SPOTT: A Predictable and Scalable Architecture for Autonomous Mobile Robot Control

John S. Zelek and Martin D. Levine

Abstract— **A robot control architecture called SPOTT[1] is proposed and implemented as a soft real-time system of concurrently executing and co-operating modules. What distinguishes SPOTT from other architectures with a behavioral control component is that (1) it is able to guarantee task completion in certain scenarios; and (2) it is scalable to different tasks, control laws, computational resources, and robot platforms. The dynamic path planning module's optimality properties give SPOTT its task completion guarantees. Scalability claims are make possible because of the (1) modularity of the behavioral control programs; and (2) SPOTT's software architecture.**

SPOTT consists of various components with both deliberative and reactive properties: a behavioral controller, a local dynamic path planner, and a global path planner. In addition other components include a map database and a graphical user interface. The behavioral control formalism is called TR+ and is based on an adaptation and extension of the Teleo-Reactive (TR) formalism. TR+ control rules (i.e., condition-action pairs) are structured as modular software components. TR+ conditions are based on sensed and internal model states and TR+ actions either affect actuator control or perform map database maintenance. The local dynamic path planner is based on a potential field method using harmonic functions, which are guaranteed to have no spurious local minima. The global planning module advises the local planning module on the local effect of the global goal. A real-time and parallel implementation of SPOTT using a message passing software package called PVM has been developed and tested across a collection of interchangeable heterogeneous workstations. Navigational experiments have consisted of moving two different robot platforms in a structured office and an unstructured laboratory environment to known spatial locations with no or a partial a priori map.

Keywords— **autonomous mobile robot, navigation, dynamic path planning, software architecture, soft real-time, distributed processing**

I. Introduction

There is a variety of potential applications for mobile robots in such diverse areas as forestry, space, nuclear reactors, environmental disasters, industry, and offices. Tasks in these environments are often hazardous to humans, remotely located, or tedious. Potential tasks for autonomous mobile robots include maintenance, delivery, and security surveillance, which all require some form of intelligent navigational capabilities. A mobile robot will be a useful addition to these domains only when it is capable of (1) functioning robustly under a wide variety of environmental conditions, (2) operating without human intervention for long periods of time, and (3) providing some guarantee of task performance. The environments in which mobile robots must function are dynamic, unpredictable and not completely specifiable by a map beforehand. In order for the robot to successfully complete a set of tasks, it must dynamically adapt to changing environmental circumstances.

Biological creatures apparently execute many tasks in the world by using a combination of routine skills without doing any extensive reasoning. In recent years researchers have used this as a guide to formulate behavioral architectures for robot control [1]. A behavior is the result of executing a collection of *situation-action* rules (i.e., *control law*'s). Researchers have tried to extend this approach by studying problems such as how the rule resultants are combined, what rules to encode, and how a behavioral program is integrated with goal-directed planners [2].

The proposed architecture - SPOTT - does not confine itself to expressing all control laws in the form of logic (i.e., as in behavioral architectures) but in addition concurrently executes a representational-based approach (i.e., potential field) for providing goal-directed navigation. The path planner can respond instantaneously giving it reactive properties but with some deliberation it is able to produce optimal paths. The behavioral programs are structured around the various aspects of the navigation problem as opposed to an arbitrary collection of rules. These programs are typically reactive but some of the conditions can require some deliberation.

SPOTT's architecture emphasizes the issues of predictability and scalability. Predictability refers to the ability to guarantee successful operation and task completion for various tasks. SPOTT's goal-directed planner encodes all possible paths, one of which is selected by performing gradient descent on the computed function. The steepest-gradient-descent trajectory is optimal in the sense of minimizing the distance to the goal subject to minimizing the probability of hitting obstacles [3]. This helps to minimize the effect of uncertainties associated with the position and size of obstacles, and the robot's position. Scalability refers to how easily the system can adapt when (1) one or more of the components in the system are changed; and (2) there is a change in the functionality of some or all of the sub-tasks in the system [4]. Tactical[2], strategic[3] and reactive[4] control laws are programmed as separate programs in the behavioral control language which easily accommodates future modifications and additions. SPOTT has been implemented using the message-passing PVM [5] package

John S. Zelek is with the Engineering Systems & Computing Program, School of Engineering, University of Guelph, Guelph, ON, N1G 2W1, Canada

Martin D. Levine is at the Centre for Intelligent Machines (CIM), Department of Electrical Engineering, McGill University, Montreal, QC, H3A 2A7 Canada.

[1] A **S**ystem which integrates **P**otential fields for planning **O**n-line with **TR+** program control in order to successfully execute a general suite of **T**ask commands.

[2] What to do at the next instance?

[3] Constrains and mediates tactical behaviors.

[4] Instantaneous response to environmental or internal stimuli.

which makes transparent the load balancing (i.e,. real-time scheduling) of processes on a heterogeneous collection of computational resources. The SPOTT system has been tested with two different robot platforms, each with different sensor configurations. All potential navigational tasks can be composed from a formal specification based on a minimal spanning basis set of navigational tasks [6], [7] as found in the English language.

II. Background

There has been a recent trend to talk about software design at the level of the organization of the overall system (i.e., architecture) [8] as opposed to the algorithms and data structures of the computation. An atypical problem within this class of research is the architecture of a mobile robot control system.

The basic requirements for a mobile robot architecture are (1) accommodation of deliberative and reactive behavior, (2) allowance for uncertainty in control and sensing, (3) accounting for dangers inherent in the robot's operation and environment, and (4) providing the designer with flexibility, allowing for modification and reconfiguration after experimentations and technological upgrades [8]. The four main architectural styles are (1) closed-loop [9], (2) a layered approach [10], (3) implicit invocation [11], and (4) a blackboard approach [12].

The simplest architecture is a closed-loop (albeit an open-loop architecture would be simpler): the controller initiates robot actions and monitors their consequences, and adjusts future controls accordingly [8], [9]. All architectures are in essence closed-loop control with an elaboration of the controller element. A layered approach nicely organizes the components required to coordinate a robot's operation into the following components in ascending order: robot control, sensor interpretation, sensor integration, real-world modeling, navigation, control, global planning, and supervisor, with the robot control layer directly communicating with the sensors and actuators [10]. Two problems come to mind when examining this architecture: (1) when fast reaction is required to incoming data, the layers may need to be circumvented; and (2) the model does not separate the data and control hierarchies. Thus they may overlap each other; for example, raw sensor input is in level 1, interpreted and integrated results are in levels 2 and 3, and the world model is in level 4, while motor control is' level 1, navigation is level 5, scheduling is level 6, planning is level 7, and user-level control is level 8. While the model is precise about the roles of the different layers, it breaks down when taken to the greater level of detail demanded by a real-time implementation [8].

The implicit invocation is embodied in the Task-Control Architecture (TCA) [11] and ORCCAD [13]. TCA is a high-level operating system supporting distributed communications, task decomposition, resource management, execution monitoring, as well as error recovery. The task decomposition is expressed as a deliberative hierarchy of action plans. It is interesting to note that subsequent uses of this architecture have evolved into a layered architecture approach [14]. TCA is still used but primarily as an interprocess communication and synchronization tool (i.e., similar to PVM [5]). The layers in ascending order of abstraction are hardware, obstacle avoidance, navigation, path planning and task scheduling. ORCCAD is also a programming methodology for robot controllers from specification to implementation.

The blackboard architecture [12] is structured around a blackboard database which is driven by various processes continually operating on the data (i.e., object-oriented). Its one drawback is that there is no control over how quickly the robot will react to sensed changes in the environment.

There is a general consensus that some kind of layering should constitute an aspect of a robot control architecture. One of the simplest layering strategies was the subsumption architecture (also referred to as behavioral) introduced by Rodney Brooks [1] at MIT. This technique was in sharp contrast to the traditional robot architecture approach in which complex models of the environment were built before planning and executing actions. Originally, the behavioral approach de-emphasized model building to the extreme by having no internal model of the environment at all. However, recent behavioral architectures do have internal models [15], [16]. Behavioral architectures possess many sensor-action streams which are executed in parallel while the traditional approach has only a single processing stream. Interesting behaviors result by executing many sensor-action rules which are usually reflexive. The biggest advantage of behavioral architectures is that they are readily responsive to environmental changes. However, they have not scaled well to more complex problems involving deliberation and cannot offer any performance guarantees.

A small community of robotics researchers is moving towards a three layered, functionally-organized hierarchical robot control architecture [17]. The lowest layer is a behavioral control system, such as the subsumption architecture [1], which is readily responsive to different sensed stimuli. The top layer is a traditional symbolic planning and modeling system, referred to as a *deliberative* layer. The middle layer, which bridges the reactive and symbolic layers, has not yet been clearly specified. Time scale is one way of differentiating the reactive and symbolic layers. The behavioral layer is in a real-time feedback control loop, while the symbolic layer is engaged with events that occur at slower intervals. An analogous method for characterizing the control layers [18] is as follows: a strategic rule-based layer for defining mission logic (i.e., symbolic), a tactical layer (i.e., middle) activating behaviors, and an execution layer (i.e., behavioral). Others [19], [20], [21] have also suggested similar layerings. There are two ways in which current research has categorized the role of the crucial middle layer. In the first approach, it is defined as a sequencing layer [22], [23], [24], [11], [25]. The sequencing transforms a procedural list of task commands into an executable set of reactive skills. The latter react to environmental changes, but are not goal-driven. Thus, there is no guarantee that the goal will be achieved. In the second approach, the middle layer is defined as a planner [26], [23]. A robot path, consisting of a set of linear segments, is planned from the start to goal position. The linear segments are then executed under behavioral control. Reactive behaviors can override the execution of this plan, but it is not clear how control is subsequently resumed by and sequenced with the path executioner. A collection of condition-action rules (i.e., as presented in a behavioral controller) cannot fully capture all possible environmental situations and contexts and thus cannot solely be used to execute a plan.

