NSF Award no. EEC-0754741

TALKING TO ROBOTS: SPEECH-DIRECTED MOTION PLANNING

NSF Summer Undergraduate Fellowship in Sensor Technologies
Anil Venkatesh (Electrical Engineering) – University of Pennsylvania
Advisor: Dr. George J. Pappas

ABSTRACT

In order to control a robot with spoken commands, concepts from speech recognition and motion planning must be integrated. This paper provides a solution to this problem, drawing on the Spoken Language Understanding Shell system developed by Schuler, and on recent work by Kress-Gazit et al. in temporal logic motion planning. Specifically, modifications and additions to the software are described, and excerpts provided, to demonstrate how the two systems are interfaced. Furthermore, successful performance of the new system in computer simulation is exhibited, allowing the user to direct a robot in complex scenarios with a minimal number of utterances. Finally, future work is discussed in multi-robot and environment-exploration applications.

1. INTRODUCTION

The task of controlling a robot with spoken commands requires the integration of concepts from the fields of speech recognition and motion planning. This paper describes such a system which is designed with the following two objectives, both stemming from general observations on the advantages of speech direction.

Spoken input requires less effort than written input. However, the benefit vanishes if the user is required to provide extensive verbal directions. In order to justify the pursuit of a speech-directed system, the inherent advantages of this medium of input must be preserved, and hence, the number and length of necessary spoken commands minimized. Therefore, the first objective of the project is economy of spoken commands.

Spoken input is more universally accessible than written input. While a speech-directed motion planner has applications where text input does not suffice, this is for naught if the system cannot easily be adapted for diverse situations. Therefore, the second objective of the project is to reduce the amount of experience in programming required to design new scenarios for the system.

This paper details the implementation of a speech-directed motion planner. Section 2 provides background information on the two fields and describes the software packages chosen for the project. Section 3 details the process of modifying and customizing the software for this project. Description and analysis of computer simulations appear in Section 4, while Section 5 contains a discussion of the results of the project and outlines immediate improvements to be made.

Finally, Section 6 addresses the viability of futures extensions to live demonstration, environment exploration, and multiple-robot scenarios.

2. BACKGROUND

2.1 The SLUSH System

The Spoken Language Understanding Shell (SLUSH) system was chosen for this project for its innovative treatment of referential semantic information. As explained in [1], SLUSH uses a single integrated phonological, syntactic, and referential semantic language model, allowing the system to narrow down its search as more information is supplied by the speaker. For example, SLUSH naturally deals with over-determined utterances like "the glass on the table in the room" without incorrectly hypothesizing a glass not contained in a room [1]. This approach is well suited to the over-determined nature of human languages.

2.2 Grammar School

Integral to the SLUSH system is its grammar file. It would not suffice to supply the system with a list of English words: any machine would soon become overwhelmed by the number of possible (though perhaps nonsensical) ways to string those words together. Instead, SLUSH uses a grammar which describes which combinations of words may occur. A typical entry could define the concept of a prepositional phrase, as follows.

```
G PP -> (IN) in NP (INPRIME)
```

Here, `G` stands for "grammar" and precedes every entry. `PP` is the user-defined name of the structure, here denoting "prepositional phrase." The words `in NP` describe what a `PP` looks like; defining `NP` below as a noun phrase like "the room," SLUSH can now recognize "in the room" as a prepositional phrase.

```
G N -> (ROOM) room
G NP -> the N
```

Meanwhile, `IN` is the user-defined name for the relation mapping the *context set* to the set of entities which can occur before this PP, where context set refers to the set of entities which satisfied the utterance up to this point. Its counterpart, the *source set*, is the set of entities which satisfy the current argument: for the PP "in the room," this would be the set of rooms in the world model. Of course, `INPRIME` is a relation on the source set instead of the context set.

However, which entities `IN` and `INPRIME` map to is not established until lexicon entries are written for them as well. The SLUSH lexicon accompanies the grammar, describing the semantics of every relation defined in the grammar. For example, consider the entry in the lexicon for the word "room":

```
L ROOM : (set-of-all i in (source-set) s-t ((ilk of i) is (room)))
```

Similarly to the grammar, L denotes "lexicon" and precedes every entry. ROOM is the same user-defined name for the relation, just like IN and INPRIME, which links this entry to the corresponding one in the grammar. The rest of the rule reads that ROOM denotes all elements in the source set that are of type (ilk) room. Of course, the so called base ontology file is requird to

define which elements in the world model satisfy this stipulation in a straightforward way; that file is included along with the grammar and lexicon in Appendix A.

