$\mathcal{Z}^{\infty}$: A Framework for Autonomous Agent Behavior Specification and Analysis

Thomas C. Henderson and Patrick Dalton

UUCS-90-018

Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

14 September 1990

Abstract

We describe a framework for the specification and analysis of autonomous agents[1]. In general, such agents require several levels of behavioral specifications, including: low-level reflex actions, mid-level controllers to deal with the physical aspects of the world, and high-level representations for goals and plans. In this study, we focus on the mid-level control problem (e.g., obstacle avoidance, grasping, etc.) and explore:

1. the use of real-time programming languages as behavior specification tools,
2. the use of formal techniques to prove properties of such specifications, and
3. the physical-based simulatoin and animation of such specifications.

[1]This work supported in part by NSF grants IRI-88-02585 and INT-89-09596.

1 Introduction

The development of robust and capable sensor-based autonomous systems requires the specification of several levels of behavior:

1. low-level servo loops,
2. mid-level reactive processes, and
3. high-level concept driven plans.

Control theory addresses the issues of low-level controller design, and artificial intelligence techniques have been proposed for high-level planning. However, mid-level control has only recently started to receive attention.

For example, our recent work has focused on the organization and integration of sensing in terms of intermediate levels of behavior – that is, activities which are not reflex, but which for the most part are not directly coupled to high-level "intelligent" behavior[10]. 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 is performed in an ongoing and automatic way.

Usually, the types of behavior involved are egocentric; i.e., they maintain spatio-temporal relations between the robot and the world. 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 eventually be used to construct goal oriented behaviors.

Our goal then is to study the specification of behaviors for autonomous agents, and from such specification (input/output or stimulus/response) models of agents, to produce a space/time trace of position, resources and properties (of agents, objects, or feature spaces) which can provide a basis for animation (generation of images), analysis of systems, and design cycle feedback.

1.1 Previous Work

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 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 [21]. The operation of these systems is represented by a finite state automaton 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[3]). Braitenberg's work was the precursor of many similar systems, including the subsumption architecture proposed by Brooks[4]. The subsumption

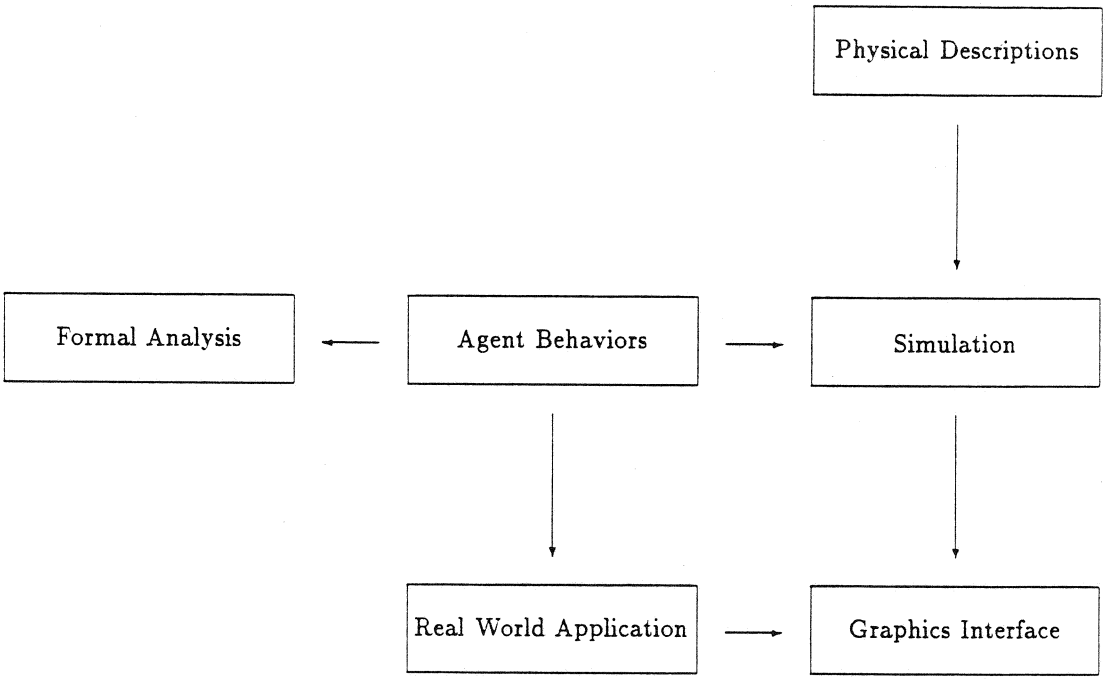


Figure 1: The $\mathcal{Z}^{\infty}$ Framework

architecture is an approach which was developed to construct systems which require composite behaviors[5, 4, 6]. Concurrent control laws are defined, each of which acquires 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. Brooks' work demonstrated a hardwired system tuned to perform a particular task, the navigation of an autonomous vehicle.

