INTERNATIONAL JOURNAL OF INTELLIGENT CONTROL AND SYSTEMS
VOL. 13, NO. 3, SEPTEMBER 2008, 162-177

Layered Mode Selection Logic for Unstructured Environments

Andrew B. WRIGHT, Traig BORN, Gabriel FERRER, and Ann M. WRIGHT

Abstract- **Robots developed from the 60's to the present have been restricted to highly structured environments such as work cells or automated guided vehicles, primarily to avoid harmful interactions with humans. Next generation robots must function in unstructured environments. Such robots must be fault tolerant to sensor and manipulator failures, scalable in number of agents, and adaptable to different robotic base platforms. Layered Mode Selection Logic (LMSL), a robot-controller architecture, addresses these concerns. The LMSL architecture has been implemented and tested on UALR's J5 robotics research platform. Objects are classified by acceleration and force measurements. Collisions are detected, and objects are classified by how difficult they are to push. J5 either avoids the obstacles or manipulates them depending on this classification. Comparable results are achieved with all sensors functioning, with only the acceleration sensor (force sensor faulted), and with only the force sensor (acceleration sensor faulted). A second demonstration of the architecture addresses the problem of border security. A team of autonomous robots are configured as a flexible sensor array. A one-dimensional experiment using Lego robots shows that the robots distribute themselves evenly along the border until an intruder penetrates the border. The robots at the point of penetration cluster, whereas the robots removed from the point of penetration remain evenly distributed, albeit with a greater inter-robot separation.**

Index Terms— **Mode Selection Logic, Border Security, Fuzzy Logic, Sensor Fusion, Fault Tolerance**

1. INTRODUCTION

An important goal in robotics research is to design robots capable of performing tasks in unstructured environments. A sample unstructured environment is shown in Fig. 1, where UALR's J5 robot platform is moving boxes with unknown mass. Such robots must be scalable, fault tolerant, and capable of being upgraded without requiring substantial redesign or recoding of the controller structure. The aspect of the robot's design which most greatly influences its ability to satisfy these requirements is the controller architecture.

The world of robot controller architectures is a spectrum with planners at one end and reactive systems at the other. Reactive systems respond quickly to stimuli and are inexpensive to implement; however, complex tasks are difficult to achieve. Implementing a planner could require a more expensive processor, but complex procedural tasks may be accomplished.

Most planner controllers rely on a world model. The world model is either preprogrammed or constructed at run-time from sensory data. Actions are chosen based on the state of the world model and predefined goals. This approach allows the designer to explicitly define the tasks and goals of the system. Planners work well in a controlled environment such as an assembly line, where the world model can either be pre-programmed or where the sensor data can be validated.

In unstructured environments, the limitations of a planner become apparent. Uncertainty in sensor data is problematic. Faulty sensor data can cause an inaccurate world model to be constructed, which interferes with the planner's ability to select an optimal course of action. Planners have been criticized for their lack of scalability to the complexity of real world tasks [1][2]. Planners are not easily scalable to multi-agent systems, as the complexity of the planner increases exponentially with additional agents.

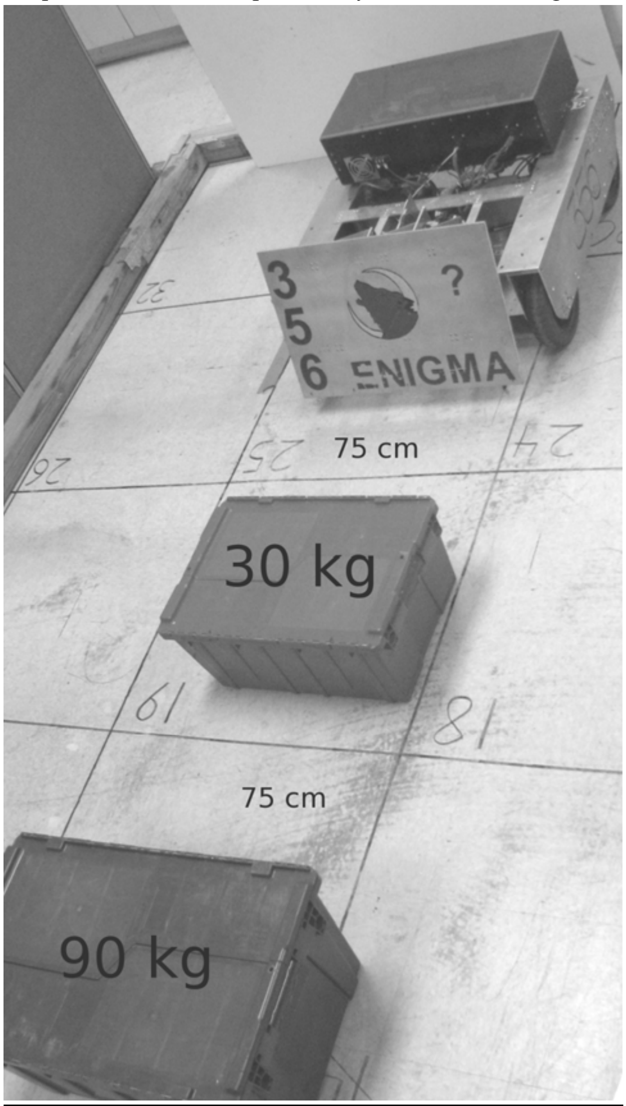


Fig. 1 J5 Robot Platform in Forward Collision Experiment

Received on Oct. 8, 2007, revised on May 1, 2008. This paper is extended from "Layered Mode Selection Logic Control for Border Security" and "Layer Mode Selection Logic Control with Fuzzy Sensor Fusion Networks" published at SPIE Defense and Security Conference, Orlando, FL, USA, April, 2007.
T. Born and A. M. Wright are with Dept. of Physics, Hendrix College, 100 Washington Ave., Conway, AR, USA (e-mail: born@hendrix.edu and wright@hendrix.edu); G. Ferrer is with the Dept. of Mathematics and Computer Science, Hendrix College (email: ferrer@hendrix.edu) and A. B. Wright is with Univ. of Ark. at Little Rock, 2801 S. University Ave., Little Rock, AR, USA (e-mail: abwright@ualr.edu).

Reactive systems use functions to map sensor inputs to actuator outputs. Since no world model is maintained, errors due to sensor inaccuracy do not accrue over time. Reactive approaches have better computational efficiency, but their inability to dynamically store data results in a lack of optimality and goal convergence [2].

Hybrid control architectures combine features from reactive systems and planner systems in an attempt to reap the benefits of each, while minimizing their inherent drawbacks. Previous work has investigated robot software architectures for integrating planning and reactive systems [3][4], as well as the application of fuzzy logic to these problems [5]. The behavior-based controller architecture, a specific example of such a hybrid, uses "basis behaviors" as building blocks to achieve more complex functionality [6]. A single basis behavior is reactive. When the robot interacts with the environment and with other robots, the higher level behavior that results is called an emergent behavior. Behavior-based architectures have been shown to be scalable to multi-robot systems [2].

Behavior-based controllers require a method for combining the outputs of the basis behaviors. Pirjanian gives an overview of methods used to combine behaviors, which he calls "coordination mechanisms" [7]. Coordination mechanisms are divided into two categories: arbitration methods and command fusion methods.

An arbitration method selects one behavior and gives it complete control. Most arbitration methods produce locally optimal decisions which may be globally non-optimal. For example, a robot may be programmed to follow a beacon, which is to the left of the robot, when an obstacle is encountered. The obstacle avoidance behavior directs the robot to avoid the obstacle by turning right. The global goal of following the beacon would have been better served by turning left to get around the obstacle.

Command fusion methods combine control signals from multiple behaviors into a single command. When the behaviors have conflicting goals, the combined command signal from conflicting behaviors may result in a non-optimal action. In the example of the robot avoiding an obstacle while following a beacon, if the obstacle avoidance behavior gives a right turn command and the follow beacon behavior issues a left turn command, the combined command would be to go straight, resulting in a collision with the obstacle.

Arkin, et al. described a "finite state machine" based behavior coordination approach [8], which is based on Brooks's subsumption architecture [9]. In Arkin's work, a "sequence of perceptual algorithms" are developed independently and combined in a finite state architecture using transition conditions based on distinct robot sensory perceptions. The Finite State Acceptor (FSA) that Arkin uses to implement perceptual coordination defines multiple operational states, including error states, and allows for four exit conditions (normal, terminal, recoverable error, fatal error). The states themselves are task-related, where the tasks are associated with types of sensory input used in the task (for instance, one state may use long-range perception to perform a task such as beacon following, while another state may use close range perception to perform a task such as docking). The FSA does not use the complexity of the task in the definition of the state, and the implementation of the states may have significant overlap in its procedures. It is possible in Arkin's FSA for two states to accomplish the same sub-task using different procedures (e.g., obstacle avoidance may be used in both long range motion and docking, but with a completely different sensor set and implementation).

Brooks's subsumption architecture is also state-based; however, Brooks builds a concept of layering into the architecture. Simple tasks are called "level 0" tasks and are closely tied to direct actuator commands. "Level 1" tasks (and higher) are built on top of the lower level tasks. Brooks introduces the concept that task complexity can be accomplished by building modules and layering them to achieve increasingly complex tasks. Brooks's states may be at varying levels of complexity and are completely independent, in the same way as Arkin's states are. Although Brooks introduces the concept of layering, it is implemented by creating independent higher level procedures, which subsume the lower level procedures by inhibiting them. There is no overlap in function between the higher level procedures and the lower level procedures.

Both Arkin's and Brooks's architectures present the design graphically as a single layer. This complicates the design effort, possibly introducing undesirable state interaction effects, and results in a large number of "building block" states which must be specifically designed. The states are usually designed "to the task" (for example, "wander" or "return to start") rather than using building blocks composed of primitive actions such as "go forward" or "turn." Both schemes envision implementation through motor commands, rather than abstracting the hardware from the control system design, and both schemes use sensory input in the state algorithm, rather than restricting sensory input to the transitions.

Brooks's architecture is specifically designed to include "scalability to task complexity" but includes this feature by

Table 1 Functional Requirements for LMSL

Functional Requirement	Design Element
● Scalable to system complexity ● Tolerant of manipulator failure ● Allow for integration of new modes	Mode Selection Logic
● Addition of manipulators	Action layer
● Scalable to number of robots	Behavior layer
● Scalable to task complexity	Layered Mode Selection Logic
● Addition of sensors ● Tolerant of sensor failure and uncertainty.	Object Classifier such as Fuzzy Sensor Fusion Network

augmenting lower level control system designs with extra states that are inherently more complex. The more complex states tend to use more advanced sensory input. More complex algorithms, such as "path plan," can be used to accommodate more complex tasks.

