N93-[illegible]
118
188163

A Natural-Language Interface to a Mobile Robot

S. Michalowski
Veterans Administration Medical Center
Palo Alto, CA 94304

C. Crangle and L. Liang
Stanford University
Stanford, CA 94305

VD614709

S0385476

1 BACKGROUND

The present work on robot instructability is based on an ongoing effort to apply modern manipulation technology to serve the needs of the handicapped [1]. The Stanford/VA Robotic Aid is a mobile manipulation system that is being developed to assist severely disabled persons (quadriplegics) in performing simple activities of everyday living in a homelike, unstructured environment. It consists of two major components: a nine degree-of-freedom manipulator and a stationary control console [2].

In clinical applications, the Robotic Aid has been used as a voice-controlled telemanipulator. To perform a task, a user gives a series of discrete, explicit motion commands, each corresponding to a degree of freedom of the robot, such as: *"Forward!"*, *"Left!"*, *"North!"*, *"Down!"*. The direction commands have to be qualified: the utterance *"Left!"*, for example, can refer to a rotation or a translation of the arm or of the mobile base, in any one of several coordinate systems. Furthermore, the command can be interpreted as an incremental motion or as a continuous velocity. A variety of qualifier commands are available to define the context of the specified directions. Experienced users of the Robotic Aid have achieved a considerable degree of skill: preparing and serving food, operating appliances, and performing various personal hygiene tasks. To an extent, however, users of the device have experienced frustration and dissatisfaction due to : 1) the low dexterity and lack of sensory control of the gripper, 2) occasional errors of the speech recognition system, 3) the necessity of constantly monitoring the robot's motion, 4) the unnatural character of the commands themselves. The first two factors are being addressed by several members of the Robotic Aid team; the last two led the authors to hypothesize that the highly formalized command structure should be replaced by simple colloquial English. This paper presents some of the design constraints, and implementation decisions, that resulted from adding a natural-language interface to the existing robot. Sections 2 and 3 describe the real-time software architecture required to produce the correct robot motion in response to verbal commands. Sections 4 and 5 describe the interpretation of the commands.

2 FUNCTIONAL DESIGN

In the work presented here, only the motions of the Robotic Aid's omnidirectional motion base have been considered, i.e., the six degrees of freedom of the arm and gripper have been ignored. The goal has been to develop some basic software tools for commanding the robot's motions in an enclosed room containing a few objects such as tables, chairs, and rugs. Given these goals and restrictions, the following are

[illegible]

intentions that an operator might wish to communicate to the robot through the use of natural-language commands: that the robot go to a given region of the room; that the robot move in a given direction; that the robot avoid a given region; that the robot stay within a given region; that the robot follow a given path; that the robot stop doing whatever it is doing at that time; that the robot perform any specific motion at slower than (or faster than) normal speed; that the robot speed up; that the robot slow down; that the robot pursue two goals simultaneously (the goals are not necessarily achieved simultaneously); that the robot pursue one goal after another has been achieved; that the robot pursue a goal until a given condition is met (the pursuit of the goal will be interrupted); that the robot repeatedly pursue a goal until a given condition is met; that the robot pursue a goal if (or when or whenever) a certain condition is met. The conditions the robot must detect are: that a given distance has been traversed; that a given time has elapsed; that the robot is beyond one region relative to another; that the robot's bumpers are hit; and that the robot is in a certain region. The robot's software architecture was designed to allow commands expressing these intentions to be interpreted.

As pointed out in a companion paper, the interpretation of even the simplest English commands to a robot must take place in a contextual framework: one that specifies the perceptual, cognitive and motor functions of the robot, as well as an abstracted model of the external world (the robot's operating environment) which can be used to resolve references to such entities as objects, trajectories, and directions.

In the present work, the environmental model takes the form of a two-dimesional map with objects represented by polygons. Admittedly, such a highly simplified scheme bears little resemblance to the elaborate cognitive models of reality that are used in normal human discourse. In particular, the polygonal model is given *a priori* and does not contain any perceptual elements: there is no "polygon sensor" on board the mobile robot. The adopted model should be viewed as a temporary device that establishes a context for the more significant developments in system design, language processing, and real-time control.

