Autochthonous Behaviors - Mapping Perception to Action

Rod Grupen and Thomas C. Henderson*
COINS
University of Massachusetts, USA

Abstract

In this paper we describe an approach to high-level multisensor integration organized around certain egocentric behaviors. The task itself determines the sequence of sensing, the sensors used, and the responses to the sensed data. This leads to the encapsulation of robot behavior in terms of logical sensors and logical actuators. A description of this approach is given as well as some examples for dextrous manipulation and mobile robots.

*Department of Computer Science, University of Utah, USA. This author would like to thank Olivier Faugeras and the Robotics and Vision Group at INRIA (Sophia-Antipolis), France for the great sabbatical year there in 1988-89. This work was supported in part by NSF Grant IRI-8802585 and DARPA Contract DAAK1184K0017. All opinions, findings, conclusions or recommendations expressed in this document are those of the author.

1 Introduction

Figure 1 depicts the universe as a quantifiable state space. Within this space, subspaces are illustrated which represent the perceivable and actuatable spaces within the universe. The perceivable subspace represents those states which may be measured within the universe, the actuatable subspace represents those states which may be altered by employing a behavior of the system. In the discrete domain, the perceivable space represents states which are fully measureable by available (logical) sensors and the actuatable space are those states which may be transformed to other states by employing (logical) actuators. The dimensionality of the perceivable and actuatable subspace may not necessarily be the same. Moreover, both must be proper subsets of the universe. We will hypothesize an ecological niche in state space for such a system as the task subspace. These are a collection of states which, for whatever reason, identify crucial events in the universe. Such events in biological systems usually correspond to survial – such as the states abstractly indicative of food or danger.

We present such a depiction of an abstract system to discuss properties of systems which directly effect the theoretical limits of the transform from perception to action. Systems employing logical sensors and logical actuators whose state space description consists of disjoint perceivable and actuatable spaces do not have the ability – regardless of training, coaching, teaching, or otherwise cajoling – to effectively map perception to action. The measurable states, in this case, will not correlate to the states from which meaningful behavior can be derived. But, consider the case when the intersection of the perceivable and actuatable spaces is not empty. This condition suggests that certain properties of the universe can be measured, and that this perceived environment can be used to stimulate action such that perceivable state changes occur. Our system now has a region in state space which, in theory, permits abstract goals to be expressed. The proximity of these goals to perceived states can be determined and state changes can be selected which minimize the *distance* to the goal.

At this point in the discussion – rather than digressing into the role of random mutation and evolution in optimizing the task space niche of biological systems – we will instead turn our attention to the design of robotic systems. A (presumabley) human designer typically selects appropriate logical sensing and actuation mechanisms with which to express the desired task domain in an uncertain and partially unpredictable universe. We will call the process the design of *logical behaviors* and will draw on research in robotic control which is cognizant of the principles illustrated above.

Many approaches to multisensor integration have been proposed ranging from low-level descriptions of geometric data sensors[28,29,30] to high-level schemes[1,51,77]. Alternatively, one can focus on the sensors[48] or particular applications[2,6].

Our recent work has focused on a mid-level problem: the organization and integration of sensing in terms of intermediate level types of behavior – that is, activities which are not reflex, but which for the most part are not directly coupled to high-level "intelligent" behavior[45]. An example of such behavior is obstacle avoidance in a mobile robot. Here, data must be integrated from cameras, sonars, and perhaps other sensors as well. However, this function must be performed in an ongoing and automatic way. This is a learned behavior.

For the most part, the types of behavior involved are egocentric, i.e., they maintain spatio-temporal relations between the robot and the world. Our analysis is organized in terms of robot goals and behavior. This is accomplished by the use of what we call: **logical behaviors**. This approach allows for active control and integratoin of multisensor information in the framework of a specific task. We provide examples of the application of these ideas to impedance control, dextrous manipulation and mobile robots.

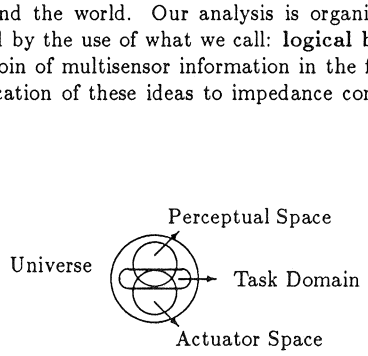


Figure 1. The Task-Perceptor-Actuator Trilogy

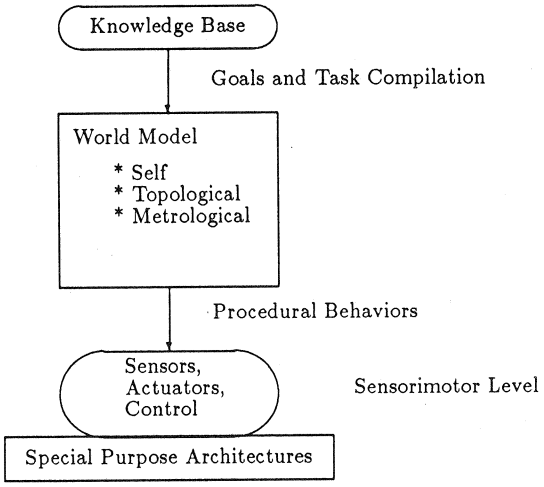


Figure 2. Autonomous Robot Research

2 Background

Multisensor integration has received a good deal of attention in recent years due to the availability of sensors, actuators, and processors. Two major testbeds for such work are:

- robotic workcell automation, and
- mobile autonomous robots.

The first of these involves applying strong knowledge-based techniques to the manufacturing environment, while the second concerns integrating several levels of information processing into a single autonomous system. We restrict our attention here to dextrous manipulation and mobile robots.

Autonomous mobile robots have been studied in a wide range of contexts. Figure 2 imposes an organization on most of the typical keywords. Obviously, the problem of navigation is basic to mobile robots and consequently has been studied by many people on specific implementations [20,22,40,44, 54,59,70,84,85,89,92]. Most such systems must use sensors (e.g., sonar or cameras [26,31,33,91]) and actuators and must control them [32,52,69,75,94,95]. The use of sensors requires the study of uncertainty management [43,78,83,86] and multisensor integration [21,30,50,65,67,72]. More global approaches to the sensorimotor problem can be found in [1,11,39], and special purpose architectures are being planned [4,88,97].

One level up, the mapping of procedural behaviors onto the sensorimotor control structure is of interest [3,5,14,46,47,48,49,93]. The world representations also exist at this level: both the metrological [7,8,73,80], where precise measurement is paramount, and topological [10,12,19,35,36,37,38,41,55,63,68, 87,90,98], where adjacency relations are useful for path planning, etc. It is even possible to study primitive forms of learning in this context [82,90].