Both architectures demonstrate the power behind state-based control schemes, especially to build failure resolution into the design. Arkin's FSA presents an elegant formalism for a state-based controller with multiple transitions; however, it lacks a clear means of layering. States are presented at the same level of complexity (i.e. a simple task might transition to a complex task) and the tasks themselves are *ad hoc*.

1.1. Functional Requirements

Every design task, including designing a robot that is capable of performing useful tasks in an unstructured environment, is facilitated by identifying the functional requirements for the problem [10]. The functional requirements must be generalized to allow for good performance in any number of tasks or environments and should be independent of each other. Each functional requirement should be satisfied by a unique design element.

Brooks [9] presents elements which he calls "dogmatic principles," several of which are functional requirements, others of which are philosophical principles which do not affect design. The CAMPOUT architecture[11] also determined several functional requirements which its specification should fulfill.

Functional requirements are part of the design effort. The Central Arkansas Mobile Robotics Consortium's (CAMRC) [1] functional requirements for a multi-agent robot functioning in an unstructured environment are summarized in Table 1. These requirements are synthesized from those presented in [9] and [11].

1.1.1. Scalability to Robot Complexity

The specifics of a robot's hardware often are embedded in the controller architecture, causing the architecture to be bound to a specific hardware configuration. Scalability to robot complexity means that the same architecture may be used to control robots of varying configuration. This specification requires that the hardware (both sensors and actuators) be abstracted from the architecture.

The robot must maintain functionality when non-critical sensors or actuators fail. A minimal set of sensors and manipulators are necessary for basic functionality. Redundant sensors can be used for critical functions, so the architecture should accommodate redundancy. Redundant actions can be used for critical functions, so actuator failures will not lead to system failure. The architecture must accommodate ease of adding functions and ease of inhibiting function when actuators have failed.

[1] http://robotics.ualr.edu

1.1.2. Addition of Manipulators

The robot must allow for expandability. Advances in sensor and actuator technology are frequent. Non-essential sensors and actuators are added to allow for enhanced capabilities. Hence, it is necessary for a robot to be capable of integrating new technologies into its existing infrastructure. The addition of the new hardware should not require radical system redesign. The addition of new procedures, which make use of the new hardware, should not disrupt existing robot procedures.

1.1.3. Scalability to number of robots

Many of the tasks performed in an unstructured environment can be done more effectively by a team of robots. "Scalability to number of robots" means that the architecture allows coordination of a group of robots without exponentially increasing the complexity of the control implementation.

1.1.4. Scalability to task complexity

"Scalability to task complexity" refers to the ability of the architecture's task specification language to allow for simple or complex tasks to be defined. In an unstructured environment, robots are required to perform tasks of varying complexity. Specifications which are *ad hoc* do not scale to task complexity, whereas specifications which build complex routines from simple building blocks are.

1.1.5. Sensor integration and uncertainty

Modern sensors are precise and accurate under controlled conditions. In unstructured environments sensors are subject to disturbances which cause sensor data to be noisy and imprecise. For a robot to perform well in an unstructured environment, the architecture must accommodate sensor noise and imprecision.

2. Layered Mode Selection Logic

CAMRC is developing an architecture, Layered Mode Selection Logic, (LMSL) which is an implementation of a

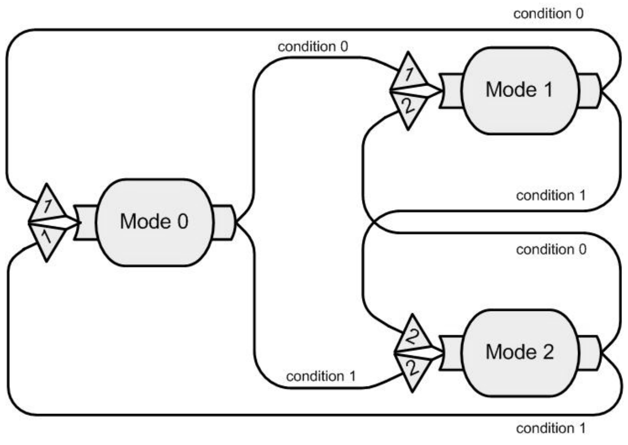


Fig. 2. Generic Mode Selection Diagram

behavior-based controller that can follow a plan. The architecture creates an abstraction layer for the robot's sensory inputs to classify objects in the environment according to attributes that are relevant to the specific problem and to classify events, such as a collision, by setting or clearing a flag. It creates an abstraction layer for the robot actuators through a reactive layer, called the Action layer. Switching among Actions based on the transitions of the discrete flags follows the rules of Mode Selection Logic (MSL) [12].

2.1. Description of Mode Selection Logic Paradigm

Mode Selection Logic (MSL) was originally proposed to control the Lycoming AGT1500 engine [13] in the M1A2 tank. Its purpose was to define safe modes into which the control system would default when sensors failed. This paradigm was successfully implemented and tested on UALR's hybrid rocket motor [12].

MSL is similar to a Finite State Machine in that a system has a finite number of states, which are called modes. In each mode, rules are defined which govern the system's behavior. Well designed modes use a minimum of sensory information to operate. Modes must be designed so that they are independent of each other, allowing a mode to be inserted or removed from the system without affecting the operation of other modes.

Each mode has a predefined set of conditions which, when satisfied, trigger the transition to another mode. When an exit condition is satisfied a transition flag is set. The MSL selects a new active mode based on which transition flags are set and their priority level.

The sensory network contains all of the sensor inputs into the system. If sensors are used only to set transition flags, then the modes become open loop reactive algorithms. In this ideal case, the two aspects of control, sensing and actuating, are decoupled. Integrating new sensors affects only the sensory network. Integrating new actuators affects only the modes.

A Mode Selection Diagram (MSD) is used to show a set of modes and their transition flags (see Fig. 2). Mode 0 is initially the active mode (i.e. the entry mode). Mode 0 will remain active until either condition 0 or condition 1 is set. If condition 0 is set then mode 1 will be selected as the active mode. If condition 1 is set then mode 2 will be selected as the active mode. A mode can have any number of transition flags. The number in the arrow indicates the priority of the transition flag, so if both flags are set, the higher priority mode will be entered.

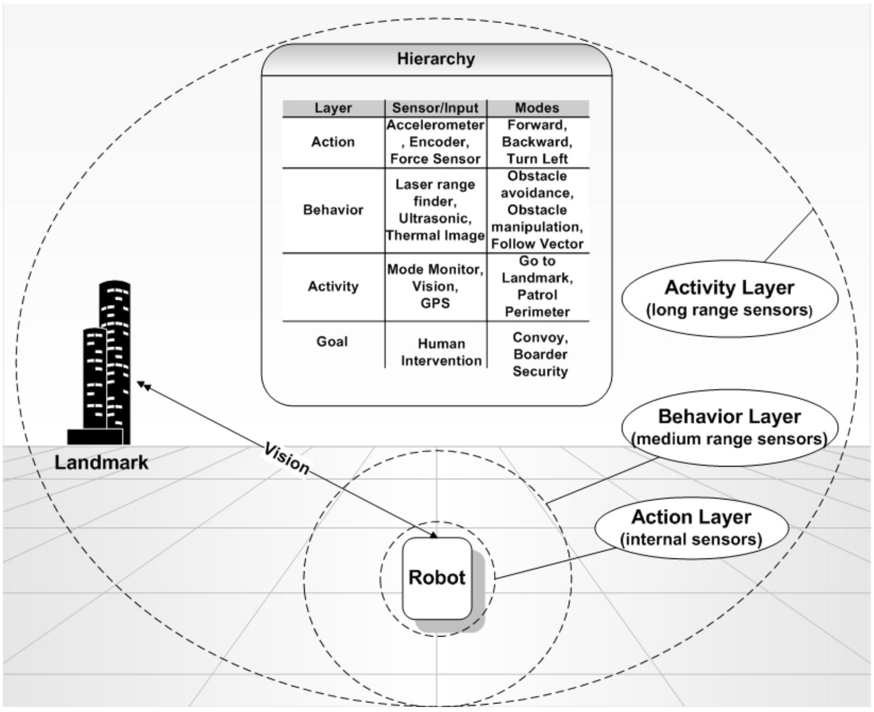
MSL satisfies the functional requirement, "Scalability to System Complexity" (see Table 1). In the hybrid rocket problem, sensors and operational modes were added without major recoding or debugging efforts. Since a control failure on that system could result in an explosion, software/hardware integration testing was emphasized in the design phase.

For a specific problem, modes and transition conditions must be defined that are relevant to the problem. The modes should be made as independent as possible (not necessarily requiring formal functional orthogonalization), and sensor inputs should be confined to the transition conditions. With these restrictions, it is straight-forward to build a library of functions, from which more complicated routines can be constructed, as described in section 2.2.

2.2. MSL as a Behavior Coordination Mechanism

Based on the work on gas turbine engine control and hybrid rocket control, the original implementation of a controller on the J5 robot platform (see Fig. 1) used the MSL design. The first set of problems attempted required that the functional activities be decomposed according to the types of sensors used on the robot. A "Robot Perception Hierarchy" was defined (see Fig. 3) wherein the different sensors were classified according to their response time, range, expense, and required computation.

Sensors which are close to the robot (such as touch sensors or force sensors) or which sense robot-specific information (wheel encoders or inertial sensors) support a reactive control approach. Information update rates are



Hierarchy

Layer	Sensor/Input	Modes
Action	Accelerometer, Encoder, Force Sensor	Forward, Backward, Turn Left
Behavior	Laser range finder, Ultrasonic, Thermal Image	Obstacle avoidance, Obstacle manipulation, Follow Vector
Activity	Mode Monitor, Vision, GPS	Go to Landmark, Patrol Perimeter
Goal	Human Intervention	Convoy, Boarder Security

Fig. 3. Sensory Hierarchy for LMSL Architecture

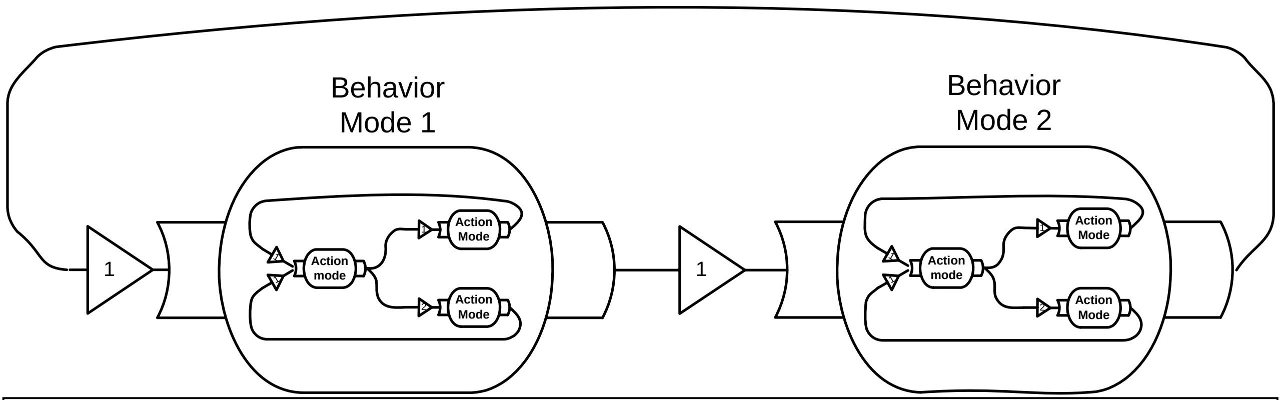


Fig. 4. Behavior Layer Mode Selection Diagram

very fast, and processing time to compute a reaction is minimal. The information is immediate, but local.