Language processing provides an interface between the user and the robot. The robot is characterized by four kinds of behavior: 1) it can move about the room in a variety of generic ways; 2) it can monitor its internal state and its state with respect to the environment; 3) it can resolve certain references to its surroundings; 4) it can execute motion sequences within certain temporal and logical constraints. Specifically, these behavior modes are implemented in the following building blocks of the robot's software architecture: MOTION ROUTINES, TEST ROUTINES, REGION ROUTINES and CONTROL STRUCTURES.

MOTION ROUTINES are simple goal-oriented algorithms that can generate or change motion. Seven of these are currently supported: `Stop`, `Pause`, `Resume`, `Piloting` (moving in a specified direction), `RegionSeeking` (translating towards, or away from, known objects or regions of space), `Orienting` (rotating with respect to objects or regions) and `Repelling` (moving around obstacles).

TEST ROUTINES (or simply TESTS) are also simple run-time routines. They do not affect the motion directly, but return a boolean result. Six TESTS are currently implemented: `RobotInRegion?` (is the robot in a specified region of space?), `Facing?` (is the robot pointing at, or away from, a specified region?), `TimeElapsed?`, `DistanceCovered?`, `AngleCovered?` (has a specified time, translation, or rotation increment elapsed?) and `BumpersHit?` (has the robot's segmented bumper system hit anything?). Any TEST, along with its arguments, can be invoked in one of two modes: *IF* and *WHEN*. In *IF* mode, the computation is performed only once and the result (TRUE or FALSE) is returned immediately. In *WHEN* mode, the calculation associated with the TEST is repeated until TRUE is obtained. Some of the MOTION ROUTINES have implicit termination TESTS, invoked automatically whenever the MOTION ROUTINE is activated.

REGION ROUTINES operate on the robot's two-dimesional world model. Real-life objects are modeled as polygons ("regions"). In addition to named objects (*window*, for instance, names the window, *chair*, the chair, etc.), regions are also used to represent areas defined with respect to objects, as demanded by natural-language expressions such as ***within six feet of the stairs, at least two feet to the right of the table***, and ***this side of the chair***. These regions can be computed relative to any one of four coordinate systems: the fixed room system; one centered on the robot; one centered on the speaker; and one embedded in an object, such as a chair, that has an intrinsic orientation. **REGION ROUTINES** are invoked by calls to procedure **DetermineRegion** with five arguments: the first names the object; the second specifies a distance (with ϵ representing a small default distance); the third specifies a direction (any of North, South, East, West, This, Other, Around, Left, Right, Forward, Back), the fourth specifies the type of region computed (< for instance indicates that the region is within the limit specified by the distance argument); and the fifth specifies the coordinate system (when none is given, the room system is chosen by default).

CONTROL STRUCTURES embody the logical and temporal constraints imposed by the command. There are seven basic forms which may be combined recursively to describe complex behavior sequences.

DO(B [until x]) This form results in the execution of **MOTION ROUTINE** B, to be terminated as soon as **TEST** x returns the value TRUE. Because some **MOTION ROUTINES** have implicit termination **TESTS**, the until clause is optional.

SEQ($S_1 \ldots S_n$) This form results in the sequential execution of daughter **STRUCTURES** $S_1 \ldots S_n$, each of the which may be of type DO, SEQ, PAR, IF, WHEN, WHENEVER or REPEAT. As shown in Section 4, language processing often produces the form SEQ(S), which simply denotes the execution of S.

PAR($S_1 \ldots S_n$) This form denotes simultaneous execution of $S_1 \ldots S_n$.

IF(x then S_1 [else S_2]) This form denotes the execution of S_1 if the **TEST** x returns TRUE immediately, i.e. at the time that the IF **STRUCTURE** is first encountered by the robot's scheduling algorithm. Optionally, if x returns FALSE, S_2 is executed.

WHEN(x S) This form is similar to (IF x S) except that **TEST** x need not be satisfied immediately: S's execution will await x being TRUE.

WHENEVER(x S) This form results in the repetition of S each time x is true, with the provision that S must terminate before x is tested again. Because this form has no explicit termination condition, it is often combined with the DO form: DO(WHENEVER(x S) until y).

REPEAT(S until x) This form results in the repetition of S until the condition x is TRUE. Each instance of S must terminate before another one can begin. The TEST x is evaluated whenever S terminates.