III. SPOTT Mobile Robot Control Architecture

The SPOTT architecture is a layered architecture but the layers are not tied directly to increasing levels of spatial and temporal data abstraction [14], [25], or are necessarily separated into distinct deliberative and reactive components [22], [23], [27], but rather revolve around around a behavioral control module and a collection of planning modules [28]. Synchronization amongst modules is achieved via a shared map database in conjunction with appropriate control laws. In addition to being layered, aspects of SPOTT also incorporate architectural styles of closed-loop, implicit invocation (e.g., module processes communicate using PVM), and blackboard (e.g., communication via a shared map). The typical division of layers (e.g., symbolic, tactical, execution) in a three-layered hierarchical architecture [18], [19] with both deliberative and reactive properties can all be encoded in the situation-action hierarchical rule set of SPOTT's behavioral controller. Either the controller[5] or planning module controls the actuator (i.e., movement of the robot) at any particular time. The planner is structured as an anytime iterative refinement algorithm [29] and can react to environmental stimuli almost immediately or produce an optimal response after some deliberation. In addition to the behavioral controller and the planning modules, SPOTT's other major components include a human-machine interface (hmi), a world model database, and perceptual processing and action primitives which interface directly with the control unit (see Figure 1).

SPOTT was designed to control a mobile robot equipped with sensors in an unmodifiable environment with the possibility - but not necessarily - of having an a priori "*partial map*" of the environment: a map of the permanent fixed structures in the environment, as would be presented by an architectural CAD drawing[6]. The assumption is made that there is an available abstract graph representation consistent with the architectural CAD map[7]. The abstract graph consists of nodes and edges, where nodes represent rooms or portions of hallway and the edges represent access ways (i.e., doors) between the nodes. SPOTT has been tested in an office and laboratory environment with and without an available a priori CAD map of the permanent structures (e.g., walls). The environment would typically be classified as structured but certain areas such as the laboratory (i.e., large open space with many to-be-discovered obstacles) can be considered unstructured.

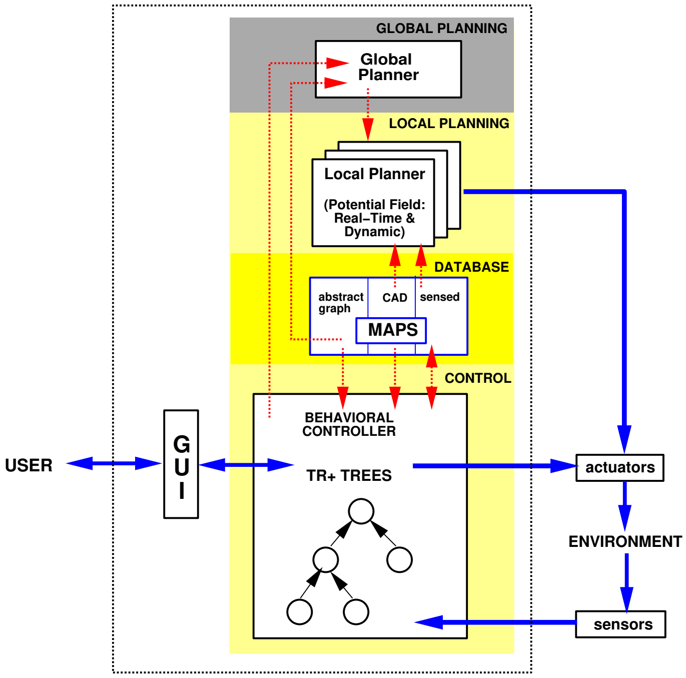

Fig. 1. **SPOTT Overview.** The main independent components of SPOTT are illustrated. Sensor-based map and robot position updates are input as TR+ conditions. Subsequently, this information is made available to the local planner. A typical operational scenario is for the local planner to control the motion of the robot and for the controller to override when necessary. The global planner provides global goal information to the local planner. Control flow is indicated by the solid thick arrow while data flow is shown by the dotted arrows.

From a system's point of view, the SPOTT robot architecture was designed to specifically address the mobile robot navigational problem [30]: solving the problems of mapping, localization, path planning and execution concurrently. The behavioral controller is responsible for monitoring sensor information, updating the map database, setting goals and if necessary controlling the motion of the robot. This is accomplished in the form of reactive rules, behavioral sequencing, and simple strategic goal setting. Rather than having an arbitrary sequence of behavioral rules, the collection and organization of rules is guided by the breakdown of the mobile robot navigation problem. The normal mode of operation is for the local path planner to determine the robot's trajectory. The role of the local path planner is to provide path predictability. Reactive rules (e.g., for safety) will override motion control from the local path planner (e.g., and move the robot away from the danger). The controller is responsible for initiating the actuator control switch and synchronization is maintained via a shared common map. Some of the *conditions* of behavioral control

[5]Even though SPOTT's behavioral controller actually executes plans of situation-action rules, it will not be referred to as being a planner.

[6]Architectural drawings are readily available in computer readable CAD formats. Note, however, that these are often not kept up-to-date and may even be inaccurate because of changes in the original plans.

[7]For the experiments, the abstract graph was created manually by visually inspecting the CAD map.

rules are responsible for executing the sensor-based localization and perceptual mapping routines. The global path planner's role is to provide global information to the local path planner by projecting a goal - if necessary - onto the border of the extent of the local planner. The global path planner performs its computations on an abstract representation of space: in the case examined, a graphical representation of the topological relationships of rooms and access ways. It is also responsible for replanning a global path if a door (e.g., edge, initially assumed to be open) is closed or the path from one access way to another (e.g., door) is blocked by a newly discovered obstacle.

A. Map Database

The modeling of an object (i.e., obstacles and goals) is subject to the 2D geometric model[8] (i.e., points, lines, ellipses, rectangles) representations used for 2D path planning. Parametric geons [31] are chosen as a qualitative description of 3D object components (i.e., for qualitative object recognition) for future use when the perceptual modules used by SPOTT have significantly matured. All the primitives (e.g., 2D, 3D) are stored as a set of vectors with associated attributes (i.e., primitive type, dimensions, spatial location in absolute coordinates, label corresponding to its real world equivalent).

The world map (i.e., database) consists of a collection of primitives (i.e., mostly line segments), the equivalent abstract graph representation in terms of nodes (e.g., rooms, hallway portions) and edges (i.e., door and their state: open or closed), the position of the robot, and the task specified by the operator. If a CAD map is available a priori, all the line segments are labeled as walls and entered into the world map.

The world map is further divided into local and global components which may be equated with short and long-term memories. The local map is a fixed-sized window (determined by the local path planner) into the global map. The spatial location of the local map within the global map changes when the robot's position is about to escape the local map's bounds. The local map is encoded in both vector and raster formats, whereas the global data is solely stored in vector format. The raster representation is an occupancy grid [10], whose free space is where the potential field is computed.

Sensor data may label existing map features (e.g., with wall or door labels) and is only included into the map if it is already not there. An occupancy grid representation simplifies the sensor fusion problem [10] at the expense of limiting accuracy to the grid's resolution. The equivalent vector representation is not subject to the occupancy grid quantization. It is included with the foresight of potentially including the capability of building an accurate map on-the-fly. Currently the issue of sensor fusion is not dealt with within SPOTT's framework and the map database is treated as a depository of a priori map information as well as sensed features.

SPOTT's implementation has each module contain its own copy of the local map. This approach minimizes access times (but increases module communication during map updates, which was found to be negligible) when compared to having one copy of the map in a central depository. Since all sensor information is input to the controller, the controller is responsible for updating all other map copies.

[8]These 2D geometric models may be the result of a projection onto the 2D navigational plane of a corresponding 3D model representing an object (e.g., door, chair, wall).

B. Task Command

SPOTT's language lexicon for specifying the task command is a minimal spanning subset for human 2D navigational tasks [6], [7]. The syntax for the task command given in BNF format is as follows:

<TASK> ::= *<VERB><DESTINATION> <DIRECTION> <SPEED>*
<VERB> ::= **go** | **find**
<DESTINATION> ::= *<TARGET>* | *<PREPOSITION> <TARGET>* | <>
<DIRECTION> ::= *<ORIENTATION>* | *<PATH><TARGET>* | <>
<SPEED> ::= **slowly** | **quickly** | **normal**
<PREPOSITION> ::= **across** | **against** | **along** | **alongside** | **around** | **at** | **behind** | **beside** | **beyond** | **by** | **far** | **in** | **inside** | **in-back-of** | **in-front-of** | **in-line-with** | **near** | **out** | **outside** | **to-the-left-of** | **to-the-right-of** | **to-the-side-of**
<ORIENTATION> ::= **forward** | **backward** | **left** | **right** | **north** | **south** | **east** | **west**
<PATH> ::= **along** | **around** | **via** | **to** | **toward** | **from** | **away-from**
<TARGET> ::= (*<PLACES>* | *<MOVEABLE-OBJ>* | *<2D-MODELS>* | *<GEONS>*) *<ATTRIBUTES>*
<PLACES> ::= **door** | **hallway** | **room** | **wall**
<MOVEABLE-OBJ> ::= **chair** | **desk**
<2D-MODELS> ::= **ellipse** | **line** | **point** | **rectangle**
<GEONS> ::= **cuboid** | **cylinder** | **ellipse**

The verb "**go**" assumes that the goal is a known spatial location in an absolute coordinate reference frame, whereas "**find**" assumes that a description of the object is known but not its spatial location. The *<ATTRIBUTES>* terminal defines the necessary 2D (or 3D) parameters and coordinates in a globally-referenced Cartesian coordinate space. The target is chosen based on what the robot (i.e., SPOTT) can perceptually recognize (e.g., door, wall), what the robot knows about (e.g., rooms, hallways in the CAD map), or 2D and 3D models. Spatial prepositions act as modifiers of the target location. Only 2D spatial prepositions are chosen because 2D mobile robot navigation has been investigated. The role of specifying the direction is to bias the local path planning execution. The path is determined by performing gradient descent in the specified direction (or as close as possible to). The lexical set of speed variables is small but a more discriminating set can be easily accommodated (e.g., very fast, very slow).

