Second Annual Conference on Advances in Cognitive Systems Poster Collection (2013) 167-183

Spatial Planning For Placement Command Understanding

Kalyan Moy Gupta KALYAN.GUPTA@KNEXUSRESEARCH.COM
Kellen Gillespie KELLEN.GILLESPIE@KNEXUSRESEARCH.COM
Justin Karneeb JUSTIN.KARNEEB@KNEXUSRESEARCH.COM
Hayley Borck HAYLEY.BORCK@KNEXUSRESEARCH.COM

Knexus Research Corp., 163 Waterfront Street, Suite 440, National Harbor, MD 20745.

Abstract

Object placement in virtual and real worlds is an important task for autonomous agents and applications. Interacting with agents using natural language commands presents an intuitive alternative to graphical and other operator control interfaces. However, understanding and interpreting language for placement actions in 3D continuous spaces is computationally hard. In this paper, we extend our previous work on resolving underspecified linguistic commands for object placement using a spatial planning approach called object placement using ordered application and simulation of constraints (OPOCS). This heuristic approach represents and utilizes practical real-world knowledge of objects an their interactions. It does this with simulation of human activities to accurately place and orient objects. We evaluated OPOCS and compared it with a naïve algorithm for understanding 10 representative spatial terms and relations in 4 different 3D office worlds. Our findings show that OPOCS significantly outperforms a naïve algorithm.

1. Introduction

A number of real and virtual world tasks involve objects being placed inside of them. For instance, constructing a virtual scene in a graphical environment requires a user to place objects in the world (e.g., Coyne & Sproat, 2001). Similarly, robots manipulating and moving objects in their environments need to pick up and place these objects in their world (e.g., Cosgun *et al*., 2011; Jiang *et al*., 2012). Using conventional graphical user or operator control interfaces for such object placement tasks can be cumbersome to learn and use; it may even be impractical in some situations. Instead, natural language (NL) commands can be issued to an application system to place objects in a real or virtual world. For instance, an interior designer could issue the following command to an interior design system "Put the printer on the desk." Although using natural language (NL) commands can be an intuitive and efficient alternative to graphical interfaces (e.g., Coyne and Sproat, 2001; and Dupuy, 2001) machine understanding of natural language commands to generate an appropriate response can be notoriously difficult due to the polysemy and vagueness of spatial terms.

Understanding natural language placement commands requires suitable semantic representations of spatial terms (Herskovits, 1986; Talmy, 2000; Bateman et al., 2010). Much research in spatial term semantics has focused on developing context free computational models (e.g., Regier and Carlson, 2001; Coventry *et al.*, 1994). These models range from quantitative (e.g., Kelleher & Costello, 2009) to qualitative (e.g., Lockwood, 2009). Although they further our understanding of the complexity and fluidity required in lexical semantic representations, they alone are insufficient for applications we described above. Consequently, in this paper, we present an approach for understanding spatial terms so that an embodied agent can act on them. We posit that actionable understanding and representation of spatial terms requires performing the computational task of spatial planning. While we focus only on the understanding of spatial terms in this paper, our ultimate goal is to support the claim that planning and simulation are core cognitive processes for *deep and embodied* natural language understanding (Bergen, 2005). This paper builds on our prior work on functional knowledge representations for spatial language understanding (Gupta *et al.* 2011). Our contributions in this paper are as follows: First, we introduce a new form of ambiguity in natural language called *pragmatic ambiguity*. Second, we identify an object placement task as a spatial planning task, which we formulate as a weighted soft constraint satisfaction problem. Third, we present a heuristic approach called Object Placement by Ordered Constraint Simulation (OPOCS). OPOCS performs constraint satisfaction by leveraging richly structured functional and pragmatics knowledge by implicitly simulating human interactions with objects. Finally, we evaluate the effectiveness of OPOCS for understanding NL placement commands containing 10 types of spatial terms to place objects in 3D office worlds. Our results show that OPOCS significantly outperforms a naïve baseline approach.

We organize the remainder of this paper as follows. We introduce the concept of pragmatic underspecification in language in the following section. In Section 3, we present OPOCS, first its knowledge representation approach, and then its constraint solving algorithm for placing an object in 3D virtual worlds. We describe our empirical evaluation and results in Section 5. We present related work in Section 6 and Section 7 concludes the paper.

2. Pragmatic Ambiguity in Natural Language Commands

Semantic ambiguities occur during natural language interpretation when there are multiple meaning representations of an utterance. The multiple meaning representations can arise structurally (e.g., argument roles and adjunct attachments) and from polysemous constituents (Egg, 2010; Frison, 2009). Although we can resolve these semantic ambiguities, that alone is insufficient to make an embodied agent ready to act as ambiguities will still remain at the *pragmatic* level. We illustrate this with an interpretation of a NL command for object placement. Consider a 3D virtual office world with a desk, a computer monitor, and a chair (see Figure 1). A user issues a command "put the printer *on* the desk", to which the system should respond with an appropriate placement. Let's assume that we have access to a suitable lexico-semantic representation of the spatial preposition 'on' which resolves to a valid placement surface (i.e., the desk). Despite this interpretation, we are presented with several choices for placement. For instance, possible placements include on the desk to the left, right, front and back of the monitor. However, the placements in front and back of the monitor are functionally invalid for a human user as they obscure functional view and are unreachable, respectively. The utterance also does

not specify the proper orientation of the printer. Without such a specification, the printer could be oriented in numerous ways in relation to the monitor and the chair, only some of which being valid. For example, the orientation shown in Figure 1 is a valid one. However, the orientations of the printer such as upside down or facing the wall would be invalid. We term this residual ambiguity arising in the context of a situation and an NL command as *pragmatic ambiguity*.

