Layered Mode Selection Logic Control with Fuzzy Sensor Fusion Network

Traig Born[a], Andrew Wright[b]
[a]Hendrix College, Department of Physics,
1600 Washington Avenue, Conway, AR 72032
[b]University of Arkansas at Little Rock, Department of Applied Science,
2801 South University Avenue, Little Rock, AR 72204

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. The Central Arkansas Robotics Consortium has developed a robot controller architecture, called Layered Mode Selection Logic (LMSL), which addresses all of these concerns. The LMSL architecture is an implementation of a behavior based controller fused with a planner. The architecture creates an abstraction layer for the robot sensors through a Fuzzy Sensor Fusion Network (FSFN), and it creates an abstraction layer for the robot manipulators through a reactive layer. The LMSL architecture has been implemented and tested on UALR's J5 robotics research platform. A FSFN combines acceleration and force signals for collision detection. The output of the FSFN switches among low level behaviors to accomplish obstacle avoidance and obstacle manipulation. 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).

Keywords: Mode Selection Logic, 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. Such robots must be scalable, fault tolerant, and capable of being upgraded without requiring substantial redesign. 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 can be seen as a spectrum with planners at one end and reactive systems at the other. On the reactive end of the spectrum response time is fast and hardware is cheap; however complex tasks are difficult to achieve. On the planner end of the spectrum design is more complex and computation is more intensive, 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 is assumed to be valid.

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 also have scalability issues when applied to multi-agent systems. The complexity of a planner increases exponentially with additional agents.

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. The behavior-based controller architecture, a specific example of such a hybrid, uses “basis behaviors” as building blocks to achieve more complex functionality. 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 current methods used to combine behaviors, which he calls “coordination mechanisms” [3]. Coordination mechanisms are divided into two categories: arbitration methods and command fusion methods.

Arbitration methods select one behavior and give it complete control. Arbitration methods can produce locally optimal decisions while at the same time creating non-optimal global results. For example, a robot may be programmed to follow a beacon. While following the beacon an obstacle is encountered. The obstacle avoidance behavior is given control, and it decides 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. Using the previous example; the obstacle avoidance behavior gives a right turn command, while the follow beacon behavior is issuing a left turn command. The combined command would be go straight, resulting in a collision.

1.1 Functional Requirements

Designing a robot that is capable of performing useful tasks in an unstructured environment is facilitated by identifying the necessary functional requirements[4]. The functional requirements must be generalized to allow for good performance in any number of tasks or environments. Layered Mode Selection Logic (LMSL) is a hybrid controller architecture designed to satisfy these functional requirements. The functional requirements are satisfied by separate design elements within the architecture [see Table 1].

Table 1. Functional requirements for generic robot control architecture

Functional Requirements	**Design Element**
Scalable to complexity of robot Tolerant of manipulator failure Allow for integration of new modes	Mode Selection Logic
Allow for integration of new manipulators	Action layer
Scalable to number of robots	Behavior layer
Scalable to complexity of Task	Layered Mode Selection Logic
Allow for integration of new sensors Tolerant of sensor failure Tolerant of sensor uncertainty	Fuzzy Sensor Fusion Network

The term scalability has many connotations in the field of robotics, all of which are beneficial in unstructured environments. Scalability to number of robots refers to an architecture’s ability to increase the number of robots being controlled without becoming exponentially more complex. Being scalable to number of robots is beneficial because many of the tasks performed in unstructured environments can be done more effectively by teams of robots.

A controller architecture formalizes how tasks are specified. 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.

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.

The robot must maintain functionality in the presence of sensor and actuator failures. A minimal set of sensors and manipulators are necessary for basic functionality. Nonessential sensors and actuators are added to allow for enhanced capabilities. The robot should be able to cope with nonessential component failure. Redundant sensors can be used for critical functions. The architecture should accommodate redundancy.

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 it must be tolerant of sensor noise and imprecision.

The robot must allow for expandability. Advances in sensor and actuator technology are frequent. 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 behaviors, which make use of the new hardware, should not disrupt existing robot behaviors.

2. METHODOLOGY

Sensor data is interpreted by a Fuzzy Sensor Fusion Network (FSFN). The FSFN generates Intermediate flags. The Intermediate flags are used by Mode Selection Logic (MSL) to determine which mode is active. The active mode determines the behavior of the robot [see Figure 1].

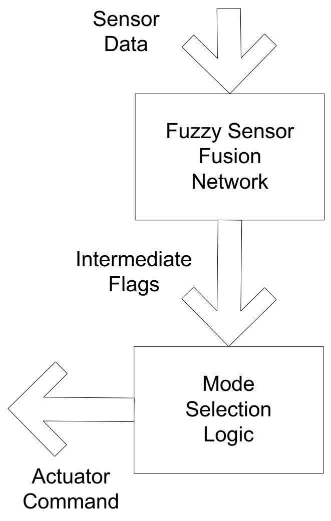

Figure 1. LMSL block diagram

2.1 Layered Modes Selection Logic

The behavior coordination mechanism used in this architecture is Mode Selection Logic (MSL). The implementation details of the general MSL mechanism are presented in reference 5. It is an arbitration mechanism because a single mode has complete control over the system's actuators. MSL is similar to a Finite State Machine in that a system has a finite number of states. For each state, the system's behavior is defined. Each state has a predefined set of conditions which, when satisfied, trigger the transition to another state.