C. Teleo-Reactive Behavioral Control

The control module is an event-driven interpreter with a collection of sensor-action rules written in a hierarchical standard programming fashion (i.e., like a C program with collection of subroutines) and a relaxed regression style[9], with the permissibility of concurrent actions. Actions of the control module not only control the actuators, but also update and maintain the map database. Conditions are based on evaluating sensor information or world model information and are typically computed as separate processes. The control module is based on a behavioral programming language called TR+ which extends the TR (Teleo-Reactive) programming language [32], [33]. Either the planner (i.e., local) or the control unit will drive the actuator (e.g., wheels) at any particular time. If the controller is not controlling the navigation of the robot, then it is supplying the necessary information to the planner (i.e., localization estimates, mapping information). Both deliberative and reactive programs are encoded in the control module. There is no constraint that the conditional parts of the rules are at one particular level of abstraction. There can be reactive rules (e.g., see Figures 7 and 8) or complex behavioral rules based on slowly changing states (e.g., see Figure 10), or even rules based on deliberative scheduling (e.g., see the room finding condition in Figure 13).

C.1 Teleo-Reactive Programs

TR programs bear some resemblance to a watered-down version of STRIPS planning [34]. A TR program is a list of hierarchically-ordered condition-action rules (i.e., like a production system). An action - a_i - either updates internal models, drives actuators, or invokes another TR subroutine. A condition - K_i - is an executable process which returns a logical value (TRUE or FALSE) derived from a state computed on sensory inputs and world models. In addition, the process can also return other (i.e., logical or non-logical) values through variables attached to the condition. Conditions are based on the K-B model proposed by Kaebling and Rosenchein [35]. In this model, the perception module monitors the environment so that the agent (i.e., robot) can establish certain beliefs of the world. Conditional predicates based on these beliefs are the inspiration for the conditions K_i in TR programs.

The collection of TR condition-action rules are ordered by subgoal relations (i.e., a plan). TR programs have a graphical representation - called a TR tree - where a condition is specified by a node and an action by an arc (see Figure 2). A hierarchical order (i.e., represented as a TR sequence, see Figure 2) on the list of condition-action rules is imposed by the designer. A TR program is constructed so that for each rule, $K_i \rightarrow a_i$, condition K_i is the regression, through action a_i, of some particular condition higher in the list. K_i is the weakest condition such that the execution of action a_i (under the most probable operating circumstances) achieves some particular condition K_j which is higher in the list ($j < i$). The condition K_1 is the goal condition the program is designed to achieve. The execution of the actions in a TR program ultimately achieves the goal. Should an action have an unexpected effect, the program will nevertheless continue working towards the goal. The TR sequence is *complete* if and only if $K_1 \vee ... \vee K_i \vee ... \vee K_m$ is a *tautology*, which is a clause containing complementary literals. A TR sequence is *universal* if it satisfies the *regression property* and is *complete*. A *universal* TR sequence will achieve its goal condition, K_1, provided that the perceptual processing of sensor data detects the necessary events that conditions monitor and the robot's execution is actually performed[10]. The regression property should be strictly adhered to when designing and planning the ordering of the events (conditions) and their associated actions. It is easier to ensure the tautology property if the number of conditions in any program (i.e., subroutine) is small. This simplifies the process of validating that all conditions are complimentary literals. The TR+ subroutines designed for robot control using SPOTT are typically concise and thus can be easily checked for *completeness* (see Section III-C.3).

TR programs also bear some resemblance to discrete event systems [36], [13] in their formulation and graphical representation. One difference is that inputs are implicit in the TR conditions while TR arcs explicitly represent just outputs (rather than both input and output, as in the arcs of Petri nets [37]). In addition, TR conditions (i.e., nodes) monitor environmental events in order to trigger actions (arcs) (as opposed to representing places and transitions as in the nodes of Petri nets). All active TR conditions are also monitored simultaneously, permitting the system to be more reactive and adaptable to environmental stimuli than discrete-event systems, where only certain events are monitored at any given instance. In addition, a TR program's close relationship to an AI planning system (such as STRIPS) gives it some goal guarantees, albeit they are weak and dependent on the formulation of the actual program.

TR programs are interpreted in a manner comparable to a production system's interpretation: the list of rules is scanned from the top for the first rule whose condition is satisfied, and then the corresponding action is executed. A TR program differs from a production system in that all the active[11] conditions are continuously evaluated and the action associated with the current highest TRUE condition at any particular time instance is always the one executed.

[9]The property of regression is used to formulate the rules, however, the property of regression does not need to be strictly followed at execution time.

[10]The robot will always have errors in its actual movements and these are corrected by localizing the robot at regular intervals.

[11]Not all conditions are continuously evaluated. A TR program is usually modular in organization, consisting of a main program and a collection of subroutines, similar to the organization of a computer program. Only the conditions in the main program (i.e., invoked by the user) and the conditions in the currently called set of subroutines are *active*.

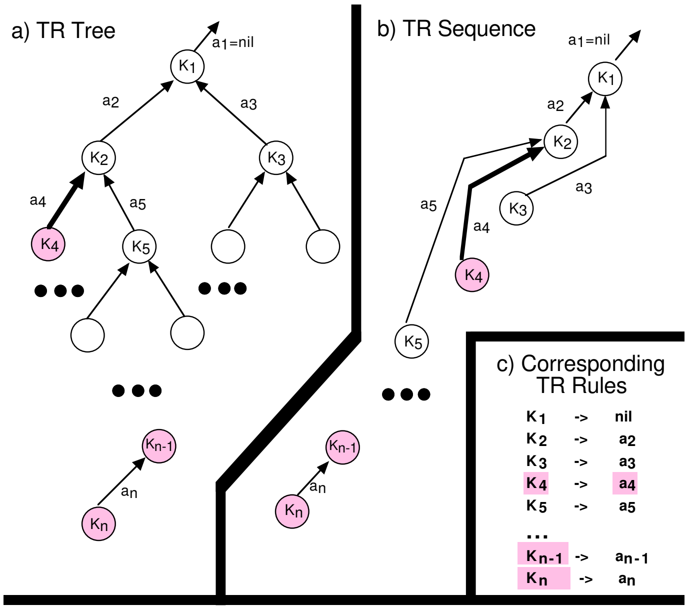

Fig. 2. **A TR Tree.** A TR tree is shown in (a). Conditions K_4,K_{n-1}, and K_n are presented as being TRUE (i.e., indicated by the shading), resulting in action a_4 being fired. The TR tree in (a) can be equivalently represented as a TR sequence as revealed in (b). The TR sequence explicitly shows the precedence ordering of the conditions' evaluations. The corresponding list of rules are displayed in (c).

The actions can be energized[12] or ballistic[13], whereas production system actions are only ballistic.

C.2 Teleo-Reactive+ Programs

A TR+ program extends a TR program by permitting concurrent actions. The additional operators were inspired by the operations used in procedural action plans [38]. The concurrent operator is specified by $\|$, and the sequential operator is symbolized by $\succ$. The concurrency operator forces the left and right side operands (i.e., actions) to be executed concurrently, while the sequential operator orders the execution of the actions from left to right. The sequential operator is similar to a TR-sequence without actions being condition-triggered and the sub-goal relation ordering.

Condition operators (e.g., AND ($\wedge$) and OR ($\vee$), and their extensions) were also added for more expressibility in the condition state and to permit the programmer to specify how and when the expression is actually evaluated. If the condition expression is evaluated from left to right, then when its logical value is known, no further conditions are processed in the expression. For example, consider the expression $A \vec{\wedge} B$, which means perform $A \wedge B$ and compute the expression from left to right. If A is false, B is not evaluated. If A is true, B is evaluated. This provides the capability of turning off the computation of certain condition processes subject to event states, and also can save computation time. If the expression is rewritten as $A \wedge B$, then A and B are evaluated concurrently. Enforcement of real-time validation to only currently changing events is achieved by a timeout specification (e.g., $\wedge_t$, $\vec{\wedge}_t$, $\vee_t$, and $\vec{\vee}_t$) for expressions: if timed out, the value of the expression is FALSE, regardless of the computed logical values of the operands.

The execution time of some conditions may be slower than the time required for a the TR+ interpreter to scan once through the list of active conditions. A condition process is computed at a certain frequency which depends on its processing complexity. The original TR formalism assumed that the logical value of a condition was always updated at each interpreter cycle. In reality, this is not always possible. In the TR+ formalism, the logical value associated with a condition process during its computation can either be set to its last computed value or to FALSE. The latter is referred to as a *ballistic* condition, while the former is called an *energized* condition. A TR+ condition's computation frequency can also be specified by the programmer. Enforcing the availability of a result from a condition to be available after t time requires the conditions to be formulated as iterative-refinement anytime algorithms [29], in addition to having an underlying real-time operating system.