Figure 1. Example 3D world for object placement

Clearly, this ambiguity is due to the omission of qualifying placement details from the command (i.e., *underspecification*). For instance, a more specific command would be "Put the printer on the desk, to the right of the monitor facing the chair". However, such a precise command formulation when multiple agents (e.g., a human operator and autonomous agent) operate in a shared environment is unnatural. Therefore, there is a need to understand and respond to *pragmatically underspecified* commands. What sorts of functional knowledge and conventions are required to recover the underspecified elements of a command? Additionally, how can a reasoner exploit them to perform placement planning? Identifying, encoding and utilizing richly structured placement heuristics to accurately place objects in the real-world practical tasks is one of primary contributions we make in this paper. We describe the OPOCS algorithm that performs this task in the next section.

2.1 Spatial Planning Task Formulation

Given a 3D world W containing a set of objects O located in various places, and a natural language command L requesting to place a target object o^t in W, find a placement (x, y, z, Ω) that *best satisfies* the intent I of the commanding agent. Let (x, y, z) represent the coordinate location of target object and let Ω represent the orientation for o^t. We postulate that the placement that best satisfies is the one that minimizes the *interaction cost* for the commanding human agent and meets the commanding agent's intent. Clearly, brute force search in a knowledge-poor manner would be expensive and is likely to result in invalid placements, as we demonstrated in our example earlier. Therefore, in the next subsection we identify the types of knowledge and the representation required and to constrain the placement. Computing the provably best placement is NP hard (Jiang *et al.*, 2012). Consequently, in OPOCS we adopt heuristic soft constraint satisfaction approach, which we present in the section below.

2.2 Knowledge Representation

Our goal in providing world knowledge to the OPOCS is to constrain the placement search. This fundamentally requires only two types of *constraints* between objects: 1) *distance* and 2) *orientation*. Since object placement commands are uttered by human agents who interact with objects in the world, we introduce the notion of a human agent, α, as the primary basis for perceiving and conceptually representing objects in the world for use in spatial planning. In the following subsection, we detail agent properties representation and follow this with object properties and interactions representation.

2.2.1 Agent Properties Representation

We assume that agent α interacts with objects using a set of *primitive actions* or perceived affordances (Gibson, 1977, Norman, 2002). Further, we assume that agent α interacts with objects in the world with a purpose or *intention*. Intentions can help constrain an underspecified placement command by implying a *locational preference*. For instance, a computer monitor for *usage* must be placed and oriented in a certain way on a work surface. Alternatively, if it is to be placed for *storage* it may be placed in a box with a different orientation. We represent the *intention of interaction* with one of the following values:

1. *Use:* This is when an agent intends to use the object. For example, default usage of chair is for sitting, for which it must be upright. However, when intended for storage it need not be oriented upright.
2. *Store*: This is when the agent intends to store the object for safe keeping. For example, chairs may be stacked up when not used. Likewise utensils such as spoons and forks have different locational preferences when intended for storage versus usage.
3. *Maintain*: This is a situation when an agent intends to perform maintenance acts on the object. For example, unclean forks and cups may be placed in the dishwasher.
4. *Build*: This is a situation when an agent intends to use it for building another object.

We assume that agents interact with objects by being located at certain places in the world called *activity stations*. Certain objects may themselves be activity stations. For example, a chair is an activity station as an agent may sit on it while interacting with other objects such as a keyboard or a mouse. Additionally, we represent certain agent properties, including *age*, *gender*, and *handedness*, which can impact the ability to reach and interact with them. OPOCS can adapt its responses to suit different agent characteristics. For instance, object placement for a child would consider a much smaller reach than that of an adult. We also consider a limited number of poses for α while they are located at the activity stations and interacting with objects in the world. These poses include sitting, standing, and lying down. We consider the following two types of agent interaction parameters to satisfy distance and orientation constraints between agents and objects:

1. *Reach*: α reaches for objects to manipulate and interact with them. For instance, an office worker may reach for pen to write with it. This type of interaction imposes a *reachability* constraint between α and an object, which we in turn transform into a *distance* constraint as an input to OPOCS. Depending on the type of object, α may prefer to use a different body part. For example, α may use a foot to interact with a ball. This subtlety can affect whether an object is reachable or not from a particular location (i.e., the underlying distance constraint). Therefore, we subcategorize the *reach* interaction as follows (see Figure 2):
    a. *Reach.Arm*: α reaches for objects with arms fully extended.

b. *Reach.Forearm*: α reaches for objects with only the forearm extended.
c. *Reach.Foot*: α reaches for object with its foot.
d. *Reach.Assisted*: α reaches for objects with tools.

2. *See*: For many tasks, α must visually interact with objects in the world. For instance, to locate an object with the intent of reaching and manipulating the object. For α to see an object it must be oriented toward the objects. This imposes a *visibility* constraint on the target placement, i.e., the object must be visible from a certain location, which we transform into *orientation and distance* constraints. In certain situations, α must be able to read the information present on the object. We represent this with the *read* action, a tighter constraint than see. An agent reads the information present on the object such as signs or writing. Clearly, this can be subcategorized to read fine print, read normal print, read large print, read poster print etc. Reading imposes a shorter distance constraint that visibility while preserving the orientation.