A mode represents a system state. In a given mode, rules are defined which govern the system. Modes are designed to be modular. They have a standard interface with each other and the sensors. A mode can be inserted or removed from the system without affecting other modes. Well designed modes confine inputs and outputs to a minimal set.

Each mode has a set of exit conditions. Sensors, timers, or counters can satisfy an exit condition. 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.

A Mode Switching Diagram (MSD) is used to show a set of modes and their transition flags [see Figure 2]. Mode 1 is initially the active mode (i.e. the entry mode). Mode 1 will remain active until

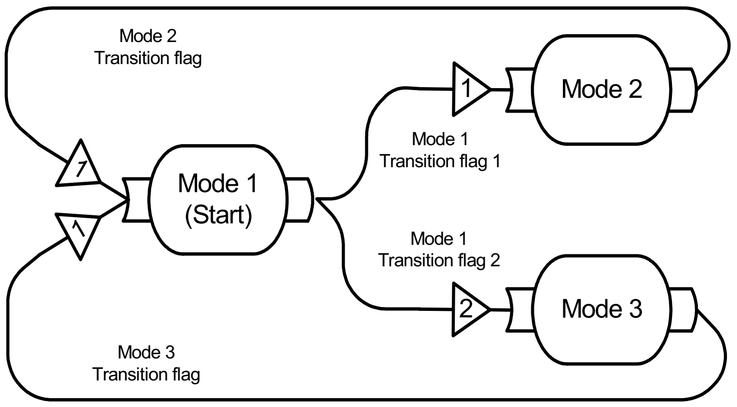

Figure 2. Generic MSD

either transition flag 1 or transition flag 2 is set. If transition flag 1 is set then mode 2 will be selected as the active mode. If transition flag 2 is set then mode 3 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.

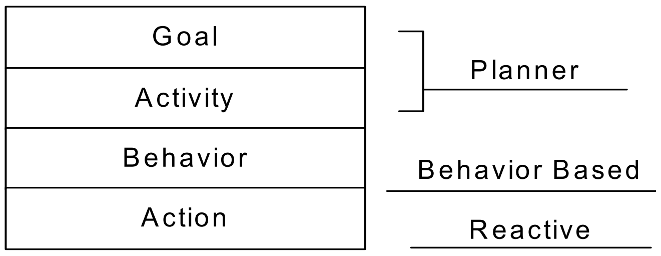

Figure 3. Layer Hierarchy

In Layered Mode Selection Logic (LMSL), modes are organized in a hierarchy. The lowest layer in the hierarchy is a reactive controller, the middle layers are behavior based, and the top layers provide planner capabilities [see Figure 3].

Each mode in the Action layer encapsulates a basic robot function, such as forward, turn left, or actuate manipulator. The Action layer modes provide hardware abstraction by using a function library to execute the low level details. This 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 easy integration of new hardware.

At the behavior layer, a behavior-based controller is implemented using the MSL paradigm. Each behavior consists of an action layer MSD [see Figure 4]. The layering of the architecture can be seen at the behavior level. As more complex tasks are required, layers are added. An activity mode consists of an MSD which is made up of behavior modes. Activities are capable of accomplishing more complex objectives than behaviors.

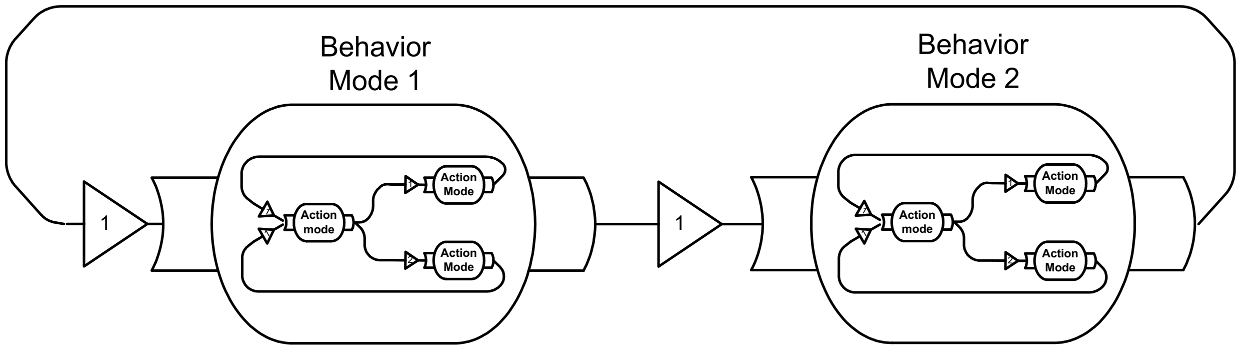

Figure 4. Behavior Layer MSD

Goals are the highest layer in the paradigm in its current implementation, although a higher level, called Personality layer, is envisioned. Transitions between goals are usually decided by human intervention. The goal layer of LMSL is well suited for controlling multi-agent robot teams. LMSL is scalable to multi-agent systems due to the behavior based component of the architecture. Unlike typical multi-agent behavior based systems, LMSL allows for top down group behavior design.