Finally, SLUSH must be told how to pronounce every word it is expected to recognize. The pronunciation file is its resource for this, listing for each word any possible pronunciations and a probability of occurrence for each. The pronunciations themselves are expressed in a standard phonetic format and the file can also be found in Appendix A.

2.2 The Most Likely Sequence

Integral to the SLUSH system is the determining of a Most Likely Sequence, or MLS, to fit the spoken input. This sequence is calculated in intervals of 10 milliseconds until the entire utterance is processed, and comes in the generic form:

```
[GO,e0.e1]STA/N_PP*;[IDENT,e0]N/alan_N;[-,-]-/-;[-,-]-/-;;/N_SIL? 5
```

`[GO,e0.e1]STA/N_PP*`
The brackets indicate that the commands "go to element e0" or "go to element e1" are hypothesized; the second component is the current *syntactic state*: in this case, the utterance so far is missing a noun (`N`) and an optional prepositional phrase (`PP*`) to be a full sentence (denoted here by `STA`).

`[IDENT,e0]N/alan_N`
The brackets denote the current hypothesis that the element e0 is the one intended by the speaker, and not e1. The syntactic state in this case indicates that the noun "alan_N" is expected (i.e. alan room—the name of a room in the example).

`/N_SIL? 5`
The number 5 indicates that this MLS was calculated during a phonetic transition: the speaker was in the middle of a word. Meanwhile, numbers less than 5 denote various syntactic-semantic transitions between actual words. In this case, it shows that the word being pronounced needs only an `N` followed optionally by silence to be identified as "alan."

Applying this method to the following MLS, we discover that it predicts the sentence "go to the alan room," between the words "alan" and "room." The spurious `e1` will disappear from the list of referents in the next MLS, after the word "room" has been fully registered.

```
[GO,e0.e1]STA/N_PP*;[IDENT,e0]N/room;[-,-]-/-;[-,-]-/-;;/R_UW_M_SIL? 2
```

2.3 Linear Temporal Logic and Motion Planning

The software implementing the algorithms in [2] is well suited to this project. Named LTLMop (from Linear Temporal Logic Motion Planner), it generates an automaton to control a robot from a set of structured English instructions, which are mapped to linear temporal logic (LTL) formulas.

As explained in [2], LTL is an extension of propositional logic to a sequence of states, corresponding to samples over time, so that the value of a proposition can change between *true*

and *false* for each new state. In order to take this new complexity into account, two operators are added to the set of negation ($\neg$) and disjunction ($\vee$): from [2], $\bigcirc\varphi$ expresses that φ is true in the next state, while $\varphi_1 \mathcal{U} \varphi_2$ denotes that φ_1 is true in every state until φ_2 becomes true. These operators can additionally be used to define $\Diamond$ (eventually) and $\Box$ (always) [2].

Rather than have the user meticulously write out linear temporal logic formulas, LTLMop uses regular expressions to support its structured English input scheme. For example, the sentence

```
If you are activating Action then visit Region
```

is converted to $\Box\Diamond(\varphi \Rightarrow \psi)$ where φ is the proposition for `Action` and ψ is the proposition for `Region`. In this case, the sentence defines a *liveness condition*, read as "always eventually, φ implies ψ." These formulas are then used to compose a controller for the robot, as detailed in [2].

3. METHODS

3.1 A New Grammar

The grammar-cum-lexicon file initially provided by SLUSH supports sentences of the form: "go to the [noun]" followed by any number of prepositional phrases; for instance, "go to the chair in the room to the left of the table." The grammar and lexicon entry covering all of this is:

```
G Simp -> go to NP (GO)
L GO : (source-set)
```

In other words, `Simp` is the name given to simple "go to" commands, which can be applied to any entity in the world model since the lexicon entry for `GO` puts no constraints on the source set.

However, the iconic if-then command, on which any meaningful robot motion planning depends, has no implementation in the default SLUSH grammar. To rectify this situation, we must introduce new grammar and lexicon entries. For example, the construction "if you see the [noun], go to…" can be added to the original grammar in the following way (courtesy of William Schuler).

```
G Simp -> PPcond Simp
G PPcond -> if Sind (IF)
G Sind -> you see NP

L IF : (context-set)
```

Again, `PPcond` and `Sind` are the user-specified names for the if-clause and the sentence it contains, respectively. Furthermore, setting `IF` to the context set in the lexicon means in this case that its referents will be whatever entities immediately follow "if you see." This way, the referents of the if-clause will not interfere with those in the subsequent "go to" clause. With this addition, the software now recognizes "if you see the table in the room, go to the chair to the left of the table" as well as a simple "go to" command as before.