Several other researchers have been trying to develop ways to specify such multi-agent systems. Overton[20] proposed schemas and that idea has been further pursued by Arkin[1]. Cox has proposed a concurrent robotic programming environment[7]. Kaelbling at SRI has worked on an architecture for intelligent reactive systems[15], but the major motivations there were to increase modularity, awareness, and robustness. A very similar project has been reported by McKenna, Pieper and Zeltzer[19] in which the motor control of a cockroach was modeled. However, none of these approaches permit any formal verification or reasoning over the components individually or over the combined system.

1.2 Our Work

We have developed a framework (called $\mathcal{Z}^{\infty}$) which permits the study of real-time intelligent controllers (behaviors). Figure 1 shows the components of the system. The realization of this framework requires the development and interfacing of several representations, e.g., agent behaviors, physical descriptions, aspects of simulation, etc. The components of the system are as follows:

- **Physical Desciption**: Each object in the world (and the world itself) needs a description of physical properties such as mass, position of sensors and position of motors for a vehicle, position in the world and intensity for a source, or the dimensions of the world and description of global physical properties such as friction, wind resistance and gravity.
- **Simulation**: Each object in the world has a characteristic motion that may be changing with time and in the presence of other objects. A force based simulator provides a means of realistically moving the objects based on the forces being applied to them internally or externally, resolving any collisions and updating the data incident on the sensors of the objects in the world.
- **Agent Behavior**:

Each agent has a formal specification of its behavior in the world. It describes the types of sensors (physical or logical) the agent's behavior is based on, as well as the responses it makes. The agent's behavior ultimately resolves into forces applied to itself or its world so that it can communicate with the simulator. However, with the use of logical sensors, the specification of the agent's behavior need not be at the force level. The specification is in a real-time programming language (Esterel).

- **Formal Analysis**:

  Since the behavior of an agent is given formally, there are formal methods for analyzing the specification. This analysis is useful for asking questions such as "Does the agent succeed in performing its behavior all the time?" and "Does the behavior fulfill these requirements?"

- **Real World Interface**:

  The agent's formal behavior specification may be given to its real world counterpart and tested in a real environment. It is another means of studying the specification of the reactive behaviors.

- **Graphics Interface**:

  The data generated by the simulator as an agent reacts to its environment must be displayed graphically. The enormous amount of information is almost incomprehensible unless displayed as a picture of the world. The interface also allows multiple views of the same data: world, agent, or sensor. The latter two are particularly useful for debugging when used in combination with a real-world interface.

There are two main issues to be addressed in this work:

1. Are there preferred approaches to performing the simulation?
   - How should the physical properties of the agents be handled?
   - How can an agent's behaviors be related to its physical components?
   - How should the physics be modeled and incorporated?
   - How can one mix simulation with real performance; for example, to help in debugging?
2. Are real-time programming languages useful for robot behavior specification?
   - Do they provide the necessary descriptive power?
   - Is formal analysis useful?
   - Does such analysis plug into simulation easily?
   - Do such specifications make composition of behaviors easier or more understandable?

We have already begun development of the framework, and are applying it to several applications of particular interest to us:

- **the INRIA mobile robot**: We are currently working on an NSF-INRIA project concerning the representation, interpretation and control of a robot's world. Our part of the project concerns developing robot behaviors. The framework will provide us the capability of testing these behaviors and studying their properties.
- **the Utah/MIT Dextrous Hand**: We are also working on the problem of dextrous manipulation. In our formulation of a solution, multiple finger agents are reacting in parallel producing an overall grasping behavior.

- the **CREATURES world**: We are exploring various aspects of the framework by specifying multiple agents reacting to stimuli (and each other) in a physically-based simulation world. This allows us to explore most of the issues of interest without being tied to a physical application.

2 Agent Behaviors

2.1 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 are currently using the Esterel synchronous programming language[2] 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 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 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 status (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.

2.2 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.). Esterel permits state tracing during execution, and this combined with access to the robot's sensory data permits rapid and accurate debugging.

We have designed and built the graphical interface so that it can be connected to either the simulation or to drivers which interface it to real applications. Other useful debugging tools that should be developed include some which permit the components of the framework to be examined; e.g., given a physical description, draw it out graphically to show the user the placement of wheels, sensors, etc.

3 Applications

We have several applications underway. These are described below:

- **INRIA 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). The geometry of the robot is: length: 1.025m, width: .7m, and height: .44m, and there are 24 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.

- **Dextrous Hand** - We have been developing a multi-agent approach to dextrous manipulation over the last few years[8, 9, 10, 13, 14], and have now reached the point where it is crucial to be able to specify several independent agents (or prerogatives) and verify formally their combined behavior. Although individual components may be well understood, it is essential to develop a theory to explain what determines the behavior of the concurrent set of autonomous elements.

  There is a vast amount of information to be processed and acted upon during a dextrous manipulation task. This ranges from knowledge about the object which can be derived either from CAD data or from sensors mounted on and about the robot, to knowledge of the task which may in fact be a function of time. Moreover, the realtime constraints imposed upon such a system make it necessary to minimize the computation and control.