3 MOTION and TEST ROUTINES

A selection of the seven **MOTION ROUTINES** and six **TEST ROUTINES** currently supported are used in the examples of Section 4. Their operation is described here.

`Piloting`
This procedure takes three arguments: the first specifies whether the movement is linear or rotational; the second specifies the direction of movement (North, for instance); and the third specifies whether the default speed of the mobile base is to be increased, decreased, or not changed at all. Thus `Piloting`(Shift, Left, +), will spawn a computation that shifts the robot to the left at a speed one unit greater than the default speed. Calls to `Piloting` are usually embedded in a DO **STRUCTURE**, and the robot will stop moving only when the condition specified in the DO becomes true. First argument values: Shift, Rotate. Second argument values: North, South, East, West, Forward, Backward, Left, Right, Clockwise, Counterclockwise. Third argument values: + (increase speed), – (decrease speed).

`RegionSeeking`
Activation of this procedure causes the robot to translate towards the nearest point on the boundary of a region. The location of the nearest point (which is not necessarily one of the vertices of the polygon that describes the region) is continually recomputed. The procedure takes three arguments: the first specifies the region (as returned by the procedure `DetermineRegion`); the second argument indicates whether that movement is towards or away; and the third argument specifies speed. First argument values: a set of vertices determined at run time. Second argument values: + (towards), − (away). Third argument values: + (increase speed), − (decrease speed). `RegionSeeking` has an implicit termination TEST: `RobotInRegion?`

`Orienting`
A procedure of three arguments: the first specifies a region that the robot turns towards or away from; the second and third arguments are as for `RegionSeeking` above. The implicit termination TEST is `Facing?`

`Repelling`
A procedure of one argument: a region (as a set of vertices) which the robot is not allowed to enter. When the robot is very close to the region, the only motion allowed is along or away from the boundary of the region.

`RobotInRegion?`
This procedure takes one argument, a region. It returns TRUE if the center of the robot is in the region; FALSE otherwise.

`Facing?`
This procedure returns TRUE if the robot is pointing at (or away from) a region. The region is the first argument and the direction (+ or −) is the second. The requirement for TRUE is that there be at least one region vertex on both sides of the robot's forward (or backwards) direction axis.

`DistanceCovered?`
This procedure takes two arguments. The first gives a distance in inches; the second gives the direction (Forward, North, etc.), along which distance is measured (the value Trajectory specifies straight line distance between the starting point and present point). The routine returns TRUE if the distance covered since the routine was activated is greater than or equal to the distance specified; FALSE otherwise.

4 COMMAND INTERPRETATION

The Language Processor accepts a user command in the form of a colloquial English sentence entered at the terminal. It produces an Interpreted Command which contains the following information:

- which of the generic robot MOTION ROUTINES are to be invoked.
- the temporal order of execution and the logical conditions under which the MOTION ROUTINES are activated and terminated, as expressed by TESTS and CONTROL STRUCTURES.
- the parameters of any polygonal regions that were referred to in the command, and are to be computed by the REGION ROUTINE.

In the following examples of Interpreted Commands, references to regions are left unresolved and implicit termination TESTS are not shown:

Go to the desk.
SEQ(`RegionSeeking`(<the region around the desk>,+))

Then go on over to the telephone when the bumpers are hit.
WHEN(`BumpersHit?`, SEQ(`RegionSeeking`(<the region around the telephone>,+)))

Slowly move backwards to within one inch of the stairs.

```
SEQ(DO(PAR( Piloting(Shift,Backward,−),
             RegionSeeking(<the region one inch around the stairs>,+,−)),
        RobotInRegion?(<the region one inch around the stairs>)))
```

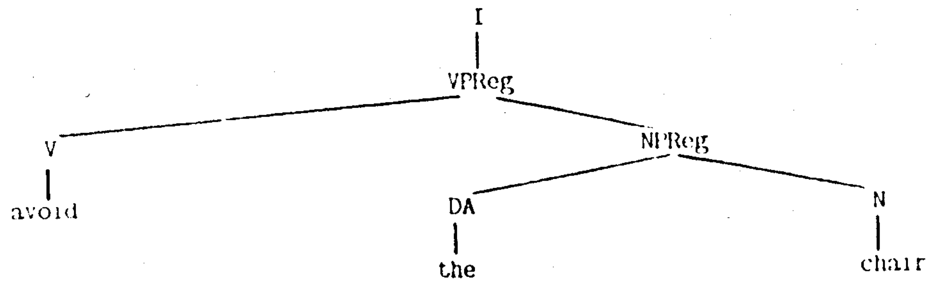
Go north west for three feet then face the chair.

```
SEQ(SEQ(DO(PAR(Piloting(Shift,North),Piloting(Shift,West)),
            DistanceCovered?(36,Trajectory))),
      SEQ(Orienting(<the region around the chair>,+)))
```