In this implementation of MSL a minimal set of modes is established. From this minimal set new modes can be added as new actuators and sensors are made available. A new mode can be integrated into existing MSD by defining rules that initiate transitions from the previously developed modes to newly defined modes. 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 MSL collapses back to the minimal set of modes, making the system fault tolerant to actuator and sensor failure.

LMSL satisfies many of the functional requirements. The action layer provides hardware abstraction, which satisfies the scalability to robot complexity functional requirement. The ability to layer modes to accomplish more complex objectives satisfies scalability to task complexity. The ability to add new modes into an existing MSD contributes to the system's modularity.

2.2 Fuzzy Sensor Fusion Network

In control systems, algorithms are used to generate command signals from sensor data. Algorithms are created from mathematical models of the system. The performance of the controller is dependent on the quality of the sensor data and the validity of the system model. Frequently in robotics, the system is complex and has too many unknown variables for a good model to be built. Fuzzy logic is well suited to these types of problems. The behavior of a fuzzy system is linguistically defined so a control system can be developed without a precise mathematical model.

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). 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 associated with a certain layer of the mode selection logic. The FSFN combines multiple sensors in a single fuzzy rules matrix. The sensors combined in an FSFN can be redundant or complimentary. In this architecture, fuzzy logic takes sensor values and maps them into 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.

3. APPLICATIONS

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. Obstacle avoidance assumes 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 this is not the case, such as search and rescue, hazardous material clean up, and battlefield reconnaissance. In these cases, it is preferable that the robot is capable of manipulating obstacles instead of avoiding them.

Obstacle manipulation enables a robot to remove an obstacle impeding its path and continue along an optimal route to the objective. This enhances the navigational capabilities of the robot by allowing it to choose a more direct route, and 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.

Obstacle avoidance was first implemented using LMSL and a FSFN on UALR's J5 robot, and manipulation was added after the avoidance MSD had been debugged. J5 has a mass of 59 kg and is .75 m wide by 1 m long by .45 m tall [see Figure 5]. It has a robust differential drive system powered by DC motors. The drive axles have optical encoders. The control system hardware consists of standard personal computer technology and custom-built circuits for motor control. It utilizes an AT motherboard and an x86 architecture processor. 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 [see Figure 5.] The wedge has a strain gauge bonded to it rear surface to provide force measurements.

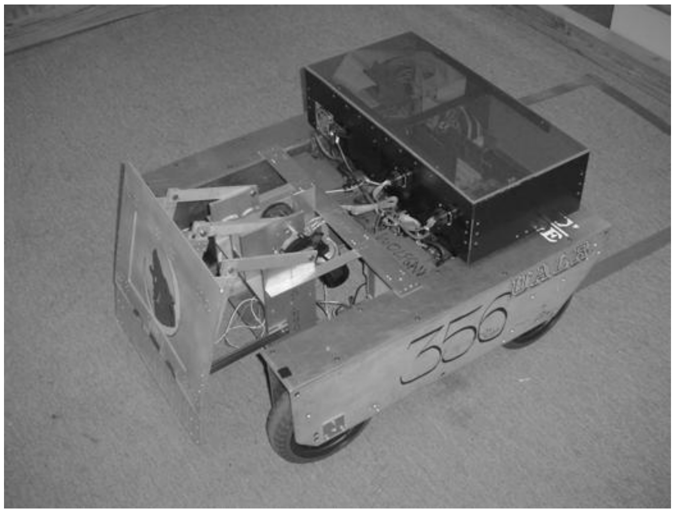
Figure 5. J5 Robot

3.1 Behavior design

In the obstacle avoidance behavior the robot enters the forward mode when initialization is complete [see Figure 6]. It continues to move forward until a forward 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.

