Logical Behaviors: A Mobile Robot Development Environment

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 and do not necessarily reflect the views of the sponsoring agencies.

Thomas C. Henderson
Deptartment of Computer Science
University of Utah, USA

In this paper we describe an approach to high-level multisensor integration in the context of an autonomous mobile robot. Previous papers have described the development of the INRIA mobile robot subsystems:

- sensor and actuator systems
- distance and range analysis
- feature extraction and segmentation
- motion detection
- uncertainty management, and

- 3-D environment descriptions.

We describe here an approach to:

- semantic analysis of the 3-D environment descriptions.

This analysis is organized in terms of robot goals and behaviors. This is accomplished by the use of logical behaviors. Such an approach allows for active control of the sensors in acquiring information.

Introduction

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 the second problem.

Autonomous mobile robots have been studied in a wide range of contexts. Figure 1 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 \cite{Chattergy85,Crowley87,Giralt84,Harmon87,Isik88,Kriegman87,Milberg87,Selfridge85,Smith87,Thorpe87,Triendl87}. Most such systems must use sensors (e.g., sonar or cameras \cite{Draper87,Elfes87,Faugeras86,Triendl87a}) and actuators and must control them \cite{Espiau85,Hirzinger86,Meyste187,Nevis84,Tuijnman87,Turau88}. The use of sensors requires the study of uncertainty management \cite{Hager88,Pertin88,Scotch87,Smith87a} and multisensor integration \cite{Chiu86,Durrant86,Henderson87b,Lowrance83,Luo88,Milet86}. More global approaches to the sensorimotor problem can be found in \cite{Albus81,Bhatt87,Giralt85}, and special purpose architectures are being planned \cite{Arkin87a,Stentz87,Whittaker87}.

One level up, the mapping of procedural behaviors onto the sensorimotor control structure is of interest \cite{Andersson88,Arkin87,Braitenberg87,Henderson85a,Henderson85b,Henderson84c,Tsuji87}.

The world representations also exist at this level: both the metrological \cite{Ayache87,Ayache88,Moravec88,Rao87}, where precise measurement is paramount, and topological \cite{Bhanu87,Boissonnat88,Chatila86,Faverjon87,Faverjon88,Gex87,Ghallab88,Goldstein87,Jarvis88,Levi87,Metee87,Sobek85,Tournagoud88,Yeung87}, where adjacency relations are useful for path planning, etc. It is even possible to study primitive forms of learning in this context \cite{Salzberg85,Tournagoud88}.

Broader studies are usually oriented towards particular applications (e.g., the nuclear industry \cite{Carlton87,White87}, road following \cite{Davis87,Dickmann88,Nasr87}) or towards well-defined, but limited goals (e.g., indoor \cite{Brady88,Faugeras87} or outdoor \cite{Kuipers88,Levitt87} navigation).

Finally, the 'highest' level involves the specification and representation of the knowledge appropriate to a given task \cite{Chandrasekaran87,Kak87,Kak88,Lenat87,Nilsson85} and its compilation into executable robot behavior (or programs) \cite{Dufay84,Henderson88,Latombe88}. 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 balckboard 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.
We 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).

In this paper we describe an approach to solving the third problem. (See cite{Ayache88} 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 `intelligent' 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 2).

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 3).

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. Techniques for the satisfaction of the relations are called `logical behaviors'.

As an example, consider the world model in Figure 4 which represents a specific office at INRIA.
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 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.

\subsection{Robot Behavior as Real-time Programming}

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

We have selected the Esterel synchronous programming language\cite{Berry88} as the specification language for the reactive kernel of the robot's behavior. A reactive system is organized in terms of three main components:

\begin{itemize}
\item \textbf{reactive kernel}: specified in Esterel and compiled into C or Common Lisp for execution,
\item \textbf{interface code}: handles drivers and sensors, and
\item \textbf{process or data handling code}: routine calculations.
\end{itemize}

The programs produced are:

\begin{itemize}
\item \textbf{deterministic}: produce identical output sequences for identical input sequences,
\item \textbf{concurrent}: cooperate deterministically, and
\item \textbf{synchronous}: each reaction is assumed to be instantaneous.
\end{itemize}

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

\begin{verbatim}
1234567 123 123 123 \kill
> {\bf every} 1000 MILLISECOND {\bf do} {\bf emit} SECOND {\bf end}
\end{verbatim}

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:

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

\subsection{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 5 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 6 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)

\end{itemize}
\begin{figure}
\vspace{2in}
\caption{The Debugging System Organization}
\end{figure}
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.

\section{Implementation}

\subsection{Mobile Robot}

Figure 7 shows the operational mobile robot at INRIA. It is similar to other mobile robots (e.g., like those at CMU or Hilare at LAAS).

Figure 8 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.

A graphical interface has been developed which permits a model of the ground floor to be specified and for the robot to be instructed to move in that environment while avoiding obstacles. Figure 9 shows a session at the Rocquencourt location of INRIA. For full details, see Robles88.

Many papers have been published describing our methods for building robust environment descriptions\cite{Ayache87,Ayache88,Faugeras87,Faugeras86}. Current capabilities include 3-camera stereo and robust multi-view fusion.