C.3 TR+ Programs for Navigation

SPOTT's TR+ programs are designed to solve the mobile robot navigation problem: mapping, localization, goal search, path planning and execution. This is contrary to most mechatronic (i.e., robotic) systems, where the decomposition of engineering specifications into tasks and states is highly subjective [39]. A sample TR+ program consisting of a main routine (see Figure 3) and twelve subroutines[14] is illustrated using the Dotty software package [40] in Figures 3 to 13. This TR+ program is an example of mobile robot control using SPOTT's framework. For the task of navigation, writing a TR+ program is relatively simple because all of the mentioned TR+ subroutine's inputs and outputs are independent of each other and may be executed concurrently. If the robot were outfitted with a manipulator, manipulation programs could be easily accommodated by being triggered by a condition indicating that the proximity of the robot is adequate for manipulating the object of interest.

One underlying assumption in the collection of TR+ programs developed is that the path planning module (i.e., local) is almost solely responsible for moving the robot. The outputs of the TR+ programs update the world model (e.g., obstacles and robot position). The exceptions to the

[12] An action is *energized* when it is sustained continually as long as its triggering condition remains TRUE.

[13] An action is *ballistic* if it executes to completion after the logical transition of its triggering condition from FALSE to TRUE.

[14] Only ten subroutines are shown. See [28] for the omitted search subroutines (i.e., *DIRECTION_SEARCH*, *RANDOM_SEARCH*). The *DIRECTION_SEARCH* program searches for the sought-after object while heading in a particular direction. The *RANDOM_SEARCH* program searches while heading in a particular direction (i.e., deduced randomly) until an obstacle is encountered, and at that time a new direction is randomly decided upon and the search continues. Both programs succeed when the sought-after object is found.

rule are the two programs that specify a reaction when collision has been detected via the bumper or infrared sensors (i.e., see Figures 7 and 8). In both of these situations, the TR+ program takes over actuator control from the path planning module (i.e., via the *SET ROBOT POSITION TO* program, the details are not shown). Control is returned to the path planner after moving the robot away from the source of collision. When control of the actuators is switched to/from a TR+ program, this is always initiated from within a TR+ subroutine.

Another assumption is that all incoming sensor information is treated as being valid for updating the map. This is reasonable considering that the main purpose of the map is for navigation rather than being an accurate reflection of the environment. As an aside, it should be noted that the sensor information has been already filtered somewhat before being added to the map. For instance, sonar data points are fused into line segments with outliers removed [41]. If one of SPOTT's roles is to provide an accurate map, then the confirming, refuting, and refining of existing map features with newly sensed multi-sensor data (i.e., to create an accurate map) will require an appropriate sensor fusion strategy and framework as well as an uncertainty representation scheme.

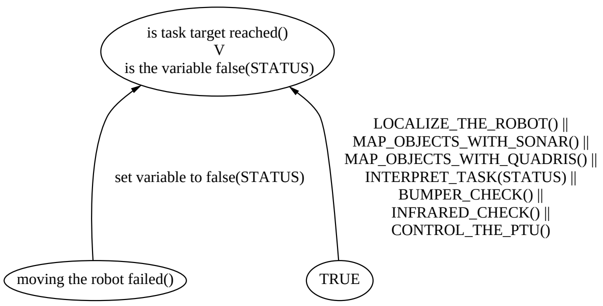

Fig. 3. **The Main TR+ Program.** Most of the subroutines called from this program are in Figures 4 through to 13. Capitalized actions are calls to subroutines. All of the TR+ trees (main and subroutines) were illustrated using the Dotty software package [40].

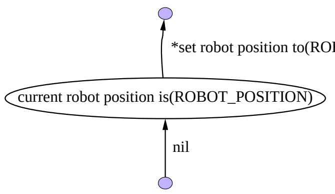

Fig. 4. **LOCALIZE_THE_ROBOT() subroutine.** The condition (i.e., node) collects sensor (i.e., sonar) data and correlates it with the existing map to determine pose [41]. The action (i.e., arc) updates the world model. All the subroutines (Figures 4 through to 13) have a bottom and top node (i.e., shown colored) that are not used and act as placeholders for attachment to the calling routine.

D. Planning

A path planner is used in conjunction with the TR+ controller because it is impossible to completely express all instances of potential obstacle avoidance as a collection of TR+ rules. Many obstacle avoidance contextual rules

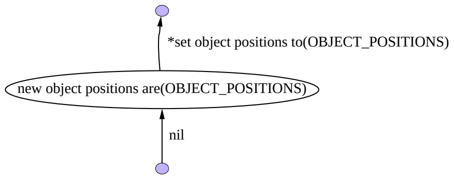

Fig. 5. **MAP_OBJECTS_WITH_SONAR() subroutine.** The condition (i.e., node) collects and processes (i.e., removes outliers, fuses into line segments) sonar data [41] and the action (i.e., arc) updates the world model.

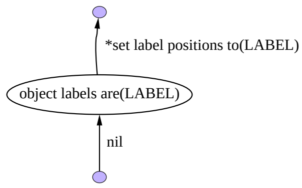

Fig. 6. **MAP_OBJECTS_WITH_QUADRIS() subroutine.** The condition (i.e., node) collects range data [42] (i.e., as range line segments) and the action (i.e., arc) updates the world model.

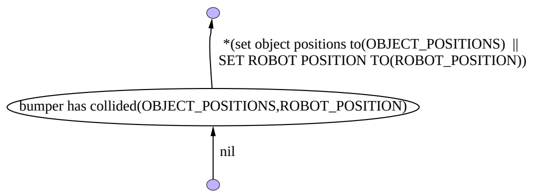

Fig. 7. **BUMPER_CHECK() subroutine.** A reactive rule which states that if the bumper senses an obstacle, then update the map and move away from the source of collision (e.g., move away a distance of ROBOT_POSITION).

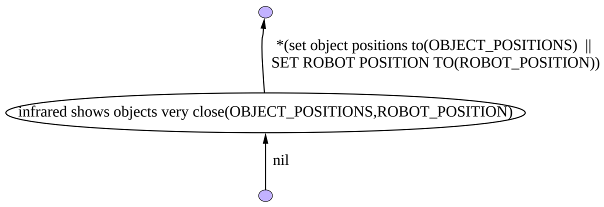

Fig. 8. **INFRARED_CHECK() subroutine.** A reactive rule which states that if the infrared senses an impending obstacle, then update the map and move away from the source of collision (e.g., move away a distance of ROBOT_POSITION).

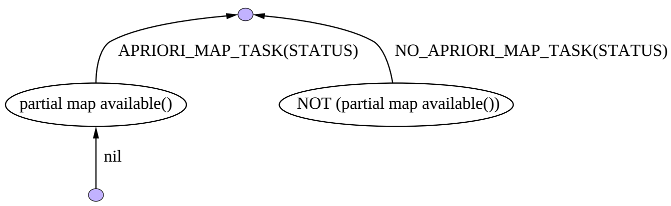

Fig. 9. **INTERPRET_TASK(STATUS) subroutine.** The goal is selected based on the availability of a priori map.

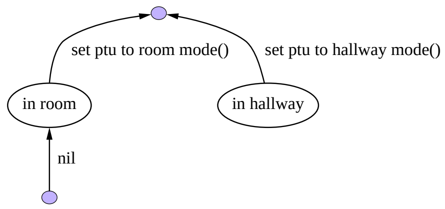

Fig. 10. **CONTROL_THE_PTU().** Set the scanning pattern of the range sensors pan-tilt head to the appropriate pattern depending on whether the robot is in a room or hallway.

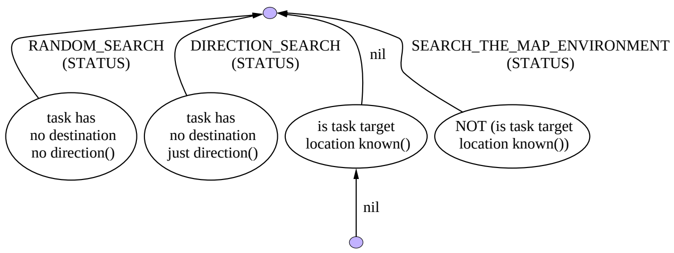

Fig. 11. **APRIORI_MAP_TASK(STATUS) subroutine.** A different goal selection strategy (i.e., subroutine) is chosen depending on a parse of the task command and the availability of an a priori map.

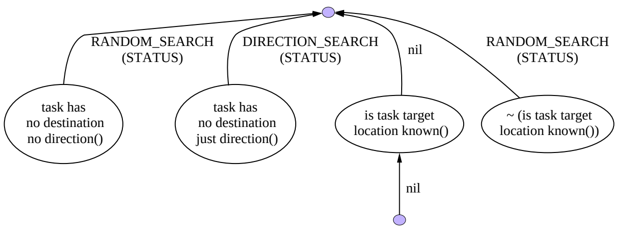

Fig. 12. **NO_APRIORI_MAP_TASK(STATUS) subroutine.** A different goal selection strategy (i.e., subroutine) is chosen depending on a parse of the task command and given that no a priori map exists.

