Modular robotic intelligence system based on fuzzy reasoning and state machine sequencing

B. Sights*, G. Ahuja, G. Kogut, E.B. Pacis, H.R. Everett, D. Fellars, S. Hardjadinata
Space and Naval Warfare Systems Center, San Diego
53560 Hull Street, San Diego, CA 92152

ABSTRACT

The fusion of multiple behavior commands and sensor data into intelligent and cohesive robotic movement has been the focus of robot research for many years. Sequencing low level behaviors to create high level intelligence has also been researched extensively. Cohesive robotic movement is also dependent on other factors, such as environment, user intent, and perception of the environment. In this paper, a method for managing the complexity derived from the increase in sensors and perceptions is described. Our system uses fuzzy logic and a state machine to fuse multiple behaviors into an optimal response based on the robot's current task. The resulting fused behavior is filtered through fuzzy logic based obstacle avoidance to create safe movement. The system also provides easy integration with any communications protocol, plug-and-play devices, perceptions, and behaviors. Most behaviors and the obstacle avoidance parameters are easily changed through configuration files. Combined with previous work in the area of navigation and localization a very robust autonomy suite is created.

Keywords: robotics, autonomy, obstacle avoidance, arbiter, fuzzy logic, state machine

1. BACKGROUND

The simplest and most basic form of robot control is *tele-operation*. During *tele-operation*, the rules for movement and arbitration among system inputs are relatively simple: user input is mapped directly to motor controller output. As robot autonomy increases, the requirements of the system architecture become increasingly complex. For example, in *reflexive tele-operation*, data from multiple sensors must be arbitrated with user input and combined into an appropriate output[1,2,3]. When moving from *tele-operation* to *reflexive tele-operation*, the space of possible inputs increases linearly and is easily handled by conventional programming techniques.

However, the addition of new behaviors and technologies under the Robotics Technology Transfer project[4] at SPAWAR Systems Center, San Diego (SSC San Diego) results in a nonlinear explosion of possible variables to arbitrate. Examples of such behaviors include *leader-follower*, *building exploration*, and searching for radioactive sources. These high-level behaviors depend on underlying component behaviors, such as obstacle avoidance, and operate concurrently with the autonomy levels described above. Complexity increases because the high-level behaviors also often need to influence the underlying component behaviors. For example, the *building exploration* behavior requires the robot to avoid obstacles encountered in its path. However, the *leader-follower* behavior requires the robot to maintain a close proximity to the leader. Clearly the obstacle avoidance in these two cases must operate differently. A robot with multiple levels of autonomy and various high level and low level behaviors must handle a potentially huge number of internal states in order to behave appropriately in any combination of autonomy level and behavior. If not well designed, the addition of just one new behavior can result in an exponential increase in complexity of the underlying software architecture.

To manage this complexity, we have built upon the *Intelligence Kernel*[5], an existing robot architecture originally developed by the Idaho National Laboratory. Our approach is based on the tight coupling of a hierarchical state machine, behavior arbiter, and an obstacle avoidance arbiter to adapt in real-time to the

*unmannedsystems@spawar.navy.mil; tel: 619-553-0707; fax: 619-553-6188;
www.spawar.navy.mil/robots

Unmanned Systems Technology IX. Proceedings of the SPIE, Volume 6561, p. 65610T (2007)

varying demands of the environment, desired behavior, user commands, and sensor inputs. The architecture can also accommodate additional behaviors or levels of autonomy with minimal modification to the existing framework.

2. SYSTEM DESCRIPTION

The system is comprised of the following main modular elements (Figure 1): communications protocols, device objects, perception objects, the robot "mediator", the obstacle avoidance arbiter, and the state machine that links together all the behaviors. Upon system initialization, the robot object reads from configuration files the communications protocols, devices, perceptions, and desired behaviors, as well as other configurable parameters. Using this information, the robot object creates and connects the proper devices, perceptions, behaviors, and communications modules together. Our approach allows developers to easily add or modify any module with minimal alterations to the rest of the system. An added benefit is the ability to rapidly compare the effect individual modules have on system performance.

