, they might direct some requests to untrained users, and other requests

"Help me" Baseline

"I think if the robot was clearer or I saw it assemble the desk before, I would know more about what it was asking me."

"Did not really feel like 'working together as a team' – For more complex furniture it would be more efficient for robot to say what action the human should do?"

"The difficulty is not so much working together but the robots not being able to communicate the actual problem they have. Also it is unclear which ones of the robots has the problem."

G^3 Inverse Semantics with S_2

"More fun than working alone."

"I was focused on my own task but could hear the robot when it needed help.... However, I like the idea of having a robot help you multitask."

"There was a sense of being much more productive than I would have been on my own."

Fig. 13 Comments from participants in our study

to "level 2" tech support. The different types of interventions provided by the experimenters compared to the subjects points to a need for the robots to model specific types of help that different people can provide, as in Rosenthal et al. (2011).

6.3 Conclusion

The goal of our evaluation was to assess the effectiveness of various approaches for generating requests for help. The corpus-based evaluation compares the inverse semantics method to several baselines in an online evaluation, demonstrating that the inverse semantics algorithm significantly improves the accuracy of a human's response to a natural language request for help compared to baselines. Our end-to-end evaluation demonstrates that this improvement can be realized in the context of a real-world robotic team interacting with minimally trained human users. This work represents a step toward the goal of mixed-initiative human–robot cooperative assembly.

Our end-to-end evaluation highlights the strength of the system, but also its weakness. Our robot used a single model for a person's ability to act in the environment; in reality,

different people have different abilities and willingness to help the robot. Additionally we assumed that the robot and person both had equal perceptual access to the objects in the environment; in practice many failures may occur due to the perceptual system not detecting an object, leading to the robot being unable to generate an accurate help request, or generating an ambiguous request because it is not aware of all the distractor objects. Developing a dialogue system capable of answering questions from people in real time could provide disambiguation when people fail to understand the robot's request. As we move from robot-initiated to mixed-initiative communication, the reliance on common ground and context increases significantly. Since our models can be expected to remain imperfect, the demand for unambiguous sentences becomes less satisfiable. In the long term, we aim to develop robots with increased task-robustness in a variety of domains by leveraging the ability and willingness of human partners to assist robots in recovering from a wide variety of failures.

Acknowledgments This research was done at CSAIL-MIT. This work was supported in part by the Boeing Company, and in part by the U.S Army Research Laboratory under the Robotics Collaborative Technology Alliance. The authors thank Dishaan Ahuja and Andrew Spielberg for their assistance in conducting the experiments.

References

Akmajian, A. (2010). *Linguistics an introduction to language and communication*. Cambridge: MIT Press. ISBN 000-0262513706.

Bollini, M., Tellex, S., Thompson, T., Roy, N., & Rus, D. (2012). Interpreting and executing recipes with a cooking robot. In *13th international symposium on experimental robotics*.

Chen, D. L., & Mooney, R. J. (2011). Learning to interpret natural language navigation instructions from observations. In *Proceedings of AAAI*.

de Marneffe, M, MacCartney, B., & Manning, C. (2006). Generating typed dependency parses from phrase structure parses. In *Proceedings of international conference on language resources and evaluation (LREC)* (pp. 449–454). Genoa.

Dorais, G., Banasso, R., Kortenkamp, D., Pell, P., & Schreckenghost, D. (1998). *Adjustable autonomy for human-centered autonomous systems on mars*. Presented at the Mars Society Conference.

Dragan, A., & Srinivasa, S. (2013). Generating legible motion. In *Robotics: Science and Systems*.

Dzifcak, J., Scheutz, M., Baral, C., & Schermerhorn, P. (2009). 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 *Proceedings of IEEE international conference on robotics and automation* (pp. 4163–4168).

Fasola, J., & Mataric, M.J. (2013). Using semantic fields to model dynamic spatial relations in a robot architecture for natural language instruction of service robots. In *2013 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE (pp. 143–150).

Fong, T., Thorpe, C., & Baur, C. (2003). Robot, asker of questions. *Journal of Robotics and Autonomous Systems*, *42*, 235–243.

Garoufi, K., & Koller, A. (2011). Combining symbolic and corpus-based approaches for the generation of successful referring expressions. In *Proceedings of the 13th European workshop on natural language generation*. Association for Computational Linguistics (pp. 121–131).

Goeddel, R., & Olson, E. (2012). Dart: A particle-based method for generating easy-to-follow directions. In *2012 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE (pp. 1213–1219).

Golland, D., Liang, P., & Klein, D. (2010). A game-theoretic approach to generating spatial descriptions. In *Proceedings of the 2010 conference on empirical methods in natural language processing*. Association for Computational Linguistics (pp. 410–419).

Goodman, N. D., & Stuhlmüller, A. (2013). Knowledge and implicature: Modeling language understanding as social cognition. *Topics in Cognitive Science*, *5*(1), 173–184.

Heim, I., & Kratzer, A. (1998). *Semantics in generative grammar*. Oxford: Blackwell. ISBN 978-0631197133.

Hertle, A. (2011). Design and implementation of an object-oriented planning language. Master's thesis, Albert-Ludwigs-Universität Freiburg.

Jackendoff, R. S. (1983). *Semantics and cognition* (pp. 161–187). Cambridge: MIT Press.

