

Beer on the Brain

Jonathan H. Connell

IBM T.J. Watson Research Center
30 Saw Mill River Road, Hawthorne NY 10532 USA

jconnell@us.ibm.com

Abstract

Speech can provide a valuable interface to a mobile robot, particularly as a simple user programming language that grounds out in whatever innate abilities the robot has. We illustrate the desired properties of such an interface via the example of getting a beverage from a refrigerator.

1. Introduction

Suppose you buy a general fetch-and-carry robot from Sears and take it home. It will come with all sorts of extraordinary innate abilities like collision avoidance, optimal navigation, object recognition, grasp planning, and visual search routines. However, to customize it to do such valuable tasks as serving beer, it needs to be told something about your environment. You might take it to the kitchen, show it the fridge, and point out where you keep the beer. You might then tell it what was expected of it in response to commands such as “Get me a Dos Equis”.

Note that the user is not doing heavy-duty programming at the level of setting joint torques or specifying convolution masks. In many ways this training is more like working with a simple macro scripting language – you define a few new items in terms of the system’s base types, and provide rewrite rules for translating commands into a parameterized sequence of built-in actions. While this is not the only way to customize a robot, nor the only use for natural language interaction, the two seem to be a good match.

2. Verbal Command Line

For control, forget dialog boxes and pull-down menus; think old-style command line interfaces. The point is to select a function and specify the arguments to it. Each utterance can be considered a complex function name based on the verb, with arguments sprinkled around it as noun phrases. For example, “Fetch a Foster’s for Fred from the fridge fast.” fits an explicit pattern “Fetch a <beer-type> for <person> from the fridge fast” which is associated with the function Beer-retrieval(<beer-type>, <person>). This sort of parsing is fairly common in natural language processing and works well for imperative statements.

Progress reporting is also important, but for our application it is even simpler. We simply use text-to-speech as a print statement. And should a function terminate abnormally, template-based speech can be used to signal exceptions, e.g. “Help, I’ve fallen in the <location> and can’t get up!”.

3. Programming and Learning

The essential part of conventional programming involves specifying functional abstractions which expand into sequential and conditional statements. This can be done straightforwardly by memorizing a goal-tagged series of statements such as: “To fetch a beer, first go to the kitchen and find the fridge. Then open the fridge and extract a bottle of beer. Finally, bring it to the requestor. Okay?”. Conditionals can be taught by treating “if” as a special keyword: “IF the fridge is empty THEN return to the requestor and announce ‘Outta luck, buddy boy’”.

Yet there are also datastructures. Naming new objects and places, and teaching the robot to recognize them may be even more important. Imagine a hybrid protocol where certain deictic references such as “here” or “this object” ground out through built-in visual primitives (e.g. where the human is wiggling his finger). One can then say “THIS object is a beer bottle”. The name of the item is taken to be whatever follows the words “is a”, namely “beer bottle”, and can simply be left as an uninterpreted tag (or even an audio sequence – there is no need to determine its written spelling). Descriptive terms can also be injected even before reification to indicate important segmentation constraints such as: “This BROWN object is a beer bottle”.

More generally, language is useful during learning because it allows one to direct the robot’s attention precisely and efficiently. Suppose one just stopped the robot and said “supporting shelf”. The robot now has to guess, first of all, whether some sort of command is being issued or a new item is being taught. It then has to figure out if “supporting shelf” is some sort of an action it supposed to learn, or if it is a spatial location, or even some kind of object. While pure induction might be able to sort out all these ambiguities given enough examples, the user is likely to get frustrated long before this occurs.

Suppose instead the user says “This LOCATION is a supporting shelf” thereby simultaneously denoting a spatial region and ascribing to it a previously known semantic type. From the type the robot can now infer that certain of the item’s properties are important, such as its position relative to some reference, whereas as other properties, such as its color, are likely to be irrelevant. In concert with the spatial bounds, this can greatly reduce the dimensionality of the description space and make induction far more tractable. Furthermore, critical features such as “It must be level horizontally” can be communicated directly, something that might be difficult (or at least tedious) to infer from examples alone.

4. Debugging and Repair

Once the robot is up and running one can not just say “Bad dog!” and expect it to figure out what it did wrong. For one thing, the search space of alternative actions is apt to be huge, leading to a lot of trial and error before the robot gets it right. This will yield unsatisfactory performance in the meantime and an impression of an excessively obtuse servant. More importantly, however, the bad-dog response is ridiculously uninformative since the user usually knows exactly what the robot did wrong or can quickly determine where the difficulty lies.