Sensors which provide medium-range information (laser range finder, ultrasonic sensor) provide more information about the area in which the robot is functioning. These sensors are more expensive and are either updated slowly or require a larger investment in computational resources. It is difficult to implement meaningful behaviors without this group of sensors.

Sensors which provide long range information, such as vision sensors, are very expensive and require dedicated computational resources to extract useful informational features. The update rates for these sensors are extremely slow relative to the reactive sensors. Vision sensors rely on ambient light or artificial light to function and are not useful in its absence, so the information may be available to the robot only for a portion of its operational time.

It became apparent that this layering of perception would best be accommodated by a layered control architecture. Since the Mode Selection Logic supported the feature of adding complexity, it was decided to design the minimal robot system and add complexity. This approach gave rise to the concept of Layered Mode Selection Logic.

The first Behavior was built using a Mode Selection Diagram (MSD), where each mode was a reactive function, called an Action. The Action layer encapsulates basic robot functions, such as forward, turn left, or actuate manipulator. The Action layer modes provide hardware abstraction by using a function library to execute the motor commands. This method allows the same Action mode to be used on robots with varied actuator sets. For example, a walking robot and a wheeled robot will both have forward mode. The abstraction facilitates integration of new hardware.

Once two behaviors were designed (Wander and Avoid) using Action layer commands (go forward, random turn, go backward) and timers to transition among the Actions, it became necessary to choose a Behavior Coordination Mechanism to transition among the Behaviors. The natural method for transitioning among the Behaviors was to treat each Behavior as a mode and to define sensor-based transition flags. A recursion on the Mode Selection Logic served as an arbitration mechanism (see Fig. 4).

At the Action layer, transitions were accomplished either through timers or very low level sensors, such as touch sensors. At the Behavior layer, more sophisticated sensory input was required. In particular, a means of detecting a collision was required to initiate a transition from Wander to Avoid. Ultimately, force sensors and wheel acceleration measurements were used to detect collisions. However, ultrasonic sensors may be used to initiate a transition from Wander to Avoid without robot/object contact.

Once the first layering was accomplished, the different classes of sensors could be accommodated at different layers in the Layered Mode Selection Logic (LMSL) hierarchy (see Fig. 3). An Activity layer was defined by transitions among Behaviors, and a Goal layer was defined by transitions among Activities. The main implementation detail remaining was to develop a methodology whereby timers, sensors, and mode monitors could initiate transitions at the higher layers.

Planner-type problems could be accomplished at the higher layers. In this respect, the LMSL architecture represents a hybrid architecture. In fact, as more complex problems were defined, the solution has been to add a layer. The work which has been accomplished at the lower layers does not need to be repeated for each new problem. Therefore, the LMSL architecture addresses the functional requirement, "Scalability to Complexity of Task."

The current hierarchy which has been implemented on UALR's J5 robot contains the Action layer (reactive), the Behavior layer, the Activity layer, and the Goal layer. For more sophisticated problems, a Personality layer, which switches among Goal layer modes is envisioned. The implementation of the Action layer on J5 has been done through a device driver, which accepts generic commands from the operating system. This gives the added bonus that specific controllers can be implemented through a data file describing the Actions and transitions at each layer. Implementing an increasingly complex controller on a specific robot does not involve recompiling code, but rather involves over-writing the data structure containing modes and mode transitions.

Since controllers are designed by drawing Mode Selection Diagrams, LMSL allows for top down goal oriented group behavior design. Design tools based on dropping and connecting predefined modes can be accomplished through a straight-forward Graphical User

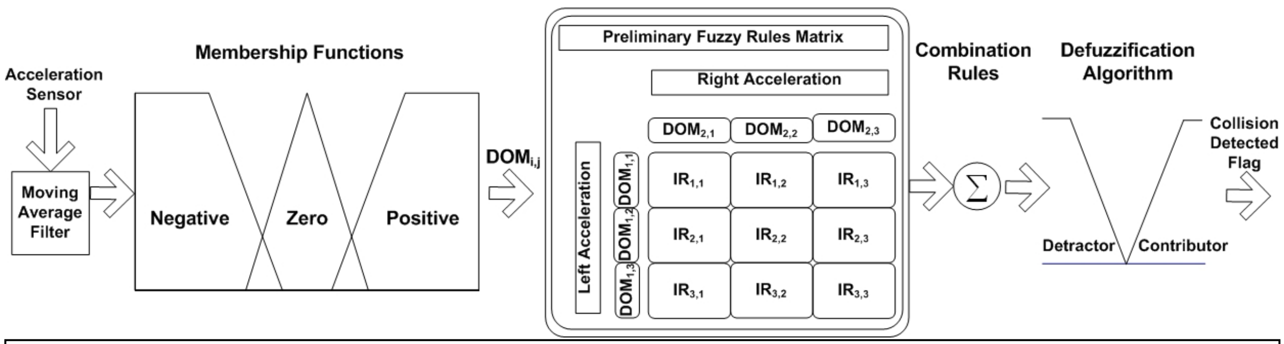


Fig. 5. Preliminary Fuzzy Sensor Fusion Network

Interface (GUI). This architecture supports graphical monitoring tools to indicate what modes a particular robot is in and to monitor the transition patterns.

If the hierarchy is implemented beginning with a minimal set of critical modes, and adding functionality, the architecture should provide inherent tolerance to actuator failure, provided some method of self-diagnosing actuator failure is implemented. If the new, non-essential actuator or sensor fails, then the MSL can disable the modes that require the new features by inhibiting transitions into those modes. As non-essential actuators and sensors fail, the controller collapses back to the minimal set of modes.

3. Object Classification

When the robot interacts with its environment, the robot must classify the objects encountered according to the different reaction choices. Sensor data is presented to a classifier whose output must be a discrete representation to be used as a Transition Flag. Different methods of classification exist in the literature. In particular, Neural Network based Classifiers, Bayesian Classifiers [14][15], and Fuzzy Logic Classifiers [16] have been used to characterize objects based on sensory information. Since Fuzzy Logic Classifiers allow multiple types of sensors to be normalized and fused in a network and are linguistically simple to implement, this option was chosen in preference to more complicated approaches.

The current research indicates that objects must be classified according to manipulability (how easily the robot can move an object), traversability (how easily a robot can climb over an object), avoidability (how easily a robot can circumnavigate an object), and changeability (the persistence of an object in the robot's space).

3.1. Fuzzy Sensor Fusion Network

In unstructured environments, sensor data is imprecise. Fuzzy logic is a type of set theory in which elements have varying degrees of membership in a set (as opposed to the standard binary type set theory where an element either belongs to a set or does not belong) [17]. Partial set membership facilitates the use of less exact definitions of the input to output relationships. Therefore, fuzzy logic improves fault tolerance to sensor imprecision.

A Fuzzy Sensor Fusion Network (FSFN) is a set of membership functions, fuzzy inference rules, composition rules, and defuzzification functions, which result in a discrete flag associated with an object property or event.

In the FSFN shown in Fig. 5, sensor values are mapped onto discrete variables rather than continuous outputs or command signals. The discrete variables signify the occurrence of an event, such as a collision. Each FSFN can trigger multiple discrete variables.

The FSFN combines multiple sensors in a Fuzzy Rules Matrix (FRM). The FRM may be designed to incorporate redundant sensors to the setting of each transition flag. If a sensor is not present or has failed, the transition flag will still be updated due to the other sensors. Signal processing algorithms to condition sensors can be incorporated in the FSFN. The membership functions can be adjusted by learning algorithms to improve the robot's ability to adapt to its environment [18][19][20].

A membership function, which is usually a piece-wise continuous polynomial, determines the degree of membership in a fuzzy set and is denoted

$$DOM_{ij} = \mu_{ij}(x),\ i \in [1,S],\ j \in [1,N_i]\ , \qquad (1)$$

where the index, i, represents the sensor type, S is the number of sensors in the network, the index, j, represents the membership function applied to that sensor, and N_i is the number of membership functions assigned to that sensor. The output of the membership function is a fuzzy variable, which is normalized, $DOM_{ij} \in [0,1]$, where zero represents that the sensor value has no membership in the fuzzy variable and one means that the sensor value has total membership in the fuzzy variable.

The degree of memberships are combined in inference rules using the fuzzy "AND" operator,

$$A \otimes B = \text{minimum}(A,B)\ . \qquad (2)$$

where A and B are the outputs of membership functions.

The product notation is used to denote the combination of many variables,

$$\prod_{j=1}^{N} A_j = A_1 \otimes A_2 \otimes \cdots \otimes A_N\ . \qquad (3)$$

An Inference Rule is a combination of fuzzy variables,

$$IR_{jk\cdots p} = DOM_{1j} \otimes DOM_{2k} \otimes \cdots \otimes DOM_{Sp}\ , \qquad (4)$$

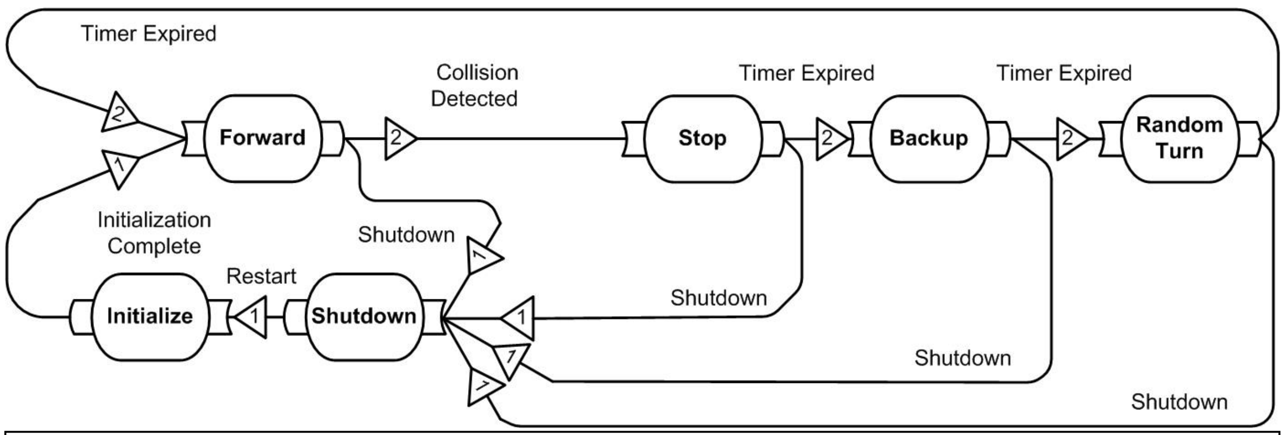


Fig. 6. Obstacle Avoidance Mode Selection Diagram

and the matrix formed by all of the inference rules is called the Fuzzy Rules Matrix (FRM). The FRM is a multi-dimensional matrix, whose dimensions are $N_1 \times N_2 \times \cdots \times N_S$.