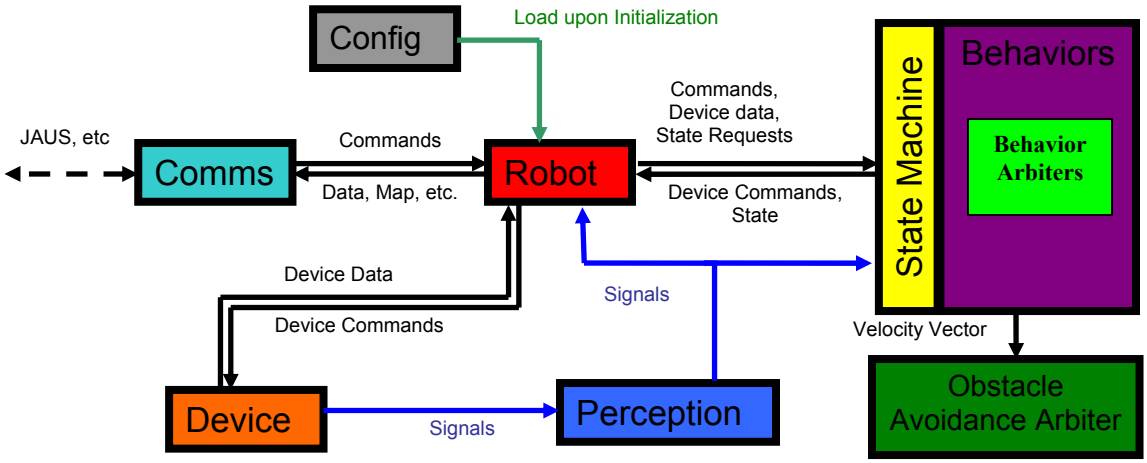


Figure 1. Diagram of the main building blocks of the system and how they are connected.

2.1. Communications

We define a communications module as a way for connecting to an external source that can influence a robot and/or accept messages from it. External sources include computer input devices, from a simple 2-axis joystick to a sophisticated sensor rich data glove, and several remote OCUs, from simple command prompt text interfaces to multi-robot, 3-D graphical interfaces. Our modular architecture allows for easy additions of any communication protocols or human-interface devices to the system, which provides the users with a major benefit of using the interface of their choice. Another byproduct of this architecture design is an easy-to-use communications programming interface that is exposed to external components, allowing for their immediate testing and evaluation before integrating them as system modules.

Currently the system supports the following input devices: all Linux supported joysticks (with a configuration file for user-desired mapping of buttons and axis to commands and controls), keyboard, and microphone for speech (using CMU's Sphinx[6] as the speech recognition software). The system also supports any JAUS compatible OCU, including MOCU[7], as well as other proprietary OCU's such as INL's 3D interface[8] and SSC San Diego's ROBART interface.

2.2. Devices

Our definition of a device is any sensor or actuator that sources data or sinks commands. Examples include ladar, sonar, encoders, motors, pan-tilt-zoom units, weapons, etc. Device abstractions are used so that

architecture is not tightly coupled to any specific device. The modular architecture also allows devices to be easily added and removed and allows the behavior of the robot to adapt based on the devices available.

2.3. Perceptions

We refer to perceptions in the context of this paper as the output of the algorithms that take raw data from a device and interpret it to extract useful information. For example, lines and corners can be gleaned from raw ladar data, or objects of interest recognized from raw video data. Perceptions can also be layered, such as our algorithm for finding open doorways using ladar data. This algorithm uses perceptions for finding lines and edges of objects and compares the orientation and composition of these lines and edges to determine if they fit the criteria consistent with a door.

Perception data can be used by behaviors to effect actions. For example, one of the *exploration* behaviors we are investigating uses an *open-space-finding* perception to determine in which direction there is an open space wide enough for the robot to move and is closest to the behavior's goal direction[9]. This particular algorithm is not included in the obstacle avoidance arbiter because we want to be able to easily ignore it when trying to get close to objects and also to keep the complexity of the obstacle avoidance arbiter minimized. The *exploration* behavior can also use the doorway perception to keep track of which rooms have been explored and label them for future reference.