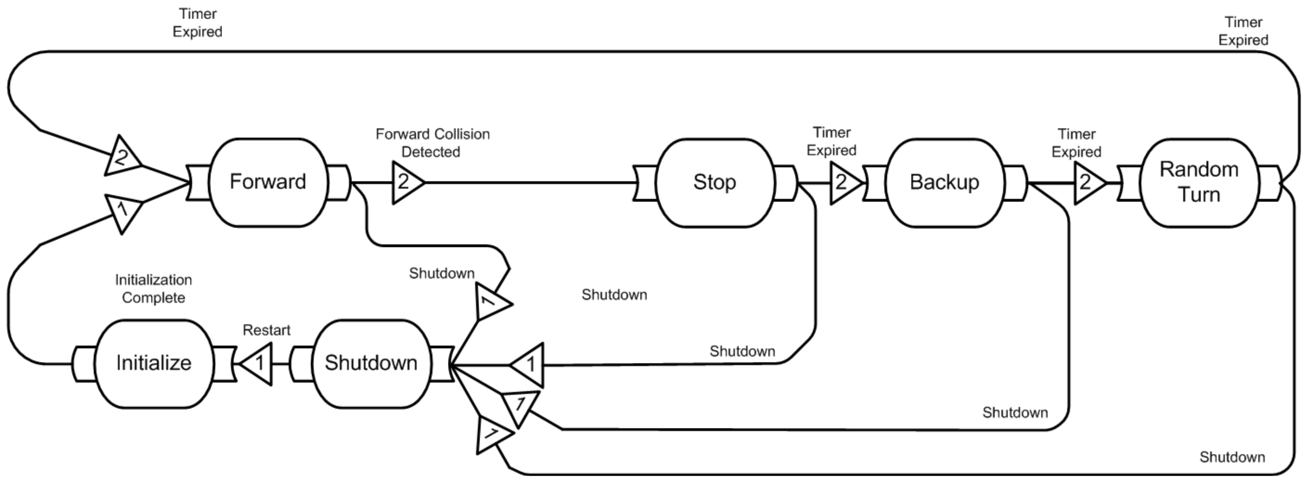

Figure 6. Obstacle avoidance MSD

To create the obstacle manipulation behavior, push mode and the immovable obstacle flag were added to the avoidance behavior [see Figure 7]. This demonstrates the ease with which new modes and improved functionality can be added.

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.

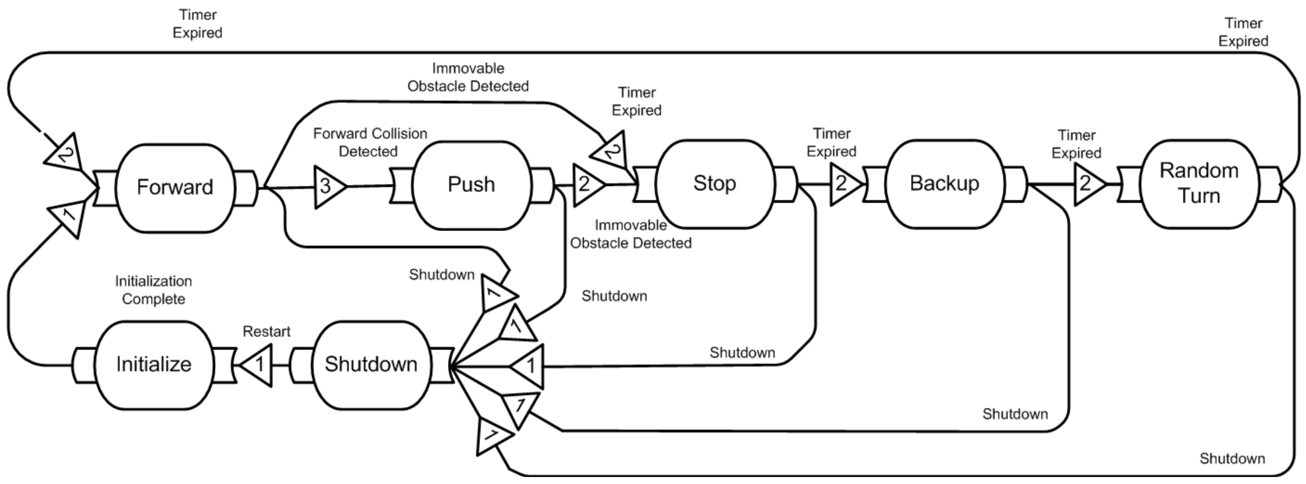

Figure 7. Obstacle manipulation MSD

3.2 Fuzzy sensor fusion network design

Obstacle manipulation requires a FSFN that is capable of detecting collisions, and identifying obstacles that can be manipulated. The Obstacle_ID FSFN uses on-board acceleration and force sensors to detect and quantify collisions.

Collision characterization experiments were conducted. The collision characterization experiments show that a negative spike in acceleration closely followed by an increase in contact force is an indication that a collision has occurred. The magnitude of the resulting force and acceleration spikes increase with the mass of the obstacle [see Figure 8]. The force vs. obstacle graph is linear. The graph of negative acceleration spikes vs. obstacle mass is monotonic, but not linear due to the wheels slipping when the obstacle is 60kg or greater.