  The simulation of the Utah/MIT Dextrous Hand requires that robot linkages be represented within the system, as well as their dynamics. Once this is accomplished, it will be possible to analyze dextrous manipulation strategies.

- **CREATURES World** - Creatures world is a hypothetical world in which many varied and strange creatures live and interact. It is a world where we can explore the different behaviors an autonomous agent needs to "survive" the perils of the world in which it lives. It also frees us from the constraints of having to rigorously enforce a specific real application's environment. In this virtual world we are free to create agents that appear to be fish, birds, or even bugs. This environment gives a testbed for studying the effects of different methods of describing and combining behaviors, and is a place to study how different behaviors interact with each other.

3.1 Prototype Implementation

A prototype system has been developed, and several examples run. We discuss three of them here.

3.1.1 Braitenberg Vehicles

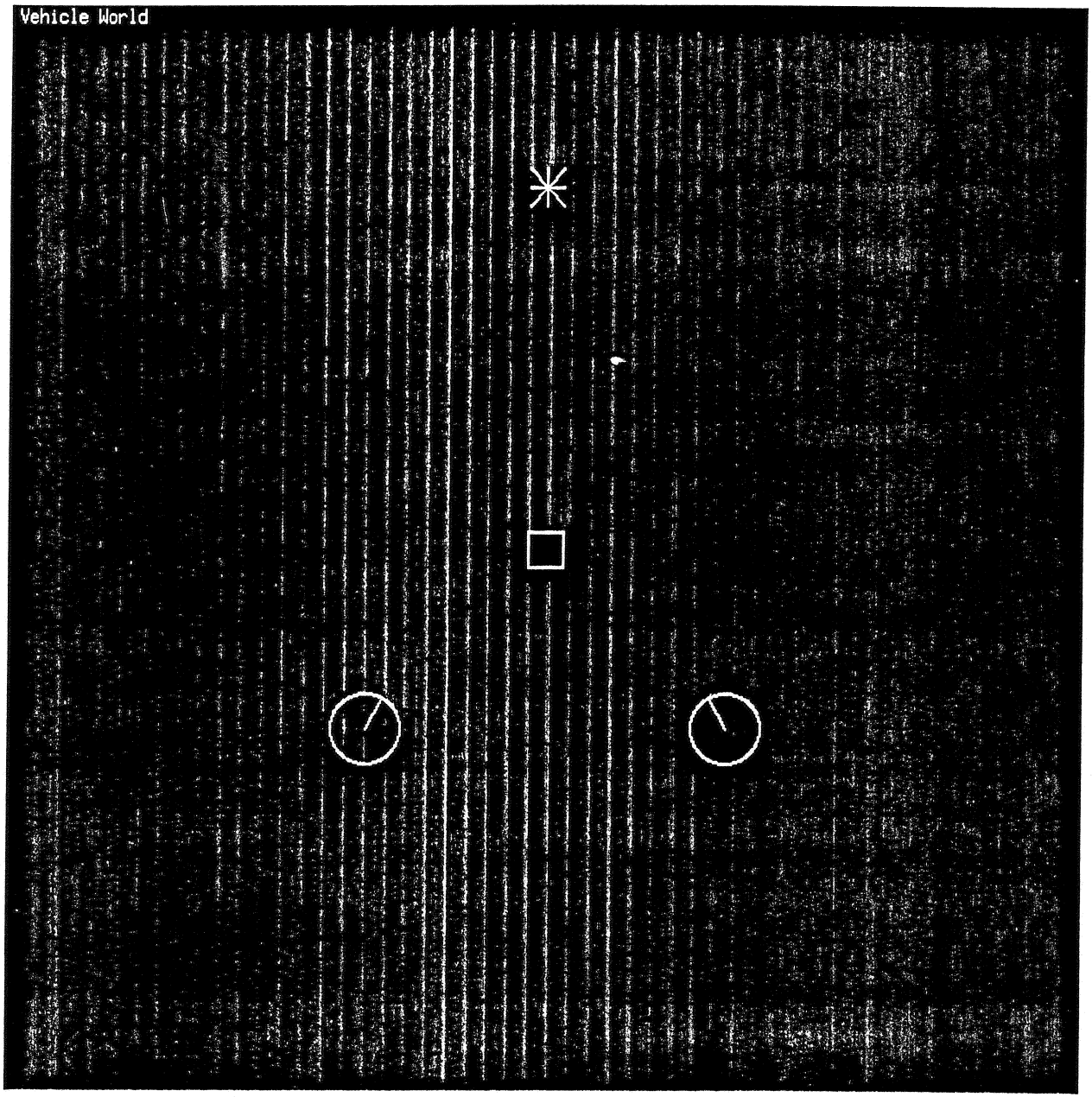
Braitenberg Vehicles are hypothetical vehicles which are described in [3]. These basic vehicles generally have several sensors pointing in various directions connected directly to the motors driving their wheels. The sensors may be connected to motors on either side of the vehicle and may have special functions describing their outputs based on the incident stimulus. In the simplist vehicles, these connections determine the behavior of the vehicle.