Each event has two combination rules associated with it, a contributor rule and a detractor rule. Determining contributor and detractor rules involves experiments. A simulation of the condition to be detected is created, and the rules in the FRM are monitored through the experiment. The inference rules which give detectable results when the event occurs are selected for the contributor rule. Inference rules which give detectable values when the event is not occurring and non-detectable values (i.e. signal-to-noise ratio between the rule and the noise floor is very low) when the event is occurring are selected for the detractor rules. Inference rules which give inconsistent results are omitted from both rules. Inference rules are combined using the fuzzy "OR" operator,

$$A \oplus B = \text{maximum}(A, B) \ , \tag{5}$$

where the summation of many fuzzy "ORs" is denoted,

$$\sum_{i=1}^{n} A_i = A_1 \oplus A_2 \oplus \cdots \oplus A_n \ . \tag{6}$$

If M specific Inference Rules from the FRM denoted by equation (4) are the set

$$\left\{ IR_{J_1 K_1 \cdots P_1}, IR_{J_2 K_2 \cdots P_2}, \cdots, IR_{J_M K_M \cdots P_M} \right\}, \tag{7}$$

then a contributor rule composed of C inference rules is

$$CR = \sum_{i=1}^{C} IR_{J_i K_i \cdots P_i} \ , \tag{8}$$

and a detractor rule composed of D inference rules is

$$DR = \sum_{i=1}^{D} IR_{J_i K_i \cdots P_i} \ . \tag{9}$$

In defuzzification, the contributor and detractor rules are compared. If the value of the contributor rule is greater than the value of the detractor rule, the detection flag is set. Otherwise, it is clear.

4. LMSL Applied to Obstacle Handling

In autonomous mobile robotics, obstacle avoidance techniques are used for motion planning. In obstacle avoidance, an algorithm steers the robot around obstacles using sensors to locate and avoid the object. These systems are designed to work in indoor environments and safely share their workspace with humans. In obstacle avoidance, it is assumed that the environment should not be altered or that the robot is incapable of altering it, and therefore the best way to handle obstacles is to go around them. Situations exist where obstacle avoidance is not the best choice, such as search and rescue, hazardous material clean up, and battlefield reconnaissance. In these cases, it is preferable that the robot is capable of handling obstacles instead of avoiding them. Object handling may refer to avoiding the obstacle, manipulating it, or traversing it.

Obstacle manipulation enables a robot to remove an obstacle impeding its path and continue along an optimal route to the objective. This behavior enhances the navigational capabilities of the robot by allowing it to choose a more direct route. In the case where all routes are blocked by movable objects, it creates a solution that would otherwise be unattainable. However, if the obstacle is massive enough that manipulating it will cause excessive drain to the batteries, then obstacle avoidance is a more efficient course of action.

UALR's J5 robot[2] was used in developing obstacle avoidance and obstacle manipulation using the LMSL architecture. J5 has a mass of 59 kg and is .75 m wide by 1 m long by .45 m tall (see Fig. 1). It has a robust differential drive system powered by DC motors. The drive axles have optical encoders. The control system hardware consists of an AT motherboard and an x86 architecture processor and custom-built circuits for motor control. J5 uses a 512 MB flash drive for non-volatile data storage. The operating system is Mandrake 9.2 Linux. J5 has an articulated inclined plane, called the Wedge, located on the front of the robot. The wedge has a strain gauge bonded to its rear surface to provide force measurements.

[2] http://theduchy.ualr.edu/txborn/

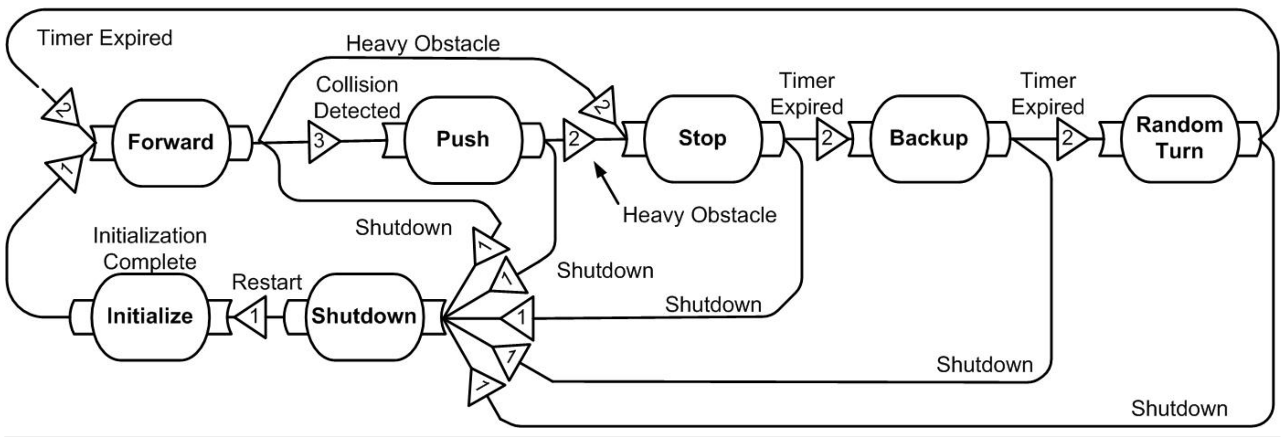


Fig. 7. Obstacle Manipulation Mode Selection Diagram

4.1. Behavior design

Obstacle avoidance was implemented using LMSL, and manipulation was added after the avoidance MSD had been debugged. In the obstacle avoidance behavior (see Fig. 6), J5 enters the forward mode when initialization is complete. It continues to move forward until a collision is detected. If a collision is detected while in the forward mode, the robot enters a stop, back up, and random turn sequence. Once the random turn is complete, the robot returns to forward mode.

To create the obstacle manipulation behavior, a new action, "Push," and a new transition flag, "Heavy Obstacle," were added to the avoidance behavior (see Fig. 7). This approach of creating more complex algorithms without disturbing existing functionality is similar to Brook's subsumption architecture approach; however, the level of complexity of manipulation in LMSL is at the same layer as avoidance. The complete behavior will be called handling, of which avoidance, manipulation, and traversal are separate components.

In the obstacle manipulation behavior, a collision-detection initiates a transition into push mode. The push mode directs the robot to make a forward arcing turn. If a heavy obstacle is detected while in either the forward or push mode, the robot enters a stop, back up, and random turn sequence. Once the random turn is complete, the robot returns to forward mode.

4.2. Fuzzy sensor fusion network design

To implement avoidance and manipulation, a fuzzy sensor fusion network to detect "collision" and "heavy obstacle" had to be designed. Wheel acceleration derived from the left and right wheel encoders were used to detect a collision. The conditioned sensor value is processed by three membership functions, each of which creates a separate fuzzy variable (negative acceleration, zero acceleration, positive acceleration).

The generic "negative" membership function is

$$\mu_{\text{neg}}(x)=\begin{cases} 1 & x<x_{\min} \\ \left(1-\dfrac{x-x_{\min}}{x_{\max}-x_{\min}}\right) & x_{\min}\le x\le x_{\max} \\ 0 & x_{\max}<x \end{cases}. \tag{10}$$

The generic "positive" membership function is

$$\mu_{\text{pos}}(x)=\begin{cases} 0 & x<x_{\min} \\ \left(\dfrac{x-x_{\min}}{x_{\max}-x_{\min}}\right) & x_{\min}\le x\le x_{\max} \\ 1 & x_{\max}<x \end{cases}. \tag{11}$$

The generic function for "zero" values is

$$\mu_{\text{zero}}(x)=\begin{cases} 0 & x<x_{\text{l}} \\ \left(1-\left|\dfrac{2x-(x_{\text{r}}+x_{\text{l}})}{(x_{\text{r}}-x_{\text{l}})}\right|\right) & x_{\text{l}}\le x\le x_{\text{r}} \\ 0 & x_{\text{r}}<x \end{cases}. \tag{12}$$

The outputs of these membership functions are combined through a FRM to generate all possible combinations of rules (in this case, two sensors, each of which has three states, the matrix is 3 by 3, see Fig. 6), and the inference rules are

$$IR_{jk}=DOM_{1j}\otimes DOM_{2k}, \tag{13}$$

where $\{DOM_{11},DOM_{12},DOM_{13}\}$ are the outputs from the left acceleration sensor membership functions and $\{DOM_{21},DOM_{22},DOM_{23}\}$ are the outputs from the right acceleration membership functions.