Figure 10 shows a typical office scene and Figure 11 shows a set of 3-D segments reconstructed from the analysis of such a scene. This 3-D description provides the basis for the development of logical sensors for object recognition and localization.

\section{Summary and Future Work}
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.

\appendix
\section{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:
\begin{verbatim}
% $Header: COMBINE.strl,v 1.1 88/12/07 tch Locked $

module COMBINE:

    type DISTANCE,
        PING,
        R_THETA,
        MOVE;

    constant OMEGA_MIN, OMEGA_MAX : integer;

    input  S;
    input MOVETIME;

    relation MOVETIME => 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
            emit GET_SONAR(?CURRENT_SONAR)
        end
    ||
    copymodule ALARM
    ||
    copymodule GET_MIN_DISTANCE
\end{verbatim}
module ALARM:

type DISTANCE,
    PING,
    R_THETA;

constant MIN_ALARM_DIST : DISTANCE;

input NEAREST_OBJ(R_THETA) ;
input GET_SONAR(PING) ;

output DISPLAY_ALARM(R_THETA);

function LESS_THAN_DISTANCE_TO_DISTANCE(DISTANCE,DISTANCE) : boolean;
function EXTRACT_R(R_THETA) : DISTANCE;
function SONAR_TO_R_THETA(PING) : R_THETA;

every immediate NEAREST_OBJ do
    if LESS_THAN_DISTANCE_TO_DISTANCE(EXTRACT_R(?NEAREST_OBJ),MIN_ALARM_DIST)
        then emit DISPLAY_ALARM(SONAR_TO_R_THETA(?GET_SONAR))
    end
end

module GET_MIN_DISTANCE:

type PING,
    R_THETA;

input GET_SONAR(PING) ;

output NEAREST_OBJ(R_THETA);
function SONAR_TO_R_THETA(PING) : R_THETA;

every immediate GET_SONAR do
    emit NEAREST_OBJ(SONAR_TO_R_THETA(?GET_SONAR))
end

module WANDER:
type MOVE;
constant OMEGA_MIN, OMEGA_MAX : integer;
input MOVE_TIME;
output MOVE_CMD(MOVE);
function rand(integer, integer) : integer;
function TURNS_TO_MOVE(integer, integer) : MOVE;
every MOVE_TIME do
  var left_wheel_turns, right_wheel_turns : integer in
    left_wheel_turns := rand(OMEGA_MIN, OMEGA_MAX);
    right_wheel_turns := rand(OMEGA_MIN, OMEGA_MAX);
    emit MOVE_CMD(TURNS_TO_MOVE(left_wheel_turns, right_wheel_turns))
end
end

The finite state machine produced for COMBINE is:

Automaton COMBINE  (Debug Format)

1. Memory allocation

  V0: boolean (boolean of signal S)
  V1: boolean (boolean of signal MOVE_TIME)
  V2: R_THETA (value of signal DISPLAY_ALARM)
  V3: MOVE (value of signal MOVE_CMD)
  V4: PING (value of signal GET SONAR)
  V5: R_THETA (value of signal NEAREST_OBJ)
  V6: integer (variable left_wheel_turns)
  V7: integer (variable right_wheel_turns)
  V8: PING (value of sensor CURRENT SONAR)
  V9: boolean (boolean of sensor CURRENT SONAR)

2. Actions

2.1 Present signal tests

  A1: V0 (signal S)
  A2: V1 (signal MOVE_TIME)

2.2 Output actions

  A3: DISPLAY_ALARM (V2)
  A4: MOVE_CMD (V3)

2.3 Assignments

  A5: V4 := (V9 ? V8 : (V9:=true, V8:=S_CURRENT SONAR() ))
  A6: V2 := SONAR_TO_R_THETA(V4)
  A7: V5 := SONAR_TO_R_THETA(V4)
  A8: V6 := rand(OMEGA_MIN, OMEGA_MAX)
  A9: V7 := rand(OMEGA_MIN, OMEGA_MAX)
  A10: V3 := TURNS_TO_MOVE(V6, V7)

2.4 Conditions
A11: LESS_THAN_DISTANCE_TO_DISTANCE (EXTRACT_R(V5), MIN_ALARM_DIST)
A12: false

3. Automaton

State 0

goto 1

State 1

if A1 then
  A5;
  if A2 then
    A8; A9; A10; A7;
    if A11 then
      A6; A3; A4;
      goto 1
    end;
  A4;
  goto 1
end;
A7;
if A11 then
  A6; A3;
  goto 1
end;
goto 1
\end{verbatim}

Multiple processes can be added to the robot by using the add\_process command in the Roboter C interface software. However, a send with APRO works better:
\begin{verbatim}
sprintf(cmd, "MOVE P RC=%d,%d P=%d \n", move.left_wheel_turns, move.right_wheel_turns, move.period);

send(cmd);
\end{verbatim}

The program must be loaded into the robot memory, and the go command issued to start it. The program then requests the user to enter a delay which corresponds to how long the program is to run (independently monitored). The robot then generates random moves (the number of turns for each wheel is independent) of not more than 20 centimeters a move every ten seconds and stops if an object is detected closer than two centimeters.

\bibliographystyle{plain}
\bibliography{general}
\tableofcontents
\end{document}