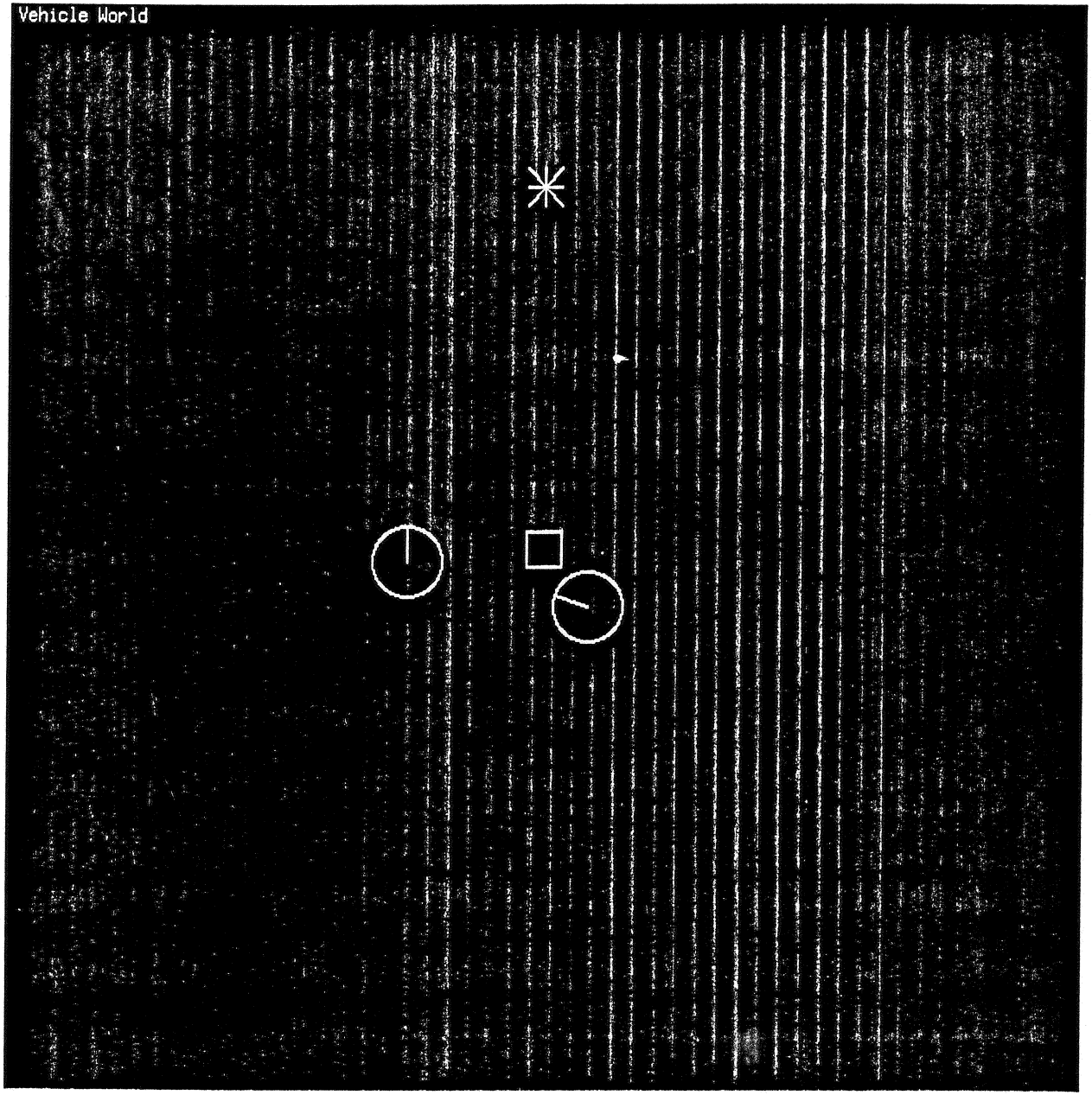
The Esterel code for a Braitenberg vehicle that is attracted to food and repelled by light is:

```
module vehicle:
```

```
type

  Double;

constant

   FACTOR : Double;

function

   Add(Double,Double):Double, Sub(Double, Double):Double,
   Mult(Double, Double):Double;


sensor

  L_Light(Double), R_Light(Double), L_Food(Double), R_Food(Double),
  Speed(Double);


input

  Update;


output

  L_Motor(combine Double with Add), R_Motor(combine Double with Add);


every Update do

  [  %% Find food

    emit L_Motor(Add(?R_Food, Mult(FACTOR, Sub(?R_Food, ?L_Food))));
    emit R_Motor(Add(?L_Food, Mult(FACTOR, Sub(?L_Food, ?R_Food))));

  || %% Avoid Bright light

    emit L_Motor(Add(?L_Light, Mult(FACTOR, Sub(?L_Light, ?R_Light))));
    emit R_Motor(Add(?R_Light, Mult(FACTOR, Sub(?R_Light, ?L_Light))));

  ]
  end. %% every Update
```

A trace of the behavior is given in the next two images.