An Interpreted Command is built up as follows. Individual words denote calls to robot **ROUTINES** and **CONTROL STRUCTURES**. These calls may be partially or fully specified. A call is fully specified when a **ROUTINE** (motion, test, or region) is named or a **STRUCTURE** is named and values are given to all its arguments. A partially specified call leaves undefined either the **ROUTINE** or **STRUCTURE** or an argument value. The process governing the synthesis of lexical calls (full or partial) to form an Interpreted Command, one that specifies robot action, is the semantic tree [7] [8]. This tree is generated from rules of semantic composition (called semantic functions) that are attached to the phrase-structure rules of the grammar. A simple example illustrates the main ideas. While the example uses a context-free grammar, semantic trees require no particular grammar or parsing method. The concept of a semantic tree does fit very well, however, with the recent developments in augmented phrase-structure grammars that have led to the theory of lexical-functional grammars and to the unification-based grammar formalisms such as D-PATR [4] [5]. The grammar presently in use for the Robotic Aid was developed using the D-PATR system available on the XEROX 1108 Workstations. A COMMON LISP version of the parser, known as CPATR, is in use in the robot system.

A brief overview of context-free grammars follows for those unfamiliar with language theory. A context-free grammar consists of a finite set of terminal symbols, W_T, a finite set of non-terminal symbols, W_N, a designated start symbol from W_N, and a finite set of production rules of the form $x \rightarrow y$, where $x \in W_N$ and y is a non-empty string in W^* where $W = W_T \cup W_N$ and W^* is the set consisting of concatenations of any finite number of members of W. The elements of W_T, the terminal symbols, are the English words used to instruct the robot. Each word belongs to a syntactic category which is designated by one of the symbols from W_N. The word *avoid*, for instance, is a verb and belongs to the category V. The word *chair* is a noun and belongs to the category N. A sentence that conforms to the rules of the grammar is said to be parsed by that grammar and the structure of that parse is shown in a parse tree. An augmented phrase-structure grammar also contains constraint equations which are attached to the symbols on the righthand side of the production rules. These equations must be satisfied for all rules appearing in the parse tree. Several examples of the use of constraint equations are to be found in Section 5.

Consider the parse tree for the imperative *Avoid the chair*. The non-terminal labels shown are I (for imperative), VPReg (for verb phrase of region), NPReg (for noun phrase of region), V (for verb), N (for noun) and DA (for definite article).