Broader studies are usually oriented towards particular applications (e.g., the nuclear industry [17, 96], road following [23,24,74]) or towards well-defined, but limited goals (e.g., indoor [13,34] or outdoor [60,64] navigation).

Finally, the 'highest' level involves the specification and representation of the knowledge appropriate to a given task [18,57,58,62,76] and its compilation into executable robot behavior (or programs) [27,

51,56,61]. The literature is quite large on most of these subjects, and these references are intended as representative of the work in this area. It should be pointed out that most system designers use a central blackboard and some form of direct production system or a compiled version (i.e., a decision tree) to represent knowledge.

From this short summary, it can be seen that the scope of autonomous robot research is indeed vast, but the difficult problems found here are yielding to the steady advance of technical and theoretical developments. In the remainder of this paper, we describe current work on the mobile autonomous robot at INRIA.

3 Behavior Based Sensing and Control

In the most general sense, a robot interacts with its environment by applying operators to the perceived state of the environment. The state and operator may be cognitive - effecting the composition of state parameters without physically altering the environment; on the other hand, elements of the robot's surface may actually be applied to the geometry of the environment. In the latter case, the contacts may be derived from the robot's wheels or bumpers in the case of a mobile cart, or from the fingertips, proximal phalanges, palm or arm of a dextrous manipulator. Characteristics of the environment, the task, and the robot kinematics may be used to construct goal oriented behaviors.

The development of controllers for robotic systems is typically a generalization of the approach used in low level feedback controllers. Elements of the system state are measured and used to quantify the error of the system with respect to a desired state. The operation of the system tends to reduce the state error to zero. The nature of the feedback variables determines the nature of the response. Adaptable control schemes can optimize the response over uncertain inputs by varying the weighting of the feedback variables; however, types of behaviors which are not defined *a priori* cannot be expressed. This suggests that a single control law is not sufficient to manage the complexity of the general purpose robot systems. Control methodologies have been developed which partition the state space of complex systems into disjoint regions, each with an associated control law [79]. The operation of these systems is represented by a finite state automata where state transitions are triggered by sensory events. This approach produces sequences of behaviors in the system.

Behavior based control schemes generalize this approach. Elemental behaviors are instantiated which span the problem domain (see Braitenberg[14]). Braitenberg's work was the precursor of many similar systems, including the subsumption architecture proposed by Brooks[15]. The subsumption architecture is an approach which was developed to construct systems which require composite behaviors[16,15]. Concurrent control laws are defined, each of which acquiring the sensory data necessary for that particular behavior. The so-called *activity producing subsystems* are integrated in a hierarchy in which primary behaviors reside at the lowest levels. Higher level behaviors are used to modulate the output produced by low level behaviors. The work demonstrated a hardwired system tuned to perform a particular task, the navigation of an autonomous vehicle.

These approaches are generalized in the society structures proposed by Minsky[71]. This proposed structure is motivated by observed problem solving behaviors in humans. Agencies are postulated which serve as *proto-specialists* over limited problem domains. The society of such agents is capable of goal oriented behaviors with dynamic priorities based on the current state of the composite system.

The logical behavior systems proposed in this paper are based on a similar perspective. Independent, elemental behaviors are defined which span the required problem domain. We generalize the notion of a behavior to any process which maps information abstracted from (logical) sensors to state transitions which may be mapped onto (logical) actuators. Once again, a logical sensor need not be linked directly or indirectly to a physical sensor, but may represent any hypothetical state from which a state transition is desired. Likewise, the logical actuator need not employ a DC motor, for example, but will transform a state in the actuatable space to some other state. This property provides a mechanism for cognitive and reflexive mappings from sensors to actuators. We also note that the distinction between planning and execution is in some sense a function of whether the logical sensor is in fact terminated at a physical sensor, and whether the logical actuator terminates at a physical actuator.

We describe below the application of this approach to dextrous manipulation and to mobile robots. The example of the design of logical behaviors for multifingered manipulator control includes complex kinematics, multi-functional mechanisms, and complex tasks. The other application is the design of an obstacle avoidance behavior for a mobile robot. We will make an effort to keep our primary focus on the even larger problem domain describing general transformations from sensors to actuators.

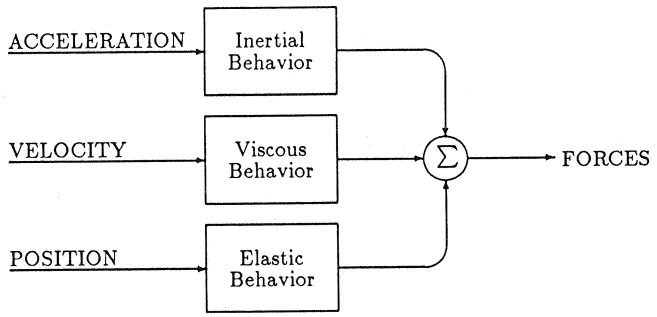


Figure 3. Three Logical Behaviors Which Constitute the Impedance Controller

4 Logical Behaviors for Generalized Impedance Control

Drake designed a passively compliant device which automatically compensates for uncertainty in certain classes of assembly operations[25]. Salisbury employs a stiffness controller to support the construction of stable grasps[81], and Lozano-Pérez *et al.* discuss the use of the generalized damper to plan fine motion assembly strategies[66]. These instances of manipulator control map sensor data to action by modeling the manipulator as an impedance relative to an environmental admittance. As such, the manipulator measures deviations from nominal positions or velocities and applies a correcting force.

To illustrate the use of logical behaviors for multisensor integration, consider an implementation of Hogan's impedance control[53]. Hogan argues that to:

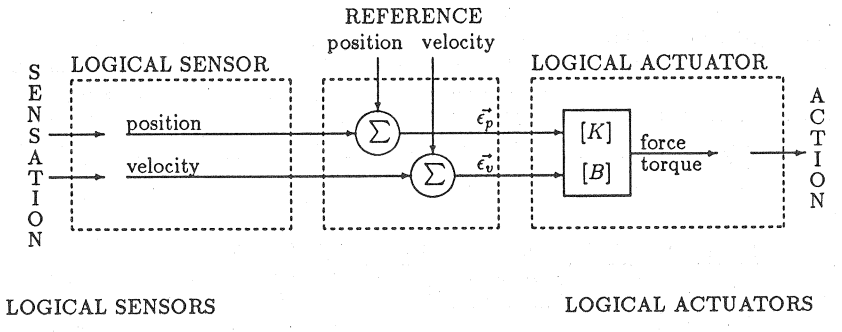
> ensure physical compatability with the environmental admittance, something has to give, and the manipulator should assume the behavior of an impedance.