Collision characterization experiments were conducted and the output of the rules matrix was observed. The four inference rules which were active prior to a collision and after a collision, but were inactive during a collision, were chosen as the collision detractor rules,

$$\{IR_{22},IR_{23},IR_{33},IR_{32}\}. \tag{14}$$

Only one rule was active during a collision, but inactive before and after it, and the collision contributor rule set consists of the single rule,

$$\{IR_{11}\}. \tag{15}$$

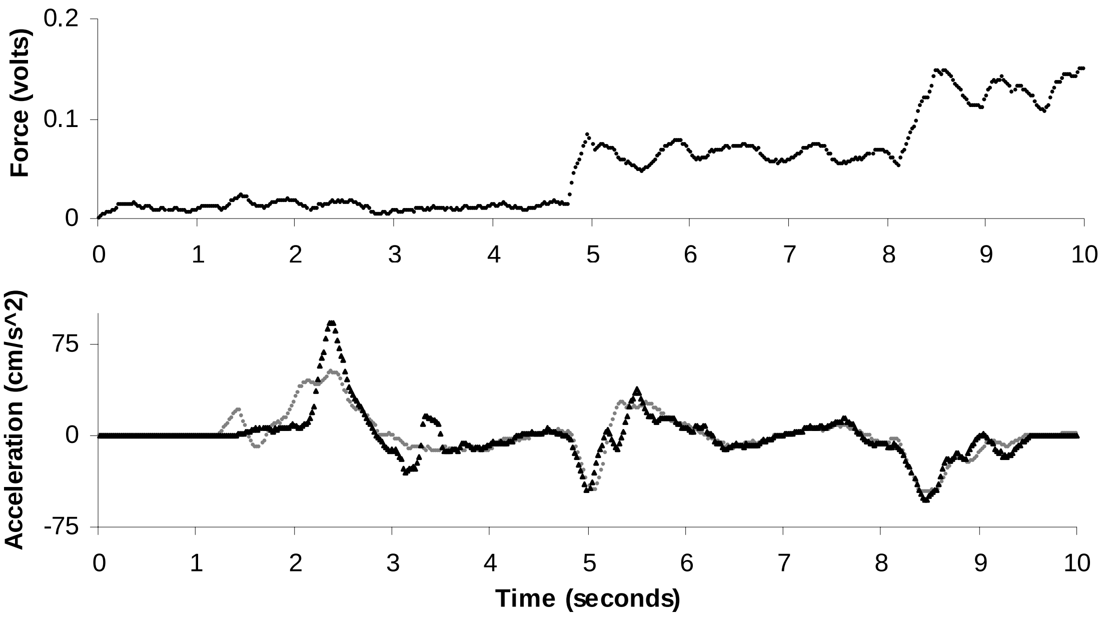


Fig. 8. Sensor response during manipulation experiment

This FSFN is capable of detecting collisions, but due to the nonlinearity of acceleration response it is not able to reliably differentiate between different obstacle masses. Tracking error in the velocity causes acceleration spikes, which are erroneously detected as collisions. Force of contact between the robot and the mass was added to the inference rule set to correct these problems. The force sensor has a linear response across a larger range of obstacle masses. The contact force verifies that the acceleration spike is due to a collision, not tracking error.

The force sensor has three membership functions (low from equation 10, medium from equation 12, and high from equation 11). Instead of creating a new rules matrix, the existing 3x3 acceleration matrix outputs are combined into a higher order 3x3x3 matrix with the force sensor. The addition of the force sensor to the FSFN augmented equation 13 to include the new DOMs.

$$IR_{jkl} = DOM_{1j} \otimes DOM_{2k} \otimes DOM_{3l}\,, \tag{16}$$

where $\{DOM_{31}, DOM_{32}, DOM_{33}\}$ are the outputs from the force membership functions.

The collision experiments were repeated and the output of the fuzzy rules matrix was observed. The inference rule set which comprises the Collision Detractor Rule becomes

$$\{IR_{221}, IR_{231}, IR_{331}, IR_{321}\}\,, \tag{17}$$

and the inference rule set which comprises the Collision Contributor Rule becomes

$$\begin{Bmatrix} IR_{111}, IR_{212}, IR_{222}, IR_{312}, IR_{112}, IR_{122}, IR_{132}, \\ IR_{113}, IR_{123}, IR_{133}, IR_{213}, IR_{223}, IR_{233}, IR_{313}, \\ IR_{323}, IR_{333} \end{Bmatrix}. \tag{18}$$

The Push behavior required the Heavy Obstacle Detected signal. Experiments showed that the Heavy Obstacle Contributor rule should be composed of the inference rules from equation 16,

$$\begin{Bmatrix} IR_{113}, IR_{123}, IR_{133}, IR_{213}, IR_{223}, \\ IR_{233}, IR_{313}, IR_{323}, IR_{333} \end{Bmatrix}, \tag{19}$$

and the Heavy Obstacle Detractor rule should be composed of the inference rule set

$$\begin{Bmatrix} IR_{111}, IR_{121}, IR_{131}, IR_{211}, IR_{221}, IR_{231}, \\ IR_{311}, IR_{321}, IR_{331}, IR_{112}, IR_{122}, IR_{132}, \\ IR_{212}, IR_{222}, IR_{232}, IR_{312}, IR_{322}, IR_{332} \end{Bmatrix}. \tag{20}$$

4.3. Manipulation Experiment

To test the Obstacle Identification FSFN, validation experiments were conducted. The robot was placed on a starting mark. A 30 kg rectangular obstacle was placed 75 cm in front of the robot perpendicular to the robot's direction of travel. A second 90 kg rectangular obstacle was placed 75 cm behind the first (see Fig. 1). The robot was directed to drive forward for 10 seconds at a velocity of 26 cm/s.

At approximately 5 seconds, the robot collided with the first obstacle, resulting in a negative acceleration spike and a step increase in force (see Fig. 8). The robot pushed the obstacle forward, until, at approximately 8 seconds, the second collision occurred. A second negative acceleration spike and a step increase in the force occurred.

The first collision is detected by the FSFN, but not identified as a heavy obstacle (see Fig. 9). The forward collision flag is set. The second collision triggers the heavy obstacle flag (see Fig. 9).

4.4. Fault tolerance

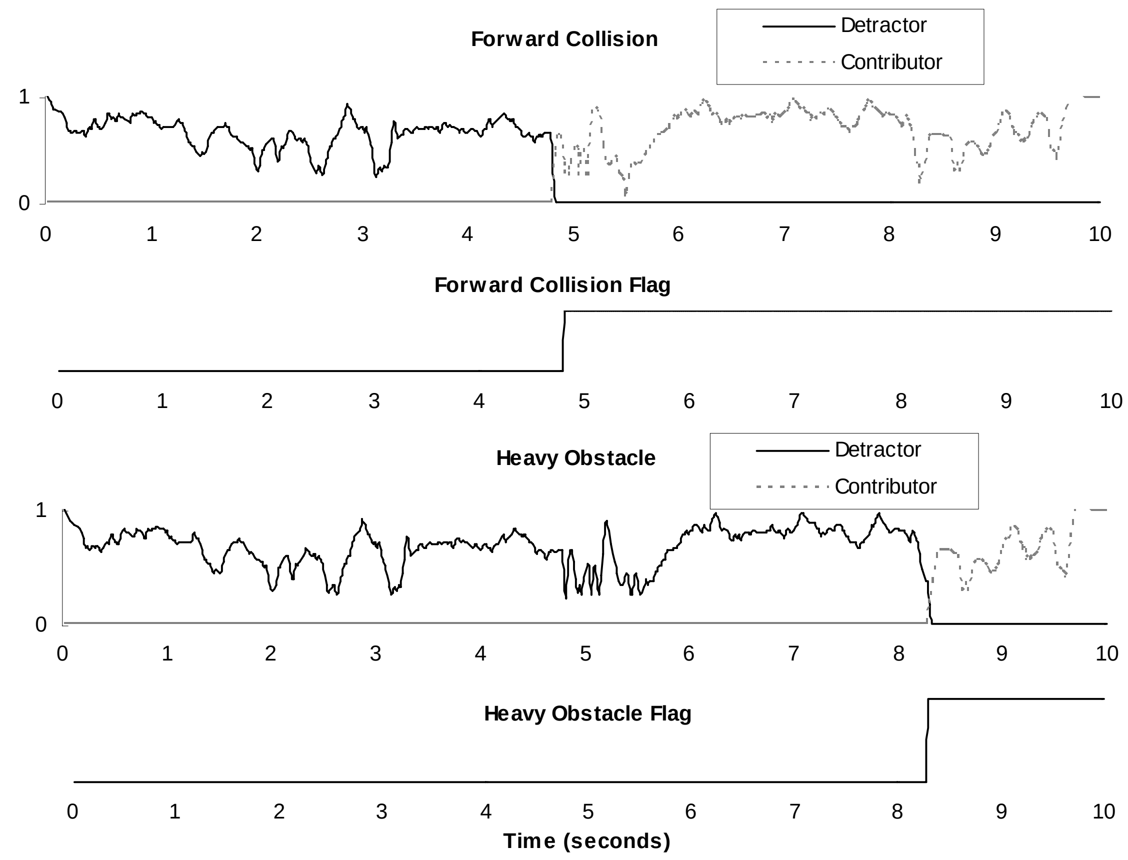


Fig. 9. Combination rules and flags during manipulation experiment

The collision detection capability of the FSFN is inherently fault tolerant to sensor failure. The method used in combining inference and composition rules allows for the rules base to automatically reduce itself to a subset that is not dependent on the faulted sensors.

4.4.1. Force Sensor Failure

If a force sensor fails, the most likely failure mode is a sensor reading of zero. The zero force reading causes the medium and high force membership functions to equal zero $\left(DOM_{32}=0,DOM_{33}=0\right)$ and the low force membership function to equal one $\left(DOM_{31}=1\right)$. The fuzzy AND operator in equation 16 causes

$IR_{jk1}=DOM_{1j}\otimes DOM_{2k}\otimes DOM_{31}=DOM_{1j}\otimes DOM_{2k}$,

$IR_{jk2}=DOM_{1j}\otimes DOM_{2k}\otimes DOM_{32}=0$, and

$IR_{jk3}=DOM_{1j}\otimes DOM_{2k}\otimes DOM_{33}=0$.

Applying these results to the Collision Detractor set, equation 17 reduces to

$$\left\{IR_{22},IR_{23},IR_{33},IR_{32}\right\}, \tag{21}$$

and the Collision Contributor rule (equation 18) becomes

$$\left\{IR_{11}\right\}. \tag{22}$$

Because medium and high force membership functions are involved in all of the Heavy Obstacle Contributor inference rules in equation 19, faulting the force sensor reduces the contributor rules to a null set. The Heavy Obstacle Detractor rule set reduces equation 20 to

$$\begin{Bmatrix}IR_{11},IR_{12},IR_{13},IR_{21},IR_{22},\\ IR_{23},IR_{31},IR_{32},IR_{33}\end{Bmatrix}, \tag{23}$$

which contains all combinations of acceleration DOMs.

Since the acceleration membership functions overlap, the value of the Detractor Rule will always be greater than zero, and a comparison between the zero Contributor Rule and the non-zero Detractor Rule will always result in the Heavy Obstacle Detected flag being zero. With the force sensor faulted, the augmented FSFN is equivalent to the FSFN without the force sensor. Since the FSFN without the force sensor could adequately detect collisions, and therefore initiate transitions into the avoidance behavior, but could not detect heavy obstacles and therefore initiate transitions into the push behavior, the behavior with the faulted force sensor degrades to the behavior of the system that did not possess the force sensor.