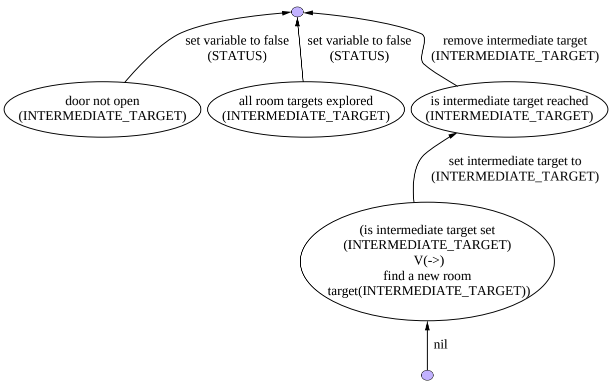

Fig. 13. **SEARCH_THE_MAP_ENVIRONMENT(STATUS) subroutine.** The subroutine used in the FIND task selects a goal for searching, given that an a priori map exists. The subroutines RANDOM_SEARCH(STATUS) and DIRECTION_SEARCH(STATUS) are similar in style to this subroutine. The 'find a new room' condition can be formulated as a scheduling algorithm of rooms to visit given the object to 'find'.

could actually be encoded as TR+ behavioral rules, but there is no guarantee that all environmental contexts would be captured.

SPOTT utilizes two concurrently executing planning modules. Planning is defined along a natural division: (1) local, high resolution, quick response, versus (2) global, low resolution, slower response. The *local* path planner executes a path based on the currently stored map encoded as an occupancy grid [10] and is based on a potential field method using harmonic functions, which are guaranteed to have no spurious local minima [43]. A *global* planning module uses a search algorithm (i.e., Dijkstra's algorithm) on a graph structure representation of the map. The role of the global planning module is to provide global information to the local path planner. This is necessary because the spatial extent of the local path planner is limited due to its computational requirements increasing linearly, proportional to the number of grid elements [44].

D.1 Local Dynamic Path Planning

The problem addressed by the local planning module is *discovery and on-line planning* [45] (i.e., dynamic path planning). In this situation, the workspace is initially unknown or partially known. As the robot moves about, it acquires new partial information via its sensors. The motion plan is generated using available partial information and is updated as new information is acquired. Typically this requires an initial planning sequence and subsequent re-planning as new information is acquired.

Planning is typically computationally expensive, especially with a *complete algorithm* (i.e., also referred to as *exact algorithm*): guaranteed to find a path between two configurations if it exists. A complete algorithm requires exponential time in the number of degrees-of-freedom [45]. A type of *heuristic algorithm* for performing dynamic path planning is the potential field technique [46]. This technique offers speedup in performance but cannot provide any performance guarantee. In addition these algorithms typically get stuck at local minima. Techniques have been devised for escaping local minima, such as the *randomized path planner* [47] which executes random walks, but the revised technique is no longer computationally efficient. A complete algorithm for performing dynamic path planning is the D^* algorithm [48]. D^* (i.e., dynamic A^*) performs the A^* algorithm initially on the entire workspace. As new information is discovered, the algorithm incrementally repairs the necessary components of the path. This algorithm computes the shortest distance path to the goal with the information available at any particular time. One problem with the D^* algorithm is that no allowances are made for uncertainty in the sensed obstacles and position of the robot.

SPOTT's local dynamic path planner is based on a potential field method using harmonic functions, which are guaranteed to have no spurious local minima [43], [49]. The obstacles form boundary conditions (i.e., Dirichlet boundary conditions are used) and in the free space the harmonic function is computed using an iteration kernel [50]. The path - determined by performing steepest gradient descent - is also optimal in the sense that it embodies the minimum probability of hitting obstacles while minimizing the traversed distance to a goal location [3]. This optimality condition permits modeling uncertainty with an error tolerance (i.e., obstacle's position and size, robot's position) provided that the robot is frequently localized using sensor data. The robot is modeled as a point but all obstacles are compensated for the size of the robot in addition to an uncertainty tolerance[15]. The computation of the harmonic function is performed over an occupancy grid representation of the local map and is executed as a separate process from the path executioner. In order to make the number of iterations linear in the the number of grid points, "*Methods-of-Relaxation*" [44] techniques such as Gauss-Seidel iteration with Successive Over-Relaxation, combined with the "*Alternating Direction*" (ADI) method were employed. A local window is used for computing the potential field for two reasons: (1) the computation time will linearly increase with the number of grid points; and (2) in long narrow corridors (e.g., hallway regions), the values of some neighboring discrete points can be undistinquishable[16]. A global path planner (discussed later) provides information about goals outside the local bounds. The process computing the iteration kernels, ingesting sensor updates, and servicing requests from the path executor uses the following algorithm:

1. Loop 1: Load initial map if available.
2. If the goal is outside the bounds of the potential field, project the goal onto the border.
3. Loop 2:
   (a) Compute an averaging iteration kernel over the entire grid space.
   (b) Poll to see if a new robot position estimate is available. If so, correct the accumulated error.
   (c) Poll to see if any newly sensed data is available. If so, update the map.
   (d) Update the robot's position based on the odometer sensor.
   (e) Poll to see if the execution process module requests a new steering direction for the robot. If so, compute it by calculating the steepest descent gradient vector at the current estimate of the robot's position.
   (f) If the robot is near the edge of the border of the potential field window, move the window so that the robot is at the center and go to Loop 1; else go to Loop 2.

This computation has the desirable property of concurrent path computation and execution, dynamic map updating (i.e., as features are sensed), as well as robot position correction.

Another approach for speeding up performance is to provide an initial guess to the solution, namely the heuristic potential field [46]. This makes the local path planner an iterative-refinement anytime approach [29] where after a short period after a map change, response is reactive but not optimal or guaranteed; and after convergence is achieved, response is optimal. Achieving fast convergence will not necessarily be possible or realistic if the path planning is done in more dimensions than two.

[15]This may appear to be heuristic, but it is based on not permitting the robot's position error to grow beyond a certain fixed limit (i.e., by re-localizing).

[16]On a sixty-four bit (i.e., thirty-two bit, double precision) computer, the harmonic function can only be accurately represented when the length-to-width ratio for a narrow corridor is less than 7.1 to 1 [28]

D.2 Interaction with Global Path Planning

The global planning module performs search (i.e., deliberation) using Dijkstra's algorithm [51] through a graph structure of nodes and arcs. A graph abstraction of the CAD map is obtained by assigning nodes to rooms and hallway portions and edges to the access ways (i.e., doors) between nodes. The global graph planner determines a path based on start and stop nodes defined by the current location and desired destination of the robot. If the current goal is outside the extent of the local path planner, then the global path planner projects the goal onto the local map's border as follows:

Let $p_g = (x_g, y_g)$ define the position of the goal in an anchored x-y spatial coordinate system (i.e., SPOTT's experiments used cm. as a unit of measure). Let $p_r = (x_r, y_r)$ define the current position of the robot. The local map is a collection of grids (i.e., nodes) but the references into this grid are always continuous rather than discrete. For example, the gradient at a particular position is calculated by interpolating amongst the neighboring grid elements. Let n_r be the node in the global map where the robot is located (i.e., p_r) and n_g be the node where the goal is located (i.e., p_g). Let $b_{pf} = ((x_1, y_1), (x_2, y_2))$ define the top-left and bottom-right points designating the spatial extent of the potential field. By definition, initially p_r is in the center of the region defined by b_{pf}. The path produced by the global path planner is given as a sequence of nodes n_j and edges e_i: $< n_r, e_i, n_j, ..., n_{m-1}, e_{l-1}, n_m, e_l, n_g >$. The following is the algorithm (see Figure 14) for projecting the global goal onto the border of the potential field:

1. If p_g is within b_{pf}, and n_g is the same as n_r then do nothing.
2. If p_g is within n_r and p_g is not within b_{pf}, then the goal is projected onto the portion of the side of b_{pf} that is entirely contained within n_r.
3. Let e_k be the first edge within the global path (i.e., moving from n_r towards n_g) that is not completely within b_{pf}. Let p_k be the intersection of b_{pf} with the line connecting the center of the edge e_k with the center of the node (i.e., n_k) last traversed before reaching that edge. The goal is projected onto the portion of the side of b_{pf} that contains both the node n_k and the point p_k.

In each of the three cases there are situations where the goal will not be reachable. In the third case, the goal will not be reachable if a doorway is closed or a node is blocked by an obstacle. In this case, the global path planner must replan the global path. In the first two cases, the goal may not be reachable within the spatial extent of the potential field due to newly discovered obstacles, but the goal may actually be physically reachable. The easiest way to deal with this situation is to make the size of the potential field such that the node is completely within the potential field's spatial extent.

IV. Implementation

SPOTT has been implemented as a collection of concurrently executing message-passing processes for purposes of (1) modularity, (2) maintaining a correspondence between processes and conceptual modules (i.e., see figure 1), (3) facilitating concurrent execution of modules, and (4) providing both synchronous and asynchronous communication amongst modules, as well as (5) a possible way of trying to meet performance specifications (i.e., by increasing the

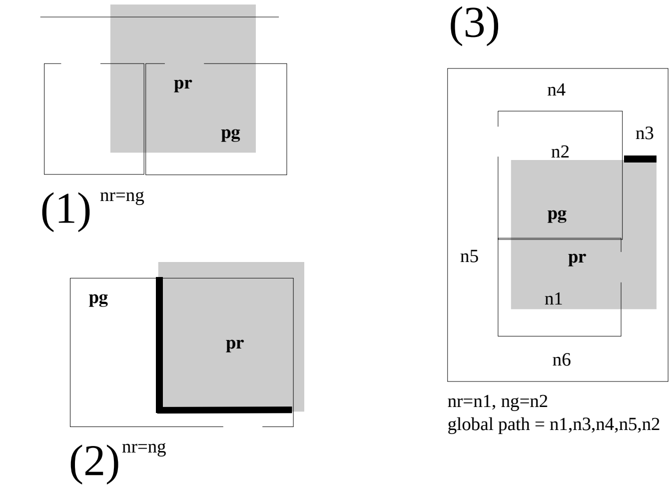

Fig. 14. **Projecting Global Goal onto Local Limits.** The three cases as presented by the algorithm in the text are presented. The shaded area corresponds to the extent of the local path planner: b_{pf}. The thick line corresponds to the projection of the goal onto the local path planner's borders. The thin lines represent wall features. In (1), the goal and robot are in the same node and b_{pf}. In (2), the robot and goal are in the same node but not the same b_{pf}. (3) is the most general case.

number of processes[17]). If only a single processor was available, an alternative computational framework would be to use threads as the modular component. In the configuration tested, the autonomous robot communicates via a radio link to SPOTT which runs on a heterogeneous computer network (e.g., SUN, SGI workstations).