Others have noted the usefulness of impedance control - loosely defined for our purposes as behaviors which map errors in position, velocity, or acceleration to forces. The terms in the impedance controller are linearly independent functions of separable state variables, $\vec{x}, \dot{\vec{x}}$, and $\ddot{\vec{x}}$. Figure 3 illustrates how the impedance controller may be considered to be a superposition of three separate impedance behaviors: an inertial behavior, a viscous impedance behavior and an elastic impedance bhavior.

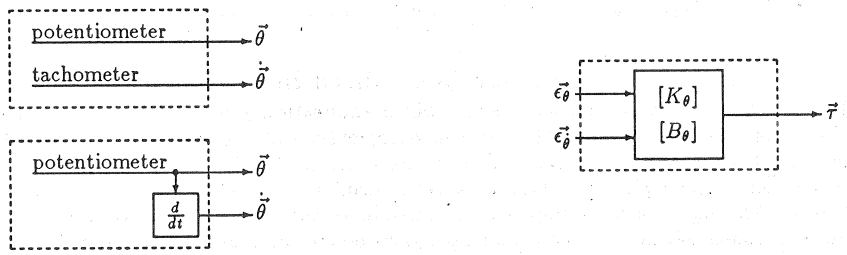
The remainder of the presentation will consider only the visco-elastic components of the generalized impedance controller. The structure of each of these logical behaviors consists of a logical sensor, an (optional) reference input, and a logical actuator. The logical sensor is any combination of hardware and software which measures and/or hypothesizes the state of the system[48]. The logical actuator is likewise, any combination of hardware and/or software which transforms the state representation. The logical actuator may simply transform the abstracted state representation, or it may actually employ hardware actuators to physically change the state of the system. To understand the utility in an abstract notion of the logical impedance behaviors, consider the various incarnations of the visco-elastic impedance controllers presented in Figure 4.

The figure defines the constraints which govern the construction of a logical behavior. In this case the behavior describes a transformation from the position/velocity domain into the force/torque domain. The behavior is represented as a combination of a logical sensor and a logical actuator. The generic logical behavior on the top of Figure 4 defines the data type consistency which must be maintained during the construction of the logical impedance behavior. In essence, all data entering or leaving the summation block in Figure 4 must be consistent. We have thus defined the type of the: characteristic output vector (cov) for the logical sensor, the reference input vector (riv), and the characteristic input vector (civ) of the logical actuator.

Figure 4(a) depicts the commonplace joint impedance controller. We illustrate two logical sensors which yield the joint space position and velocities required. The logical actuator simply computes



(a) Visco-Elastic Joint Position Servoes



(b) Visco-Elastic Cartesian Endpoint Control

(c) Visco-Elastic Object Pose Control

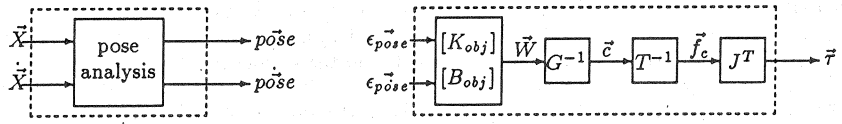


Figure 4. Implementations of Logical Impedance Behaviors

torques which supp
two commonly used
presented in Figure
impedance paramet
somewhat different
and a logical actuat
Cartesian pose imp
the set of Cartesian
The corresponding
restoring wrenches,
well. The inverse of
applied through eac
wrenches, such that:

then,

and, the force comm

Finally, the forces a
appropriate manipul

Beyond the relat
is an instance of the
sensor and logical a
multisensor control.

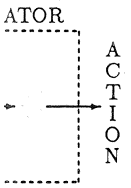
5

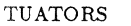
A primary moti
abstraction that is p
will first suggest se
manipulation. The r
a basis for autonom
impedance behavior
that follows. These
[42].

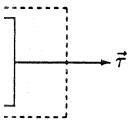
The **wrench clo**
the equilibrium posi

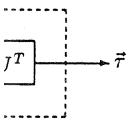
1. the geometry c
2. the type(s) of c
   and
3. the pose of the

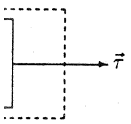
This *sensor* data ma
knowledge of the ma
is expressed analytic
of the object is know
in wrench space whi
object's six degrees
the logical actuators
by manipulator cont
descent in the error
The **isotropic m**
the wishes of the wr

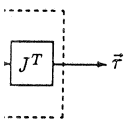












torques which suppress errors in joint space. Figure 4(b) presents logical behavior expressions of two commonly used Cartesian endpoint controllers. The second simply employs the logical sensors presented in Figure 4(a) and a logical actuator which suppresses joint space errors using Cartesian impedance parameters as suggested by Salisbury[81]. The first Cartesian controller shown employs a somewhat different logical sensor which expresses the position and velocity directly in Cartesian space and a logical actuator which supresses errors in Cartesian space. Figure 4(c) presents an object frame Cartesian pose impedance behavior. Another stage is added to the logical sensor which transforms the set of Cartesian contact positions and velocities into an object frame pose and velocity in 6-space. The corresponding logical actuator maps errors in the pose parameters relative to the reference into restoring wrenches, $\vec{W}$. This wrench is in n-space and contains internal (null space) components as well. The inverse of the Grip Jacobian is used to map the restoring wrench into individual wrenches applied through each contact location. If the transform T is defined to map contact forces into contact wrenches, such that:

$$\vec{w}_i = T\vec{f}_i$$

then,

$$\vec{f}_i = T^{-1}\vec{w}_i$$

and, the force commands at each contact are expressed as:

$$\vec{f}_i^c = T^{-1}(\vec{c}^T\vec{w}_i).$$

Finally, the forces applied at each contact location are transformed into actuator torques by way of the appropriate manipulator Jacobian.

Beyond the relative complexity of the impedance controllers presented in Figure 4, each controller is an instance of the same logical behavior. Each instance represents an effective assemblage of logical sensor and logical actuators for some combination of task objectives and constraints in the context of multisensor control.

5 Logical Behaviors for Grasping and Manipulation

A primary motivation for developing the logical behavior formalism is the modularity and data abstraction that is provided in the design of complex robotic controllers. To illustrate this feature, we will first suggest several behaviors which are understood to be useful in the context of grasping and manipulation. The resulting behaviors may be used as a programming language for manipulation, or as a basis for autonomous behavior composition. These behaviors are in fact supported by the Cartesian impedance behaviors described in Section 4. We will employ two principal behaviors in the discussion that follows. These behaviors and others useful in grasping and manipulation are described in detail in [42].