Fault tolerance was verified experimentally by disconnecting the force sensor and repeating the manipulation experiment (see section 4.3). Collisions were detected consistently, but heavy obstacles identification

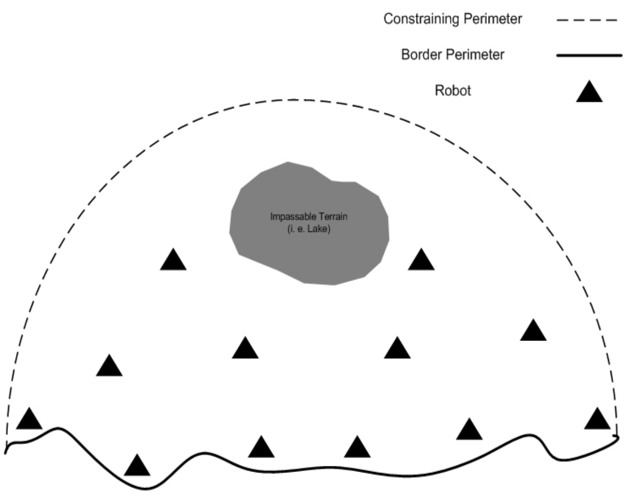


Fig. 10. Definition of Perimeters

failed. A similar result occurs for each acceleration measurement. The manipulation experiment was repeated with the acceleration sensors faulted. Collisions were detected consistently, and heavy obstacles identification was successful.

5. LMSL APPLIED TO BORDER SECURITY

There are many applications that can be used to drive the development and validation of a robot controller architecture, such as Search and Rescue, Mine Sweeping [21], and Planetary Exploration [22][23]. With the rising tide of illegal immigration overlaid with terrorism threats at home, the task of border security is becoming increasingly important. Fixed walls or fixed security monitors can be undermined, broken down, or evaded. Manned patrols are costly and impractical. Technological advances in the field of robotics may provide a flexible sensor array which can enhance monitoring of the borders.

Table 2. Modes for border security

Behavior	Activity	Goal
Pursue	More Flex	Border Security
Capture	Balance	
Patrol	Less Flex	

A flexible sensor array refers to a team of mobile robots carrying a sensor package, such as a thermal imager, which can reliably detect a human signature. This array is spread across a perimeter, and each robot attempts to maintain a fixed distance with its neighboring robots, such that a group of humans passing in the vicinity of the robots will be sensed by the sensor package (see Fig. 10).

As the signature moves towards the robots, the robots are designed to move away from the border and to maintain contact with the signature. Combined with the array's rule of maintaining a fixed distance with its neighbors, when one robot moves out of position, the section of the sensor array will deform towards the interior. This represents a detectable pattern, which either a human border agent with a heads up display can monitor or which algorithms can infer from robot positions and states. The differences between an animal crossing the border versus a mass of humans crossing the border can be distinguished.

Table 3. Transition Flags for border security

Flags	Layer	Source
Perimeter_Detected, Intruder_Detected, Intruder_Escaped	Activity	FSFN
Long_Pursuit, Short_Pursuit	Activity	Timer
Low_Density, Medium_Density, High_Density	Goal	Mode Monitor

The implementation of border security using LMSL will require Actions, Behaviors, Activities, and Goals to be developed. The library of Activity layer modes and Behavior layer modes for the general case (see Table 2) are not yet fully developed. The Behaviors will be described in terms of their desired outcomes.

FSFN's that are capable of detecting boundaries, robots, and intruders are required. The flags which will be needed to implement the proposed border security scheme are presented in Table 3.

The system will recognize two types of perimeters. The perimeter that the team is assigned to guard is called the Border Perimeter. A second perimeter, which serves as a limit to keep the robots from dispersing too far is called the Constraining Perimeter. The intersection of the Border perimeter and the Constraining perimeter bounds a closed domain (see Fig. 10).

5.1. Behavior layer modes

In the Patrol behavior, robots are directed to maintain a centroid of distance between themselves and neighboring robots. The robots are attracted to the Border perimeter and may not cross the Constraining perimeter. The combination of these two rules results in the robots settling into a pattern along the Border perimeter with a more or

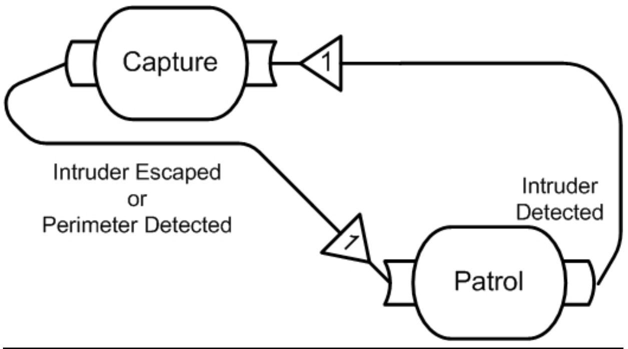


Fig. 11. Less Flex Activity MSD

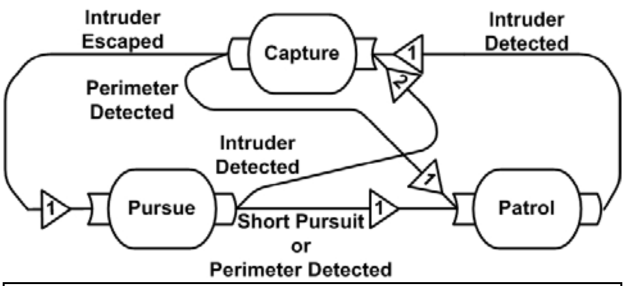


Fig. 12. Balance Activity MSD

less fixed spacing among themselves, depending on the terrain, number of robots, and length of the Border perimeter (see Fig. 10).

If a thermal or visual signature representative of a possible human is detected, the Capture behavior directs the robot to maintain a fixed distance from an intruder. While in Capture behavior, the presence of other robots is ignored and the Border perimeter attraction is ignored; however, the Constraining Perimeter may not be crossed.

If a robot loses the signature from the intruder, it may enter the Pursue behavior. This behavior directs the robot to move in the direction of the last known heading of the intruder. Other robots are ignored. The Border Perimeter attraction is ignored, and the Constraining Perimeter remains in effect.

5.2. Transition flags

The transition flags for the Activity layer are Intruder_Detected, Intruder_Escaped, Perimeter_Detected, Short_Pursuit, and Long_Pursuit (see Table 3). Intruder_Detected indicates that a potential human has been encountered. The Intruder_Detected flag may be generated by a FSFN using an array of complimentary sensors such as vision and thermal imager.

The Intruder_Escaped flag is triggered when the robot loses sight of the intruder. This flag is only triggered after an intruder has been detected. Perimeter_Detected indicates that the robot is nearing a Perimeter. At the Activity Layer no distinction is made between Border and Constraining perimeters. The Short_Pusuit and Long_Pursuit flags are set by timers, which start when the robot enters the Pursuit mode.

5.3. Activity layer modes

In the Less_Flex activity, the robot favors the Patrol behavior (see Fig. 11). The initial state of the robot is the Patrol behavior. If an intruder is detected, the robot enters the Capture behavior. The robot returns to Patrol behavior when either the intruder escapes or the robot encounters the Border perimeter or the Constraining perimeter. In Less_Flex, the robot does not pursue an intruder, which minimizes the robot's displacement from the border.

In the Balance activity (see Fig. 12), the robot gives equal treatment to Pursue and Patrol. The robot begins in Patrol behavior. When an intruder is detected, the robot enters the Capture behavior. If the Border perimeter or the Constraining perimeter is encountered while the robot has the intruder captured, the robot will not cross either perimeter, but will return to Patrol behavior. If the intruder escapes and no perimeter is detected, the robot will Pursue. While in Pursue behavior, if an intruder is detected, the robot will return to Capture. While in Pursue behavior, if a perimeter is encountered or the short pursuit timer expires, then the robot will return to Patrol.

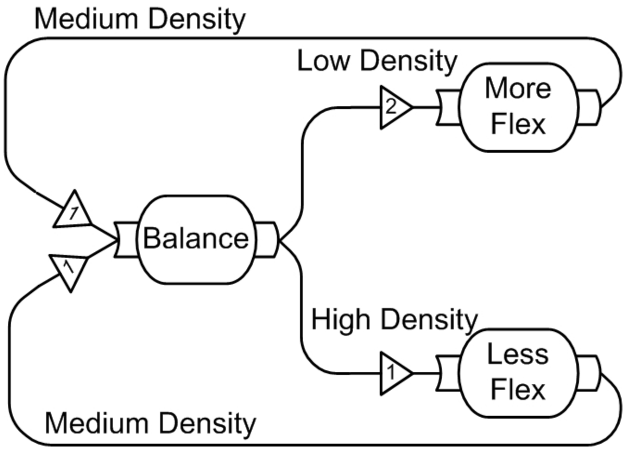


Fig. 13. Border Security Goal MSD

In the More Flex activity the robot prioritizes the Pursue behavior (see Fig. 14). The MSD for More Flex is similar to the Balance mode MSD. Instead of the short pursuit timer, a long pursuit timer is used. This gives the robot more time to reacquire its target before returning to Patrol.

5.4. Goal Layer

The Goal Layer, Border Security, is operating on each robot; however, the inputs to generate transition flags are initiated by a combination of intruder detection from the team of robots. The Border Security goal (see Fig. 13) transitions among the Activities, "Balance," "More Flex," and "Less Flex."

The purpose of the Border Security goal is to keep the robot formation dispersed without losing the integrity of the patrolled border. This dispersion allows a minimum number of robots to adequately monitor an area. It will vary with the intruder density. When more intruders are

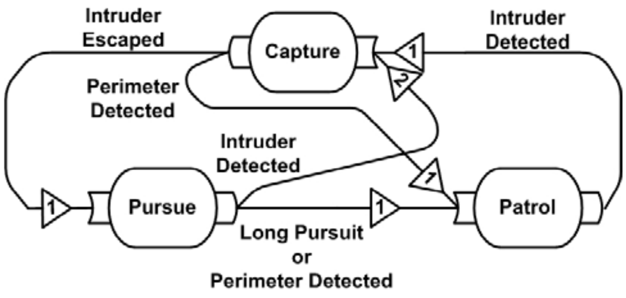


Fig. 14. More Flex MSD

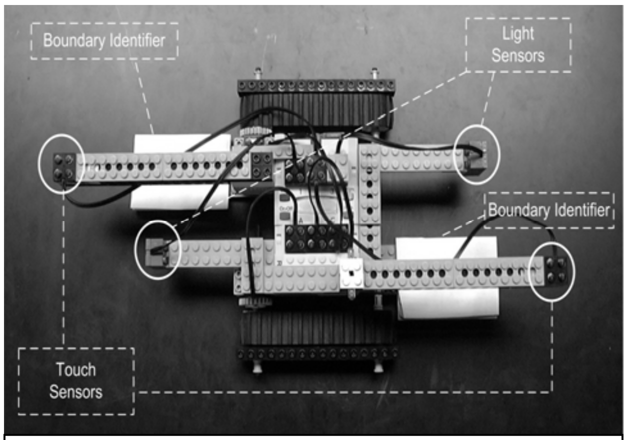


Fig. 15. Lego border security robot

present, the robots guarding that section must be more concentrated to track them. When fewer intruders are present, the formation can be more dispersed.