The software tool PVM[18] (Parallel Virtual Machine) [52] was used to distribute the control, planning, and graphical user interface across a collection of existing processor resources. PVM is a message passing library which permits the harnessing of a collection of heterogeneous processors into a single transparent, cohesive and unified framework for the development of parallel programs. The PVM system transparently handles resource allocation, message routing, data conversion for incompatible architectures, and other tasks that are necessary for operation in a heterogeneous network environment. PVM offers excellent price-performance characteristics compared to massively parallel processors [53]. One limitation of PVM is its simplified process allocation scheme (i.e., processes are evenly distributed across the processors in the order of their initiation) which does not take into account the dynamic changes in processor and memory usage (i.e., load balancing). In addition, PVM has not been implemented to execute on a real-time operating system (i.e., only UNIX and Windows implementations currently exist).

One way to represent TR+ programs is in a tree graph format. The *Dotty* [40] toolkit has been used to create software for visually programming TR+ programs as trees, and for monitoring the execution of TR+ programs.

During execution, there is a wealth of on-line graphical information displayed, namely, the current state of: the TR+ control programs; the equi-potential contours of the potential function used by the local path planner; the global path in the abstract map; the planar map of the robot and its environment; as well as the graphical user interface for control. The software package PVaniM [54] is also used to provide online visualization (i.e., of processor and memory usage) of SPOTT's distributed execution with PVM. In addition, the robot vocalizes key actions it initiates such as acquisition and recognition of sensor data. This wealth of information was found essential for debugging, monitoring and verifying proper robot execution.

V. Experiments

SPOTT has been tested in the CIM laboratory and office environment with a limited set of general navigational tasks. Experimentation was conducted with both a Nomad 200 mobile robot as well as a RWI B-12 robot. The RWI B-12 was equipped with a ring of 12 sonar transducers around its circumference. The sensors on the Nomad 200 include a ring of 16 ultrasonic sensors (i.e., sonar), a ring of 16 infrared proximity, a ring of 20 tactile sensors (i.e., pressure sensitive, bumper), and two controllable range sensors (i.e., BIRIS) mounted on pan-tilt heads (QUADRIS) [55], [56]. Sonar and QUADRIS are used for mapping. Furthermore, QUADRIS is specialized for object recognition. In addition, sonar was used for localization (i.e., estimating the position of the robot) [41]. The bumper and infrared sensors are primarily used for safety and accomplish this by mapping very close objects which may be missed by the other two sensors. In addition, the data from these sensors is readily available and requires minimal analysis.

The TR+ interpreter was found to take on average 25μsec/cycle to evaluate a single condition. The time taken to converge to a solution by the local path planner, using a 35 by 35 grid on a collection of 5 SGI workstations for 3 obstacles and a goal, is approximately 2 seconds. In addition to five processors being allocated for local path planning computation, one processor was dedicated to the TR+ interpreter and the additional processors were used for the various range and sonar perceptual processing algorithms. Ideally, all of the processes should be able to execute on a single processor (i.e., if threaded) if the computational requirements of the local path planner can be reduced. Reducing the computation time by using irregular grid sampling is under current investigation. In addition, it was found that in highly structured environments (e.g., narrow hallways) it is computationally advantageous to use just TR+ rules for navigation (e.g., move forward and stay centered in the hallway) as opposed to requiring the full computational power of the local path planner. Typically if an obstacle is discovered in a narrow hallway, the hallway is blocked from passage and a new global path needs to be recomputed, thus negating the need for the local path planner.

[17] It should be noted that it is not always possible to just add processors to increase performance. In addition, system performance may not improve by adding processors because of the underlying communication overhead.

[18] This is an ongoing project carried out by a consortium headed by the Oak Ridge National Laboratory.

Experimentation has been conducted for a variety of tasks under two different scenarios: (1) no map is available a priori; and (2) a partial map of the permanent structures (i.e., walls) is available a priori. An edited *AUTOCAD* architectural drawing of the CIM office and laboratory space was used as a partial map of the permanent structures. It was found that a CAD map typically contains redundant data (e.g., more than two lines to represent the same wall) and large map databases tend to slow down access time. Therefore, the CAD map was manually edited to remove redundant data.

One experiment was performed assuming that the spatial location of the goal was known with no a priori map. The robot moved at 10 cm/sec and updated its map of the environment from sonar data every 5 seconds. Bumper and infrared data were monitored every 2 seconds. The robot was localized every 10 seconds[19]. The map built up during the execution of the task is given in Figure 15. Figure 16 shows selected frames during the execution of the task.

Fig. 15. **Dynamic Mapping and Trajectory Determination.** As the robot moves towards a pre-defined spatial location, sonar data update the map and alter the trajectory.

Fig. 16. **Autonomous Navigation with No A Priori Map.** The robot successfully navigated around various obstacles as it traversed towards a pre-defined spatial location.

A second experiment, shown in Figure 17, illustrates the robot navigating to a predefined location. In this experiment, the robot uses an available CAD map as a priori knowledge. However, during task execution the robot discovers features (e.g., the partition) that are not present in the CAD map. The robot first localizes itself to the corner identified in the original CAD map assuming that the robot's location is known to be in that vicinity to begin with. In this example, the robot moved at 50 cm/sec. In addition to acquiring sonar data, rangefinder data were also acquired at 1 Hz.

The sonar range line features were produced by fusing sonar data points and removing outliers [41] at a regular frequency, a process which could take a few seconds and be very time consuming. Localization (i.e., determining the robot's pose) was performed by matching the sonar features to the existing CAD map at pre-defined areas. These areas were typically hallway corners that were highly unlikely to be cluttered by yet-to-be-discovered obstacles. Localization was also slow due to the matching process (e.g., another few seconds). Sonar processing is typically slow because a dense scan equivalent to a ring of 144 ultrasonic sensors is taken and then processed (i.e., fused and outlier removal) to achieve a map of line segments. Trading off accuracy against performance speed is a matter of future investigation. Acquiring and processing sonar data has been a definite bottleneck in achieving faster robot speeds. This is because actuator commands and sensor feedback share a common communication link. Ideally, the goal is to move the robot at an above-average human walking pace (i.e., 1 m/sec), which is feasible if the communication bottleneck is removed.

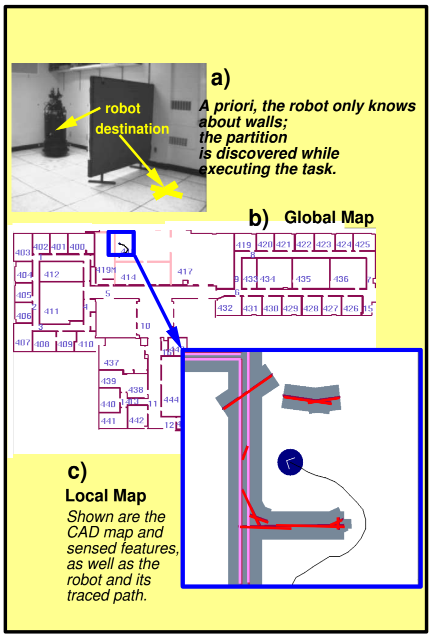

Fig. 17. **Autonomous Navigation with a Partial A Priori Map.** A priori, the robot only knows of the walls in the CAD map. (a) shows the initial configuration and the desired destination. (b) shows the location of the local map in the global map. (c) shows the local map at task completion, along with the CAD (light lines) and sensed features (darker lines), as well as the executed trajectory. The padding of the features, to account for uncertainty and the local path planner modeling of the robot as a point, is illustrated by the grey regions surrounding the line (i.e., wall) features.

Acquisition of QUADRIS range data with embedded simple object recognition algorithms executes at approximately 1 Hz. Horizontal and vertical object range profiles are used to recognize features such as doors, chairs and desks [42]. A limitation of the technique is that successful

[19] This experiment was carried out when the localization frequency could only be specified with respect to time. Currently, the rate can also be specified with respect to distance traversed, as well as with respect to pre-defined map locations that are typically uncluttered and therefore good locations for performing sensor-to-map matching.

recognition of an object (e.g., chair) also requires that the robot be in a particular position and pose [42]. The control of the pan-tilt heads is a predefined scanning pattern depending on the context[20].

All of the TR+ programs tested [28] so far are based on knowing the spatial location of the goal beforehand. This implies that the condition "*is task target location known*" in the TR+ subroutines shown in Figures 11 and 12 is always TRUE. The task is completed successfully when the robot arrives at the desired position, as is the case when the task command is "*GO*". However, when the task command is "*FIND*", a search needs to be initiated in the environment. For the "*FIND*" task, the types of objects which can be found are limited by the recognition capabilities of processing QUADRIS range data. Recognition with moderate levels of success at this time is limited to chairs, doors, walls, and wooden objects in the shape of geons [57].