The **wrench closure behavior** is defined to construct 6 dimensional constraint envelopes about the equilibrium position of the object. The logical sensors for this behavior must determine:

1. the geometry of the graspable surface,
2. the type(s) of contact(s) delivered to the object's surface by the manipulator and the environment, and
3. the pose of the object.

This *sensor* data may be derived from a variety of physical sensors, or derived from basic principles and knowledge of the manipulator and the object. In the simulations presented, the geometry of the object is expressed analytically, the contact type is uniformly a point contact with friction model, and the pose of the object is known a priori. The reference input to the logical behavior is a 6-dimensional volume in wrench space which expresses both positive and negative sense wrench magnitudes in each of the object's six degrees of freedom. This *goal* represents a stable grasp objective. The error submitted to the logical actuators represents the difference between the current state of the wrench volume generated by manipulator contacts and the reference input. The logical actuator in this case performs a gradient descent in the error space toward suitable wrench closure states.

The **isotropic manipulator behavior** is designed to condition the manipulator to be compliant to the wishes of the wrench closure behavior. This behavior employs a logical sensor to compute a scalar

manipulability index for the hand. This state description is derived from the kinematic configuration of all the fingers in a grasp. The scalar index is normalized so that the reference goal for this behavior is implicitly to generate a unity index for the hand. This behavior performs a gradient ascent in the manipulability index toward isotropic hand states.

Logical Sensors	*Logical Actuators*	
Object Geometry	Closure Gradient Descent	contact positions
Contact Positions	Hand Index Gradient Ascent	position/orientation

Examples are presented for cylindrical and spherical test objects. All examples employ the Utah/MIT hand geometry. A top view and a side view of the hand/object system are presented. Intermediate positions for the hand frame y- and z-axes are shown. The initial hand frame position is identified in bold print; for clarity, only the final finger configuration is shown. Typical computation time for the four fingered grasps is approximately 10 *seconds* of CPU time on a VAX 750.

The original task stack submitted to the system for the initial grasp task is illustrated in Figure 5.

type:	approach
screw task:	NA
contact set:	all fingers
nomadic contacts:	all fingers
↓	
type:	condition
screw task:	hypercubic wrench volume
contact set:	all fingers
nomadic contacts:	all fingers
↓	
NIL	

Figure 5. The Task Stack Representation of the Initial Grasp Task

Figures 6, 7 and 8 present the grasps of the cylinder produced when 2, 3 and 4 fingertip contacts are applied to the task, respectively. In Figure 8, the third finger (the ring finger) does not oppose the thumb, but goes to a position where it more effectively complements the wrench subspaces produced by the other contacts. In (b), the task is modified dynamically by adding an incremental task which is the wrench subspace produced by the index finger. The trajectories of the other fingers are then tethered elastically to the index finger. The fingers are essentially grouped into a virtual finger. Which grasp is better for the robot hand is not clear, but (b) appears more anthropomorphic.

Figures 9, 10 and 11 present the grasps of a sphere produced when 2, 3 and 4 fingertip contacts are applied to the task, respectively. In Figure 11 the super symmetric object and the functionally redundant manipulator produced convergence difficulties in the geometry synthesis. There appear to be closely spaced meta-stable states in the contact geometry solution. The convergence problem was controlled by designating a virtual finger over the middle and ring fingers of the hand; the result is not intuitively satisfying. This example suggests the application of additional constraints in the form of virtual fingers and/or geometrical restrictions limiting the portion of the object surface which may be used to address a particular task.

6 Logical Behaviors for Obstacle Avoidance

In this section we consider the following problem: suppose that our mobile robot is wandering through an unknown indoor environment. The robot must:

- **incrementally build a 3-D representation of the world** (i.e., determine its motion and integrate distinct views into a coherent global view),
- **account for uncertainty in its description** (i.e., explicitly represent, manipulate and combine uncertainty), and
- **build a semantic representation of the world** (i.e., discover useful geometric or functional relations and semantic entities).

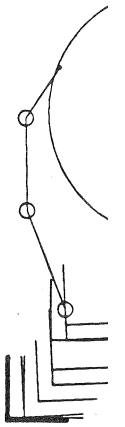


Figure 6. A Two F
Volume in Six DO

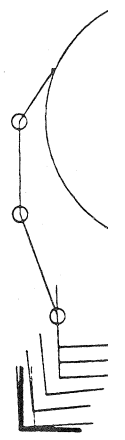


Figure 7. A Three
DOF

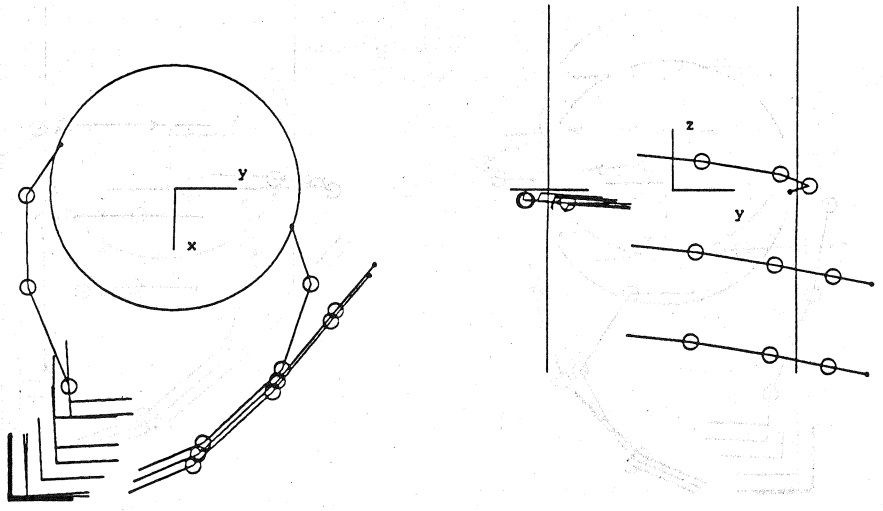


Figure 6. A Two Fingered Grasp on a Cylinder 4 *inches* in Diameter; the Task is a Hypercubic Wrench Volume in Six DOF

