Grounding Language in Spatial Routines

Stefanie Tellex and Deb Roy
MIT Media Lab
Cambridge, MA 02141
stefie10@media.mit.edu dkroy@media.mit.edu

Abstract

This paper describes a spatial language understanding system based on a lexicon of words defined in terms of *spatial routines*. A spatial routine is a script composed from a set of primitive operations on sensor data, analogous to Ullman's visual routines. We hypothesize that a set of primitives that underlie spatial language could be used to succinctly express the meaning of spatial terms in a way that can be used to interpret natural language commands. This hypothesis is tested by using spatial routines to build a natural language interface to a real-time strategy game, in which a player controls an army of units in a battle. Spatial routines are used as a top down control mechanism for the system, providing a mapping from natural language commands to system behavior. Although the current implementation has significant limitations as revealed in an initial evaluation, we believe that the concept of spatial routines holds promise as a way to ground spatial language semantics in terms of embodied sensory-motor interaction.

Introduction

Spatial competence is a central aspect of human intelligence, and a crucial component of any system that needs to intelligently move itself or objects in the world. Spatial language is a window to spatial cognition: humans use spatial language to describe spatial situations, to refer to objects, and to request other humans to take actions. We are interested in developing computational models of the structures and processes that underlie spatial language interpretation.

This paper describes a system that tries to obey natural language commands about unit movement in a real-time strategy (RTS) game. In the RTS game used in this work, the player uses a mouse and keyboard interface to control an army in a battle against a computer opponent. When speaking to another human in order to control units in an RTS game, a human commander uses quite complicated language including conditional expressions, negatives, and statements about higher level strategy. In order to simplify the domain, we focused on simple commands that specified a set of units to move and a target location such as "have everybody go below the lake". Our system grounds the words in its vocabulary in terms of a lexicon of words defined in terms of *spatial routines*, analogous to the visual routines described by Ullman (1983). A spatial routine is a script composed of a set of primitive operations on data extracted from the RTS game state. The system is evaluated using a corpus collected by recording the commands that one human issued to another who was playing the game using a keyboard and mouse interface.

The system presented in this paper is controlled by a top-down architecture: natural language commands are converted to actions in the game via a lexicon of words defined in terms of scripts of primitive operations. This architecture makes the mapping between language and action straightforward, but makes it harder to express actions the system could take on its own.

Related Work

The system presented here uses a lexicon of words defined in terms of a set of primitive spatial operations to obey natural language commands. Wierzbicka (1996) defines a set of linguistic semantic primitives, searching for a set of words that have lexical forms in every language, and which can be used to define any word in any language. Our work moves below the linguistic level of meaning, resulting in definitions that can be used to obey natural language commands. The field of cognitive semantics and others have described sets of sub-linguistic primitives that can be used to encode the meaning of words. (Talmy 2005; Jackendoff 1985) Their approaches operate at the level of abstract schemas whereas ours operates at a more concrete level of sensory-motor programs.

Ullman (1983) suggests a set of primitive operations that combine to form visual routines which process images to perform visual tasks. His goal is to find a flexible framework for visual processing that could be used to explain the wide variety of human visual competencies. The primitives operate on an image bitmap and points within the map. For example, the *indexing* primitive finds locations

with particular features such as color or line orientation; humans do something analogous when they find odd-colored objects in a group of other objects. The *coloring* operation spreads activation in the base representation, stopping at boundaries, and can be used to identify whether a point is inside a bounded region. The compositional nature of Ullman's primitives suggests applying a set of primitives in order to define the meaning of linguistic terms. Since language is compositional, defining words in a compositional substrate yields a mechanism for combining word meanings in phrases and sentences.

Rao (1998) presents the "Reverse Graphics" system which uses a visual routine-based architecture to process black and white images. This builds on Ullman's ideas by refining the set of primitives and adds a concrete implementation. Rao focuses on using a set of primitives to develop algorithms that process and label video data. In contrast, the current work describes a system that understands natural language commands in a game environment.