The robot could even change what task it is performing based on the type of perception data it has received. For example, if a perception detects a weapon signature while the robot is enacting the *search* behavior, the robot switches to a *goto* behavior to inspect the weapon more closely. Sign reading is another example: once a sign is detected, the robot sequentially switches to the *goto* and *turnto* behaviors in order to position itself directly in front of the sign. Then the robot reads the sign and adds the information to its world model. Perception data can also be sent to the user or to other robots to provide relevant information about the environment. Figure 2 describes some of the common perceptions used and how they are linked to other modules of the system. An example of a user interface depicting robot perceptions is found in Figure 3.

Perception Name	Description	Subscribing Behaviors
Open Space	Directions that are safe for the robot to move in based on its size.	Autonomous, Shared, Goto , Waypoint
Laser Feature	Extracts lines, corners, and edges of objects from ladar data.	Laser Doorway, Landmark, Target Tracking, etc.
Laser Doorway	Uses Laser Feature data to find open doorways.	Exploration, Mapping
Laser Wall Follow	Uses lines extracted in the Laser Feature algorithm to calculate a heading to follow the closest wall at a particular distance.	Autonomous, Shared, Goto, Waypoing, etc.
Laser Pursuit	Uses ladar data to follow a target	Pursuit
Laser Tracking	Uses a data association algorithm to track edges of moving targets extracted in the Laser Feature algorithm.	HPDA, Pursuit, Exploration
Landmark	Tracks Perceptual and Device Data Features in the Environment	Localization, Mapping
Mapping	Uses range data to create a map and localize (SLAM)	Higher Level Intelligence, User, Explore Behavior
Vision	Uses Video Data to Recognize Objects	Pursuit, Target Tracking, User, Mapping

Figure 2. A list of the most commonly used perceptions, how they are linked to device data, other perceptions, and behaviors.

One key feature of the system architecture is that signals link devices, perceptions and behaviors in any combination through a publish-subscribe architecture. This allows automatic update of the perceptions whenever new data is available from devices upon which they rely. In the same regard, the robot and behaviors can be signaled whenever the perception data is new. This eliminates the need to specifically pass or check data in the main control loop of the program and, instead, triggers actions "subconsciously" inside the robot much like the human brain. For example, communications messages are automatically generated whenever new items of interest are found, such as weapons or human presence, and then notify the user of these important discoveries. The same signal could also cause the robot to change into a *pursuit* behavior to track down the intruder, or an *inspect* behavior to get a better view of the weapon.

Figure 3. Overlay of the robot's mapping, door finding, and intruder detection perceptions from the interior of an underground World War II bunker on Google Earth.

2.4 Robot Behaviors

The discussion on behaviors is divided into three subsections:

- State Machine – Handles requests to change robot behavior, and modularizes code used to implement behaviors.
- Arbiter – Behaviors that take inputs from multiple sources (e.g., operator commands, goal locations, wall-follow heading, ladar-based human tracking, vision-based human tracking, etc.) use a fuzzy logic arbiter to decide the best action to take given the inputs.
- Obstacle Avoidance – A generic fuzzy logic obstacle avoidance class has been developed. The output of the arbitrated behavior command is fed into the obstacle avoidance algorithm, which feeds the commands to the motor controllers/drivers.

2.4.1 State Machine

Algorithms for performing relatively high-level tasks, such as *mapping* or *person-following*, generally take place within variations of a closed-loop control system that monitors sensors and produces appropriate output. These control systems are largely event-based. The code executed during a given loop depends on events such as detected obstacles or user input in combination with the current state of the robot. All but the simplest robotics systems often dedicate a large amount of code to this control loop. However, during any given pass through such a loop, only a small percentage of this code is usually being executed. The task of ensuring that the correct code is selected for any given combination of robot state and event can be overwhelming. Traditional programming methods using state variables and "switch-case" type control structures often result in excessively complex code.