Vehicle World


Vehicle World

3.1.2 A Spring-Mass System

This spring model demonstrates some of the versatility of the system. It contains a single agent which is a mass with a spring apparently attached to the center of the world. The agent has a sensor for spring displacement. Its behavior is to act like a spring mass system with undamped oscillations. More simply, it applies a an undamped spring mass equation to its sensor input for its motor ouput.

The Esterel code for the spring is:

```
module spring:

type Vector;

sensor  Displacement(Vector);

input   Update;
output  MotorPower(Vector);


every Update do
  emit MotorPower(?Displacement)
  end.
```

3.1.3 The INRIA Mobile Robot

The INRIA mobile robot is as described in the applications section. The behavior specification is to move forward until it reaches an obstacle; then it tries to turn right if it can, otherwise, turn left, and if that fails, then back up. This behavior does not prevent the robot from getting stuck since it does not turn in place. However, it is a behavior complex enough to be of higher order than simple stimulus response. The robot senses obstacle from the combined inputs of its 24 perimeter sonar sensors.

The Esterel code for this behavior is:

```
module robot_vehicle:

type  Double;

constant POS_WHEEL_POWER, ZERO_WHEEL_POWER, NEG_WHEEL_POWER : Double;
constant COAST_TIME : integer;

procedure Update_Clear_Actions()();


input  Update, Ready, ClearFront, ClearRightTurn, ClearLeftTurn, ClearRear;

output MoveLeftWheel(Double), MoveRightWheel(Double);

relation ClearFront => Ready, ClearRightTurn => Ready,
         ClearLeftTurn => Ready, ClearRear => Ready;


signal MoveForward, MoveBackward, TurnRight, TurnLeft, ActionComplete in

  every Update do
    call Update_Clear_Actions()()
  end
||
  loop
    await Update;
    present ClearFront
      then emit MoveForward
      else present ClearRightTurn
             then emit TurnRight
             else present ClearLeftTurn
                    then emit TurnLeft
                    else present ClearRear
                           then emit MoveBackward
                           end %% present ClearRear
                    end %% present ClearLeft
             end %% present ClearRight
      end;%% present ClearFront
    await ActionComplete
```

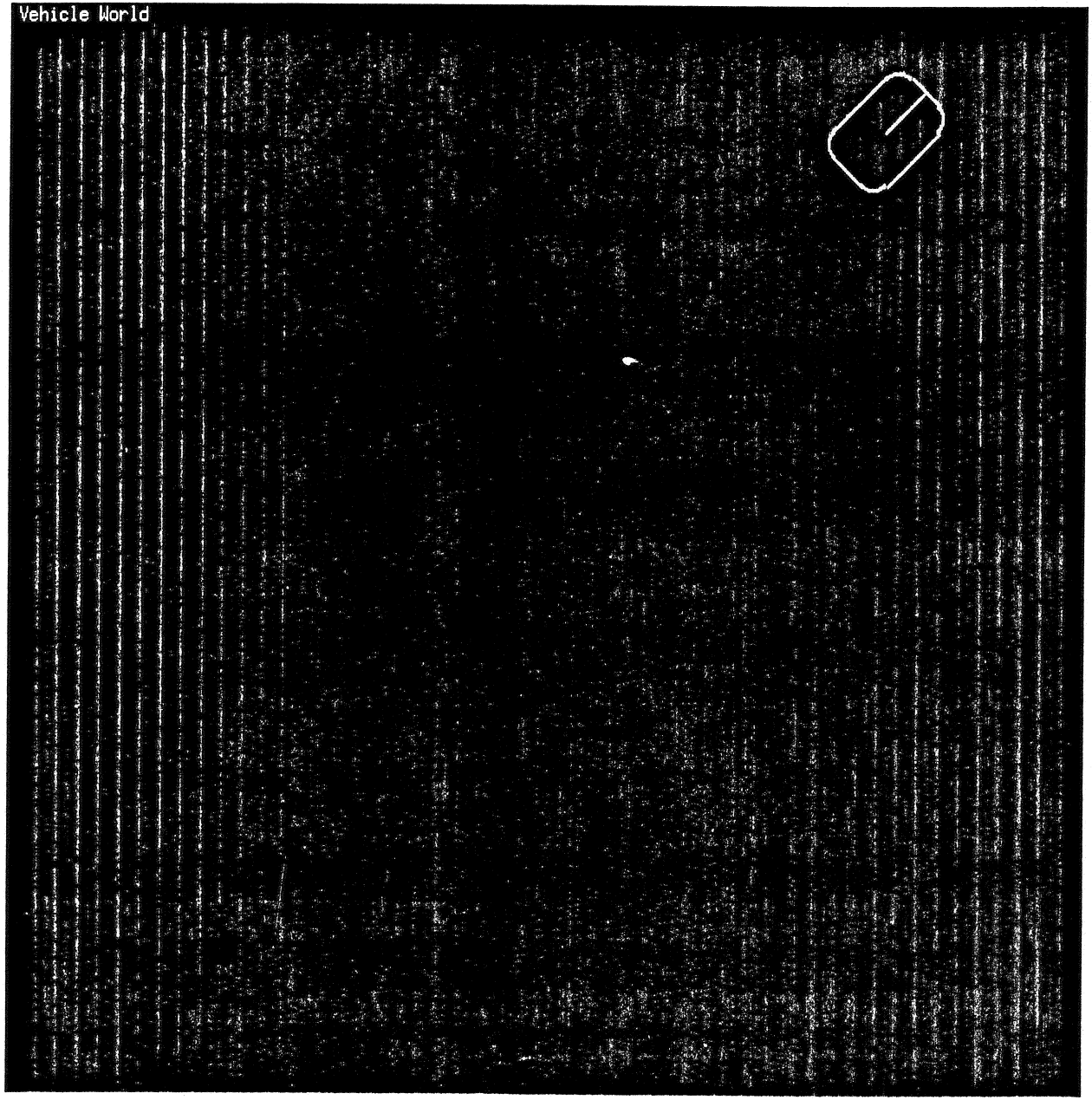
```
  end %% loop
||
  every MoveForward do
    emit MoveLeftWheel(POS_WHEEL_POWER);
    emit MoveRightWheel(POS_WHEEL_POWER);

    do
      every Update do
        emit MoveLeftWheel(ZERO_WHEEL_POWER);
        emit MoveRightWheel(ZERO_WHEEL_POWER)
        end %% every
    upto COAST_TIME Update;

    emit MoveLeftWheel(NEG_WHEEL_POWER);
    emit MoveRightWheel(NEG_WHEEL_POWER);

    emit ActionComplete
    end %% every
||
  every MoveBackward do
    emit MoveLeftWheel(NEG_WHEEL_POWER);
    emit MoveRightWheel(NEG_WHEEL_POWER);

    do
      every Update do
        emit MoveLeftWheel(ZERO_WHEEL_POWER);
        emit MoveRightWheel(ZERO_WHEEL_POWER)
        end %%every
    upto COAST_TIME Update;

    emit MoveLeftWheel(POS_WHEEL_POWER);
    emit MoveRightWheel(POS_WHEEL_POWER);

    emit ActionComplete
    end %% every
||
   every TurnRight do
     emit MoveLeftWheel(POS_WHEEL_POWER);
     emit MoveRightWheel(ZERO_WHEEL_POWER);

     do
       every Update do
         emit MoveLeftWheel(ZERO_WHEEL_POWER);
         emit MoveRightWheel(ZERO_WHEEL_POWER)
         end %% every
     upto COAST_TIME Update;

     emit MoveLeftWheel(NEG_WHEEL_POWER);
```