The mobile robot community has created systems that can understand natural language commands. Much of this work focuses on spatial language to control the robot's position and behavior, or to enable it to answer questions about what it senses. In general, previous work in this area has focused on developing various command sets for mobile robots and robotic wheelchairs, without directly addressing aspects of language that are context sensitive. (See (Skubic *et al.* 2004; Pires & Nunes 2002; Yanco 1998; Gribble *et al.* 1998)). In our previous work, we created a spatial-routines based system that obeyed natural language movement commands by using the environment to plan a context-sensitive trajectory based on available pathways. (Tellex & Roy 2006) For example, if the robot was in an empty room with a doorway off to its left, it would go left through the doorway, while if it was in a hallway approaching an intersection, it would go forward and to the left.

Many have collected corpuses in order to drive the implementation of systems that obey natural language commands in a simulated or real-world environment. (See (MacMahon, Stankiewicz, & Kuipers 2006; Bugmann *et al.* 2004; Gorniak & Roy 2004).) Bugmann *et al.* (2004) and MacMahon, Stankiewicz, & Kuipers (2006) do not describe an end-to-end evaluation on a held out test set. Both systems mapped natural language utterances to a static set of motor commands. Spatial routines, in contrast, define words in terms of operations on sensor data, which then compose to create a plan for motor action. Gorniak & Roy (2004) created a system that uses natural language instructions to select objects in a scene. In contrast to the current work, their system only supports selecting objects in a simple environment, not moving objects in a richer domain.

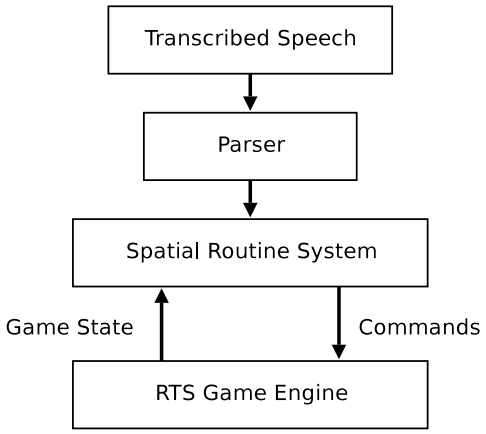


Figure 1: The architecture of the language understanding system.

System Architecture

The spatial language understanding system described in this work consists of a parser created by Gorniak & Roy (2006), using a custom grammar. The parser finds the largest grammatical island in the input sentence, and sends the output the spatial routines system for evaluation against the current game state as reported by the RTS game engine. The spatial routine system then sends the results of that evaluation to the RTS game engine in the form of a set of units and a location to which they should be moved, and the game engine executes the command. The system architecture is shown in Figure 1.

For example, if the user says "Move the flamethrowers on top to the left of the marines," the parser creates the following representation:

```
go(on(flamethrowers(),top()),
   of(left(),marines()))
```

This expression is evaluated by the spatial routines system, and the result is a set of units and a goal point to which they should move. These are converted to commands for the RTS game engine to execute. Because words in the lexicon are defined in terms of scripts that operate on information from the game state, commands can be obeyed in a context sensitive way, and the same command can lead to different behavior in different situations.

Words in the system's lexicon are defined as scripts of primitive operations that run on data extracted from the game state at the time the command was issued. Game state used by the system includes unit locations, map visibility, unit ownership, and terrain information. The primitives are defined as functions that take grids, points, or numbers as arguments. Grids can be either numeric grids, masks, which take boolean values, or regions, which are masks with a sin-

gle continuous area selected. A subset of the primitive operations is shown in Figure 2, and the complete list can be found in Tellex (2006).