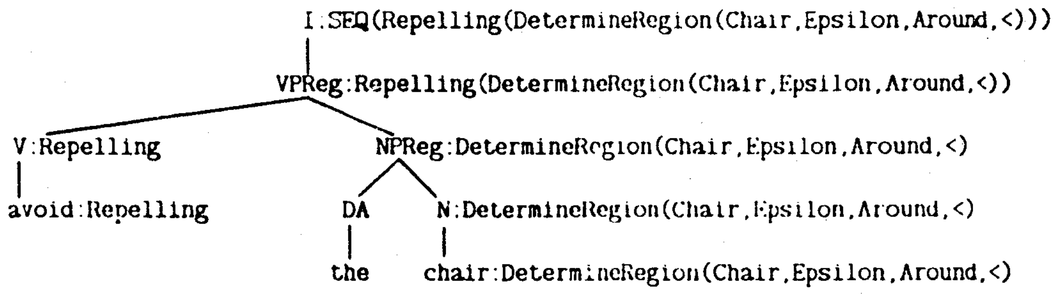


This parse is produced by a context-free grammar which is extended by assigning at most one semantic function to each production rule of the grammar. The semantic functions show how the denotation at a node of the tree is derived from the denotations of its daughter nodes. In the grammar, square braces show denotations. For instance, [NPReg] stands for the denotation of NPReg, [*chair*] for the denotation of *chair*. In this example, the denotation of *chair* is the routine that computes the region around the chair. The denotation of *avoid* is the MOTION ROUTINE `Repelling`. The semantic function [V]([NPReg]) designates the operation by which [NPReg] is set as the argument value for the [V] routine. Note that the SEQ STRUCTURE is introduced by the semantic function attached to the imperative rule. The definite article has no denotation in this simplified example. (It is clear from work on discourse understanding that cues given earlier in the discourse help fix the reference of a noun phrase such as *the chair*. Crangle and Suppes [3] describe the use of an appropriate discourse mechanism, but in this paper the semantic role of the definite article is ignored.)

Production Rule	Semantic Function
I --> VPReg	[I] = SEQ([VPReg])
VPReg --> V + NPReg	[VPReg] = [V]([NPReg])
NPReg --> DA + N	[NPReg] = [N]
V --> *avoid*	[V] = [*avoid*] == Repelling
N --> *chair*	[N] = [*chair*] == DetermineRegion(Chair,Epsilon,Around,<)

Some semantic functions, such as the operation described above for specifying argument values, are implemented in a straightforward manner in the D-PATR system using the operation of unification and representing calls as feature-value pairs in the D-PATR notation. Other semantic functions are accommodated in an extension made to D-PATR and CPATR. This extension also allows the execution of LISP functions that, in interaction with the user, help determine the appropriate interpretation of commands that are semantically ambiguous or semantically incomplete. One such example is discussed in the next section along with several production rules and semantic functions from the robot's grammar. Full details of the implemented grammar are in a longer technical report now in preparation.

The extended grammar yields the following semantic tree for *Avoid the chair.* To the left of the colon at each node is the terminal or non-terminal label. To the right of the colon is the denotation of that label. At the top of the tree, the Interpreted Command specifies robot action for the English command *Avoid the chair.*


I:SEQ(Repelling(DetermineRegion(Chair,Epsilon,Around,<)))
VPReg:Repelling(DetermineRegion(Chair,Epsilon,Around,<))
V:Repelling
NPReg:DetermineRegion(Chair,Epsilon,Around,<)
avoid:Repelling
DA
N:DetermineRegion(Chair,Epsilon,Around,<)
the
chair:DetermineRegion(Chair,Epsilon,Around,<)

When the rules of semantic composition are attached to the production rules of the grammar, the approach is often called "syntax-driven translation." This label is somewhat inappropriate for this work, however, since the production rules are strongly constrained by the demands of the semantics, suggesting rather the label "semantic grammars." In such grammars, the categories are not those of grammars that concern themselves mainly with syntactic phenomenon. For instance, the grammar for the mobile base of the Robotic Aid does not simply have verb phrases, but verb phrases of direction, of compound direction, and of region. One result of this subcategcrization is that there are many more production rules. At the same time, other conventional syntactic categories such as prepositional phrases are often missing, with the result that relatively flat parse trees are produced for many sentences. An argument for such grammars, and a discussion of their computational consequences, may be found in [6].