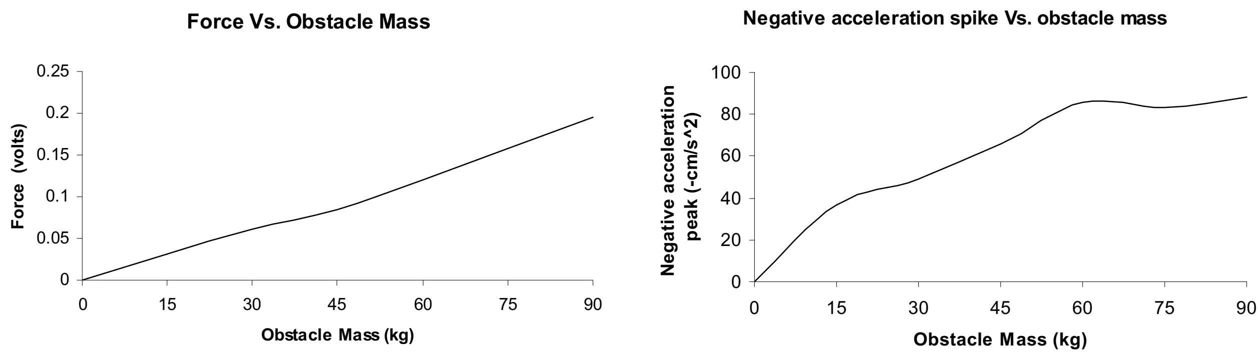

Figure 8. Obstacle Collision Experiments

Membership functions were created for the Obstacle_ID FSFN [see Figure 9] using data from the collision characterization experiments. Each sensor is associated with a fuzzy variable, which maps into three fuzzy sets. The fuzzy variables are left wheel acceleration, right wheel acceleration, and force. The fuzzy sets which acceleration maps into are negative, zero, and positive. The fuzzy sets for force are low, medium, high.

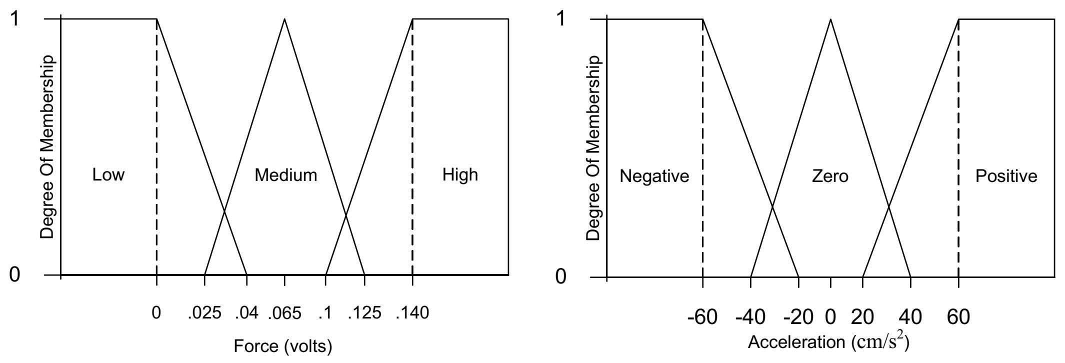

Figure 9. Obstacle_ID FSFN Membership Functions

The degree of membership for a fuzzy set is denoted as:

$$DOM(j_k) \tag{1}$$

Where $k \in \{left\_wheel\_acceleration, right\_wheel\_acceleration, contact\_force\}$ and

$$j_k \in \begin{cases} \{neg, zero, pos\} & k=1,2 \\ \{low, med, high\} & k=3 \end{cases} \tag{2}$$

The degree of memberships are combined in inference rules using the fuzzy AND operator. The fuzzy operator "AND" is $A \otimes B = \text{minimum(A,B)}$ and the combination of many fuzzy "ANDs" is denoted $\prod_{i=1}^{n} A_i = A_1 \otimes A_2 \ldots \otimes A_n$.

An inference rule is denoted as:

$$inference\_rule(j_1,...j_n)=\prod_{k=1}^{n} DOM(j_k) \tag{3}$$

A preliminary set of inference rules was created for the Obstacle_ID FSFN using only the left wheel acceleration and right wheel acceleration DOMs.

$$preliminary\_obstacle\_ID(j_1,j_2) \quad j_1\in\{1,2,3\}, j_2\in\{1,2,3\} \tag{4}$$

This preliminary_obstacle_ID 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 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. Also, the presence of a contact force verifies that any acceleration spike is due to a collision, not tracking error.