Of course, the grammar must be further modified, since it is reasonable to expect a robot to do more than visit locations on visual triggers. The result and its expressivity are detailed in the next section.

3.2 Adapting SLUSH

Clearly, processing the MLS is the key to extracting spoken commands from SLUSH. By first filtering the phonetic transitions, the words can then be extracted by cross-referencing by the phonetic representation in the pronunciation dictionary (see Appendix A). But the true power of the SLUSH system lies in the incorporation of semantic information directly into its language model, allowing for successful interpretation of phrases like "the chair to the left of the table." For this reason, it is also crucial to extract, for example, the referent `e0` from `[GO,e0]`, since SLUSH may have deduced this perfectly precise command from highly imprecise language.

In order to indicate the semantics more transparently in the MLS, we adapt the form of the conditional clause as follows (# indicates removal):

```
G Simp -> go to NP (GO)
G Simp -> PPcond Simp
#G PPcond -> if Sind (IF)
#G Sind -> you see NP
G PPcond -> if you see NP (IF-SEE)
G PPcond -> if you are in NP (IF-ARE)
G PPcond -> if you catch the convict (IF-CATCH)

#L IF : (context-set)
L IF-SEE : (context-set)
L IF-ARE : (context-set)
L IF-CATCH : (context-set)
```

In this way, each type of condition is clearly marked along with its referents in the MLS, for example:

```
[IDENT,e0.e1]STA/PPcond_Simp;[IF-SEE,e12]PPcond/sil;[-,-]-/-;[-,-]-/-;;/SIL 2
```

which results at the end of the clause "if you see the chair in the alan room" (this chair is indeed denoted by `e12`).

Using regular expressions, the words and referents are extracted from the MLS, resulting in an output of the following form (where the beta room corresponds to `e1`):

```
if you see the chair in the alan room go to the beta room

Command Referents: e1
Conditional Referents: e12
```

It will prove advantageous to demarcate the two clauses with the word "then"; accordingly, a provision is included in the code for MLS processing to insert the line `/DH_EH_N_SIL?` into the MLS dump in the transition from if-statement to command. As long as this word is added to the pronunciation file, it will be appended to the sentence automatically, resulting in the following.

```
if you see the chair in the alan room then go to the beta room
```

```
Command Referents: e1
Conditional Referents: e12
```

The final step of MLS processing is oddly necessitated by the proficiency of the SLUSH system in reducing information like “the chair in the alan room” to `e12`, the element to which the phrase refers. While this is the primary strength of SLUSH, it also obscures the location of that chair, which could be a point of interest for a motion planner. In order to retrieve this information as well, we must scan the MLS dump for the entry where the word “in” is resolved, leaving its inverse, INPRIME, and the corresponding room referent. The following example is from the same sentence as above.

```
…[IF-SEE,e12]PPcond/PP_PP*;[INPRIME,e0]PP/sil;[-,-]-/-;;/SIL 3
```

This information is appended to the processed MLS except when the clause deals with an action (such as `IF-CATCH` above); in that case, recording the location is sufficient. We finally arrive at the following format.

```
if you see the chair in the alan room then go to the chair in the beta
room

Command Referents: e1/e10
Conditional Referents: e0/e12
```

3.3 Quasi-English Templates

While LTLMop provides a way for the user to write LTL formulas in a quasi-English medium, even simple scenarios such as those described in [2] require perhaps twenty such formulas to be fully specified. For example, the sentence

```
If you are activating Action_1 then visit Region_1
```

would be of little use in most applications if not accompanied by additional rules to govern when `Action_1` should be activated, perpetuated, and deactivated. This presents a problem for spoken input, since the advantage of delivering commands verbally is quickly outweighed by the burden of keeping so many particular instructions in mind.

To address this concern, a system to generate many quasi-English sentences from a single spoken instruction has been developed. Every sentence delivered by SLUSH is identified as one of the supported commands, and its information is expressed by an appropriate set of quasi-English template sentences. For example, the verbal direction “if you are in the yard, go to the office” generates the following four sentences.

```
If you are in yard then do yardFlag
If you activated yardFlag and you were not in office then do yardFlag
If you are activating yardFlag then visit office
If you were in office do not yardFlag
```

The custom proposition `yardFlag` is introduced in the first sentence so that the robot "remembers" that it had visited the region yard. The second sentence assures that `yardFlag` is reactivated at every new stage, since its value will otherwise change serendipitously. Of course, the third sentence describes the intended behavior, that the robot visit the office; and the fourth sentence absolves the robot of its duty once this task has been performed.