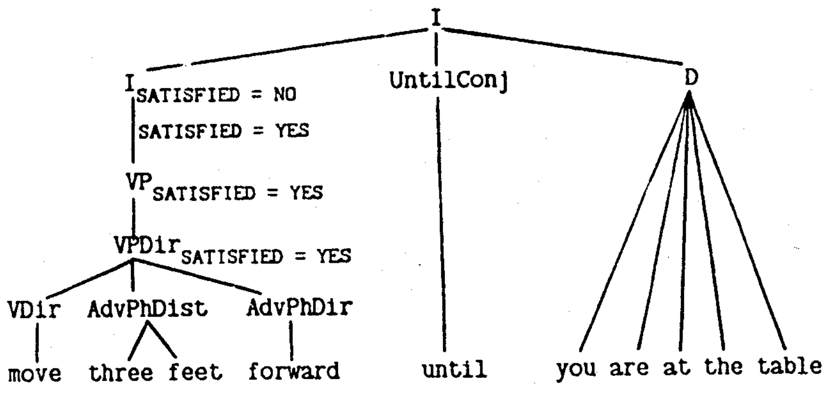
5 RULES AND CONSTRAINTS

Production Rule	Semantic Function	Constraint Equations
(1) VPDir --> VDir + AdvPhDir	[VPDir] = SEQ([VDir]([AdvPhDir]$_2$))	VPDir SATISFIED = NO
(2) VP --> VPDir	[VP] = [VPDir]	VP SATISFIED = VPDir SATISFIED
(3) I --> VP	[I] = SEQ([VP])	I SATISFIED = VP SATISFIED
(4) I --> I + UntilConj + D	[I] = DO([I] , [D])	I(rhs) SATISFIED = NO I(lhs) SATISFIED = YES
(5) VPDir --> VDir + AdvPhDist + AdvPhDir	[VPDir] = DO([VDir]([AdvPhDir]$_2$) , [AdvPhDist]([AdvPhDir]$_2$))	VPDir SATISFIED = YES
(6) VPDir --> VDir + AdvPhDir	[VPDir] = DO([VDir]([AdvPhDir]$_2$) , DistanceCovered?(AskHowFar,[AdvPhDir]))	VPDir SATISFIED = YES
(7) VPDirC --> VDir + DAdv + DAdv + AdvPhDist	[VPDirC] = DO(PAR([VDir]([DAdv]$_2$),[VDir]([DAdv]$_2$)) , [AdvPhDist](ArcLength$_2$))	
(8) AdvPhDist --> ForP + AdvDist	[AdvPhDist] = DistanceCovered?([AdvDist]$_1$)	
(9) VPReg --> VRegS + AdvDir + ToP + NPReg	[VPReg] = DO(PAR(Piloting(Shift$_1$,[AdvDir]$_2$),[VRegS]([NPReg]$_1$,[ToP]$_2$)), RobotInRegion?([NPReg]))	

Several examples are now discussed using the nine rules above which are taken from the semantic grammar for the Robotic Aid. The production rules of this grammar, in the spirit of lexical-functional grammars, are augmented by equations that constrain the values that features can take. In the rules above, for instance, the feature SATISFIED is constrained to take either the value YES or the value NO. (A note on notation used in the semantic functions: When only some argument values are set, subscripts are used to indicate argument positions. For instance, Routine-X(value-a_2,value-b_3) indicates that the second and third arguments of Routine-X are set to value-a and value-b respectively with other argument values left unaltered. When all argument values are set, no subscripting is used.)

The first five rules illustrate the role played by the constraint equations. Rule (1) parses a verb phrase such as *Move forward* which on its own is considered to be semantically incomplete in that it specifies neither how far to move nor for how long. The verb phrase of direction (VPDir) is therefore marked with the SATISFIED feature set to NO. *Move forward* may be embedded within a command such as *Move forward until you are at the table* which is semantically complete and is parsed by rules (1), (2), (3), and (4) without inconsistency in the assignment of values to the SATISFIED feature. (*You are at the table* is parsed as the declarative D.)

The verb phrase *Move three feet forward*, parsed by rule (5), is also semantically complete and so the SATISFIED feature is set to YES in rule (5). The command *Move three feet forward until you are at the table*, on the other hand, cannot be parsed by rules (2), (3), (4), and (5) because SATISFIED is set to YES by rule (5) and maintained with that value by rules (2) and (3), whereas by rule (4) the I (for imperative) constituent appearing in the righthand side of the equation must have SATISFIED set to NO. What the user might have intended by that command is more accurately expressed by the command *Repeatedly move three feet forward until you are at the table* or *Continue moving forward three feet at a time until you are at the table.*



The sixth rule illustrates our handling of a command such as *Move forward* when it is not used in a context that completes it semantically. The rule contains a call to the LISP routine `AskHowFar` which asks the user to specify a distance to be used in the **TEST** `DistanceCovered?`. (This routine is executed only if rule (6) appears in the final parse tree; it is not executed during the parsing process itself when rules are successively tried and discarded.) As discussed extensively in the companion paper in this volume, interaction between the robot and the user (and between the robot and its perceptual environment) is essential to the interpretation of natural-language commands. This example illustrates just one of the many occasions that call for interaction. Other obvious examples are the use of words such as *left* in *Go left* which can be interpreted relative to the robot or the speaker.