The addition of the force sensor to the FSFN required only that a new set of membership functions be created and the DOM set in eqn. 4 be augmented to include the new DOMs.

$$Obstacle\_ID(j_1,j_2,j_3) \quad j_1\in\{1,2,3\}, j_2\in\{1,2,3\}, j_3\in\{1,2,3\} \tag{5}$$

This FSFN identifies two events: “forward collision” and “heavy object detection”. Each event has two combination rules associated with it, a contributor rule and a detractor rule. The contributor rule contains inference rules which contribute to the occurrence of the event. The inference rules which detract from the occurrence of an event are combined in the detractor rule. Combination rules combine inference rules using the fuzzy “OR” operator.
The fuzzy operator “OR” is $A\oplus B$ = maximum(A,B) and the summation of many fuzzy “ORs” is denoted $\sum_{i=1}^{n} A_i = A_1\oplus A_2 ...\oplus A_n$.

The inference rules contained in each combination rule set were selected by analyzing the data from the collision experiments. Inference rules which gave inconsistent results were omitted from both rules. The inference rules which gave strong results during the collision were selected for the contributor rule.

$$\begin{aligned} forward\_collision\_contributor &= Obstacle\_ID(1,1,1)\oplus Obstacle\_ID(2,1,2)\oplus Obstacle\_ID(2,2,2)\\ &\oplus Obstacle\_ID(3,1,2)\oplus \sum_{j_2=1}^{3} Obstacle\_ID(1,j_2,2)\\ &\oplus \sum_{j_1=1}^{3}\sum_{j_2=1}^{3} Obstacle\_ID(j_1,j_2,3)\end{aligned} \tag{6}$$

Inference rules which gave strong values before the collision were selected for the detractor rules. (7)

$$\begin{aligned} forward\_collision\_detractor &= Obstacle\_ID(1,3,1)\oplus Obstacle\_ID(2,2,1)\oplus Obstacle\_ID(2,3,1)\\ &\oplus \sum_{j_2=1}^{3} Obstacle\_ID(3,j_2,1)\end{aligned}$$

The heavy obstacle contributor rule consists of all inference rules containing the force high DOM, i.e. j_3 =3.

$$heavy\_obstacle\_contributor = \sum_{j_1=1}^{3}\sum_{j_2=1}^{3} Obstacle\_ID(j_1,j_2,3) \tag{8}$$

The heavy obstacle detractor rule contains all rules with either force low or force medium.

$$heavy\_obstacle\_detractor = \sum_{j_1=1}^{3} \sum_{j_2=1}^{3} \sum_{j_3=1}^{2} Obstacle\_ID(j_1, j_2, j_3) \tag{9}$$

In defuzzification the contributor and detractor rules are compared. If the contributor rule firing strength is greater than the detractor, the detection flag is set. This maps the continuous fuzzy sensor fusion values into a discrete flag set.

4 MANIPULATION EXPERIMENT

To test the Obstacle_ID FSFN and the obstacle manipulation MSD, 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 Figure 10]. The robot was directed to drive forward for 10 seconds at a velocity of 26 cm/s. At approximately the 5 second mark the first collision occurred, resulting in a negative acceleration spike and a step increase in force [see figure 11]. The robot continued to move forward pushing the first obstacle. At approximately the 8 second mark 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 Figure 12]. The forward collision flag is set, and a mode transition from forward (mode 1) to push (mode 6) is initiated. The second collision triggers the heavy obstacle flag, which initiates a transition into stop (mode 2). Backup (mode 3) is triggered when the stop timer expires.