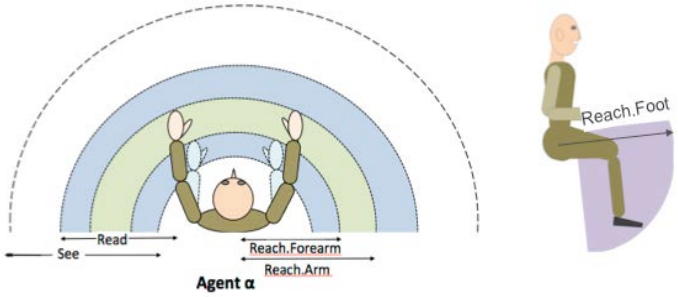

Figure 2. Agent interaction parameters

2.2.2 Object Properties & Interaction Constraints Representation

The objects afford certain interaction actions and functions for human agents and thereby can constrain the placement of functionally related objects. For the purposes of agent-to-object and object-to-object interaction we consider the *surfaces* of an object and their orientations as the primary basis of our representation. We use it to represent applicable distance and orientation constraints between objects in our knowledge base. Furthermore, we specify additional knowledge and preferences about placement heights. We represent the following properties for object surface (see Figure 3):

- *Intrinsic axes and their canonical geo-orientations*: We anchor three-dimensional axes at the centroid of a surface, with the positive local *z*-axis as its outside normal, the positive local *y-axis* as its intrinsic right and the positive local *x*-axis as its intrinsic top. Figure 3 shows the representation of two surfaces on the table: `table.top.face` and `table.leg.face`, as well as their intrinsic axes. We represent the canonical geo-orientations with the axis annotations 'top' and 'bottom'. Clearly, many objects and their surfaces do not have a canonical orientation, such as a tabletop.

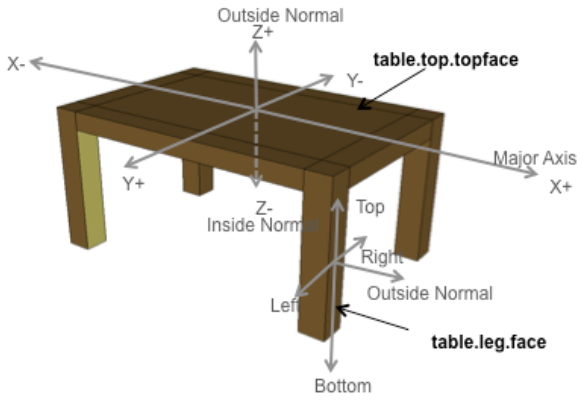

Figure 3. Example surfaces and their intrinsic axes representations for a table

- *Preferred height*: We specify the preferred height constraint of a surface with categorical labels representing the agent attributes with the following labels: floor, knee, waist, chest and eye.
- *Functional interaction constraints*: There are two types of functional interaction constraints, those with an agent and those with other objects.
  - We represent the interaction of a surface with an agent with the following values (see Figure 4):
    - *Activity surface*: A surface may be a region where the agent is located while performing interaction activities. For instance, a chair's seat is an activity surface where an agent is located in a sitting pose while interacting with other objects. Likewise, a floor is an activity surface where the agent may interact with other objects such as a table while in a standing pose.
    - *Reachable*: The surface must be reachable by an agent for some activity surface
    - *Visible*: The surface must be visible from an activity station
    - *Readable*: The surface must be readable from some activity station. For example, the face of a computer monitor must be readable by an agent.

```
''kitchen_table" : {
   "support" : True
   "containment" : False
   "default_intent" : "USE"
   "installable" : False
   "activity_station" : True
   "attracts" : ("book" "laptop" ... )
   "height_preference" : "FLOOR"
   "one_per_activity_station"   : True}
```

Figure 4. Example functional knowledge for a kitchen table

  - We represent interaction constraints between objects with the following labels:

- *Support*: One surface supports another surface, such as a `table.top.face` may support a `monitor.stand.bottom` surface as the table is supporting the monitor itself.
- *Share activity region*: Certain objects may be collocated because they share the same activity region. For instance, monitor, mouse and keyboard surfaces may share activity regions that involve a workstation (e.g., an office chair seat)
- *Attract same kind*: Certain objects such as books and plates may attract their own types and are consequently collocated. This value specifies such a preference.
- *Installable*: Certain objects are installable allowing OPOCS to relax vertical surface placement constraints. For instance, pictures may be placed on vertical surfaces such as walls.

2.3 OPOCS Algorithm

Overview

Based on a given 3D world comprising stationary objects and a NL command to place an object, the algorithm proceeds as follows. First, it identifies a set of possible surfaces on which it can place the *target object*. Next, it identifies a set of activity surfaces and subsequently segments them to select a set of *relevant activity stations*. This simulates a human agent's activity and interaction with relevant objects in the world. Intuitively, relevant activity stations are those that allow for the easy reachability and manipulation of a number of related and collocated objects. Based on the specified spatial constraints and the world, the algorithm then *infers the applicable placement constraints* by using its knowledge. Using the identified activity stations and the applicable placement constraints, it computes a placement that best satisfies the expressed and inferred constraints. We present the details of these steps below.

Inputs

1. O, a set of objects located in W.
2. o^t, the target object to be placed (e.g., a printer).
3. *LC*, a linguistically expressed placement constraint (e.g., on the desk). This typically comprises a landmark or reference object o^l (in this example, 'the desk') and a spatial relation *r* (in this example, 'on').
4. KB, the functional interaction knowledge base containing the agent and object interaction knowledge covering all objects in W (O and o^t). (see Section 3.2)
5. Representative agent α for whom the placement is to be performed.

Output

The best placement p<*x, y, z*, Ω>, where *best* implies the highest *constraint satisfaction score*.

Processing steps

1. *Get candidate placement surfaces* (*CPS*): A candidate placement surface *cps* is one on which the target object o^t could be placed for the given command *LC*. We identify O^l as the set of all objects associated with the landmark object o^l. We then identify all the surfaces associated with o^l and the spatial relation *r* as the initial set *CPS*. We filter them based on the following criteria to obtain the final set of *CPS*:
   a. If o^t is not installable, then we only consider those surfaces whose outside normal points upwards, else we also consider surfaces whose outside normal points horizontally.
   b. The height of the surface must be within the height preference range of o^t.

c. The smallest dimension of the surface must be larger than the largest dimension of the target object.