Lexicon of Words Defined as Spatial Routines

The system uses a lexicon of words defined in terms of these primitives in order to obey natural language commands. Each entry in the lexicon consists of a symbol and an associated script composed from the set of primitives. The notation $Subscript(argnames)$ is used to specify subscripts, which is like a lambda expression, a procedure that takes the specified arguments, and when called returns a result. Some spatial routines in the lexicon return a subscript as their result. Sample entries in the lexicon are shown in Figure 3; detailed information about the lexicon can be found in Tellex (2006).

Corpus Collection

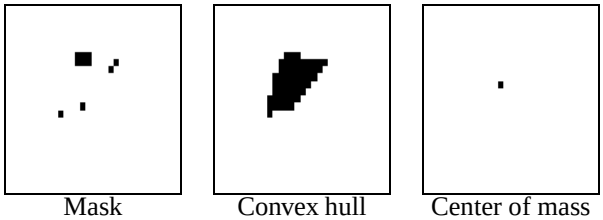
Normally a person who plays an RTS game discovers the game state through the game interface, makes strategic decisions based on this information, and then implements those decisions using a keyboard and mouse to control their units. In order to collect data to design and evaluate our system, these tasks were split between two people, who used language to coordinate their actions. One participant, the *commander*, watched the game unfold and gave verbal instructions to the interpreter, telling him how to play the game. The other participant, the *interpreter*, followed those instructions using the keyboard and mouse interface.

Using this paradigm, data was collected from fourteen games, averaging 6 minutes and 30 commands each. Each game had a different commander, but the same person served as interpreter across all games in order to ensure a standardized interpretation of the commands that participants issued. Before recording any sessions for the corpus, the interpreter played through the game on his own, and recorded a test session as a commander.

The game runs on an RTS game engine called Stratagus. [1] For our purposes, a custom game was developed based on one of the games written using Stratagus, Battle of Survival. [2] The game designed for this evaluation has three types of units: marines, assassins and flamethrowers, balanced as in the game "Rock Paper Scissors": marines easily defeated assassins and were defeated by flamethrowers; assassins easily defeated flamethrowers but lost to marines, and flamethrowers beat marines but lost to assassins. Players controlled 30 units total: 10 of each unit type, and were told how the units were balanced. In addition, each unit in the game was labeled with one of the words "rock", "paper" or

[1] http://stratagus.sourceforge.net

[2] http://bos.eul.org



ConvexHull(mask) Returns the convex hull of a mask. A visualization of the primitive's operation is shown above.

CenterOfMass(mask) Returns the center of mass of a mask, computed by averaging the points in the mask. A visualization of the primitive's operation is shown above.

ColorRegion(region, grid, function(grid, point)) Calls a function on each point in a region. Writes the output of the function to an output grid.

TraceLine(start_point, direction, function(grid, point)) Calls a function on each point in a ray. Writes the output of the function to an output grid.

UnmaskCircle() Returns a region unmasked around a specified point and a specified radius.

Max/Min(grid) Returns the point where a grid takes on its maximum/minimum value.

Distance(p1, p2) Computes the distance between two points.

Divide(grid1, grid2) Divides one occupancy grid by another.

MaskUnion/MaskIntersect/InvertMask(mask1, mask2) Functions to manipulate masks.

IndexMask Returns a list of regions, one for each connected component in the input mask.

ClosestRegion(listOfRegions, point) Returns the region closest to a point in a list.

ScoreRegion(listOfRegions) Sorts regions according to a function.

Direction(grid, point) Returns the height of each point in a direction.

Angle(point1, point2) Returns the angle between two points.

AverageDirection(path_grid) Finds the average direction of a path grid.

Figure 2: A subset of the primitive operations used to define words in the system's lexicon, and a visualization of their operation.

go (arg1, arg2)

if $arg2$ is passed

if $arg1$ is a Numeric Grid

$goal = arg1$, $subject = arg2$

if $arg2$ is a Numeric Grid

$goal = arg2$, $subject = arg2$

else $goal = arg1$, $subject = selectedunits$

Now there are variables for $subject$ and $goal$.