More elaborate commands involve a greater number of the template sentences, a complete list of which is included in Appendix B. In order to reduce the number of sentences necessary, the symbols { } are introduced around optional information, which is either removed or retrieved depending on the semantics of the utterance.

Once all utterances have been delivered, the sentences are compiled into a list. Furthermore, any actions which were mentioned are listed separately; the same goes for sensors, which arise in utterances of the form `if you see Sensor_1 then...`. Finally, custom propositions such as `yardFlag` above are indexed in order of utterance, resulting for instance in `yardFlag1`. This ensures that every custom proposition is unique.

In addition to the automatically generated sentences, certain basic conditions are specified by the user beforehand, in the same way that the grammar is customized for any particular scenario. These may include the initial conditions `Environment starts with false` and `Robot starts with false`. Furthermore, liveness conditions such as "visit every room" are currently not supported for spoken input, and must also be specified beforehand.

4. OPERATION AND SIMULATIONS

The system developed in this project provides an intuitive way to issue a set of verbal instructions to a robot. Guided by the SLUSH system map of the world model (Figure 1), the user depresses the control key and delivers a command, repeating as necessary. She then loads the generated file into LTLMop, which automatically creates a controller for the robot, satisfying the specifications of the user. The automaton can then be tested in Stage or Gazebo simulation.

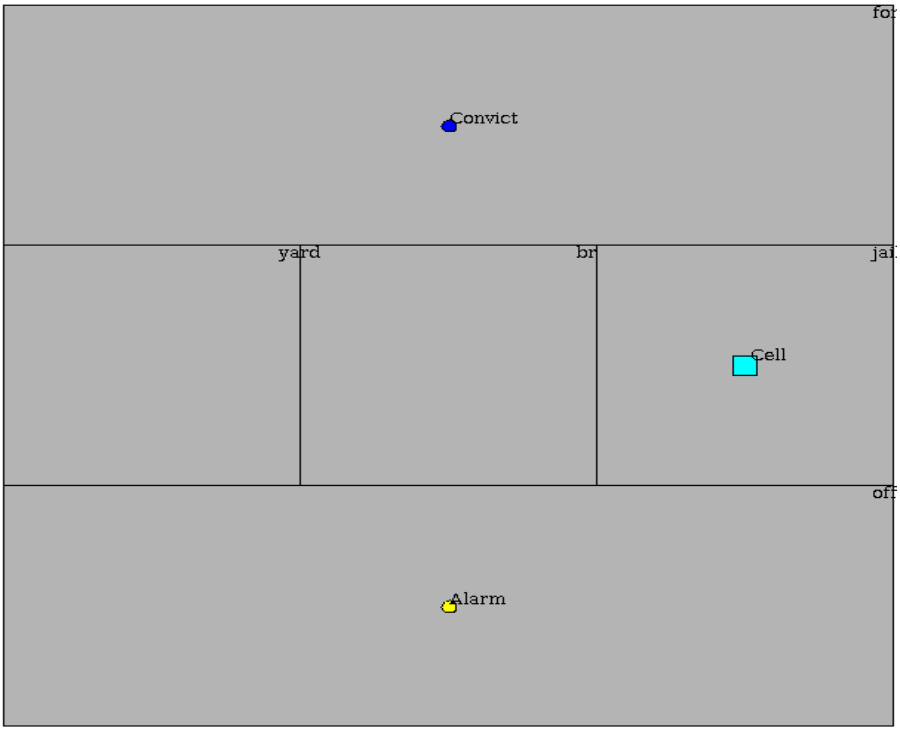

Figure 1 - SLUSH System Map

The following simulations are based on this scenario: a robot is moving about in the world depicted in Figure 2. An alarm is ringing in the office and a convict is hiding in the forest. We issue the following commands:

1. If you see the alarm in the office, catch the convict in the forest.
2. If you catch the convict in the forest, lock him up in jail.