2. *Get candidate activity surfaces* (*CAS*): An activity surface is one on which agent α is located while interacting with other candidate objects. For example, this may be the floor or furniture such as a seat. A *cas* is one that is within α's reachable distance from at lease one *cps*.
3. *Get all objects in contact with CAS and CPS* (*OCON*): For all the surfaces in *CPS and CAS*, we identify all objects that are in contact with these surfaces. We then recursively identify all the objects that are in contact with the objects thus far identified. Next, we identify their categories as well to retrieve placement constraints associated with each type of an object.
4. *Get activity surface constraints* (*ASC*): Activity surfaces are locations in the world where an agent may be located while interacting with an object. For every object in *OCON*, we identify whether it is an activity surface and an activity station, how many objects of the type are allowed in a particular activity station (*as*) (i.e., the object per station cardinality constraint), and whether it is a self-category attractor. For example, a chair seat is an activity surface and an activity station, and a book is not. Typically, only one chair may be placed per activity station, but many books may be placed per activity station. Furthermore, a book attracts other objects of its own type. In addition, certain objects may be co-located within the reach of the same activity stations, while others may not. In this step, for each pair of objects in *OCON*, we establish whether they can belong to the same *as* attached to a different *one.*
5. *Get applicable placement constraints* (*PC*): For all pairs of objects in *OCON* including the agent and their relevant surfaces, we identify and retrieve all the constraints.
6. *Get scored activity stations (SAS*): The goal of this step is to identify activity stations as sub regions on activity surfaces. To compute activity stations, we place a grid of cells, called the *cas-grid,* over each *cas* in *CAS*. We control the *cas-cell* size in a *cas-grid* with a parameter. We use a smaller size for a finer degree of control for placement. However, smaller cell size exponentially increases the computation time. Next, for each object o^i in *OCON*, we place agent α at the object and vote on the cells in the grid as follows. First, if the cas-cell is located on an activity station (e.g., seat of a chair), then it gets and high positive vote (e.g., 5). Next, if a cas-cell is reachable from a voting object o^v in *OCON* that should not be in the same activity station as o^t, a large negative vote (e.g., -100) is cast on the cas-cell; otherwise, the cell gets a positive vote (e.g. +1). For example, a printer may not share the same activity station as a laptop. In that case, this approach ensures that a printer and a laptop are not placed together. Also, if o^v is of the same kind as o^t, and the cardinality constraint for the o^v for the station is 1, then it casts a large negative vote (e.g. -100) to prevent using the same activity station. We total the votes for each cas-cell in the grid into a *cas-cell-score*. Contiguous cells in a grid with a score > 0 comprise an *as*, which is added to the SAS. The magnitude of a vote indicates the *weight or influence of a particular type of constraint.* Figure 5a shows SAS computation in the office world. Light areas around the furniture show most probable activity regions. The dark areas show forbidden and low probability regions.
7. *Get scored placement regions* (*SPR*): A placement region, *pr,* is a region on a *cps* where o^t may be placed to satisfy as many constraints in *PC* as possible. We score the *pr* to obtain a scored placement region *spr* as follows (see Figure 5b): First, we place a grid on each *cps*, known as a *cps-grid*. Next, from each *cas-cell* belonging to *CAS*, we vote on all the cells (i.e. cps-cell), in the *cps*-grids. If a *cps-cell* is reachable from a *cas-cell* then the *cas-cell* votes on the *cps-cell* with its *cas-cell-score*. All relevant objects in *OCON* also vote on cps-cells based on their interaction constraints with the o^t that belong to PC. If a *cps-cell* is located on

an o^v, and as determined if, o^v and o^t are of the same category and, if the category is a self-category attractor for the given intent then the cps-cell gets a large positive vote (e.g., +20). For example, a book attracts a book to be placed together, or 'stacked'. Since many *cas-cells* can vote on a *cps-cell*, we retain the max vote value as the cps-cell score. The constraints are available in *PC* that we obtained in step 6. Please note that an object in *OCON* can cast a negative vote if it needs to prevent placement to maintain visibility. For example, a projector may cast a negative vote in front of its lens for every object other than a projection screen. All contiguous *cps* cells with votes more than a certain threshold are collected as an *spr* and added to SPR. This process is repeated till all the contiguous regions have been identified. Figure 5b shows SPR visualization. Bright green areas show SPRs with high scores and dark green and orange depicts areas with negative scores.

8. *Get the geo-orientation* (*go*): We treat getting the geo-orientation as an independent task from identifying a *spr*. We use the surfaces of o^t and examine their constraints in *PC* in order to select a preferred geo-orientation. Using this, we set the geo-orientation of the target object surface. For example, the preferred geo-orientation of a book's cover could be facing up so it can be seen. This is set as one of the orientation components in Ω.
9. *Get the interactive orientations* (*io*): The valid interactive orientations of our target object's surfaces are those that conform to various orientation constraints such as visibility and readability contained in *PC* and applied from the *as* associated with each *spr*. To obtain an interactive orientation (*io*) at an *spr*, we begin with an orientation in the horizontal plane that aligns the axis of an object to the edge of the placement surface and incrementally rotate it (e.g., 45 degrees) and check for the applicable constraints with objects in OCON. Whenever a constraint is not met, that particular orientation is given a penalty based on the constraint's priority (low, medium, or high). Once all possible orientations are scored the one with the smallest penalty score is selected as *io* and added to complete the omega.
10. *Get a best placement* (*p*): We select the SPR with the highest score and its corresponding best orientation at the placement