Finite state machines (FSMs) are a tool to help manage this problem. FSMs have long been used in robotics for a variety of purposes. Examples include Rodney Brooks' augmented finite state machines (AFSMs), which were used in the construction of the subsumption architecture[10]. More recently they have been used in a variety of swarm robot applications to help manage the complexity of swarming behaviors.

Our architecture uses a hierarchical state machine (HSM) to help solve problems that arise when implementing multiple high-level behaviors. As in most implementations of state machines in robotics, the formalism of a state chart eliminates ad-hoc "state variables" and control code, replacing them with well-defined states and transition events. The code specific to accomplishing the control tasks within a given behavior is explicitly included as part of the state. This promotes readability and effectiveness. However, robot tasks or behaviors may share a large common code base. The hierarchical state machine allows common sub-states to be re-used by multiple super states, avoiding code duplication and greatly simplifying the resulting state chart. To facilitate discussion we use terms analogous to robot behavior to name and describe states. A "behavior" state describes the top-level super state of the state machine, where "tasks" describe the reusable substates used to compose *behaviors*. While our current state chart includes 8 behaviors, only one relatively simple behavior, *goto*, is shown in the state chart diagram (Figure 4) to demonstrate the modularity and composition of the state machine. In the state chart, states are prepended with "S_' and events are prepended with "E_." The substates, "Turn To", "Move To", and "Get Unstuck" are reused in other behaviors without modification.

2.4.2 Arbiter

When a person decides to locate and retrieve an object in a particular room of a building, he switches between different behaviors to accomplish the overall goal: go to the room, locate the object, move towards the object, grab the object, etc. As part of traveling to the room, the person subconsciously recognizes walls, doorways, and obstacles, as well as the difference between his current location and his desired location in the building. The brain fuses all of this data into a motion strategy that will get him closer to his goal. Depending on the actual environment, the person will rely on different perceptions. At their residence, a person would rely on familiar landmarks to keep track of their location and obstacle detection to keep from stepping on their children's toys. In contrast, on a battlefield, the person might primarily rely on visual motion detection and hearing to safely traverse a building. On the robot, the changing of behaviors is handled by the state machine, but the fusion of inputs from multiple perceptions and device data, based on what behavior is being performed, is handled by an arbiter (Figure 5).

The quintessential example of this concept is when the robot is given a goal location in world coordinates, typically referred to as a *goto* command. In this case, the goal input into the arbiter is the distance and heading error to the goal. The other inputs into the arbiter are the open space angle closest to the desired heading and the angle error to the closest wall. The fuzzy rule sets for these inputs are shown in Figure 6 and some example degree-of-membership (DOM) functions are shown for *Distance to Target* and *Angle to Target* inputs in Figure 7. There are 525 fuzzy associate memory (FAM) rules for output rotational speed and 15 FAM rules for translational speed (it takes *Distance to Target* and *Arbitrated Rotational Speed* as parameters), which can be described using the following premises:

1. If we are *Not Close* to the target, then follow the open space angle that is closest to the desired heading, and travel at a *Positive Fast* translational speed.

2. If we are *Close* to the target, follow the heading to the target. If the output rotational speed is *Positive Fast* or *Negative Fast*, then the translational speed should be *Positive Slow*, otherwise it should be *Positive Fast*.
3. If we are *Very Close* to the target, turn aggressively to follow the heading to the target. If the output rotational speed is *Negative Slow* or *Positive Slow*, then travel at *Positive Slow* translational speed. If the output rotational speed is *Negative Fast* or *Positive Fast*, then travel at *Zero* Translational speed.
4. If we are within a specified radius of the target, signal achievement of goal condition.

An interesting byproduct of this method is that by simply changing the FAM rules and adding the distance to the closest wall as an input, we could make the robot hug the closest wall until it came upon a doorway or other tight passageway. This would create a "tactical" *goto* behavior whereby the robot would protect itself by staying close to the structure, instead of driving down the center of hallways and rooms where it is a much easier target.