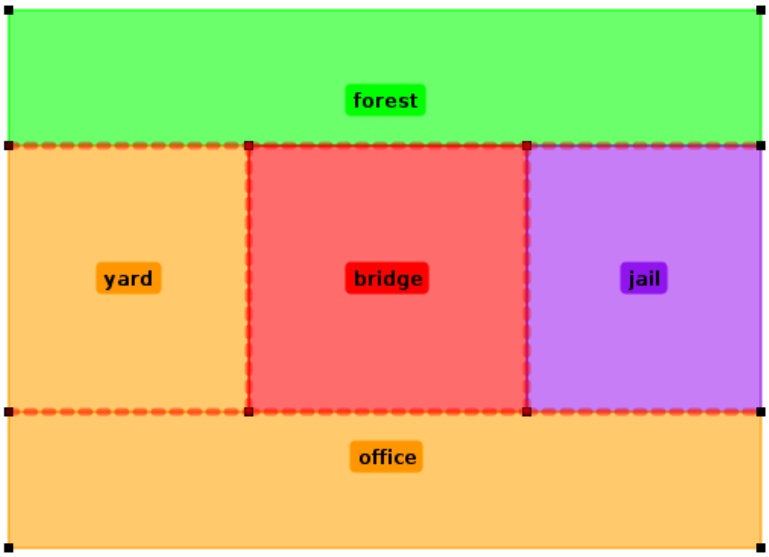

Figure 2 - World Model

MLS processing yields the following two entries. Note that the referents of actions `CATCH` and `LOCK` are suppressed since each action uniquely refers to the convict.

```
if you see the alarm in the office then catch the convict in the forest

Command Referents: forest
Conditional Referents: office/alarm
-
if you catch the convict in the forest then lock him up in the jail

Command Referents: jail
Conditional Referents: forest
```

Next, the appropriate quasi-English template sentences –twelve are required here– are added to the initial and liveness conditions already specified, resulting in the nineteen sentences given in Appendix B. Separate lists of sensors, actions, and custom propositions for this example are given in Appendix B as well.

In this case, we have chosen to specify beforehand that the robot should start in the yard and keep visiting every region. Accordingly, it roams around and interrupts its business to catch the convict whenever it happens to hear the alarm (which is when it enters the office). Figure 3 illustrates this behavior.

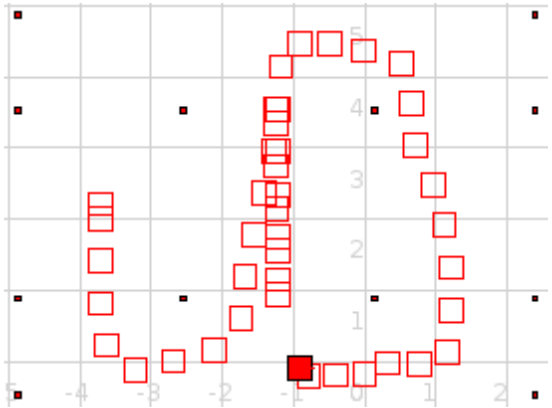

Figure 3 - Hearing the alarm for the second time

In the second simulation, we give the same verbal instructions but add that the robot should not visit the bridge by changing the structured English sentence `Visit bridge` to `Always not bridge`. Figure 4 shows how its course is adjusted by this constraint, although it still completes its task.

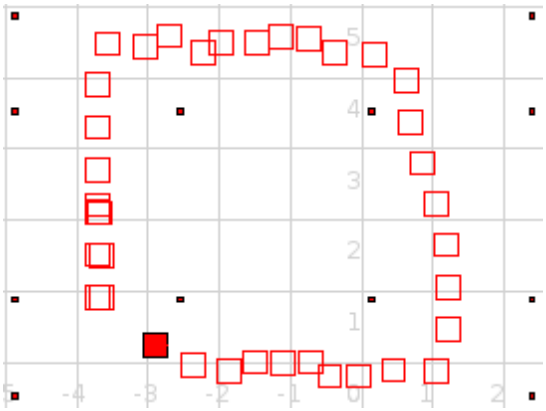

Figure 4 - Bridge closed for repairs

5. DISCUSSION

The expressivity achieved in this project is summed up in Table 1. Of course, any of the clauses in the first column may be paired clauses in the second. The final grammar, which implements the various combinations expressed in above, appears in Appendix A.

Table 1: Expressivity of final grammar

IF	THEN
SENSOR [in LOCATION_1]	go to LOCATION_2
in LOCATION_1	ACTION_2 [in LOCATION_2]
ACTION_1 [in LOCATION_1]	

The two primary goals of this project were met. Using template sentences and semantic analysis of each utterance, the number of spoken commands necessary has been kept small. Furthermore, the system is designed to accommodate customization by the user; a new scenario can be crafted solely by modifying the SLUSH files in Appendix A, and drawing the world map in the graphical region editor of LTLMop.