Much richer feedback can be provided through a debugging tool such as a stack prober. To implement such a facility the robot needs to be able to answer “why” it did a particular action. As in expert systems, this can be done by plugging the information from various call levels into standard pseudo-English response templates. In addition to straight dependency-directed reporting, it should also be possible to access the “comments” included during programming as a justification for the step from a higher authority, e.g. R: “What kind of beer do you want?”, U: “I don’t know, surprise me ... Aaack, why did you dump beer in my lap?”, R: “To surprise you.”, U: “Yes, but WHY?”, R: “Because Joe told me this would be surprising.”

Once the source of a difficulty has been pinpointed there are several things that might be done to correct it. Again, a verbal interface to at least the simplest level of an editing facility would be desirable. Often the confusion will involve the mis-identification of an object. This can typically be cured by adding or deleting conditions from a rule: “Show me a beer bottle ... No, a beer bottle has to be brown.” or, conversely, U: “Why isn’t this a beer bottle?” R: “It is not upright.”, U: “Oh, that’s not important.”

Other times, extra meta-control information may need to be supplied (e.g. to resolve rule conflicts in a production system such as ACT* [Anderson 1983]). Suppose the robot was asked to bring a beer from the kitchen to the living room. If it knew the spatial layout of the house its path planner might decide that proceeding directly through the door connecting the two rooms was the most efficient route. Yet the user might object to this and attempt to reprogram the robot through a dialog such as: U: “How ELSE could you get to the living room?”, R: “Through the hallway.”, U: “That is better because you won’t block the TV.” Here a preference is set up for selecting a route, along with a comment-like justification (which the robot merely needs to parrot later).

5. Language and Speech

Natural language processing systems have been applied to a number of domains. The success of these systems often depends on some strong underlying constraint. For instance, query answering systems map questions into database formulae, story understanding attempts to extract and follow causal chains, thematic gisting relies on the

existence of stereo-typed frames or scripts, and dialog understanding is analyzed in terms of epistemic speech acts. However, in our particular robot task these particular constraints are usually not available or relevant.

Despite this, some of the machinery developed for these other tasks can be useful. For instance, limited anaphora resolution makes verbal programming less tedious: “This is the recycling bin. Empty bottles go HERE”. Similarly, simple ellipsis processing makes the interface more habitable: “Bottles go here, cans [go] there.” One of the most useful techniques is the ability to define paraphrases: “ ‘Thirsty’ means ‘Get me a beer’ ”. This could be implemented at the basic grammar level by an innate template: X MEANS Y. Substring or subcategory matching could then be done on X and Y to variabilize the expression properly. Many of the abilities mentioned here were implemented long ago in the LIFER system for query answering [Hendrix 1977].

Similarly, speech recognition typically comes in one of two basic flavors [Microsoft 1998], neither of which is ideal for robot communication. Full dictation systems work with unlimited vocabularies and use a lot of statistical knowledge to help resolve homonymic sequences. But an exact translation is generally not needed by a robot – so what if it thinks you said “ear bot” instead of “beer bottle”? As long as it is reasonably consistent there should be no problem. The second option, command-and-control grammars, often give a more reliable parse since preceding words provide a heavy bias as to what the next word is likely to be. The drawback is that all the terms and valid sentences patterns have to be known ahead of time.

The ideal grammar should be extensible at run-time to accommodate newly learned object type phrases and their enclosing verb-based function templates. For instance, it would be useful to have a constrained grammar with a special symbol that could match with and bind the phonetic transcription of new “open class” words (the transcription is useful for later speech output). Barring this, the system should at least provide special sentence patterns to allow the user to “declare” new identifiers and types.

6. Conclusion

Speech can be particularly useful for robots in several capacities: command, programming, learning, debugging, and editing. Think scripting, not database queries.

References

Anderson, John R., The Architecture of Cognition, Lawrence Erlbaum, 1983.

Hendrix, Gary G., “Human Engineering for Applied Natural Language Processing”, Proc. IJCAI-77, 183-191.

Microsoft Corp., “Microsoft Speech API 4.0”, www.microsoft.com/iit/documentation/online.htm, 1998.