The Low_Density, Medium_Density, High_Density intermediate flags used in the Border Security goal are determined by a Mode Monitor. Each robot monitors the Behavior layer state of its near neighbors. The more near neighbors who are in the Capture or Pursue behaviors, the more intruders are present in the robot's vicinity. This value is a continuous variable that is fed into a FSFN to set the "High_Density" and "Low_Density" intermediate flags. More robots in Patrol behavior would indicate that few intruders are being detected and therefore this section of border has a low intruder density. This value is fed into a FSFN to determine the "Low_Density" intermediate flag.

Initially all robots are in the Balance activity. When the "Low_Density" intermediate flag is set, the Activity is switched to "More Flex" which allows the robots to disperse over a wider territory. This activity should be the normal Activity in which the flexible sensor array remains when no intruders are present.

If the robots begin to detect intruders, the "Medium Density" flag will be set. If the number of intruders is large, both the "Medium Density" and "High Density" flags will be set. Once the "Medium Density" flag is set, the Activity switches to Balance, which decreases the amount of time in which the robot will remain in the Pursue behavior upon losing contact with an intruder. The likelihood that robots will remain in position increases, as each robot passes off an intruder to its near neighbors. The overall sensor array flexes in the direction of the intruders due to the Capture and Pursue behaviors of the robots actively engaged with intruders. The neighbors of these robots also adjust their positions, since the Patrol behavior requires them to maintain a centroidal distance with their near neighbors.

In the event of mass intrusion, the "High Density" flag will be set, which will transition robots into the "Less Flex" activity. In "Less Flex," robots may Patrol or Capture, but may not Pursue. This tightens the sensor grid in the vicinity of a mass crossing.

As the intruder density decreases (either because the intruders return across the Border perimeter or because the

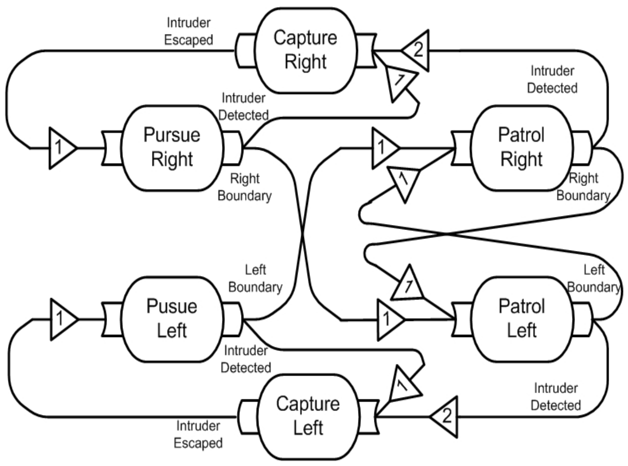


Fig. 16. One Dimensional border security MSD

have crossed the Constraining perimeter), the "Low Density" flag will be set, and robots return to the "More Flex" activity. The attraction of the Border perimeter will eventually return the flexible sensor array to its original configuration along the border.

Because the robots are sharing information about intruder contacts, at the point of many contacts, the robots will have the "High Density" flag set. Away from the contacts, robots who are not engaged with intruders will still have the "Medium Density" flag set.

The current prototype implementation of the Border Security goal is discretized into three Activities. If this proves to be too coarse a discretization, additional Activities (for instance "Even More Flex" and "Ludicrous Flex") can be added to improve performance.

5.5. Behavior Prototype Experiment

A prototype border security behavior has been developed and tested. The general border security problem was simplified for the prototype system. This prototype behavior exhibited emergent behaviors which support its ability to maintain a secure perimeter.

The border region was simplified by reducing it to a single dimension. A one dimensional border allows for straightforward control of the orientation of the robots during interactions. The types of interactions are reduced by combining robot and perimeter detections into a single type called a *boundary detection*. All other interactions are considered intruders. These interactions can be detected by using a simple set of sensors.

The prototype behavior was tested using Lego robots (see Fig. 15). The robots use a pair of light sensors on either side to detect boundaries. A boundary is designated by a white boundary identifier. Each robot has a boundary identifier on both sides such that when it interfaces with the other robots they detect each other as boundaries. Similar boundary identifiers are located at the limits of the perimeter.

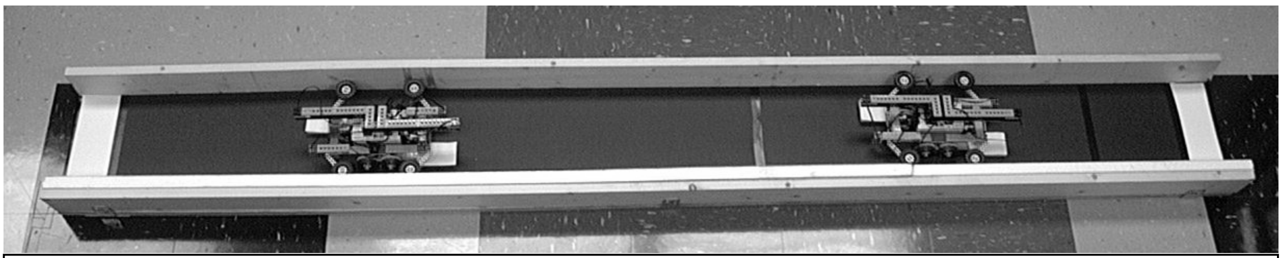


Fig. 17. Border Security Robots in One Dimensional Border

The robots use a pair of touch sensors to detect intruders. The touch sensors are oriented such that when the robots interact, the light sensor will allow them to identify each other before the touch sensor is activated. Any non-boundary detected is considered an intruder. A block of wood served as an intruder in this experiment.

The prototype MSD (see Fig. 16) is an implementation of the Balance activity (see Fig. 12). The general behaviors (Patrol, Pursue, and Capture) are made specific by adding left and right directions to them. The boundary detected transition flags are also made specific by the addition of a direction (Left Boundary, Right Boundary). The Long Pursuit and Short Pursuit intermediate flags from the general case were omitted for simplification.

Each robot begins one of the two Patrol modes. While in Patrol, the robot travels in the specified direction (left or right) until one of two events occur:

1. A light sensor detects a boundary (another robot or a perimeter). A transition to the complementary patrol mode is initiated, causing the robot to move away from the boundary. In the absence of intruders the robot will cycle between Patrol Left and Patrol Right modes.

2. A touch sensor indicates the presence of an intruder. A transition into the Capture mode corresponding to the left or right touch sensor is initiated. While in Capture mode, the robot maintains a fixed distance to the intruder. The distance is limited by the range of the sensor. Since a touch sensor is used in this experiment, the range is zero, and the robot will stay in contact with the intruder.

When the touch sensor is released, the Intruder Escaped flag is set, causing the robot to transition into the corresponding Pursue mode. It pursues in the same direction it had been patrolling when it performed the capture. The robot will remain in Pursue until the intruder is recaptured or a boundary is encountered.

Two robots running the prototype behavior were placed in a 2.25 meter track (see Fig. 17). The position of the center of each robot was measured at 0.5 second intervals using a video capture software called LoggerPro. A single stationary intruder was inserted at various positions on the track. Three distinct emergent behaviors were observed:

1. When no intruders were present, the robots patrolled the entire perimeter (see Fig. 18). The emergent behavior shows that the formation will uniformly distribute itself within the bounded region. In Fig. 18, each robot oscillates towards the center position. When it encounters the other robot, it changes direction until it encounters the boundary.

2. When an intruder was placed between the robots, the robots captured the intruder from both sides (see Fig. 19). The emergent behavior shows that the robots will converge on an intruder, but leave the border un-patrolled. The spacing between the robots at 10 seconds is the width of the intruder plus one robot width.

3. When an intruder was placed on the outside of either robot, the robot nearest the intruder captured it while the neighboring robot patrolled the area that the capturing robot no longer visited (see Fig. 20). This emergent behavior shows that the formation will adapt to the presence of intruders.

When no intruders are present, every position along the patrol border should be visited an equal amount of time by at least one robot. When an intruder is captured, one or

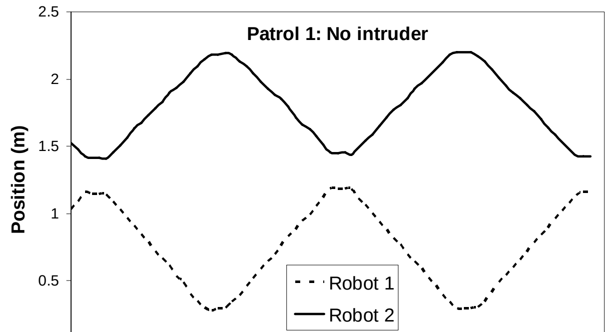


Fig. 18. No intruder; patrol behavior.

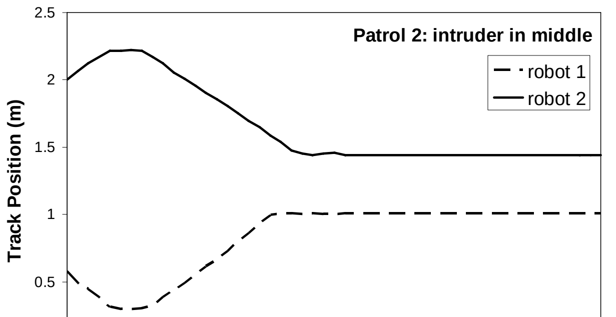


Fig. 19. Intruder in middle; capture behavior.

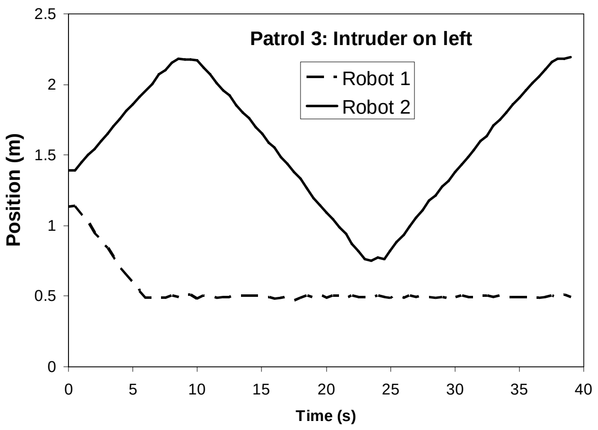


Fig. 20. Intruder on left; capture behavior.