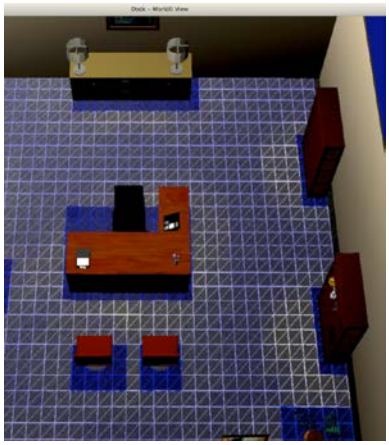

Figure 5a. SAS visualization

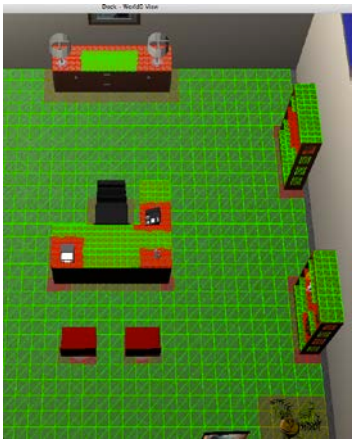

Figure 5b. SPR visualization

3. Evaluation

Our *objective* was to evaluate the placement performance of OPOCS and compare it to a naïve placement approach for resolving pragmatic underspecification of spatial prepositions. We present our evaluation approach and the results in the subsections below.

3.1 Data Sets

We created four unique single-room office world data sets for testing. These included: i) Manager Office, ii) Executive Office, iii) Home Office, and iv) Satellite Office. The office worlds contained the same 13 types of objects typically found in offices (see Figure 6). For instance, office desks, chairs, and shelves as furniture, office equipment such as computers and whiteboards, and decorative items such as flower vases and pictures. Despite having the same semantic types, the objects varied across worlds in terms of their shapes and sizes. For instance, different desk models were used across the worlds, but all had a semantic term 'desk'.

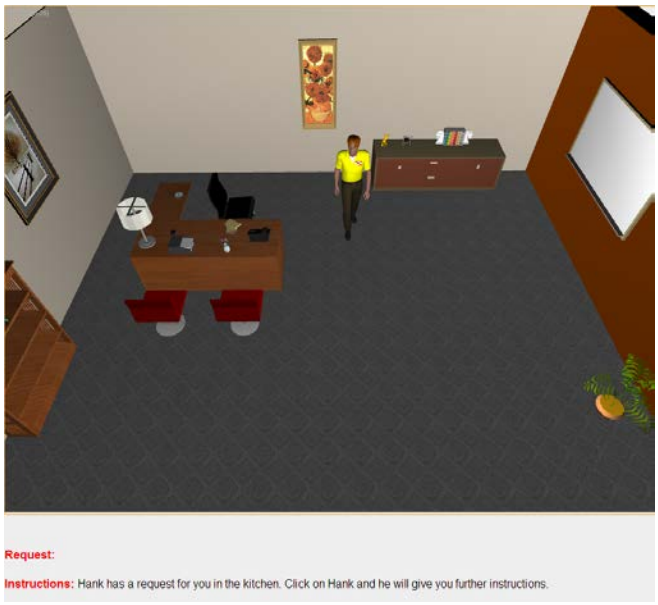

Figure 6. Example office world displayed in IDEA

We varied the placement and the layout of objects as well so that they imposed a different set of placement constraints in each world. In each world we created a set of 74 test cases across 10 spatial relations. Table 1 shows example test commands. The spatial relations we evaluated comprised the following four categories:

1. *Functional spatial prepositions*: These included the English prepositions "in" for containment and "on" for support and placement functions.

2. *Projective spatial prepositions*: These include projective prepositions in the horizontal plane left, right, in front of, and behind, and projective prepositions in the vertical dimension over and under (in the back of)
3. *Proximal or distal prepositions*: This included near
4. *Configurable preposition*: This included beside.

We checked the test cases were checked for reasonableness of request in the environment.

Table 1. Example test commands

Put the printer to the right of the microwave
Put the filled cup under the shelf
Put the notepad in front of the sofa
Put the picture behind the table
Put the clock over the whiteboard

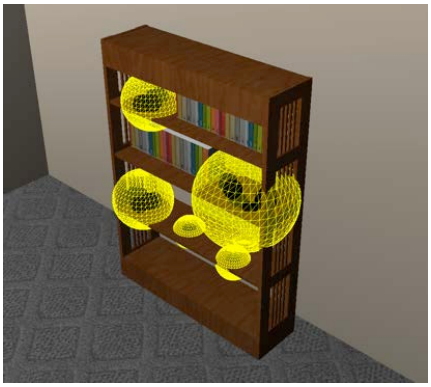

Figure 7. Visualization of multiple WPR for a test case.

For each test case, we created weighted ground truth placement using crowd sourcing. We developed a web-based Integrated 3D Environment for Annotation called IDEA (see Figure 6). We presented each test case to 20 independent annotators. IDEA presented placement commands visually by means of a speech bubble associated with a request avatar. The annotators then used their keyboard and mouse to place and orient target objects in response to a NL placement command. Prior to doing the actual assignments, users had the opportunity to learn and practice using IDEA for placement annotation with an online tutorial. We created the placement ground truth, which included a set of valid placement regions with preferred orientations. Each placement region had a score. This is a marked departure form the majority of existing evaluation approaches where annotators must agree on a single correct response (i.e., ground truth) to a test case. Although a single solution may be appropriate for classification problems, it is not so for a synthesis task such as planning or design. For the object placement task multiple solutions are

practically acceptable. Therefore, we admitted multiple possible answers and clustered them for probability instead. This also eliminated the issue of inter annotator agreement that arises from unrealistic admittance of only one solution.