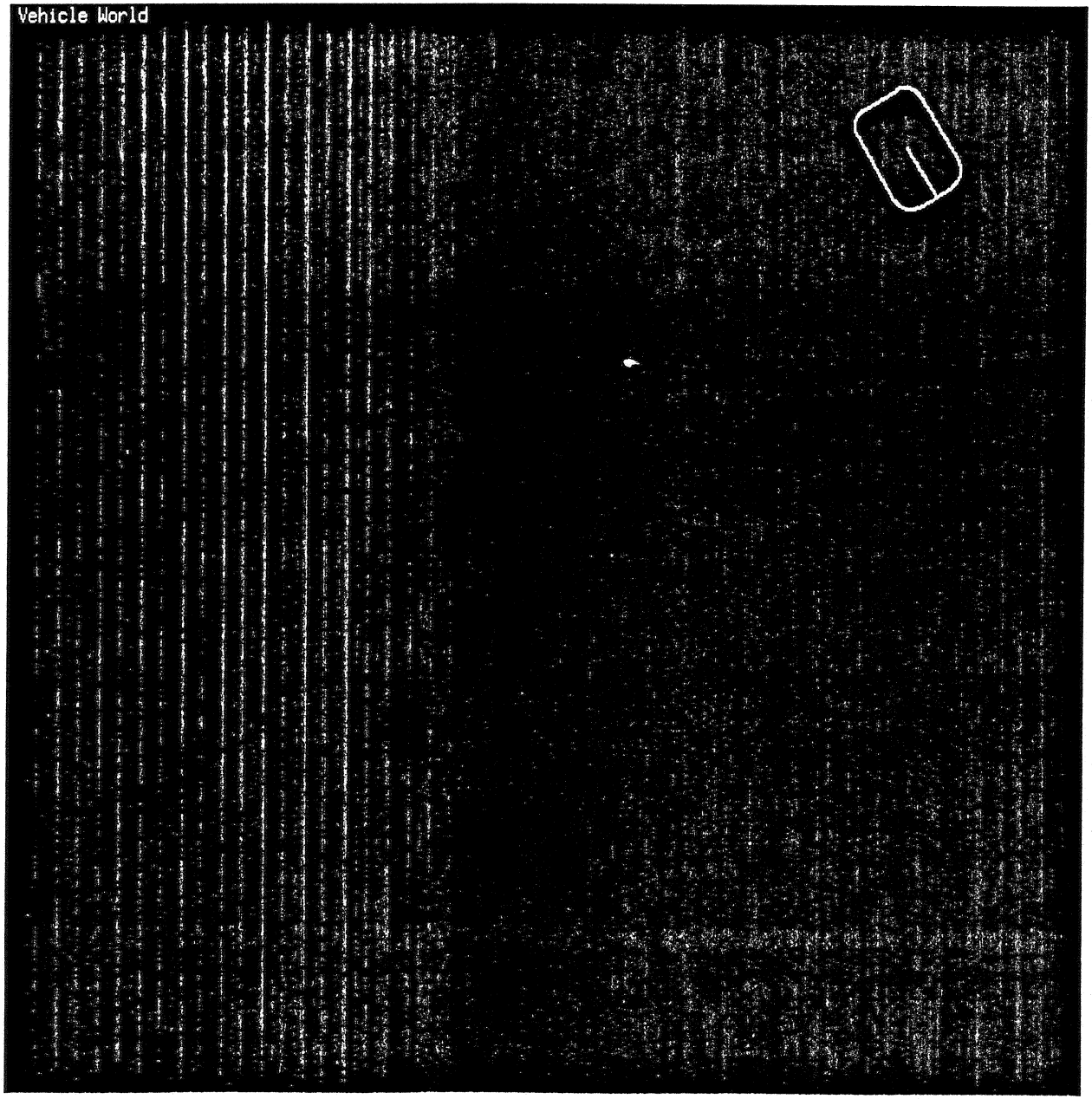
```
    emit MoveRightWheel(ZERO_WHEEL_POWER);

    emit ActionComplete
    end %% every
||
  every TurnLeft do
    emit MoveLeftWheel(ZERO_WHEEL_POWER);
    emit MoveRightWheel(POS_WHEEL_POWER);

    do
      every Update do
        emit MoveLeftWheel(ZERO_WHEEL_POWER);
        emit MoveRightWheel(ZERO_WHEEL_POWER)
        end %% every
    upto COAST_TIME Update;

    emit MoveLeftWheel(ZERO_WHEEL_POWER);
    emit MoveRightWheel(NEG_WHEEL_POWER);

    emit ActionComplete
    end %% every

end.%% signal
```