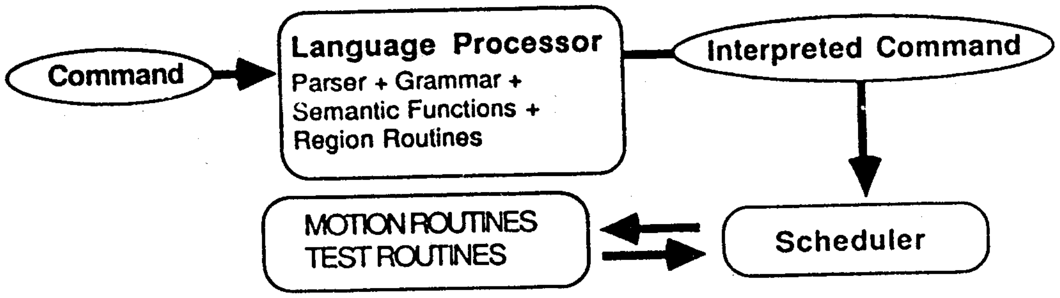
The seventh, eighth, and ninth rules, shown here without their constraint equations, illustrate something of the "fit" that has to be found between the surface structure of English commands and the robot routines. Rule (7) parses a verb phrase such as *Move north west for three feet.* (It is convenient semantically to treat *northwest* as two separate words.) The robot's `Piloting` routine knows only about the four compass directions ~~of~~ north, south, east, and west given by the room coordinate system. The only way to accomplish movement in the northwest direction in response to this command is to simultaneously execute a `Piloting(North)` and a `Piloting(West)`. This parallel STRUCTURE is then embedded in a DO STRUCTURE with the adverbial phrase of distance (AdvPhDist) contributing the TEST `DistanceCovered?` with its first argument value set in rule (8). Notice that in rule (7), the second argument value of `DistanceCovered?` is set to the value Trajectory.

Rule (9) parses verb phrases such as *Go left towards the table*, invoking the simultaneous execution of a `Piloting(Left)` routine and a `RegionSeeking(<the region around the table>)` routine embedded in a DO STRUCTURE with a `RobotInRegion?` TEST to determine when the robot is at the table. Note that a simple PAR STRUCTURE of `Piloting` and `RegionSeeking` is not sufficient: in that case, `RegionSeeking` would end when the robot reached the table, but `Piloting` would not, and the robot would continue moving left. It makes sense to issue this command only if in moving leftwards the robot would indeed reach the table.

6 IMPLEMENTATION AND COMMAND EXECUTION

The mobile robot consists of a six degree-of-freedom Unimation PUMA 260 robotic arm, equipped with a simple gripper and mounted on a unique three-wheeled motion base. The 12-inch diameter wheels of the base are located at the vertices of an equilateral triangle with 17.3-inch sides. The circumference of each wheel consists of 20 free-wheeling rollers which allow the wheel to move in a direction parallel to its axis of rotation. An onboard processor generates position commands to the wheel controllers at a rate of 15 Hz. By choosing a set of three rotational wheel velocities, any combination of translations and rotations can be achieved. The robot possesses complete freedom of motion in the plane, unlike other vehicles whose wheels have to be re-oriented before the direction of motion can change.

The onboard computer is a Digital Equipment Corporation LSI 11/73. The motion computation routines were written in MicroPower PASCAL, a dialect of PASCAL that explicitly supports a high level of multi-process concurrence on a single CPU. This is achieved through a system of dynamic priority assignments and inter-process synchronization and communication functions such as semaphores and mailboxes. In addition, multiple invocations of a given re-entrant process can run concurrently, each in its own priority and memory-mapping context. Typically, at any given time, a dozen processes are competing for access to the CPU, the exact number of processes depending on the specific command being executed. A MicroPower PASCAL kernel assigns access based on priority. Although this system has the desired flexibility, it requires that the individual routines be as simple as possible to reduce the overall CPU load. The kernel contributes an approximate CPU overhead of 20%, with additional processing time needed for boilerplate functions such as kinematic computations and communication with the Language Processor.