Jurafsky, D., & Martin, J.H. (2008). Speech and language processing (2 ed.). Pearson Prentice Hall. ISBN 0131873210.

Knepper, R.A., Layton, T., Romanishin, J., & Rus, D. (May 2013). IkeaBot: An autonomous multi-robot coordinated furniture assembly system. In *Proceedings of IEEE international conference on robotics and automation*. Karlsruhe.

Kollar, T., Tellex, S., Roy, D., & Roy, N. (2010). Toward understanding natural language directions. In *Proceedings of ACM/IEEE international conference on human-robot interaction* (pp. 259–266).

Krahmer, E., & Van Deemter, K. (2012). Computational generation of referring expressions: A survey. *Computational Linguistics*, *38*(1), 173–218.

MacMahon, M., Stankiewicz, B., & Kuipers, B. (2006). Walk the talk: Connecting language, knowledge, and action in route instructions. In *Proceedings of national conference on artificial intelligence (AAAI)* (pp. 1475–1482).

Maitin-Shepard, J., Lei, J., Cusumano-Towner, M., & Abbeel, P. (2010). Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *Proceedings of IEEE international conference on robotics and automation*. Anchorage, AK.

Matuszek, C., FitzGerald, N., Zettlemoyer, L., Bo, L., & Fox, D. (2012). A joint model of language and perception for grounded attribute learning. Arxiv preprint arXiv:1206.6423.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). PDDL—the planning domain definition language. Technical Report CVC TR98003/DCS TR1165, Yale Center for Computational Vision and Control. New Haven.

Reiter, E., Dale, R. (2000). Building natural language generation systems. Cambridge University Press. ISBN 9780521620369.

Rosenthal, S., Veloso, M., & Dey, A. K. (2011). Learning accuracy and availability of humans who help mobile robots. In *Proceedings of AAAI*.

Roy, D. (2002). A trainable visually-grounded spoken language generation system. In *Proceedings of the international conference of spoken language processing*.

Simmons, R., Singh, S., Heger, F., Hiatt, L.M., Koterba, S.C., Melchior, N., & Sellner, B.P. (2007). Human-robot teams for large-scale assembly. In *Proceedings of the NASA science technology conference*.

Striegnitz, K., Denis, A., Gargett, A., Garoufi, K., Koller, A., & Theune M. (2011). Report on the second second challenge on generating instructions in virtual environments (give-2.5). In *Proceedings of the 13th European workshop on natural language generation*. Association for Computational Linguistics (pp.270–279).

Sutton, C. A., & McCallum, A. (2012). An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, *4*(4), 267–373.

Tellex, S., Kollar, T., Dickerson, S., Walter, M., Banerjee, A., & Teller, S., et al. (2011). Understanding natural language commands for robotic navigation and mobile manipulation. In Proceedings of AAAI.

Tellex, S., Knepper, R., Li, A., Rus, D., & Roy, N. (2014). Asking for help using inverse semantics. In *Robotics: Science and systems*, (Best Paper.).

Vogel, A., Bodoia, M., Potts, C., & Jurafsky, D. (2013a). Emergence of gricean maxims from multi-agent decision theory. In *Proceedings of NAACL*.

Vogel, A., Potts, C., & Jurafsky, D. (2013b). Implicatures and nested beliefs in approximate Decentralized-POMDPs. In *Proceedings of the 51st annual meeting of the association for computational linguistics*. Association for Computational Linguistics. Sofia.

Wilson, R. (1995). Minimizing user queries in interactive assembly planning. *IEEE transactions on robotics and automation*, *11*(2).

Ross A. Knepper is an assistant professor in the Department of Computer Science at Cornell University. His research focuses on the theory, algorithms, and mechanisms of automated assembly. Previously, Ross was a Research Scientist in the Distributed Robotics Lab at MIT. Ross received his M.S and Ph.D. degrees in Robotics from Carnegie Mellon University in 2007 and 2011.

Stefanie Tellex is an assistant professor at Brown University. The aim of her research program is to construct robots that seamlessly use natural language to communicate with humans. She completed her Ph.D. at the MIT Media Lab in 2010, where she developed models for the meanings of spatial prepositions and motion verbs. Her postdoctoral work at MIT CSAIL focused on creating robots that understand natural language. She was named one of IEEE Spectrum's AI's 10 to Watch and won the Richard B. Salomon Faculty Research Award at Brown University.

Adrian Li received an M.Eng. degree from the University of Cambridge in 2014 in the Engineering Tripos. He spent a year abroad visiting MIT EECS in 2012–2013.

Nicholas Roy is an Associate Professor in the Department of Aeronautics & Astronautics at the Massachusetts Institute of Technology and a member of the Computer Science and Artificial Intelligence Laboratory (CSAIL) at MIT. He received his Ph.D. in Robotics from Carnegie Mellon University in 2003. His research interests include mobile robotics, decision-making under uncertainty, human-computer interaction, and machine learning.

Daniela Rus is a professor in the EECS Department at MIT. She is the Director of CSAIL. She holds a PhD degree in computer science form Cornell University. Her research interests include distributed robotics, mobile computing, and programmable matter. She has several research activities in environmental robotics. She is the recipient of an NSF Career award and an Alfred P. Sloan Foundation fellowship. She is a class of 2002 MacArthur Fellow. She is a fellow of AAAI and IEEE.