We computed the *weighted placement regions* (WPR) using agglomerative clustering (Clust, 2013). We compute the weight of a cluster i w_i as follows:

$$w_i = \#a_i / \max(a)$$

where, $\#a_i$ is the number of annotations contained in the cluster and max(a) is the maximum number of annotations over all the clusters. Therefore, the weight of a cluster is [1,0). Similarly, we clustered the orientations into a weighted orientation angle (WOA). WOA weight computation was the same as WPR. Figure 7 shows multiple WPRs for a test case using yellow globes.

Many of our test cases had more than one WPR and WOA. We hypothesize that the number of WPRs is an indicator of task difficulty and underspecification for a particular spatial relation. Our analysis shows that functional prepositions (i.e., in and on) are the most underspecified (7 and 6 WPRs respectively), followed by distal or proximal prepositions (2-4 WPRs). The projective prepositions are the least underspecified. By the same token, the positioning task is harder than the orientation task (5 vs. 2.5) clusters.

3.2 OPOCS Knowledge Base

The OPOCS knowledge base comprised 39 types of objects typically found in offices. These ranged from books to white boards. Table 2 shows representations data and constraints meta data for three objects. Regarding the number of spatial constraints in the knowledge base, the orientation constraints were the most frequent, followed by visibility, and the distance constraints.

Table 2. Example representation records

Properties	Wall Clock	Office Desk	White board
Support	No	Yes	No
Containment	Yes	No	No
Installable	Yes	No	Yes
Activity Station	No	No	No
Attracted Objects	0		
Preferred Height	Eye	Floor	Eye
Constraint Types	Orientation, visibility, distance	Orientation	Orientation, and visibility

3.3 Measures

We assessed the object placement performance with the following three measures:

1. *Position Accuracy* (PA): This measures the accuracy of placement by comparing it with WPRs. If the computed placement is contained in one of the WPRs then it gets the score of the WPR it is contained in or else it gets 0.

2. *Orientation Accuracy* (OA): This measures the orientation accuracy by comparing the orientation generated by a placement planner with WOA. If the orientation is contained in one of the WOAs it gets the weight associated with WOA as its score otherwise it is 0.
3. *Computation Tim*e: This is the computation time in milliseconds.

3.4 Test Execution Environment

We ran the test cases using our CoASTeR testbed to evaluate algorithms that perform complex spatio-temporal reasoning tasks such as a placement planning in response to a natural language command. CoASTeR (Communicative Agents for Spatio-Temporal Reasoning) Workbench includes a multi-agent simulation server with 3D visualization and a Testbed. The Testbed can automatically load test case scripts, execute them against one or more reasoning agents, and compare their performance against specified metrics. We ran all the test cases against OPOCS and a naïve Baseline (BL) algorithm. The BL algorithm randomly selected a candidate surface with a random placement and orientation.

3.5 Results

Table 3 shows the *position accuracy* of OPOCS vs. BL across all the 4 sets on 10 relations. It shows that OPOCS significantly outperformed BL (0.29 vs. 0.0008, $p < 0.01$). Performance on functional prepositions (i.e., the most underspecified) ranges from 0.08 (Home) to 0.67 (Executive) with an average 0.37 for "in", and 0.1 (Home) to 0.51 (Manager) for on with an average of 0.34. The performance on the projective prepositions ranged from 0 (Front in Home) to 0.8 (Behind in Manager). Across the data sets, on average, it ranged from 0.17 (Front) to 0.54 (Over). The performance on distal relation near ranged from 0 (Home) to 0.14(Manager) with and average of 0.12.

Table 3. OPOCS vs. BL Position Accuracy Performance on Office Worlds

Relations	Manager		Executive		Home		Satellite		Mean	
	OPOCS	BL	OPOCS	BL	OPOCS	BL	OPOCS	BL	OPOCS	BL
Behind	0.80	0	0.73	0	0.00	0	0.20	0	**0.43**	0
Beside	0.63	0.09	0.38	0	0.04	0	0.55	0	**0.46**	0.025
Front	0.17	0	0.23	0	0.00	0	0.26	0	**0.17**	0
In	0.42	0	0.67	0	0.08	0	0.31	0	**0.37**	0
Left	0.19	0	0.33	0	0.11	0	0.22	0	**0.21**	0
Near	0.14	0.05	0.19	0	0.00	0	0.16	0.09	**0.12**	0.03
On	0.51	0.03	0.57	0	0.10	0.03	0.17	0	**0.34**	0.01
Over	0.75	0	0.13	0	0.50	0	0.75	0	**0.54**	0
Right	0.51	0.13	0.33	0	0.07	0	0.13	0	**0.26**	0.03
Under	0.17	0.17	0.17	0	0.33	0	0.19	0.	**0.21**	0.04
Mean	**0.43**	**0.04**	**0.37**	**0**	**0.12**	**0.003**	**0.29**	**0.009**	**0.29**	**0.00008**

The variation in the test worlds caused significant variations in the performance. The Satellite and Home office worlds were more difficult than Manager and Executive worlds as reflected by their

mean scores across all relations; 0.43 and 0.37 vs. 0.12 and 0.29 respectively. This difficulty was in part caused by multiple instances of landmarks such as lamps in the Home and satellite offices that caused landmark referential ambiguities. This was compounded by markedly different arrangements. A close examination of 0 valued performance cases revealed a number of near misses, which we judged as acceptable placements, but our metric assessed them as 0.

Table 4 shows the *orientation accuracy performance* of OPOCS vs. BL. On average OPOCS significantly outperforms BL on orientation accuracy (0.66 vs. 0.15 $p < 0.01$). The orientation accuracy ranges from 0.26 (Over in Manager) to 0.93 (Behind in Executive). Clearly, the orientation performance of OPOCS is substantially higher than the placement performance because the orientation task is relatively easier. Also it is noteworthy that there is little variation in orientation performance despite variations in the data sets.