A number of features could be added to the system to improve its functionality. The additions require modest amounts of coding and should be supported in the next version.

For example, it would be worthwhile to implement support for utterances with multiple referents. It is reasonable to expect that the user may not care to specify a single entity in the world model every time. Accordingly, the MLS processor should generate LTL formulas involving all possible referents, which can be done with no increase in the number of logical propositions.

Conjunction support is another useful aspect for the next revision. By adding

```
G Simp -> Simp and Simp
```

to the grammar, some modifications to the MLS processing algorithm will allow this considerable enrichment to the expressivity of the system.

Furthermore, the current system requires text input beforehand to determine liveness conditions. The project encountered difficulties in automatically generating such sentences because the conditions governing them may be complicated: ultimately, there could be a unique formula governing the visitation of each individual region in the world model. Given that the philosophy of the project is economy of spoken commands, the specification of such a scenario would be burdensome indeed. However, support for sentences such as "go everywhere but [region]" would allow the user a measure of control over "visitation rights" without unduly increasing the number of utterances required.

6. RECOMMENDATIONS

The success of this project in simulation prompts a demonstration with real robots. Since LTLMop supports Player, this should be a relatively straightforward extension in the future.

Multi-robot scenarios also provide and interesting opportunity for future work. This is a natural progression from the current system with each robot treating the others as part of its environment [2]. The prospect of instructing many robots with a single command is an intriguing one, which will certainly require attention to the way the temporal logic formulas are currently generated.

A more radical extension to this project is the idea of a robot exploring a previously unmapped area. Establishing a feedback loop between speech recognizer and motion planner (that is, essentially between robot and human), the robot in this new system could coin new terms for the objects and regions it discovered, sending maps and images back to the user. Meanwhile, the user could remotely, verbally direct the robot in further exploration.

7. ACKNOWLEDGMENTS

I would like to thank my advisors, Dr. George Pappas and Hadas Kress-Gazit of the University of Pennsylvania, for their advice and encouragement. I would also like to thank Dr. William Schuler, Stephen Wu, and Luan Nguyen of the University of Minnesota for help and collaboration regarding their SLUSH system. Finally, I would like to thank the National Science Foundation for their continued support of the SUNFEST program.

8. REFERENCES

1. W. Schuler, S. Wu, L. Schwartz, A Framework for Fast Incremental Interpretation during Speech Decoding, *Computational Linguistics,* in press.
2. H. Kress-Gazit, G.E. Fainekos, G.J. Pappas, Where's Waldo? Sensor-Based Temporal Logic Motion Planning, *IEEE Conference on Robotics and Automation, Rome, Italy, April 2007.*

APPENDIX A

Grammar

```
G START -> STAA sil
G STAA -> STA
G STA -> sil Simp
G Simp -> Comm
G Comm -> go to NP (GO)
G Comm -> catch the convict PP (DO-CATCH)
G Comm -> catch him PP (DO-CATCH)
G Comm -> lock up the convict PP (DO-LOCK)
G Comm -> lock him up PP (DO-LOCK)
G NP -> the N PP*
G PP* -> PP PP*
G PP* -> PPneg PP*
G PP* -> sil
G PP -> (CONTAIN) containing NP (CONTAINPRIME)
G PP -> (IN) in NP (INPRIME)
G PP -> sil
G PPneg -> not PP (NOT)
G PPneg -> that are not PP (NOT)
G Simp -> PPcond Comm
G PPcond -> if you see NP (IF-SEE)
G PPcond -> if you are in NP (IF-ARE)
G PPcond -> if you catch the convict PP (IF-CATCH)
G PPcond -> if you catch him PP (IF-CATCH)
G PPcond -> if you lock up the convict PP (IF-LOCK)
G PPcond -> if you lock him up PP (IF-LOCK)
G N -> (YARD) yard
G N -> (OFFICE) office
G N -> (FOREST) forest
G N -> (JAIL) jail
G N -> (BRIDGE) bridge
G N -> (CONVICT) convict
G N -> (ALARM) alarm
G N -> (CELL) cell
```