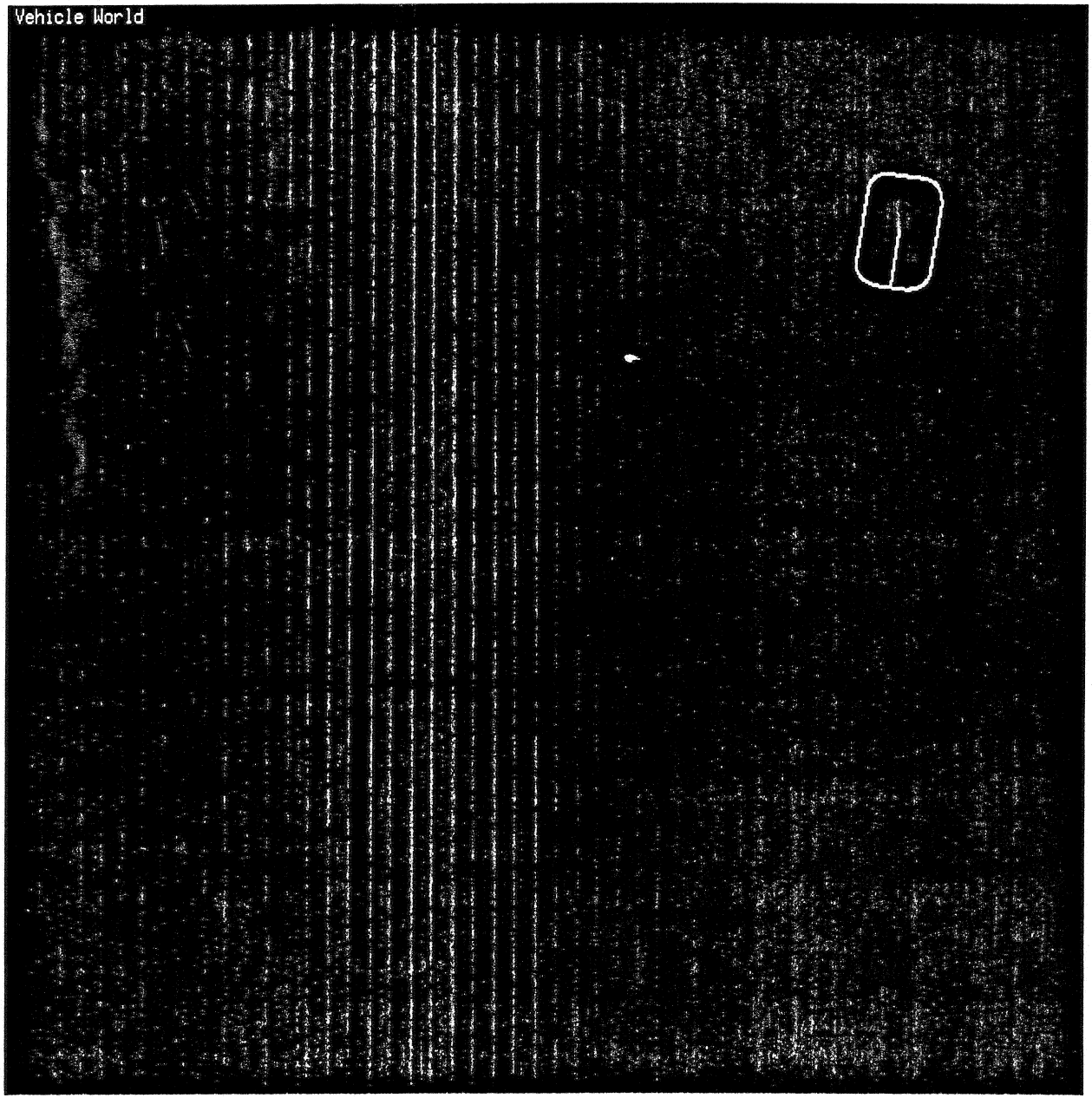
The following five images trace its behavior.