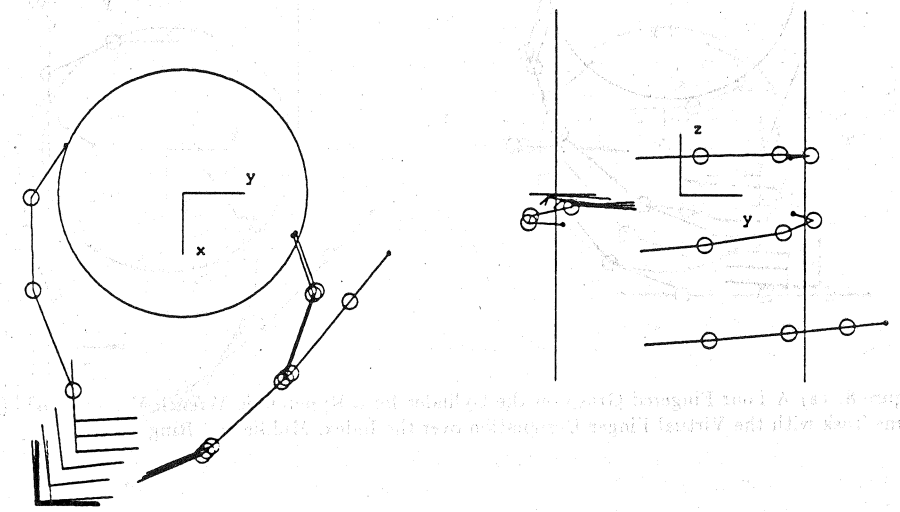


Figure 7. A Three Fingered Grasp on the Cylinder; the Task is a Hypercubic Wrench Volume in Six DOF

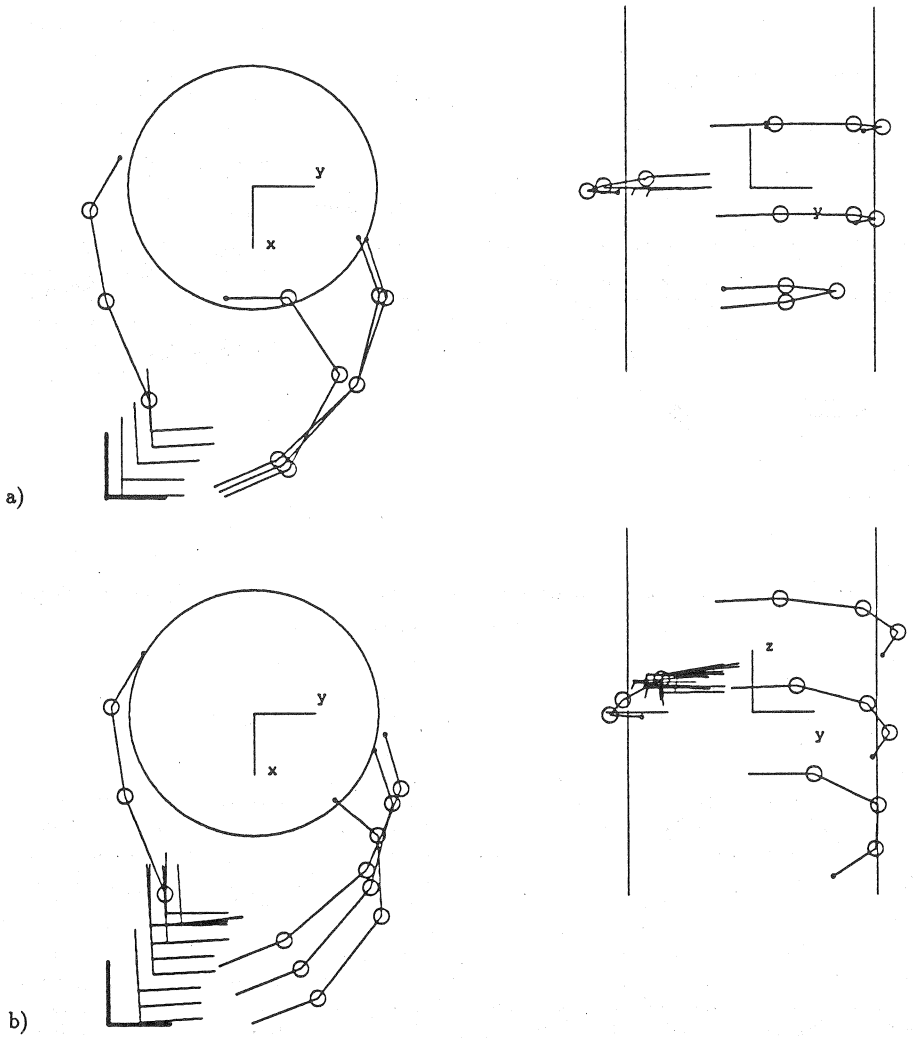


Figure 8. (a) A Four Fingered Grasp on the Cylinder for a Hypercubic Wrench Volume Task; (b) the Same Task with the Virtual Finger Designation over the Index, Middle and Ring Fingers

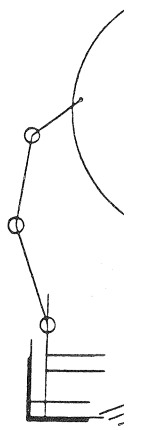


Figure 9. A Two I
Volume in Six DO

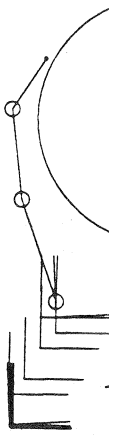


Figure 10. A Three
DOF

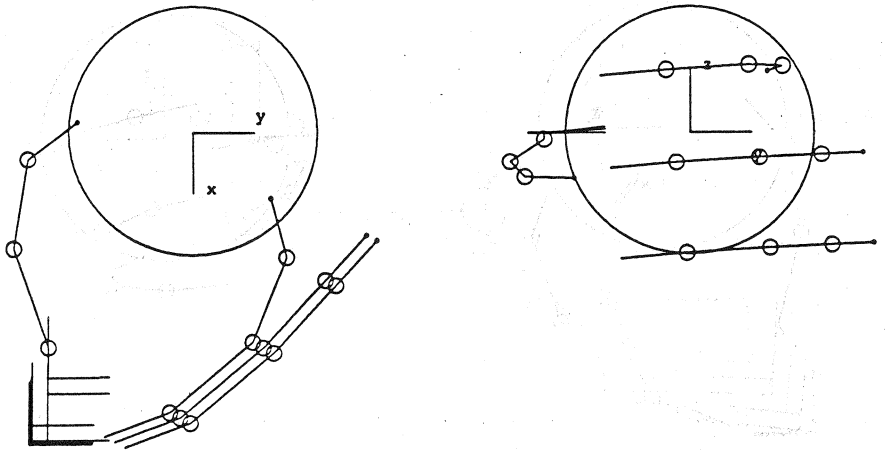


Figure 9. A Two Fingered Grasp on a Sphere 4 *inches* in Diameter; the Task is a Hypercubic Wrench Volume in Six DOF