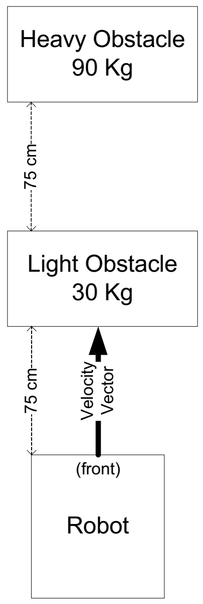

Figure 10. Manipulation experiment schematic

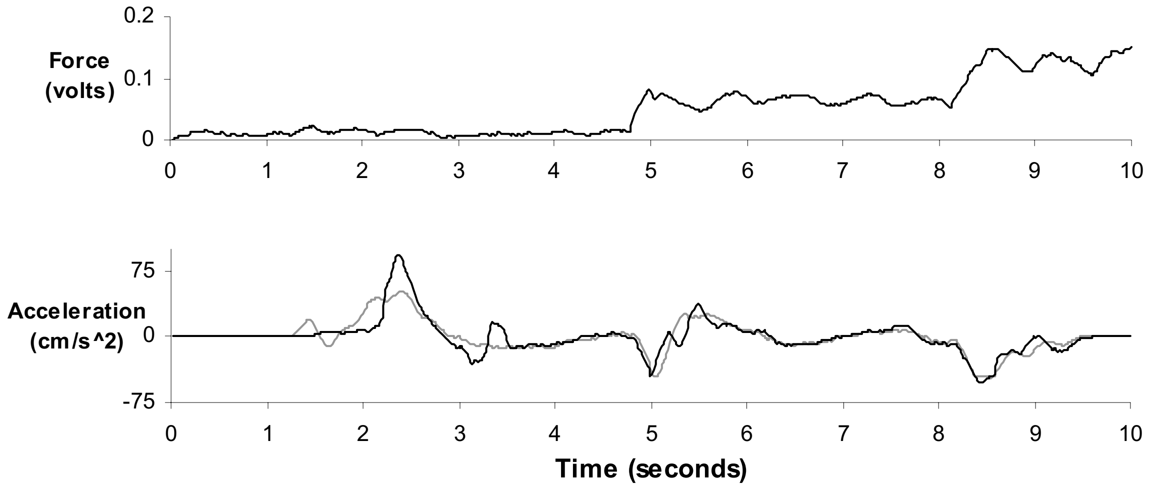

Figure 11. Sensor data for manipulation experiment

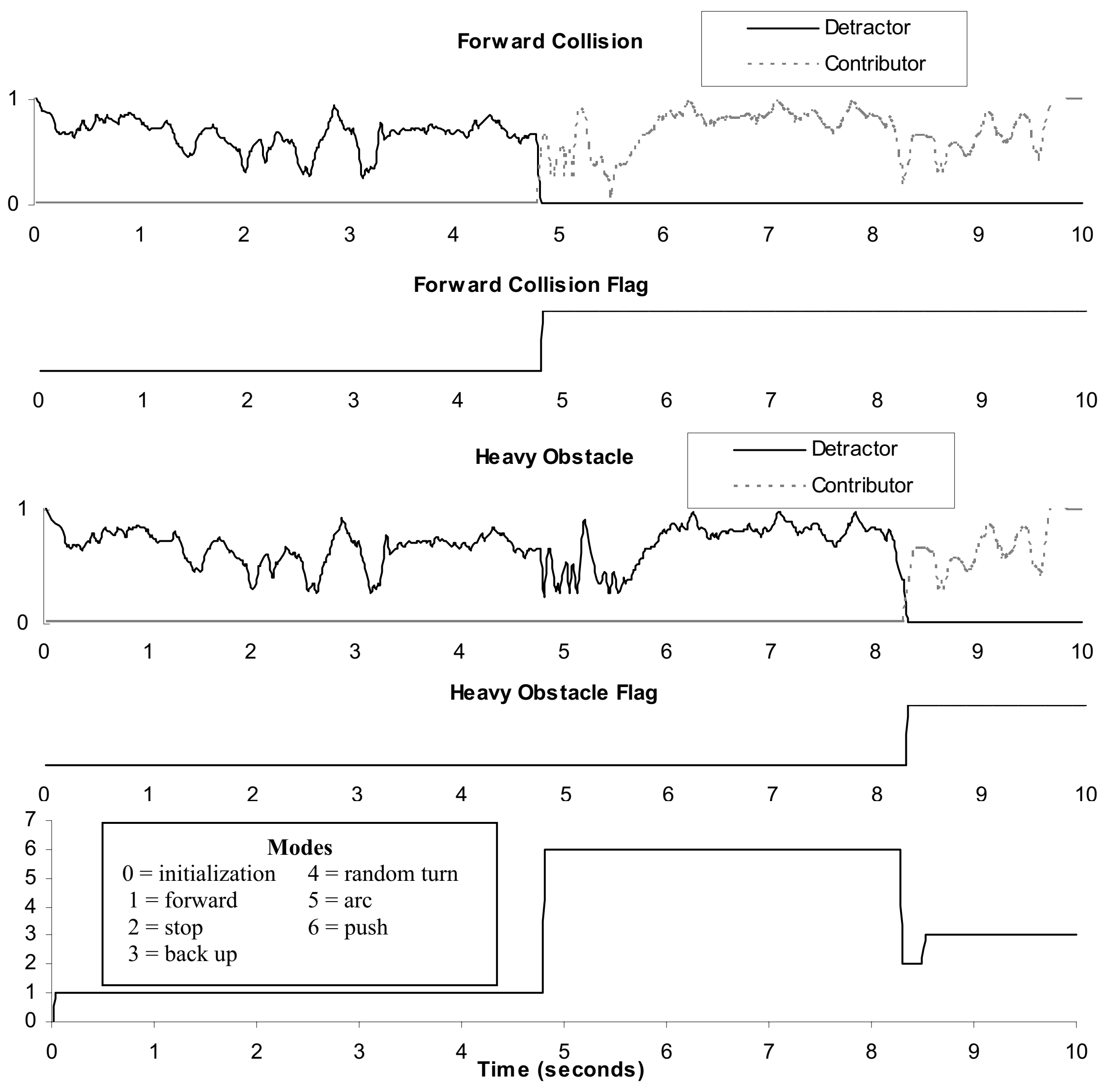

Figure 12. FSFN variables for manipulation experiment

4.1 Fault tolerance

The forward collision detection capability of the Obstacle_ID FSFN is inherently fault tolerant to sensor failure, due to the way the inference rules and combination rules are combined. The fuzzy AND operator used in the inference rules causes a rule to be set at a 0 value if any of its DOMs are zero. The fuzzy OR operator used in the combination rules allows all 0 value inference rules to be masked out of the equation.