Other experiments demonstrated, as suspected, that the SPOTT system is highly dependent on the robustness and reliability of the sensor data, namely sonar and QUADRIS. An artificially created environment made up of partitions connected with metallic tubes with notches proved problematic for sonar mapping. These notches produced multiple reflections and caused doorway features to be entered larger-than-life into the map database. Mapping doorways with sonar data is known to be problematic [58]. QUADRIS also experienced some problems by producing features that were the result of lighting effects as opposed to actual physical objects. The lights in the room had to be dimmed and the curtains drawn for satisfactory behavior.

Moving obstacles are currently treated as static objects and a single object can be repeatedly mapped as several objects at different locations during a time interval. Future plans include investigating perceptual routines for mapping moving obstacles by employing a visual sensor. The occupancy grid can accommodate moving obstacles by shifting the occupied grids when necessary. Incorporating sensor fusion techniques into SPOTT also needs to be investigated, especially if a significant number of additional sensors and perceptual algorithms are added.

VI. Conclusions

A. Predictability

The ability of SPOTT to guarantee task completion depends on the type of task - specified by the task lexicon - and the amount of a priori information. SPOTT's a priori knowledge can vary from none to a partial map of the environment (e.g., an architectural CAD map). SPOTT can guarantee task completion (see Figure 18) when the task is *GO* and a CAD map is available a priori, or during the tele-operation mode. The tele-operation mode is based on frequent operator-specified directional commands (e.g., *go forward*). SPOTT responds by guiding the robot in the specified direction, avoiding any newly encountered obstacles and typically stopping when a previously known structure is present in the original path (e.g., wall) or when told to stop by the operator.

[20] For example, a different scanning pattern is used for hallways and rooms.

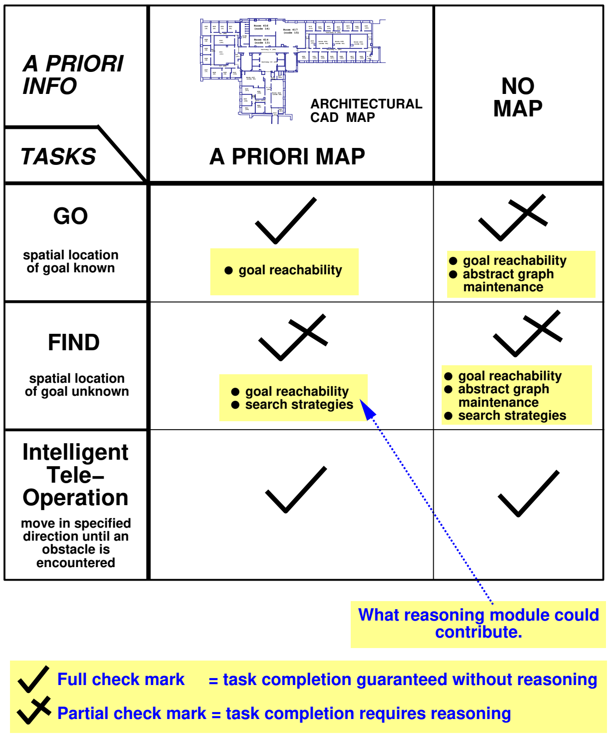

Fig. 18. **What SPOTT Can and Cannot Do.** A check mark in a matrix entry indicates the situations where SPOTT can guarantee task completion. A partial check mark indicates that SPOTT cannot guarantee task completion in these situations and the associated caption indicates what role an external reasoning module could fulfill in this situation.

Guaranteeing task completion when issuing the "*FIND*" command or when no a prior map is available is problematic for SPOTT. In these situations, if the task can be rephrased in the context of a "*GO*" command with a known map, then it can also be guaranteed. "*FIND*" reduces to a collection of "*GO*" tasks with a set of intermediate goal locations (i.e., the places to visit). The goal object will be found if all possible locations in the map are visited and from those locations, all possible vantage points are observed. This assumes that there is a map that takes care of storing all the visitation sites and what part of the environment is gazed upon. In addition, adequate perception is a necessity. SPOTT uses the range sensor for recognition and its current capabilities are limited to wooden geons [57].

The other troublesome aspect of guaranteeing successful task completion is in the case of no a priori map. SPOTT's map database is a depository of all sensed information. If the map database is to be also used for map building in addition to just navigational control (i.e., current only role), then the issue of sensor fusion and the deduction of an accurate CAD map and a companion abstract global map from the sensed features needs to be addressed. If these issues are dealt with, then the task could be treated as if an a priori map existed. It is envisioned that a concurrently executing reasoning module would perform map building in addition to possibly monitoring controllability, observability and reachability issues.

Issues of predictability related to real-time performance are delegated to proper scheduling and load balancing procedures at the level of distributed process allocation (i.e., PVM).

B. Scalability

The scalability of the SPOTT architecture has been shown by (1) the accommodation of changing control laws (programmable in the the TR+ language); (2) the independence of robot platforms and sensors (i.e., SPOTT was tested with two different platforms and sensor configurations); (3) and the independence of computing platforms (i.e., PVM permits an interchangeable collection of heterogeneous processors). The previous section suggested using another component - a reasoning module - for building an accurate map while performing the navigation task. The process of reasoning should not interfere or slow down the real-time operation of the robot. Thus, it should be performed on the area void of the current local map.

The inclusion of a manipulator into the robotic platform would require minimal changes to the SPOTT architecture. A separate collection of TR+ subroutines would be initiated by a condition indicating that the robot is in close proximity to engage in manipulation. A separate module dedicated to manipulator path planning would also be required.

C. In Closing

The SPOTT architecture has been shown to have both predictability and scalability properties. The TR+ formalism was found useful for programming control laws at different levels of abstraction, for expressing hierarchies of control through subroutines, and for monitoring execution by a visualization provided by a graphical representation. The major modification to the TR formalism was the inclusion of condition and action concurrency and the necessary operators. The TR+ programs' structure was not arbitrary but was guided by key facets of the mobile robot navigation problem (i.e., mapping, localization, path planning and execution). The local path planning module provides the source of task guarantee but at a computational cost. In highly structured areas such as hallways, it may prove beneficial to use a collection of TR+ rules, delegating the use of the local path planner to highly unstructured areas (e.g., large unknown open spaces). The SPOTT architecture provides a principled and organized methodology for programming and controlling a mobile robot. A proof of concept was demonstrated with actual mobile robot platforms. Some gaps remain to bring closure to the issue of predictability, namely an accurate map building strategy, and suitable sensor recognition strategies. Unfortunately, a specification of SPOTT's real-time necessities cannot be handled at this time, but this is an issue for real-time computing within the message-based distributed (or potentially threaded) implementation (i.e., PVM).

Acknowledgments

This research is partially supported by the *Natural Sciences and Engineering Research Council* and by the *National Centres of Excellence Program* through IRIS (*Institute for Robotics and Intelligent Systems*). This research is part of the *Dynamic Reasoning, Navigation and Sensing for Mobile Robots* project under the ISDE (*Integrated Systems in Dynamic Environments*) theme.

The authors would also like to thank the following members of the ISDE project for their technical and theoretical input: Marc Bolduc, Thierry Baron, Don Bui, Paul MacKenzie, Gregory Dudek, Peter Caines, Carlos Martinez-Mascarva, Tom Mackling, Nicholas Roy, and Michael Daum.

References

[1] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE Transactions on Robotics and Automation*, vol. 2, pp. 14–23, March 1986.

[2] A. Saffiotti, K. Konolidge, and E. H. Ruspini, "A multivalued logic approach to integrating planning and control," *Artificial Intelligence*, vol. 76, no. 1-2, pp. 481–526, 1995.

[3] P. G. Doyle and J. L. Snell, *Random Walks and Electric Networks.* The Mathematical Association of America, 1984.

[4] R. Yerraballi and R. Mukkamalla, "Scalability in real-time systems with end-to-end requirements," *Journal of Systems Architecture*, vol. 42, pp. 409–429, December 1996.

[5] A. Geist, A. Beguelin, J. Dongarra, W. Jian, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing.* MIT Press, 1994.

[6] B. Landau and R. Jackendoff, "What and where in spatial language and spatial cognition," *Behavioral and Brain Sciences*, vol. 16, pp. 217–265, 1993.

[7] G. A. Miller and P. N. Johnson-Laird, *Language and Perception.* Harvard University Press, 1976.

[8] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline.* Prentice-Hall, Inc., 1996.

[9] T. Lozano-Perez, *Autonomous Robot Vehicles.* Springer-Verlag, 1990.

[10] A. Elfes, "Sonar-based real-world mapping and navigation," *IEEE Journal of Robotics and Automation*, vol. 3, pp. 249–265, 1987.

[11] R. G. Simmons, "Structure control for autonomous robots," *IEEE Transactions on Robotics and Automation*, vol. 10, pp. 34–43, February 1994.

[12] R. Liscano, R. Fayek, A. Manz, E. Stuck, and T. Tigli, "Using a blackboard to integrate multiple activities and achieve strategic reasoning for mobile robot navigation," *IEEE Expert*, pp. 24–36, 1995.

[13] D. Simon, B. Espiau, E. Castillo, and K. Kapellos, "Computer-aided design of a generic robot controller handling reactivity and real-time control issues," *IEEE Transactions on Control Systems Technology*, vol. 1, pp. 213–229, December 1993.

[14] R. Simmons, R. Goodwin, K. Z. Haigh, S. Koenig, and J. O'Sullivan, "A layered architecture for office delivery robots," in *First International Conference on Autonomous Agents*, (Monterey, CA), Feb. 1997.