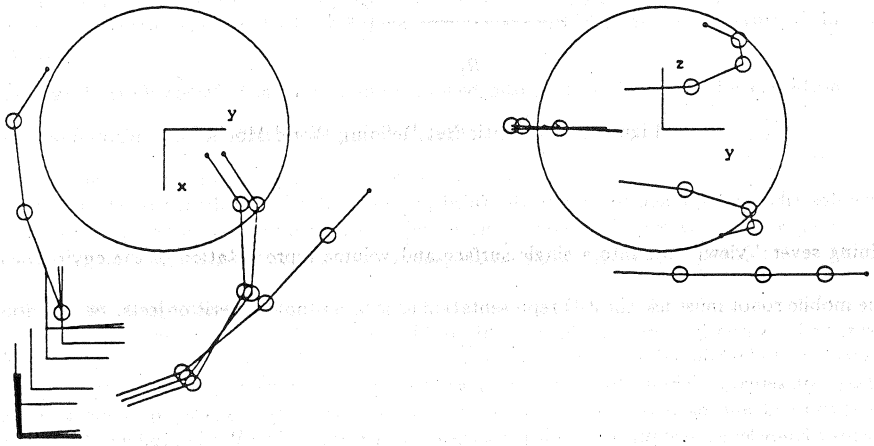


Figure 10. A Three Fingered Grasp on the Sphere; the Task is a Hypercubic Wrench Volume in Six DOF

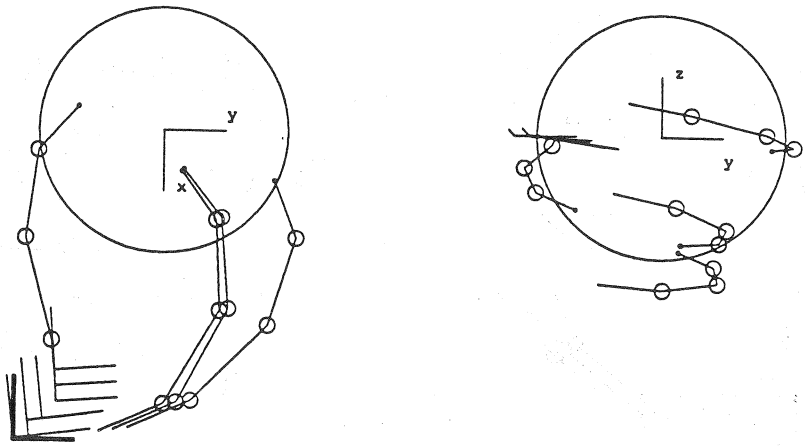


Figure 11. A Four Fingered Grasp on the Sphere

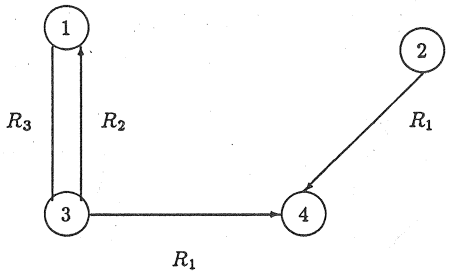


Figure 12. Semantic Net Defining World Model

Here we describe an approach to solving the third problem. (See [6] for details on efficient techniques for producing a local 3-D map from stereo vision and structure from motion as well as a method for combining several viewpoints into a single surface and volume representation of the environment and which accounts for uncertainty.)

The mobile robot must use the 3-D representation to locate simple generic objects, such as doors and windows, and eventually more complicated objects like chairs, desks, file cabinets, etc. The robot can then demonstrate task-based behavior such as going to a window, finding a door, etc. The representation should contain semantic labels (floor, walls, ceiling) and object descriptions (desks, doors, windows, etc.).

The proposed approach is straightforward and exploits our previous work on logical sensors, the Multisensor Knowledge System, and multiple semantic constraints. The World Model is defined in terms of a semantic network (e.g., see Figure 12). The nodes represent physical entities and the relations are (currently) geometric. "Behind" each node is a logical sensor which embodies a recognition strategy for that object. The relations are simply tabulated.

A goal for the robot is defined by adding a node representing the robot itself and relations are added as requirements (see Figure 13). This method permits the system to focus on objects of interest and to exploit any strong knowledge that is available for the task. The added relations are satisfied (usually) by the robot's motion.

As an example, consider the world model in Figure 14 which represents a specific office. The addition of the robot and the "Next_to" relation fires the "Find_door" logical sensor. This in turn causes the strategy for door finding to be invoked. Such a strategy may attempt shortcuts (quick image cues) or may cause a full 3-D representation to be built and analyzed. Logical behaviors are then the combined

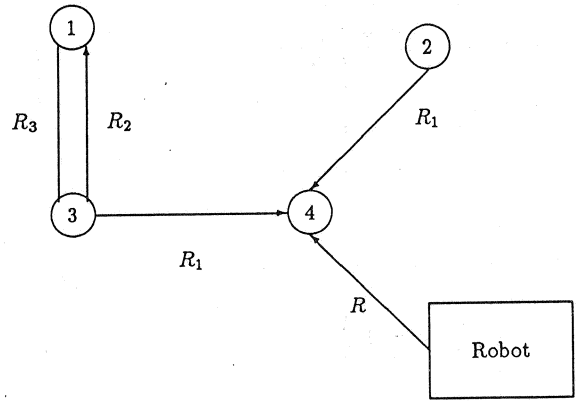


Figure 13. Defining the Robot Task

logical sensors and motion control required to satisfy the "Next_to" relation.

Note that it is in the context of such a strategy that high-level multisensor integration occurs in goal-directed behavior. We are currently implementing a testbed for experimentation.

6.1 Robot Behavior as Real-time Programming

Robots must maintain a permanent interaction with the environment, and this is the essential characteristic of *reactive programs*. Other examples include real-time process controllers, signal processing units and digital watches.

We have selected the Esterel synchronous programming language[9] as the specification language for the reactive kernel of the robot's behavior. A reactive system is organized in terms of three main components:

- **reactive kernel**: specified in Esterel and compiled into C or CommonLisp for execution,
- **interface code**: handles drivers and sensors, and
- **process or data handling code**: routine calculations.

The programs produced are:

- **deterministic**: produce identical output sequences for identical input sequences,
- **concurrent**: cooperate deterministically, and
- **synchronous**: each reaction is assumed to be instantaneous.

Interprocess communication is done by instantly broadcasting events, and statements in the language take time only if they say so explicitly; for example:

every 1000 MILLISECOND **do emit** SECOND **end**

In this example, a SECOND signal is sent every thousand milliseconds.

Thus, Esterel provides a high-level specification for temporal programs. Moreover, the finite state automata can be analyzed formally and give high performance in embedded applications. They help encapsulate the specification of sensing and behavior from implementation details. This simplifies simulation, too.