Similar arbiters are also employed for other behaviors as well. For example, when the robot autonomously explores under *mixed-initiative control*, a human can intervene to influence the robot's behavior. In this case, the FAM rules are designed to follow the closest open-space heading to continue moving forwards, but if the user intervenes, then the rules are designed to favor the user commands over other perceptual data when deciding which velocity vector to send to the obstacle avoidance.

Parallel arbiters for different actuators can also be employed. For example, a robot equipped with a weapon could simultaneously arbitrate the drive control system to move into an ideal position to target an object while the weapon movement could be arbitrated with information from the vision system to keep the targeted object "in the crosshairs"[13].

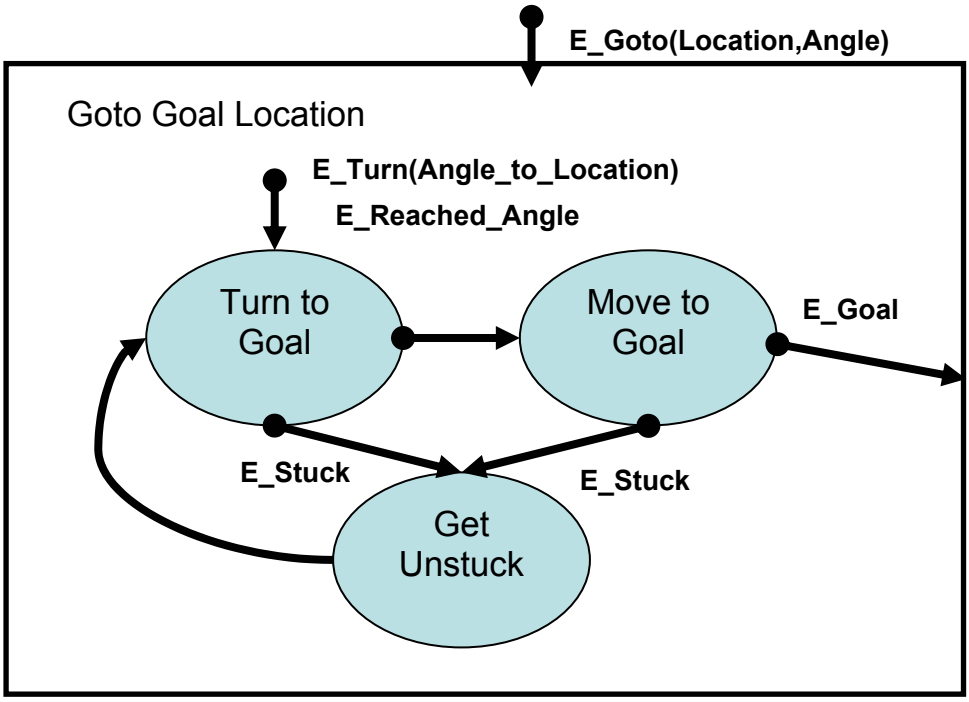


Figure 4. State machine diagram showing the modularity and composition of the *goto* behavior.