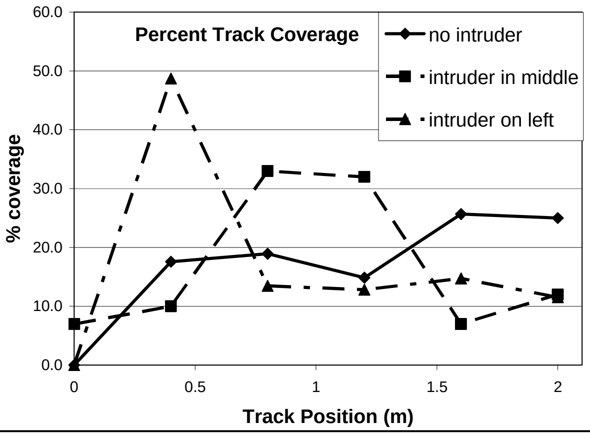


Fig. 21. Percentage of time that a track location is covered.

both robots will stay at the intruder's location until given further instructions. As a result, the rest of the patrol area may be covered less frequently or not at all.

The percentage of time each location of the track is covered by a robot during the trials graphed in Figures 18-20 is shown in Fig. 21. The track was split into six bins, each 40 cm wide. The bin width was selected to be slightly larger than the width of one robot. For each track position bin, the amount of time that the bin was occupied by a robot was recorded and a percentage of total time was calculated. Given a sufficiently long running time, this graph will have a flat line if no intruders are present, or a peak at the location of an intruder. This graph can be used to set a threshold value, based only on the position of the robot, to determine whether an intruder is present and what its location is. If the percentage peaks above that threshold value, the human controller can be alerted that an intruder is present at a specified location. The controller may then elect to send out new instructions. This information may be used at the Goal Layer to adjust the number of robots in the vicinity of the intruder to adjust their Activity setting.

CONCLUSIONS

The LMSL architecture has been implemented and tested on UALR's J5 robot. It has been verified that certain functional requirements were satisfied. The ability to incorporate new sensors into an existing infrastructure was verified by adding the force sensor into the Obstacle_ID FSFN after a preliminary FSFN was built using only acceleration. The ability to incorporate new modes was verified by adding the pushing mode to the obstacle avoidance behavior to create the obstacle manipulation behavior. The fault tolerance of the FSFN to sensor failure has been verified experimentally for both acceleration and force sensors.

An application to Border Security was phrased in the LMSL paradigm. The Behavior, Activity, and Goal Mode Selection Diagrams have been developed. A one-dimensional prototype was implemented in hardware, and emergent behaviors consistent with the Border Security design have been observed.

FUTURE WORK

The formalism of the Layered Mode Selection Logic and it application to cooperative mobile robotics problems is in an early stage of development. The CAMRC has defined many of the routines for implementation, but is still developing the functional requirements and the algorithms to respond to those requirements. CAMRC has begun work on a Matlab based Graphical User Interface for writing the tables used in the layers in the MSL. It is developing object classification algorithms from sensor inputs to determine appropriate transition flags. It is developing a library of debugged Actions, Behaviors, Activities, and Goals to be used in the GUI to enhance the design effort.

The border security is being transitioned to a two dimensional verification of the algorithms, using a small robot based on the Vex microcontroller.

Although the LMSL should provide tolerance to actuator failures, this feature has not yet been tested, since current robots used in this work do not have redundant features. The movable wedge on the J5 robot platform will allow us to explore traversability of some terrain. When this feature has been activated, the concept of actuator failure will be explored.

Additional quantification of fault tolerance to sensor failures is being developed by adding ultrasonic sensors to augment the touch sensor with a non-contact collision detection. This sensor provides a much more dramatic difference between sensor types and will make for a more thorough test of this feature.

ACKNOWLEDGEMENT

This work was supported by Arkansas Space Grant Consortium, NASA grant number NCC5-260, and University of Arkansas at Little Rock College of Engineering and Information Technology.

The authors acknowledge Armand Tomany for the mechanical design and fabrication of the original J5 robot. Mr. Tomany performed this design work as part of his

Mechanical Engineering Technology senior design project in conjunction with the 2001 FIRST Robotics Competition.

REFERENCES

[1] Mataric, M. "Behavior-Based Control: Examples from Navigation, Learning, and Group Behavior", *Journal of Experimental & Theoretical Artificial Intelligence*, Vol. 9, 1997, pp. 323-336.

[2] Zielinski, C. "Reactive Robot Control Applied to Acquiring Moving Objects", *Proceedings of The 3rd International Symposium on Methods and Models in Automation and Robotics*, Vol. 3, 1996, pp. 893–898.

[3] Bonasso, R. P., Firby, R. J., Gat, E., Kortenkamp, D., Miller, D. P., Slack, M. G., "Experiences with an Architecture for Intelligent, Reactive Agents"**,** *Journal of Experimental & Theoretical Artificial Intelligence*, 1997.

[4] Brill, F. Z., Wasson, G. S., Ferrer, G. J., and Martin, W. N. "The Effective Field of View Paradigm: Adding Representation to a Reactive System", *Engineering Applications of Artificial Intelligence,* Vol. 11, 1998, pp. 189-201.

[5] Driankov, D. and Saffiotti, A. (Eds), Fuzzy Logic Techniques for Autonomous Vehicle Navigation, Springer-Verlag, 2001.

[6] Arkin, R. C., Behavior-based Robotics, The MIT Press, Cambridge, MA, 1998.

[7] Pirjanian, P. "Behavior Coordination Mechanisms State of the Art", *Institute for Robotics and Intelligent Systems University of Southern California Tech Report*, IRIS-99-375, 1999.

[8] Arkin, R. C., MacKenzie, D., "Temporal Coordination of Perceptual Algorithms for Mobile Robot Navigation," IEEE Transactions on Robotics and Automation, Vol. 10, No. 3, 1994, pp. 276-286.

[9] Brooks, R., "A robust layered control system for a mobile robot," IEEE Trans. on Robotics and Automation, v. 2, n. 1, 1986, pp. 14-23.

[10] Suh, N. The Principles of Design, Oxford University Press, 1990.

[11] Pirjanian, P., Huntsberger, T. L., Trebi-Ollennu, A., Aghazarian, H., Das, H., Joshi, S. S., Schenker, P. S., CAMPOUT: A Control Architecture3 for Multi-robot Planetary Outposts," Proceedings of SPIE 2000, Sensor Fusion and Decentralized Control in Robotic Systems III, v. 4196, pp. 221-230.

[12] Wright, A., Teague, W., Wright, A., Wilson, E. "Instrumentation of UALR Labscale Hybrid Rocket Motor", *Proc. of SPIE Sensors for Propulsion Meas. Appl.*, Vol. 6222, 2006, No. 622202, pp. 1-12.

[13] Ericksen, J., Godere, E., Wright, A., "Digital Controller Improves Power and Flexibility of Gas Turbine Driven M1A1 Tank," IGTI Publication 91-GT-295, 1991.

[14] Thrun, S., "Bayesian landmark learning for mobile robot localization," Mach. Learning, vol. 33, no. 1, 1998, pp. 41-76.

[15] Vandapel, N., Huber, D. E., Kapuria, A., and Hebert, M., "Natural Terrain Classification using 3-D Ladar Data," Proceedings of the 2004 IEEE International Conference on Robotics and Automation, New Orleans, LA April 2004.

[16] Jagielska, I., Matthews, C., Whitfort, T., "An investigation into the application of neural networks, fuzzy logic, genetic algorithms, and rough sets to automated knowledge acquisition for classification problems," Neurocomputing, Vol. 24, 1999, pp. 37-54.

[17] Passino, K. M., Yurkovich, S., Fuzzy Control, Addison Wesley Longman, 1998.

[18] Berenji, H. R., Khedkar, P., "Learning and Tuning Fuzzy Logic Controllers through Reinforcements," *IEEE Trans. On Neural Networks*, Vol. 3, No. 5, 1992, pp. 724-740**.**

[19] Lin, C. T. and Lee, C. S. G., "Reinforcement structure/parameter learning for neural-network-based fuzzy logic control systems," *IEEE Trans. Fuzzy Systems*, Vol. 2, No. 1, 1994, pp. 46-63.

[20] Ye, C., Yung, N. H. C., and Wang, D. W., "A fuzzy controller with supervised learning assisted reinforcement learning algorithm for obstacle avoidance," *IEEE Transactions on Systems, Man and Cybernetics-Part B: Cybernetics*, Vol. 33, No. 1, 2003, pp. 17-27.

[21] Ulam, P., Endo, Y., Wagner, A., and Arkin, R. "Integrated mission specification and task allocation for robot teams-testing and evaluation," Georgia Inst. Tech., Tech Rep. GIT-GVU-07_02, 2007.

[22] Howard, A., Seraji, H., Tunstel, E., "A Rules-Based Fuzzy Traversability Index for Mobile Robots," Proc. 2001 IEEE Intl. Conference on Robotics and Automation, 2001, Seoul, Korea.

[23] Huntsberger, T., Pirjanian, P., Trebi-Ollennu, A., Nayar, H. D., Aghazarian, H., Ganino, A. J., Garrett, M., Joshi, S. S., Schenker, P. S., "CAMPOUT: A Control Architecture for Tightly Coupled Coordination of Multirobot Systems for Planetary Surface Exploration," IEEE Transaction on Systems Man and Cybernetics Part A Systems and Humans, 2003, Vol. 33, pp. 550-559.

Andrew Wright received his BS in Mechanical Engineering from the University of South Carolina in 1982, his MS in Mechanical Engineering from the Massachusetts Institute of Technology in 1988, and his PhD in Mechanical Engineering from Rensselaer Polytechnic Institute in 1996. He is an Associate Professor in the Department of Applied Science at the University of Arkansas at Little Rock. From 1988-1992 he worked at Textron Lycoming, developing control algorithms for gas turbine engines. Dr. Wright mentored FIRST Team Number 356 from its inception in 2000 until the present, and has designed many robots and robotic mechanisms in conjunction with this endeavor. Dr. Wright is a member of ASME, the Acoustical Society of America, Tau Beta Pi, and Phi Beta Kappa.

Traig Born received his BS in Systems Engineering at the University of Arkansas at Little Rock in 2002 and his MS in Applied Science in 2008. He was a mentor on FIRST Team 356 from 2000 to 2006. He is employed at Hendrix College as the Lab Manager in the department of Physics. His research interests include robot design and control systems.

Gabriel J. Ferrer is an Associate Professor of Computer Science at Hendrix College. He received his BA in Computer Science from Rice University in 1994, and his MS and PhD in Computer Science from the University of Virginia in 1996 and 2002 respectively. He is developing automated software verification algorithms, mobile robot planning algorithms, and mobile robot software architectures.

Ann Wright is an Associate Professor and Chair of the Department of Physics at Hendrix College in Conway, AR. She received her BS from the Massachusetts Institute of Technology in 1991, and her PhD in Physics from Rensselaer Polytechnic Institute in 1996. Her research includes measuring combustion properties in a hybrid rocket. She mentored FIRST Team 356 since 2000.