Lexicon

```
L YARD : (set-of-all i in (source-set) s-t ((name of i) is (Yard)))
L OFFICE : (set-of-all i in (source-set) s-t ((name of i) is (Office)))
L FOREST : (set-of-all i in (source-set) s-t ((name of i) is (Forest)))
L JAIL : (set-of-all i in (source-set) s-t ((name of i) is (Jail)))
L BRIDGE : (set-of-all i in (source-set) s-t ((name of i) is (Bridge)))
L CONVICT : (set-of-all i in (source-set) s-t ((full of i) is (1)))
L ALARM : (set-of-all i in (source-set) s-t ((empty of i) is (1)))
L CELL : (set-of-all i in (source-set) s-t ((ilk of i) is (chair)))
L CONTAIN   : (reach-of (source-set) in (dir-graph-from-all i to j s-t ((i is-not j) and (((abs-
of ((y of i) minus (y of j))) l-t-eq (yradius of i)) and ((abs-of ((x of i) minus (x of j))) l-t-
eq (xradius of i))))))
L CONTAINPRIME : (intersection-of (context-set) with (reach-of (source-set) in (dir-graph-from-
all i to j s-t ((i is-not j) and (((abs-of ((y of i) minus (y of j))) l-t-eq (yradius of j)) and
((abs-of ((x of i) minus (x of j))) l-t-eq (xradius of j)))))))
L IN   : (reach-of (source-set) in (dir-graph-from-all i to j s-t ((i is-not j) and (((abs-of ((y
of i) minus (y of j))) l-t-eq (yradius of j)) and ((abs-of ((x of i) minus (x of j))) l-t-eq
(xradius of j))))))
L INPRIME : (intersection-of (context-set) with (reach-of (source-set) in (dir-graph-from-all i
to j s-t ((i is-not j) and (((abs-of ((y of i) minus (y of j))) l-t-eq (yradius of i)) and ((abs-
of ((x of i) minus (x of j))) l-t-eq (xradius of i)))))))
L NOT : (exclusion-of (source-set) from (context-set))
L GO : (source-set)
L DO-CATCH : (source-set)
L DO-LOCK : (source-set)
L IF-SEE : (context-set)
L IF-ARE : (context-set)
L IF-CATCH : (context-set)
L IF-LOCK : (context-set)
L CONTAIN   : (reach-of (source-set) in (dir-graph-from-all i to j s-t ((i is-not j) and (((abs-
of ((y of i) minus (y of j))) l-t-eq (yradius of i)) and ((abs-of ((x of i) minus (x of j))) l-t-
eq (xradius of i))))))
L CONTAINPRIME : (intersection-of (context-set) with (reach-of (source-set) in (dir-graph-from-
all i to j s-t ((i is-not j) and (((abs-of ((y of i) minus (y of j))) l-t-eq (yradius of j)) and
((abs-of ((x of i) minus (x of j))) l-t-eq (xradius of j)))))))
L IN   : (reach-of (source-set) in (dir-graph-from-all i to j s-t ((i is-not j) and (((abs-of ((y
of i) minus (y of j))) l-t-eq (yradius of j)) and ((abs-of ((x of i) minus (x of j))) l-t-eq
(xradius of j))))))
L INPRIME : (intersection-of (context-set) with (reach-of (source-set) in (dir-graph-from-all i
to j s-t ((i is-not j) and (((abs-of ((y of i) minus (y of j))) l-t-eq (yradius of i)) and ((abs-
of ((x of i) minus (x of j))) l-t-eq (xradius of i)))))))
```

```
L NOT : (exclusion-of (source-set) from (context-set))
```

Pronunciation

```
P alarm : /AH_L_AA_R_M_SIL? = 1
P are : /ER_SIL? = 1
P bridge : /B_R_IH_JH_SIL? = 1
P catch : /K_AE_CH_SIL? = 0.5
P cell : /S_EH_L_SIL? = 1
P containing : /K_AH_N_T_EY_N_IH_NG_SIL? = 1
P convict : /K_AA_N_V_IH_K_T_SIL? = 0.5
P convict : /K_AH_N_V_IH_K_T_SIL? = 0.5
P forest : /F_AO_R_AH_S_T_SIL? = 0.5
P forest : /F_AO_R_IH_S_T_SIL? = 0.5
P go : /G_OW_SIL? = 1
P having : /HH_AE_V_IH_NG_SIL? = 1
P him : /HH_IH_M_SIL? = 0.5
P him : /IH_M_SIL? = 0.5
P if : /IH_F_SIL? = 1
P in : /IH_N_SIL? = 1
P is : /IH_S_SIL? = 1
P it : /IH_T_SIL? = 1
P jail : /JH_EY_L_SIL = 1
P lock : /L_AA_K_SIL? = 1
P not : /N_AA_T_SIL? = 1
P office : /AH_F_AH_S_SIL? = 0.5
P office : /AH_F_IH_S_SIL? = 0.5
P on : /AH_N_SIL? = 1
P see : /S_IY_SIL? = 1
P sil : /SIL = 1
P that : /DH_AE_T_SIL? = 1
P the : /DH_AH_SIL? = 0.7
P the : /DH_IY_SIL? = 0.3
P then : /DH_EH_N_SIL? = 1
P thing : /TH_IH_NG_SIL? = 1
P to : /T_AH_SIL? = 0.3
P to : /T_IH_SIL? = 0.2
P to : /T_UW_SIL? = 0.5
P up : /AH_P_SIL? = 1
P yard : /Y_AA_R_D/SIL? = 1
P you : /Y_UW_SIL? = 1
```