Other advantages include the fact that synchrony is natural from the user's viewpoint; e.g., the user of a watch perceives instant reaction to pushing a control button on the watch. Synchrony is also natural to the programmer. This reconciles concurrency and determinism, allows simpler and more rigorous

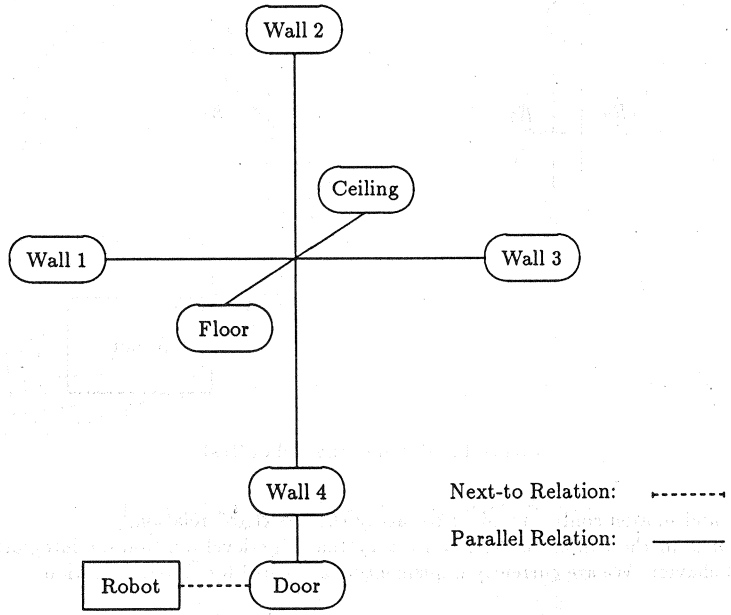


Figure 14. A Representation of the Command: "Go to the office door"

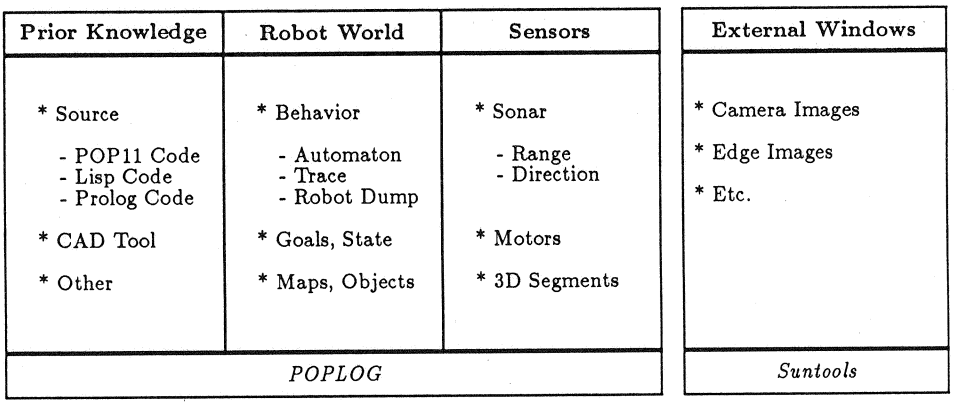


Figure 15. The Debugging System Organization

programs and separates logic from implementation. Finally, such automata are easily implemented in standard programming languages.

Details of the language are not given here; however, a brief summary is in order:

- **variables**: not shared; local to concurrent statements.
- **signals**: used to communicate with environment or between concurrent processes; carry staus (present or absent) and value (arbitrary type).
- **sharing law**: instantaneous broadcasting; within a reaction, all statements of a program see the same status and value for any signal.
- **statements**: two types:
  1. standard imperative style, and
  2. temporal constructs (e.g., **await** *event* **do**).

An extremely useful output from Esterel is a verbose description of the automaton. This can be used for debugging purposes. Esterel also produces a C program which implements the automaton.

Another useful output is a CommonLisp version of the automaton. This makes simulation straightforward, so long as reasonable functions can be written which simulate the world and the physical mechanisms of the robot. But these, too, can be specified in Esterel and then combined.

6.2 Robot Behavior Debugging Environment

In developing Esterel specifications for robot behavior and sensor control, we are faced with the problem of integrating diverse kinds of knowledge and representations. In particular, debugging robot behaviors requires knowledge of the world model, the robot's goals and states, as well as the behavior specification, and sensor data (intensity images, sonar data, 3-D segments, etc.). Figure 15 shows the current implementation organization. We use POPLOG (an interactive environment which combines CommonLisp, Prolog and Pop11) to support manipulation and display of prior knowledge, the robot world, and some sensor data, while other Suntool-based utilities support display of the trinocular stereo camera images, etc.

Figure 16 shows a representative collection of windows which provide:

- POPLOG source code (window management, etc.)
- Prolog source (semantic entity definition; e.g., walls, doors, etc.)
- sensor data display (e.g., sonar range data, 3D segments)
- Esterel generated automaton (e.g., COMBINE.debug)

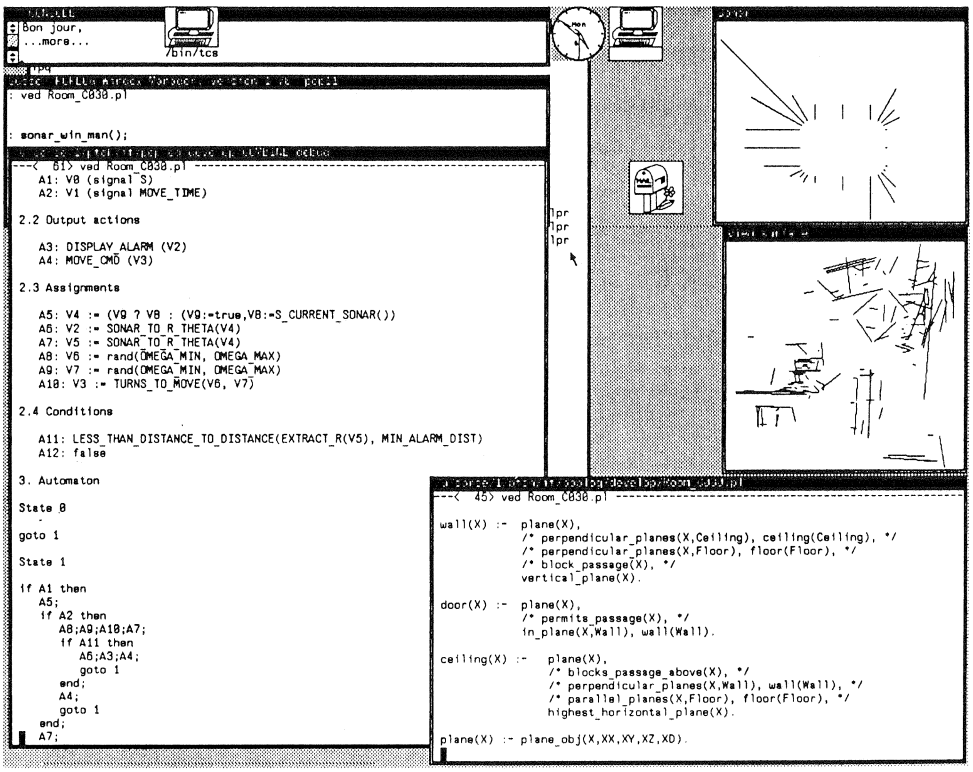


Figure 16. Collection of Windows for Debugging

Esterel permits state tracing during execution, and this combined with access to the robot's sensory data permits rapid and accurate debugging. In Appendix A we give the details for the specification of a wandering robot which must avoid colliding with objects in the world. This specification has been compiled and loaded onto the robot and successfully executed.