Table 4. OPOCS vs. BL Orientation Accuracy Performance

Relations	Manager		Executive		Home		Satellite		Mean	
	OPOCS	BL	OPOCS	BL	OPOCS	BL	OPOCS	BL	OPOCS	BL
Behind	0.81	0.11	0.93	0.29	0.81	0	0.81	0.14	**0.84**	0.13
Beside	0.68	0.02	0.74	0.05	0.65	0.26	0.69	0.19	**0.71**	0.12
Front	0.73	0.17	0.74	0.06	0.71	0.17	0.59	0.26	**0.69**	0.16
In	0.70	0.16	0.70	0.05	0.81	0.05	0.62	0.16	**0.71**	0.11
Left	0.54	0.23	0.60	0.05	0.59	0.14	0.60	0.15	**0.58**	0.14
Near	0.53	0.29	0.56	0.07	0.59	0.20	0.68	0.21	**0.59**	0.19
On	0.58	0.13	0.89	0.17	0.87	0.21	0.61	0.40	**0.74**	0.22
Over	0.26	0.25	0.27	0.02	0.26	0.25	0.28	0.25	0.27	0.19
Right	0.61	0.19	0.75	0.14	0.73	0.13	0.56	0.13	**0.66**	0.15
Under	0.81	0.01	0.83	0	0.72	0.19	0.73	0.02	**0.77**	0.05
Mean	**0.63**	**0.15**	**0.70**	**0.09**	**0.67**	**0.16**	**0.62**	**0.19**	**0.66**	**0.15**

The mean computation time for OPOCS was 0.9 seconds on a Macbook pro. This was deemed acceptable in our application. The computation time varied across different relations and appears to be correlated to the number of WPRS, which is consistent with our expectations.

4. Related Work

Bateman et al (2010) introduced a spatial ontology layer that considerably refines and extends conventional lexical semantics with numerous task-based features. Their ontology contains spatial modality features such as distance, direction, path and motion. Although this is useful as a semantic knowledge when utilized in a semantic parser, this approach will still result in pragmatic ambiguity. In contrast, we focused on representation of functional knowledge for recovery of underspecified distance and orientation constraints. We rely on incorporating the contextual information via world knowledge along with relatively simple lexico-semantic representations, similar to Roy et al. (2005).

Wordseye (Coyne & Sproat, 2001) performs scene construction (i.e., object placement) via natural language commands. Wordseye includes a knowledge base of functional interactions

between human agents and object using a notion of spatial tags. Their scene construction or depiction process exploits handcrafted rules along with spatial tags to place objects. Like Wordseye, our approach uses a knowledge base of function interactions to generate constraints, albeit with widely different representations. In particular, we represent orientation and distance constraints between object surfaces and agents using functional relations, activity stations and their cardinalities. We then use a human activity simulation to satisfy contextual constraints in least cost manner, which Wordseye does not. Furthermore, unlike Coyne and Sproat, 2001, we present and extensive evaluation of our approach across 10 static relations.

Kelleher *et al.* (2009) presents a spatial reasoning approach for transforming a qualitative semantic representation of locative expressions into a geometric representation. They present computation models for proximal and projective prepositions inspired by potential fields. They perform spatial reasoning to aggregate the effects of contextual or distractor objects to find the optimal fields. Their potential field aggregation approach is similar to OPOCS compute SPR step. However, OPOCS differs significantly from their approach in the following aspects. First, OPOCS uses a simple unweighted potential field but not only uses distance interaction constraints but also orientation constraints between objects using specific object categories. In contrast, Kelleher only uses object attributes to compute fields. Second, OPOCS uses the notion of activity stations and implicitly simulates human interactions with the environment, which Kelleher *et al.* do not. Kelleher *et al.* (2009) evaluate their computational models in a 2D environment with simple geometric shapes and disregarding function. In contrast, we evaluate our placement approach in a 3D environment with functional objects.

Jiang et al (2012) present an approach for learning to place new objects in real world. Their approach includes knowledge of real world placement preferences and strategies such as stability, and stackability. They use an integer programming approach with linear programming relaxation to find least cost placements. This is, in principle, similar to OPOCS that exploits human placement preferences via default intentions and candidate surface heights. However, OPOCS uses a heuristic approach with simulation to find candidate placement. Our work also differs in objective that it is directed to resolving underspecification spatial commands. One important difference between OPOCS and their work is that OPOCS has only been applied to virtual environments where it has complete encoded knowledge of objects in the environment. In comparison, Jiag et als' placement algorithm learns placement knowledge and performs placement in noisy and real environments with a real robot.

Pandey and Alami (2010) present an approach for computing weighted convenience maps. The maps called Mightability maps contain spatial information about reachability and visibility for use in human robot cooperative tasks. Their approach amounts to simulating perspective taking to detect what would be reachable and visible to a human. In OPOCS, a very similar computation is performed in Step 7 (get Scored Placement Surface). In addition, in OPOCS, we utilize the knowledge of when and where to apply visibility and reachability constraints for spatial planning. Furthermore, we use the same to search for relevant activity stations.

Cosgun *et al.* (2011) present a planning algorithm for placing objects on a table surface. Unlike OPOCS, their focus is on rearranging (i.e., pushing) existing objects to place a new object. They evaluate their push planning approach on 2D maps of objects. In contrast, OPOCS is focused on placing objects with linguistics commands on available spaces. Kurup and Cassimatis (2010) present a qualitative spatial reasoning approach that can be used for spatial planning. However, their approach is qualitative and limited to 2D worlds.