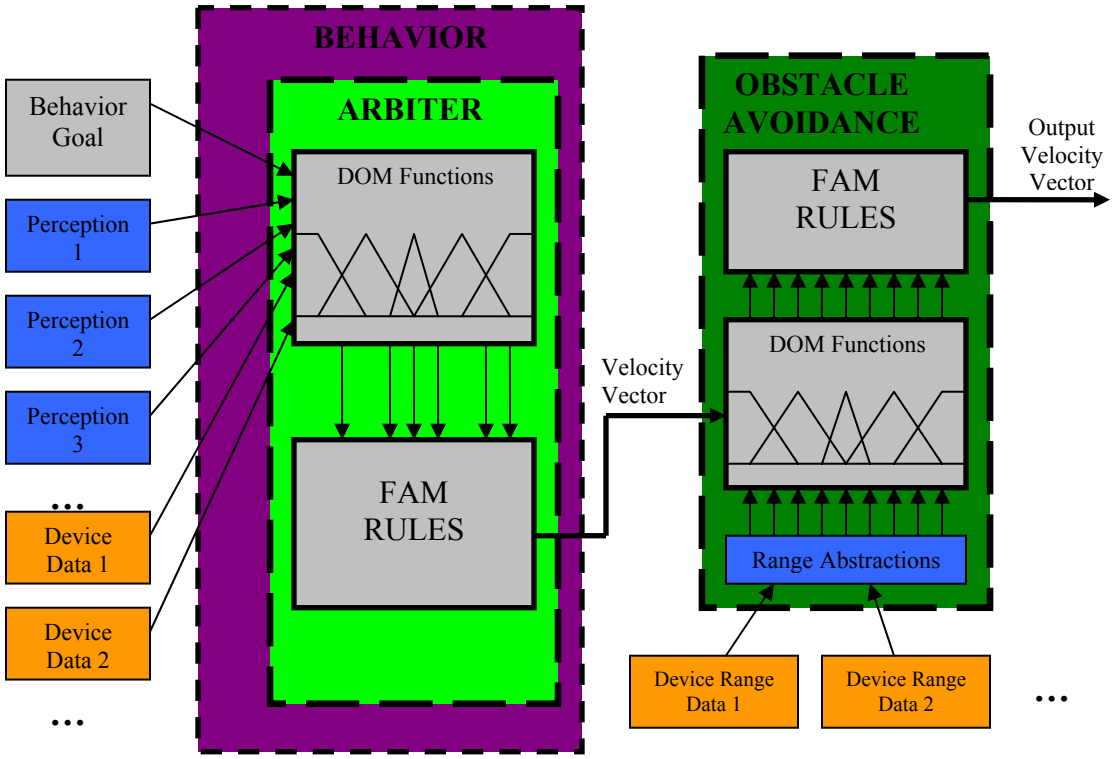


Figure 5. Diagram of how the behavior goal, perception data, and device data are arbitrated into a single command for the obstacle avoidance, which outputs a velocity vector for the drive control system.

2.4.3 Obstacle Avoidance

In contrast to Rodney Brook's early subsumption architecture, whereby collision/obstacle avoidance was the lowest layer and knew nothing of what higher-level behaviors wanted[14], our system takes the output of the behavior arbiter as an input to the obstacle avoidance. This eliminates the problem of the obstacle avoidance deciding between possible output velocity vectors when approaching an obstacle, and choosing one that does not bring the robot closer to its goal. Another benefit of our obstacle avoidance system is that the overlap of the fuzzy DOM functions allows for smooth sharing and handoffs between different rules as they activate and then move out of activation, much like how the human brain works[11]. This is much different than "if-then" rules commonly found in intelligence programming, whereby only one decision can be made at a given time. Instead, we can fuse the outputs of several rules based on the strength of how well the situation matches the conditions for the triggering of that rule.

Our obstacle avoidance changes how it reacts to obstacles depending on the behavior the robot is performing by changing the FAM rules it uses to make decisions as follows:

1. Reactive Obstacle Avoidance takes desired translational and rotational speed into account, but will override these commands in order to continue moving and avoid a collision.
2. Passive Obstacle Avoidance takes desired translational and rotational speed into account, but will stop the robot if it gets too close to an object. It will not try to provide a better velocity vector to keep the robot moving.
3. Goal Based Obstacle Avoidance is a hybrid of the reactive and passive methods. It takes the desired translational and rotational speeds to approach a target and will avoid obstacles if they are not in the direction of the target, but it will not turn away from objects which may be the target. This type of obstacle avoidance is particularly useful when trying to get to a goal location which is close to another object, such as a table or wall, or when trying to perform leader-follower behaviors.