[15] E. Gat, R. Desia, R. Ivlev, J. Loch, and D. Miller, "Behavior control for robotic exploration of planetary surfaces," *IEEE Transactions on Robotics and Automation*, vol. 10, pp. 490–503, August 1994.

[16] M. J. Mataric, "Integration of representation into goal-driven behavior-based robots," *IEEE Transactions on Robotics and Automation*, vol. 8, pp. 304–312, June 1992.

[17] H. Hexmoor and D. Kortenkamp, "Issues on building software for hardware agents," *Knowledge Engineering Review*, vol. 10, no. 3, pp. 301–304, 1995.

[18] R. Byrnes, "The rational behavior software architecture for intelligent ships," *Naval Engineers Journal*, March 1996.

[19] J. Bellingham and J. Leonard, "Task configuration with layered control," in *Proceedings of the IARP 2nd Workshop on Mobile Robots for Subsea Environments*, (Monterey, CA), pp. 193–302, May 1994.

[20] F. R. Noreils and R. G. Chatila, "Plan execution monitoring and control architecture for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 11, pp. 255–266, April 1995.

[21] F. F. Ingrand, M. P. Georgeff, and A. S. Rao, "An architecture for real-time reasoning and system control," *IEEE Expert*, pp. 34–44, December 1992.

[22] R. J. Firby, "An investigation into reactive planning in complex domains," in *Proceedings AAAI-87 Sixth National Conference on Artificial Intelligence*, pp. 202–206, AAAI, July 13-17 1987.

[23] E. Gat, "Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots," *Proceedings of the AAAI92*, 1992.

[24] M. P. Georgeff and A. L. Lansky, "Reactive reasoning and planning," in *Proceedings AAAI-87 Sixth National Conference on Artificial Intelligence*, pp. 677–682, AAAI, July 13-17 1987.

[25] R. P. Bonasso, R. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack, "Experiences with an architecture for intelligent reactive agents," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, no. 1, 1997.

[26] R. C. Arkin, "Integrating behavioral, perceptual, and world knowledge in reactive navigation," *Robotics and Autonomous Systems*, vol. 6, pp. 105–122, 1990.

[27] R. C. Arkin, "The impact of cybernetics on the design of a mobile robot system: A case study," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, pp. 1245–1257, November/December 1990.

[28] J. S. Zelek, *SPOTT: A Real-time Distributed and Scalable Architecture for Autonomous Mobile Robot Control.* PhD thesis, McGill University, Dept. of Electrical Engineering, 1996.

[29] S. Zilberstein, "Using anytime algorithms in intelligent systems," *Artificial Intelligence Magazine*, pp. 73–83, fall 1996.

[30] J. J. Leonard and H. F. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons," *IEEE Transactions on Robotics and Automation*, vol. 7, pp. 376–382, June 1991.

[31] K. Wu, *Computing Parametric Geon Descriptions of 3D Multi-part Objects.* PhD thesis, McGill University, Dept. of Electrical Engineering, 1996.

[32] N. J. Nilsson, "Toward agent programs with circuit semantics," Tech. Rep. STAN-CS-92-1412, Department of Computer Science, Stanford University, Stanford, California 94305, January 1992.

[33] N. J. Nilsson, "Teleo-reactive programs for agent control," *Journal of Artificial Intelligence Research*, vol. 1, pp. 139–158, 1994.

[34] R. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, pp. 189–208, 1971.

[35] L. P. Kaebling and S. J. Rosenschein, "Action and planning in embedded agents," *Robotics and Autonomous Systems*, vol. 6, pp. 35–48, June 1990.

[36] J. Kosecka and R. Bajcsy, "Discrete event systems for autonomous mobile agents," *Robotics and Autonomous Systems*, vol. 12, pp. 187–198, 1994.

[37] N. Nissanke, *Realtime Systems.* Prentice Hall, 1997.

[38] D. M. Lyons, "Representing and analyzing action plans as networks of concurrent processes," *IEEE Transactions on Robotics and Automation*, vol. 9, pp. 241–256, June 1993.

[39] D. M. Auslander, "What is mechatronics?," *IEEE/ASME Transactions on Mechatronics*, vol. 1, no. 1, pp. 5–9, 1996.

[40] E. Koutsofios and S. C. North, "Applications of graph visualization," in *Proceedings of Graphics Interface 1994 Conference*, (Banff, Canada), pp. 235–245, May 1994.

[41] P. Mackenzie and G. Dudek, "Precise positioning using model-based maps," in *IEEE International Conference on Robotics and Automation*, vol. 2, pp. 11615–1621, San Diego, CA: IEEE, May 1994.

[42] D. Bui, "Quadris: A range sensor for navigation and landmark recognition," Master's thesis, McGill University, Dept. of Electrical Engineering, in preparation.

[43] C. I. Connolly and R. A. Grupen, "On the applications of harmonic functions to robotics," *Journal of Robotic Systems*, vol. 10, no. 7, pp. 931–946, 1993.

[44] W. F. Ames, *Numerical Methods for Partial Differential Equations.* Academic Press Inc., 1992.

[45] D. Halperin, L. Kavraki, and J.-C. Latombe, "Robotics," ch. 41, pp. 755–778, Boca Raton, FL: CRC Press, 1997.

[46] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, vol. 5, pp. 90–98, Spring 1986.

[47] J. Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach," *International Journal of Robotics Research*, vol. 10, pp. 628–649, 1991.

[48] A. Stentz, "The focussed d* algorithm for real-time replanning," in *Proceedings of the International Joint Conference on Artificial Intelligence*, (Montreal, PQ), Aug. 1995.

[49] L. Tarassenko and A. Blake, "Analogue computation of collision-free paths," in *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, pp. 540–545, IEEE, April 1991.

[50] A. van de Vooren and A. Vliegenthart, "On the 9-point difference formula for laplace's equation," *Journal of Engineering Mathematics*, vol. 1, no. 3, pp. 187–202, 1967.

[51] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms.* Addison-Wesley Publishing Co., 1983.

[52] A. Geist, A. Beguelin, J. Dongarra, W. Jian, R. Manchek, and V. Sunderam, "Pvm 3 user's guide and reference manual," Tech. Rep. ORNL-TM-12187, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, USA, May 1993.

[53] V. Sunderam, G. Geist, J. Dongarra, and R. Manchek, "The pvm concurrent computing system: Evolution, experiences, and trends," *Journal of Parallel Computing*, vol. 20, no. 4, pp. 531–546, 1993.

[54] B. Topol, J. T. Stasko, and V. Sunderam, "Integrating visualization support into distributed computing systems," Tech. Rep. Technical Report GIT-GVU-94/38, Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA, October 1994.

[55] F. Blais, M. Rioux, and J. Domey, "Optical range image acquisition for the navigation of a mobile robot," in *Proceedings 1991 IEEE International Conference on Robotics and Automation*, pp. 2574–2586, 1991.

[56] M. Bolduc, "The quadris sensor," Tech. Rep. CIM-TR-96-??, McGill Research Centre for Intelligent Machines, McGill University, Montreal, PQ, Canada, July 1996.

[57] R. Ng, "Geon recognition using a mobile robot visual system," Master's thesis, McGill University, Dept. of Electrical Engineering, July 1998.

[58] J. Borenstein, D. Wehe, L. Feng, and Y. Koren, "Mobile robot navigation in narrow aisles with ultrasonic sensors," in *ANS 6th Topical Meeting on Robotics and Remote Systems*, (Monterey, California), Feb. 5-10 1995.

John S. Zelek (M'86-S'91-M'97) received the Ba.Sc. degree in Systems Design Engineering from the University of Waterloo in 1985, and the Ma.Sc. degree in Electrical Engineering from the University of Ottawa in 1989 and the Ph.D. degree in Electrical Engineering from McGill University in 1996. He is currently an Assistant Professor in the Engineering Systems and Computer Program at the University of Guelph. During 1996-97 he was a Visiting Assistant Professor in the Computer Engineering department at Wright State University and during 1997-98 he was a Visiting Assistant Professor in the Computer Science department at Brock University. His research interests include autonomous mobile robotics control and computer vision. Dr. Zelek received the *Outstanding paper award* at the 1995 Conference on Systems, Man and Cybernetics held in Vancouver.

Martin D. Levine(S'59-M'66-SM'74-F'88) received the B.Eng. and M.Eng. degrees in Electrical Engineering from McGill University, Montreal, in 1960 and 1963, respectively, and the Ph.D. degree in Electrical Engineering from the Imperial College of Science and Technology, University of London, London, England, in 1965. He is currently a Professor in the Department of Electrical Engineering, McGill University and the Director of the McGill Center for Intelligent Machines (CIM). During

1972-1973 he was a member of the Technical Staff at the Image Processing Laboratory of the Jet Propulsion Laboratory, Pasadena, CA. During the 1979-1980 academic year, he was a Visiting Professor in the Department of Computer Science, Hebrew University, Jerusalem, Israel. His research interests encompass computer vision and robotics and he has consulted for various government agencies and industrial organisations in these areas. He has authored Vision in Man and Machine and has coauthored Computer Assisted Analyses of Cell Locomotion and Chemotaxis. Dr. Levine is on the Editorial Board of Computer Vision and Understanding, having also served on the Editorial Boards of the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE and Pattern Recognition. He is also the Editor of the Plenum Book Series on Advances in Computer Vision and Machine Intelligence. He was the General Chairman of the Seventh International Conference on Pattern Recognition held in Montreal during the summer of 1984 and served as President of the International Association of Pattern Recognition during 1988-1990. Dr. Levine is a Fellow of the IEEE and the International Association of Pattern Recognition.