5. Conclusions

Developing robust and accurate approaches for interacting with autonomous agents using NL is an active area of research and so is developing lexical representations and ontologies of spatial planning in 3D environments. However, polysemy and vagueness in language combined with the continuous nature of realistic 3D environments make this a computationally challenging task. In this paper, we introduced a new layer of linguistic ambiguity beyond the conventional semantic ambiguity called pragmatic ambiguity and underspecification. We then proposed and constrained spatial planning approach for resolving pragmatic underspecification. We presented OPOCS that represents and exploits functional knowledge of objects and human activities to recover unspecified pragmatic and contextual constraints to effectively place objects. Our evaluation of OPOCS on 4 office worlds over 10 representative spatial terms demonstrated that it significantly out performs a naïve baseline. We also introduced a novel weighted multi-ground truth computation approach for test data using crowd sourcing. This is a significant departure from conventional evaluation approaches where a single ground truth is considered. We argue that, for spatial planning, which is a type of synthesis task, multiple ground ranked truths must be considered as we did in OPOCs.

OPOCS and our evaluation are not without limitations. First, we evaluated OPOCS in worlds with around 40 objects. However, placement tasks only concerned a dozen types of objects. Although, our objects and placement tasks were realistic their variety was not large enough to test the scalability and robustness of OPOCS. Our future evaluations will consider a much larger set of objects in the world and the placement tasks. Algorithmically, OPOCS only used hand-engineered knowledge and assumed complete world knowledge. In future, we will develop approaches for learning placement knowledge and the impact of degrading the world knowledge on OPOCS placement performance. We are presently working on case-based approaches for learning object placement knowledge and heuristics. We intend to utilize the evaluation data obtained from crowd sourcing for learning. Given the complexity of evaluation we limited ourselves to a small set of representative static spatial terms. Our future work will extend representation of dynamic and configurable spatial terms such as along, across and between. Finally, we will explore application of OPOCS to real environments.

Acknowledgements

This research was funded by Office of Naval Research. We thank the reviewers for helpful comments and suggestions.

References

Bateman, J.A., Hois, J., Ross, R., and Tenbrink (2010). A linguistic ontology of space for natural language processing, Artificial Intelligence, Vol 174, pp. 1027-1071.

Baylog J.G., Wettergren, T.A., Hyland, J.C., & Smith, C.M, (2008). Robust Search for Structured Object Placement Using Unmanned Vehicles, OCEANS 2008: IEEE

Bergen, B. (2005) Mental Simulation in Spatial Language Processing, Proceedings of the Twenty-Seventh Annual Conference of the Cognitive Science Society.

Clust(2013). Hierarchical Clustering, Retrieved from http://en.wikipedia.org/wiki/Hierarchical_clustering on 10 August 2013.

Cosgun, A., Hermans T., Emeli, V., and Stillman, M. (2011). Push Planning for Object Placement on Cluttered Table Surfaces, IEEE/RSJ Conference on Intelligent Robots and Systems (IROS), San Francisco

Coventry, K.R., Carmichael, R. & Garrod, S.C. (1994). Spatial prepositions, object-specific function and task requirements, *Journal of Semantics*, 11, 289-309.

Coyne, B., & Sproat, R., (2001) WordsEye: An automatic text-to-scene conversion system, SIGGRAP'01, *Proceedings of the 28th annual conference on computer graphics and interactive techniques*, pp. 487-496, New York, NY: ACM.

Dupuy, S. (2001). Generating a 3-D simulation of a car accident from a written description in natural language: The CARSIM system. *Proceedings of the Workshop on Temporal and Spatial Information Processing*, pp.1-8.

Egg, M., (2010) Semantic underspecification: concepts and applications, Language and Linguistic Compass, Vol 4(3) pp 166-181

Frison, S. (2009). Semantic Underspecification in Language Processing, Language and Linguistic Compass, Vol 3 (1). pp. 111-127

Gibson, K.J. (1977). The Theory of affordances. In R. Shaw and J Bransford (Eds.), Perceiving, acting, and knowing: Toward and ecological psychology(pp. 67-82). Hillsdale, NJ: Erlbaum.

Gupta, K.M, Schneider, A., Klenk, M., Gillespie, K., Karneeb, J. (2011). Representing and Reasoning wth Functional Knowledge for Spatial Language Understanding. Workshop on Computational Models of Spatial Language Interpretation-2, Boston, MA: CogSci-2011

Herskovits, (1986) Language and spatial cognition, Cambridge University Press, Cambridge, MA

Jiang, Y., Lim, M., Zheng,C., & Saxena, A. (2013) Learning to Place New Objects in a Scene, International Journal of Robotics Research (IJRR).

Kelleher, J. and Costello, F. (2009). Applying computational models of spatial prepositions to visually situated dialog, Computational Linguistics, Vol 35, No.2. pp. 271-306

Kurup, U., & Cassimatis, N.L., (2010). Quantitative spatial reasoning for general intelligence, *Proceedings of the Third Conference on Artificial General Intelligence Conference*, pp. 1-6, Lugano, Switzerland: AGF.

Lockwood, K. (2009) Using Analogy to Model Spatial Language and Multimodal Knowledge Capture, PhD Dissertation, Northwestern University, Department of Computer Science, Evanston, IL

Norman, D. (2002). The design of everyday things, New York, NY: Basic Books.

Pandey, A.K. & Alami, R. (2010). Mightability maps: A perceptual level decisional framework for co-operative human-robot interaction. In the proceedings of the IEEE Conference on Intelligent Robots and Systems (IROS)

Regier, T., & Carlson, L. (2001). Grounding spatial language in perception: An empirical and computational investigation. *Journal of Experimental Psychology: General, 130*, 273-298.

Roy, D. (2005) Grounding words in perception and action: computational insights, Trends in Cognitive Sciences, Vol 9. No 8.

Talmy (2000). How Language Structures Space, in *Toward a Cognitive Semantics-Vol.1,* MIT Press, Cambridge, MA.