$subject = UnmaskIntersect(subject, myunits)$

#(because it is only possible to move units the player controls.)

$subject\_com = CenterOfMass(subject)$

if $goal$ is a function

$goal\_point = Max(goal(subject))$

if $goal$ is a Grid

$goal\_point =$
$closest\_point(max\_region(goal), subject)$

if $goal$ is a Point

$goal\_point = goal$

$Select(subject)$

return $goal\_point$

direction(direction, target) This is used to define north, south, left, right, top, bottom, etc. The definition for each of these words in the lexicon passes the vector corresponding to that word to this function.

$grid$ = Assign each point the value of the direction function. (Uses ColorRegion primitive)

if $target$ is not passed

return $grid$

if $target$ is GridMask (e.g., "The north marines")

Find all regions in the mask.

Compute the center of mass of each region.

Score each region based on the grid's value at the region's center of mass.

Return the highest scoring region.

towards(destination)

$script = Subscript(subject)$

if destination is a Grid

$destpoint = Max(destination)$

else

$regions = IndexMask(destination)$ Find all regions in the mask.

$region = ClosestRegion(regions)$

$destpoint = CenterOfMass(region)$

$subjectpt = CenterOfMass(subject)$

$angle = Angle(subjectpt, destpoint)$

return $direction(angle)$

return $script$

Figure 3: Pseudocode for a subset of the lexicon used in the system.

"scissors." Participants played against an enemy army consisting of the same units. The enemy was controlled by a simple algorithm that waited for an attack, and then moved nearby units to defend the units under fire. The object of the game was to destroy all enemy units, and the game ended as soon as one of the armies was completely destroyed.

When playing the game, commanders watched a screen showing the game view and mouse movements of the human interpreter. They were asked not to touch the keyboard and mouse at all. Instead they spoke through a microphone to the interpreter, who was playing the game using a conventional keyboard and mouse interface in another room. The interpreter could not talk back to the commander. Instead, he communicated only via the game interface.

The speech corpus was segmented using a pause-structure based segmenter, then transcribed with an internally developed speech transcription tool. The speech segmenter created a speech event marking each place that it detected speech within the audio stream. When transcribing the speech, we sometimes modified the automatically created speech events, splitting and merging them so that each logical command was contained in its own event. Testing sessions were not transcribed until all development was complete. Detailed statistics about the corpus can be found in Tellex (2006).

Evaluation Procedure

A simple algorithm was developed to provide a baseline for system performance. We created an algorithm that takes fixed action based on spotting certain keywords in the command. After parsing the command, the semantic representation is searched for the strings "north", "south", "east", "west", and other direction words such as "left", "right", "top", and "bottom". A direction, represented as an angle, is generated from a lookup table based on the presence of one of these words. The units to be moved are the currently selected units, if any are selected. If no units are selected, it moves all units. It moves the units half a screen length in the specified direction.

The evaluator rated the performance of the baseline and spatial routines systems for commands in the testing portion of the corpus. For each command, the evaluator saw the command text and the state of the game when the command was issued. Then he watched as the spatial routine or a baseline system tried to obey it. Commands were shown in random order, and each command was shown twice, once for the spatial routines system and once for the baseline. Both the baseline and routine systems selected the units they planned to move, and moved the view port to center on those units. Then the units were moved towards the goal point. After the units started moving, the view port was moved again to center on the goal point. The evaluator rated the performance on a Likert scale from 1-7, with one labeled as "Less Ac-

	Training	Testing	Total
Keyword	66	16	82
Parser error	87	37	124
Routine error	18	23	41
Analyzed Commands	104	75	179
All Commands	275	151	426

Table 1: Shows the number of commands rejected from the evaluation by each rule, and the number of commands used in the evaluation.

ceptable" and seven labeled as "More Acceptable."