Base Ontology

```
V yard ilk : room
V yard name : Yard
V yard color : light_gray
V yard x : 200
V yard y : 400
V yard xradius : 100
V yard yradius : 100
V yard shape : r
V office ilk : room
V office name : Office
V office color : light_gray
V office x : 400
V office y : 600
V office xradius : 300
V office yradius : 100
V office shape : r
V forest ilk : room
V forest name : Forest
V forest color : light_gray
V forest x : 400
V forest y : 200
V forest xradius : 300
V forest yradius : 100
V forest shape : r
V jail ilk : room
V jail name : Jail
V jail color : light_gray
V jail x : 600
V jail y : 400
V jail xradius : 100
V jail yradius : 100
V jail shape : r
V bridge ilk : room
V bridge name : Bridge
V bridge color : light_gray
V bridge x : 400
```

```
V bridge y : 400
V bridge xradius : 100
V bridge yradius : 100
V bridge shape : r
V Alarm ilk : glass
V Alarm x : 400
V Alarm y : 600
V Alarm empty : 1
V Alarm shape : e
V Alarm color : yellow
V Convict ilk : glass
V Convict x : 400
V Convict y : 200
V Convict full : 1
V Convict shape : e
V Convict color : blue
V Cell shape : r
V Cell color : cyan
V Cell ilk : chair
V Cell x : 600
V Cell y : 400
V Cell xradius : 8
V Cell yradius : 8
```

APPENDIX B

Template Sentences

```
1. If you are in Location1 then do Location1Flag
2. If you activated Location1Flag and you were not in Location2 then do Location1Flag
3. {If you are activating Location1Flag then visit Location2}
4. If you were in Location2 do not Location1Flag
5. Do Action2 if and only if you are activating Location1Flag {and you were in Location2}
6. If you did not activate Action2 and you are activating Action2 then stay there
7. If you activated Action2 then do not Location1Flag
8. If you are sensing Sensor {and you were in Location1} then do SensorFlag
9. If you activated SensorFlag and you were not in Location2 then do SensorFlag
10. {If you are activating SensorFlag then visit Location2}
11. If you were in Location2 then do not SensorFlag
12. If you activated SensorFlag and you did not activate Action2 then do SensorFlag
13. If you activated Action2 then do not SensorFlag
14. Do Action2 if and only if you are activating SensorFlag {and you were in Location2}
15. If you activated Action1 {and you were in Location1} then do Action1Flag
16. If you activated Action1Flag and you were not in Location2 then do Action1Flag
17. {If you are activating Action1Flag then visit Location2}
18. If you were in Location2 then do not Action1Flag
19. If you activated Action1Flag and you did not activate Action2 then do Action1Flag
20. If you activated Action2 then do not Action1Flag
21. Do Action2 if and only if you are activating Action1Flag {and you were in Location2}
```

Structured English Example

```
Environment starts with false
Robot starts with false

Visit yard
Visit office
Visit forest
Visit jail
Visit bridge

If you did not activate Catch and you are activating Catch then stay there
If you are sensing alarm and you were in office then do alarmFlag1
If you are activating alarmFlag1 then visit forest
If you activated alarmFlag1 and you did not activate Catch then do alarmFlag1
If you activated Catch then do not alarmFlag1
Do Catch if and only if you are activating alarmFlag1 and you were in forest

If you did not activate Lock and you are activating Lock then stay there
If you activated Catch and you were in forest then do catchFlag2
If you are activating catchFlag2 then visit jail
If you activated catchFlag2 and you did not activate Lock then do catchFlag2
If you activated Lock then do not catchFlag2
Do Lock if and only if you are activating catcFlag2 and you were in jail
```

Propositions

```
Sensors: alarm
Actions: Catch, Lock
Customs: alarmFlag1, catchFlag2
```