In the case of a force sensor failure the most likely failure mode is a reading of 0. A 0 force reading causes force low (DOM(1_3))to equal 1, and the force medium(DOM(2_3)), and force high (DOM(3_3)) to equal 0. In the inference stage the rules are combined using the OR (minimum) operator so all rules containing DOM(2_3) and DOM(3_3) reduce to 0 and rules containing DOM(1_3)are unchanged. All the inference rules in the forward collision detractor rule contain DOM(1_3)

hence it remains unchanged. The forward collision contributor rule only has one inference rule containing DOM(1_3) so it reduces to:

$$forward\_collision\_contributor = Obstacle\_ID(1,1,1) \tag{10}$$

This reduction of the rules causes the collision detection to become solely dependent on acceleration [see Figure 13]. With force faulted the FSFN is functional but becomes equivalent to the preliminary_obstacle_ID FSFN and has the same limitations. It is unable to identify heavy obstacles and less reliable due false detections from to velocity tracking error.

Fault tolerance was verified experimentally by disconnecting the force sensor and repeating the collision detection experiments. Collisions were detected consistently.

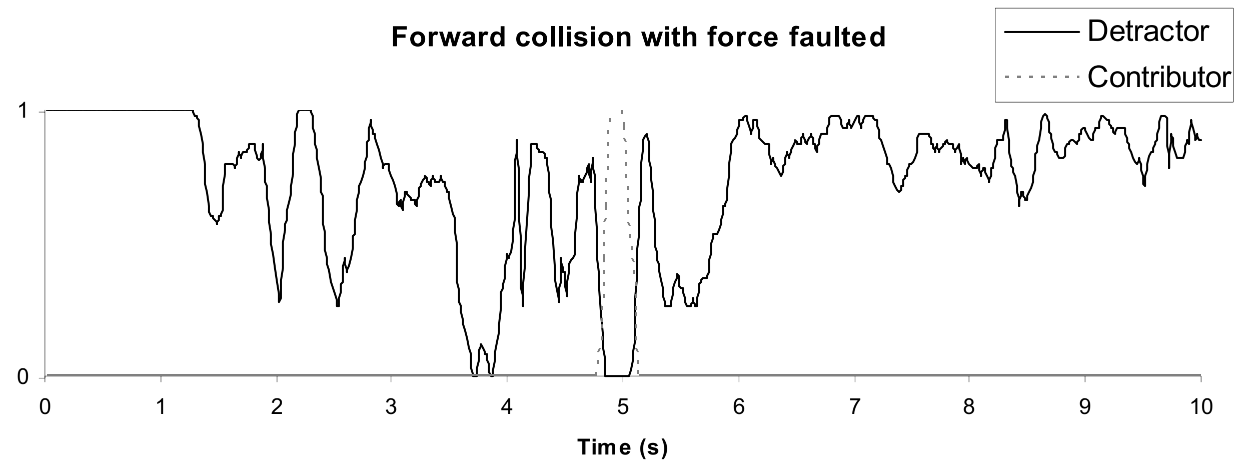

Figure 13. Force faulted collision

The most likely failure mode for the acceleration sensor is a reading of 0. As a result DOM(1_1), DOM(1_3), DOM(2_1) and DOM(2_3) =0; Also DOM(1_2) and DOM(2_2) =1. This allows the collision rules to be reduced to:

$$forward\_collision\_contributor = Obstacle\_ID(2,2,2) \oplus Obstacle\_ID(2,2,3) \tag{11}$$

$$forward\_collision\_detractor = Obstacle\_ID(2,2,1) \tag{12}$$

This reduction of the rules causes the collision detection to become solely dependent on force.

Fault tolerance was verified by conducting forward collision experiments with the acceleration sensors faulted, and collisions were consistently identified.

5. CONCLUSION

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.

The next stage of this work will begin by building a comprehensive library of actions, behaviors, and activities. The incorporation of learning algorithms will allow the FSFN's to be self tuning. Software tools will be built to automate the FSFN design process. Coordination among mechanisms on the same robot and coordination among multiple robots will be done through synchronization of MSDs, perhaps though a feedback structure. This development requires additional theoretical developments.

Acknowledgements

This work was supported by Arkansas Space Grant Consortium, NASA grant number NCC5-260, and University of Arkansas at Little Rock College of Information Science and Systems Engineering. 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, 9, 323-336, 1997.

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, 3, 893–898, 1996.

3. 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.

4. Suh, N. The Principles of Design, Oxford University Press, 1990.

5. Wright, A., Teague, W., Wright, A., Wilson, E. "Instrumentation of UALR labscale hybrid rocket motor" Sensors for Propulsion Measurement Applications, Proc. of SPIE Vol. 6222, 622202, 2006.