When an Interpreted Command is produced by the Language Processor, it is acquired by a program called the Scheduler which is in charge of activating **TESTS** and **MOTION ROUTINES** based on the content of the Interpreted Command. Typically, inspection of the Interpreted Command will result in invoking one or more **TESTS** and/or **MOTION ROUTINES**. This is done by sending a coded command (containing the identifier of the desired routine(s) and its arguments) to the robot. A utility routine on the robot maintains a variety of tables and other data structures that keep track of the state of the system which, at any instant, is defined by the currently executing set of **MOTION ROUTINES** and **TESTS**. The instantaneous state of the robot persists until one of the **TESTS** returns a value. Then, and only then, is the Scheduler reactivated. It examines the Interpreted Command to determine the appropriate response — usually the invocation or termination of other **MOTION ROUTINES** and/or **TESTS**.

The Language Processor and the Scheduler were implemented in COMMON LISP on a stationary Microvax II computer. The **MOTION ROUTINES** and **TESTS** (implemented as PASCAL procedures) run on the mobile robot's LSI 11/73. To guarantee smooth motion of the robot, these procedures are executed at a fixed rate of 15 Hz. Whenever the Scheduler activates or terminates a **MOTION ROUTINE** or **TEST**, a command packet is sent to the robot via the radio link, specifying the name of the routine and its arguments. The robot responds whenver a **TEST** returns TRUE (if it was invoked in *WHEN* mode) or TRUE/FALSE (if *IF* mode was selected).

During most instances of command execution, a number of **MOTION ROUTINES** execute concurrently. For example, in the case of the command *Move to the front of the desk while facing the window and avoiding the rug and the lamp*, four **MOTION ROUTINES** are active simultaneously: `RegionSeeking`(<the region in front of the desk>), `Orienting`(<the region around the window>), and two instances of `Repelling`: one with argument (<the region around the rug>), the other with argument <the region around the lamp>). Each **MOTION ROUTINE** contributes to the overall motion of the robot in the form of a three-dimensional (two translations and one rotation) velocity vector in one of two coordinate systems: the fixed room system (directions: north, south, east, west, turn clockwise, turn counterclockwise), and the moving base system (directions: forward, backward, left, right, turn left, turn right). As noted earlier, all process computations are iterated at 15 Hz.

7 ACKNOWLEDGEMENTS

This work was funded by the United States Veterans Administration through the Rehabilitation Research and Development Center of the VA Medical Center in Palo Alto. The authors wish to gratefully acknowledge the use of the following Stanford University facilities: Center for Design Research, Center for the Study of Language and Information, and the Institute for Mathematical Studies in the Social Sciences. The contributions made by Michael Moore and Sami Shaio are very much appreciated.

References

[1] L.J.Leifer, S.J.Michalowski, H.F.M.Van der Loos. *Development of an Advanced Robotic Aid: from Feasibility to Utility.* Proceedings of the Ninth Annual Conference on Rehabilitation Technology RESNA'86.

[2] S.J.Michalowski. *Progress Towards a Robotic Aid for the Severely Disabled.* Proceedings of the Sixth CISM-IFToMN Symposium on Theory and Practice of Robots and Manipulators ROMANSY'86.

[3] C.Crangle, P.Suppes *Studies in Natural Semantics for Instructable Robots: Part 1.* Technical report 308, Institute for Mathematical Studies in the Social Sciences. Stanford University. 1986.

[4] R.Kaplan, J.Bresnan *Lexical-functional Grammar: a Formal System for Grammatical Representaion.* in J. Bresnan(ed) The Mental Representation of Grammatical Relations. pp 173-281 MIT Press, Cambridge, Mass. 1982.

[5] S.M.Scheiber, F.C.N.Pereira, L.Karttunen, N.Kay *A Compilation of Papers on Unification-based Grammar Formalisms.* Report CSLI-86-48 Center for the Study of Language and Information. Stanford University. 1986.

[6] P.Suppes, C. Crangle *Context-fixing Semantics for the Language of Action.* in C.Taylor, J.Dancy, J.Moravcsik(eds) Language and Value (in press) Stanford University Press.

[7] P. Suppes *Semantics of Context-free Fragments of Natural Languages.* in K.J.J.Hintikka, J.Moravcsik, P.Suppes(eds) Approaches to Natural Language. Dordrecht: Reidel. 1973.

[8] P.Suppes, E.Macken *Steps Towards a Variable-free Semantics of Attributive Adjectives, Possesives and Intensifying Adverbs.* in K.E.Nelson(ed) Children's Language. Gardner Press, New York. 1978.