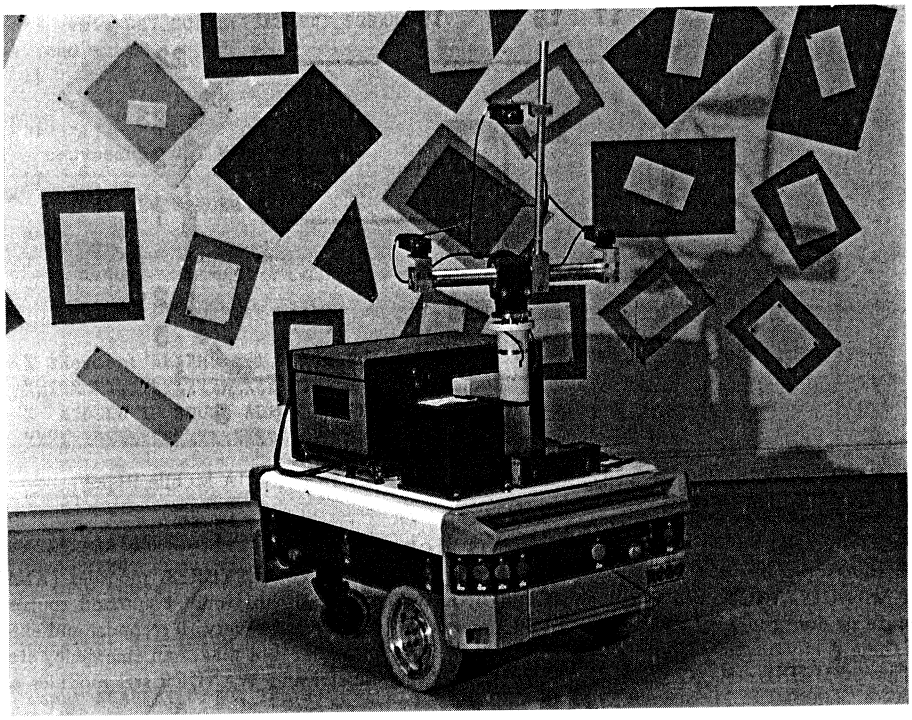


Figure 17. The INRIA Mobile Robot

6.3 Mobile Robot

There is an operational mobile robot at INRIA (Sophia-Antipolis). It is similar to other mobile robots (e.g., like those at CMU or Hilare at LAAS). Figure 18 shows the geometry of the robot (length: 1.025m, width: .7m, and height: .44m) and the locations of the sonar sensors. The two rear wheels drive the robot.

The onboard processing consists of two M68000 series microprocessors on a VME bus; one controls the sonar sensors, and the other runs the real-time operating system, Albatros. The two main wheels are controlled separately, and the system has an odometer.

High-level multisensor integration must be investigated in the context of real-world problems. We have described current work on an autonomous mobile vehicle under development at INRIA. We propose "logical behaviors" as an approach to robot goal representation and achievement.

We intend to continue development of algorithms, architectures and systems for multisensor robotic systems. Moreover, we are currently investigating the simulation of such systems; this involves embedding the reactive kernel in a modeled robot world. Finally, as can be seen by the rough nature of the definitions of walls, doors, etc., we must develop a suitable formal model of the world in which the robot finds itself. We intend to exploit optimized refinements of conceptual clusters defined in first order predicate calculus.

7 Summary

We have proposed **logical behaviors** as a technique for organizing multisensor integration and control. Such an approach allows encapsulation of sensing and actuation and relates those activities to a specific task. Several examples have been presented ranging from impedance control for a robot manipulator to obstacle avoidance for a mobile autonomous robot. We are currently extending the environment to include better debugging and simulation environments.

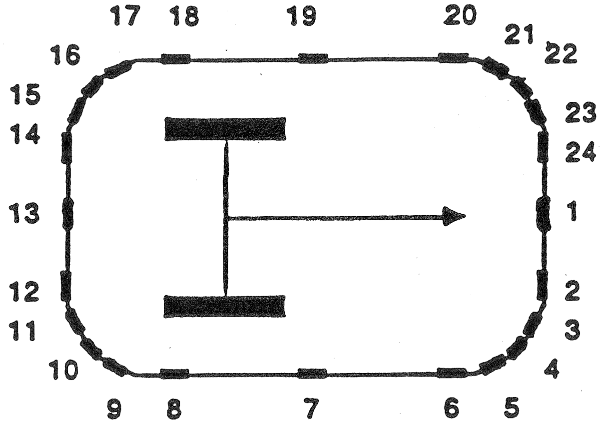


Figure 18. The Geometry and Sensor Placement on the INRIA Mobile Robot

A Wandering Robot Example

In this appendix, a system is developed which combines several ESTEREL modules (ALARM, GET_MIN_DISTANCE, WANDER and COMBINE) with the on-board robot command routines to generate random robot movement. The robot generates a random move every 10 seconds, and executes it; if there is any obstacle closer than the predefined threshold, the robot makes an emergency stop.

The COMBINE.strl, ALARM.strl, GET_NEAREST_OBJ.strl and WANDER.strl modules are as follows:

```
% $Header: COMBINE.strl,v 1.1 88/12/07  tch Locked $
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  MODULE TO COMBINE READING WITH WATCHING THE SONAR SENSORS%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

module COMBINE:

type DISTANCE,
     PING,
     R_THETA,
     MOVE;

constant OMEGA_MIN, OMEGA_MAX : integer;

input S;
input MOVE_TIME;

relation MOVE_TIME => S;

sensor CURRENT_SONAR (PING);

output DISPLAY_ALARM(R_THETA);
output MOVE_CMD(MOVE);

[
signal GET_SONAR(PING), NEAREST_OBJ(R_THETA) in

  every S do
```