INPUTS		OUTPUTS	
Distance To Target		**Translational Speed, Rotational Speed**	
VC	Very Close	NF	Negative Fast
CL	Close	NS	Negative Slow
NC	Not Close	ZE	Zero
		PS	Positive Slow
		PF	Positive Fast
Angle To Target			
NVL	Negative Very Large		
NLA	Negative Large		
NSM	Negative Small		
ZER	Zero		
PSM	Positive Small		
PLA	Positive Large		
PVL	Positive Very Large		
Open Space Angle, Wall Follow Angle			
NL	Negative Large		
NS	Negative Small		
ZE	Zero		
PS	Positive Small		
PL	Positive Large		

Figure 6. Input and output Fuzzy sets for the Goto behavior.

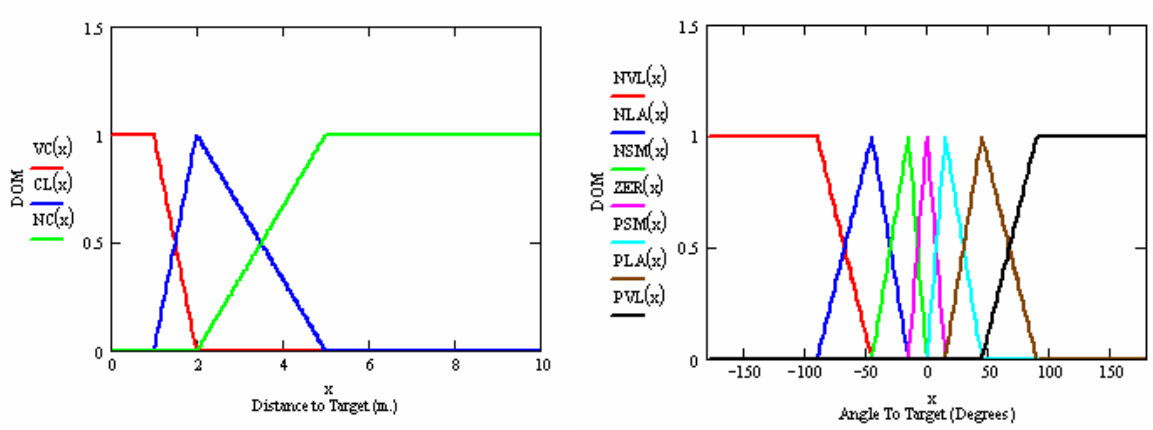
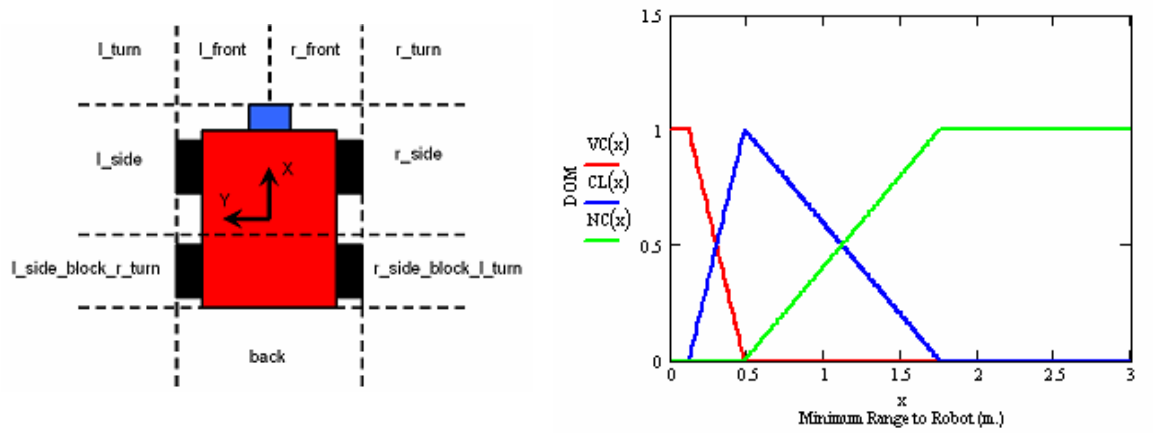


Figure 7. DOM Functions for the *Distance to Target* and *Angle to Target* fuzzy sets in the Goto Arbiter.