Vehicle World

Vehicle World


Vehicle World


Vehicle World


Vehicle World

4 Future Work

We are working on several other aspects of the framework, including:

- incorporate tendons,
- define multi-link agents,
- perform physical measurements over simulations,
- add several graphical interface and analysis tools; e.g.:
  - see world from agent's place,
  - view sensor data directly, and
  - view agent's state.

References

[1] Ronald C. Arkin. Motor schema-based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112, August 1989.

[2] Geŕard Berry and Georges Gonthier. The esterel synchronous programming language: Design, semantics, implementation. Research Report 842, INRIA, Sophia Antipolis, France, May 1988.

[3] Valentino Braintenberg. *Vehicles: Experiments in Synthetic Psychology.* MIT Press, Cambridge, Massachusetts, 1987.

[4] R.A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[5] R.A. Brooks. Intelligence without representation. In *Proceedings of the Workshop on the Foundations of Artificial Intelligence.* MIT Press, 1987.

[6] J.H. Connell. A behavior-based arm controller. *IEEE Transactions on Robotics and Automation*, 5(6):784–791, Decmeber 1989.

[7] I.J. Cox and N.H. Gehani. Concurrent programming and robotics. *International Journal of Robotics Research*, 8(2):3–16, April 1989.

[8] R.A. Grupen. *General Purpose Grasping and Manipulation with Multifingered Robot Hands.* PhD thesis, University of Utah, Merrill Engineering Building, Salt Lake City, UT 84112, August 1988.

[9] R.A Grupen, Thomas C. Henderson, and Ian D. McCammon. A survey of general-purpose manipulation. *International Journal of Robotics Research*, 8(1):38–62, February 1989.

[10] Rod Grupen and Thomas C. Henderson. Autochthonous behaviors: Mapping perception to action. In Thomas C. Henderson, editor, *NATO ASI on Traditional and Non-Traditional Robotic Sensors*, page to appear, Maratea, Italy, August 28 - September 2 1989. Springer-Verlag.

[11] Z. Har'El and R.P. Kurshan. Automatic verification of coordinating systems. In *Workshop on Automatic Verification Methods for Finit State Systems*, pages 1–16, 1989.

[12] Z. Har'El and R.P. Kurshan. Software for analytical development of communication protocols. *ATT Technical Journal*, page to appear, January 1990.

[13] T. Henderson and R. Grupen. A survey of dextrous manipulation. Technical Report UUCS-TR-007, The University of Utah, Department of Computer Science, July 1986.

[14] Thomas C. Henderson and Rod Grupen. Logical behaviors. *Journal of Robotic Systems*, page to appear, 1990.

[15] Leslie P. Kaelbling. An architecture for intelligent reactive systems. In M.P. Georgedd and A.L. Lansky, editors, *Proceedings of the Workshop on Reasoning about Plans*, pages 395–410, Timberline, Oregon, June 30 - July 2 1987.

[16] R.P. Kurshan. Reducibility in analysis of coordination. In *Discrete Event Systems: Models and Applications*, LNCIS 103, pages 19–39, Berlin, 1987. Springer-Verlag.

[17] R.P. Kurshan. Analysis of discrete event coordination. In *Discrete Event Systems: Models and Applications*, LNCIS, page to appear, Berlin, 1990. Springer-Verlag.

[18] R.P. Kurshan and K. McMillan. A structural induction theorem for processes. In *8th PODC*, pages 239–247, 1989.

[19] M. McKenna, S. Pieper, and D. Zeltzer. Control of a virtual actor. *ACM SIGGRAPH - Computer Graphics*, 24:165–174, 1990.

[20] K. Overton. *The Acquisition, Processing, and Use of Tactile Sensor Data in Robto Control.* PhD thesis, University of Massachusetts, Amherst, Massachusetts, May 1984.

[21] M.H. Raibert. *Legged Robots that Balance.* MIT Press, Cambridge, MA, 1986.