Corpus Filtering

Because many of the commands in the corpus were quite complex, some of the commands were excluded from the evaluation using automatic heuristics. For example, many interpreters asked the commander questions or issued conditional commands. Such commands are out of the scope of the problems the system is trying to solve, although it would be worthwhile to expand that scope in future work. Other commands used vocabulary not yet encoded in the system, but that could easily be added. Excluding these commands focuses the evaluation on how well the system works on vocabulary that was encoded. The corpus was filtered automatically using three rules. Table 1 shows the number of commands filtered by each rule. The first rule excluded a command if it contained the strings "if", "not", "don't", "follow", and "?". The second rule excluded commands that failed to parse. In practice this often acted as a vocabulary based filter, since the parser often failed to parse commands with unknown words. Finally if the command was parsable, the routines system attempted to obey it. If an error occurred during this process, that command was not included in the evaluation. Sometimes this happened when one of the words in the lexicon was called with incorrect argument. In other cases the game state made the command impossible to obey because the command did not make sense. E.g., the command "Move assassins to the right" was rejected from the evaluation because at the time this command was issued, all the assassins had been killed, so the routine system failed. Samples of commands excluded by each heuristic can be found in Tellex (2006). Only 42% of the commands in the corpus were included in the evaluation, mostly from parser and routine errors. This many were excluded due to the complexity of the language collected in the corpus and the fact that the lexicon of words was coded manually. A system that learns commands from training data could be more robust and could increase coverage of the data set.

Results and Discussion

The spatial routines algorithm performed only slightly better than the baseline on the training and testing data sets. The average ratings and significance values are shown in Table 2.

	Training		Testing	
	Baseline	Routine	Baseline	Routine
Mean	3.75	4.75	3.86	4.32
Std. Dev.	2.15	1.91	2.27	2.25
# Samples	104	104	75	75
Paired One-Tail T-Test	$P = 0.00002$ ($T = -4.14$)		$P = 0.07$ ($T = -1.49$)	
T-Test	($df = 65534$)		($df = 65534$)	

Table 2: Average ratings for the baseline and routine systems, on a scale from 1-7.

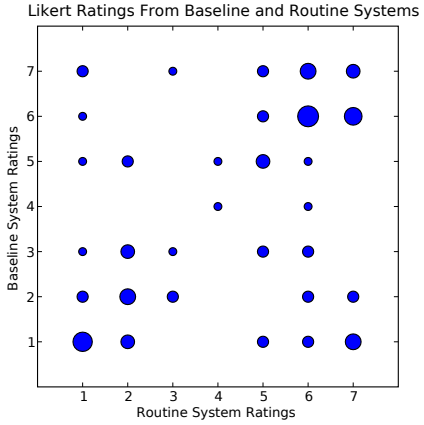


Figure 4: A scatter plot showing the annotator's ratings of each command in the test set. The area of the circle at each point is proportional to the number of commands at that point.

A one factor ANOVA among all four columns was significant at $P = 0.004$ ($F = 4.53$, $df = 3$).

Figure 4 shows commands from the test set, organized by the performance of the routine and baseline systems. Although the systems performed similarly for many commands (shown by the correlation between the baseline and routine ratings), there was a set of commands for which performance of the two systems differed. Table 3 shows a subset of these commands, where the difference in performance by the two systems was large. The commands that worked better on the baseline system than the routines system are all because of some kind of failure in the routine system, either parser error, caused by the presence of out of vocabulary words or words combined in unanticipated ways. The routine system worked well compared to the baseline on commands whose syntactic structure combined to form the meaning of the command, and where the parser was able to successfully extract that structure and send it to the routine system.

High for Baseline, Low for Spatial Routines		
Likert Score		Command
Baseline	Routine	
7	1	and go around to the left this big group of trees
7	1	and to the right to the green patch
6	1	have the marines attack move towards the right
5	1	and four flamethrowers attack these enemies
7	3	go back around the trees to the right

High for Spatial Routines, Low for Baseline		
Likert Score		Command
Baseline	Routine	
1	7	and since the enemy is here let's bring the flamethrowers this way also
1	7	take four assassins
1	7	back to the main group of guys
1	7	get four marines
2	7	move four of the flamethrowers to the left
2	7	have everybody go below the lake

Table 3: Commands where the difference between the baseline system's performance and the spatial routine system's performance was large.

Future Work

Although the spatial routines system successfully interpreted a range of spatial commands, as rated by a human, it only slightly exceeded the performance of a baseline system. The language collected in the corpus rarely contained deeply nested compositional structures such as "move the marines on top below the lake." where we expect spatial routines to perform much better than a baseline system. Instead it consisted of rich dialog expressing goals and conditionals. Perhaps by taking into account the context of the goal structure and history in addition to the game state would increase system performance. In addition, manually programming the lexicon led to it being too brittle, failing in unexpected situations and failing to adequately cover even the training set. This problem could be alleviated by exploring algorithms that automatically learn to create the lexicon of words based on training data and game state.

The spatial language corpus that we collected using video games provided a very difficult modeling challenge. The language interpretation system successfully interpreted a significant portion of the corpus despite many limitations of the current implementation. We continue to believe that spatial routines are a compelling means of capturing the complex semantics of spatial language although much further effort will be required to translate this concept into robust implementations.

Acknowledgements

We would like to thank Mike Fleischman and Piotr Mitros for their helpful comments on earlier drafts of this paper. All remaining errors are, of course, our own.

References

Bugmann, G.; Klein, E.; Lauria, S.; and Kyriacou, T. 2004. Corpus-based robotics: A route instruction example. In *Proceedings of Intelligent Autonomous Systems*, 96–103.

Gorniak, P., and Roy, D. 2004. Grounded semantic composition for visual scenes. *Journal of Artificial Intelligence Research* 21:429–470.

Gorniak, P., and Roy, D. 2006. Perceived affordances as a substrate for linguistic concepts. In *Twenty-eighth Annual Meeting of the Cognitive Science Society*.

Gribble, W. S.; Browning, R. L.; Hewett, M.; Remolina, E.; and Kuipers, B. J. 1998. Integrating vision and spatial reasoning for assistive navigation. *Assistive Technology and Artificial Intelligence: Applications in Robotics, User Interfaces, and Natural Language Processing* 1458:179.

Jackendoff, R. S. 1985. Semantics and cognition.

MacMahon, M.; Stankiewicz, B.; and Kuipers, B. 2006. Walk the talk: Connecting language, knowledge, and action in route instructions. In *Proceedings of the 21st National Conf. on Artificial Intelligence (AAAI-2006)*.

Pires, G., and Nunes, U. 2002. A wheelchair steered through voice commands and assisted by a reactive fuzzy-logic controller. *Journal of Intelligent and Robotic Systems* 34(3):301–314.

Rao, S. 1998. *Visual Routines and Attention*. Ph.D. Dissertation, Massachusetts Institute of Technology.

Skubic, M.; Perzanowski, D.; Blisard, S.; Schultz, A.; Adams, W.; Bugajska, M.; and Brock, D. 2004. Spatial language for human-robot dialogs. *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews* 34(2).

Talmy, L. 2005. The fundamental system of spatial schemas in language. In Hamp, B., and de Gruyter, M., eds., *From Perception to Meaning: Image Schemas in Cognitive Linguistics*. Mouton de Gruyter.

Tellex, S., and Roy, D. 2006. Spatial routines for a simulated speech-controlled vehicle. In *Human-Robot Interaction 2006*.

Tellex, S. 2006. Grounding language in spatial routines. Master's thesis, Massachusetts Institute of Technology.

Ullman, S. 1983. Visual routines. Technical Report AIM-723, Massachusetts Institute of Technology.

Wierzbicka, A. 1996. *Semantics: Primes and Universals*. Oxford University Press.

Yanco, H. A. 1998. Wheelesley: A robotic wheelchair system: Indoor navigation and user interface. *Assistive Technology and AI* 256–268.