2.4.3.1 Steps of Obstacle Avoidance – Range Abstraction

When the robot is initialized, the obstacle avoidance "range abstraction" function is linked to range measurement devices (such as ladar, sonar, stereo vision) using signals. Whenever any of these devices receives new data, the obstacle avoidance object is signaled and automatically updates the range abstractions. This is done by using the length and width of the robot to determine into which region (Figure 8a) the obstacle fits. Also, there is overlap between regions that is not depicted in Figure 8a, which takes into account the ambiguity inherit in some range sensors, such as sonar. Next, the distance from the obstacle to the robot is calculated, and if it is the closest object in that region, then the value is stored as the minimum distance for that region.



a. **b.**

Figure 8. a.) Range Abstraction Regions. **b.)** Degree of Membership Function for the *l_front* and *r_front* fuzzy sets of the ATRV Sr. platform when the maximum speed is 0.6 m/s with an assumed minimum deceleration of 1 m/s^2.

2.4.3.2 Steps of Obstacle Avoidance – Fuzzifying the Data

Each region has its own Fuzzy DOM function, giving us 9 DOM rule sets. The front regions all have three DOM fuzzy set values (Figure 8b.) while the sides and back regions only have two. This is mainly due to the fact that our robot primarily moves along the X axis and generally moves forwards, so reducing the number of fuzzy set values in regions that do not factor as prominently into the decision making process means that we do not have to write as many rules. Another aspect of this decision is that when turning, the robot only wants to know if something is within a range where a collision is possible, so using only *Very Close* and *Not Close* fuzzy set values makes more sense. On the other hand, as the robot drives forward, it needs to know if there is an object in the way at all, if it should slow down to navigate around the object, or if it needs to stop to avoid it all together. This leads to using *Very Close*, *Close*, and *Not Close* fuzzy set values for the front regions.

Note that the minimum distance the robot will get to an object before the range reading is completely a member of the *Very Close* fuzzy set is based on the difference between the robot's diagonal length and overall length dimensions. This ensures that the robot will be able to turn if it has to stop. The distance at which ranges begin to belong to the *Very Close* fuzzy set is dependant on the robot's current maximum translational speed and the robot's minimum deceleration (e.g., at what rate will it stop if I give it a stop command).

2.4.3.3 Steps of Obstacle Avoidance – Making Decisions

After each minimum range crisp value has been fuzzified using the corresponding DOM function, it is evaluated using the FAM rules for the type of obstacle avoidance currently being used. Currently, there are 5315 rules, with reactive obstacle avoidance having 2430 (Figure 9), passive obstacle avoidance having 455, and goal based obstacle avoidance having 2430. You can see that as the complexity of robot's behavior increases, the number of rules increases.

Upon initialization of the obstacle avoidance module, the FAM rules are read from a configuration file. This means that it is possible to create different configuration files for different types of robots. For example, most of our laboratory robots are skid-steered, but many of the larger vehicles at SSC San Diego are not. If we wanted to test our software on an Ackerman-steered vehicle, we would simply create a new configuration file by changing the appropriate rules, and then read rules from that configuration file instead.

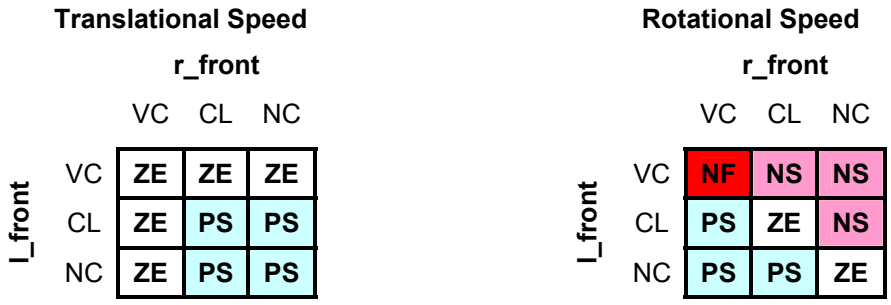


Figure 9. Example fuzzy FAM Rules for translational and rotational speeds in reactive obstacle avoidance with a cross section of l_turn, r_turn, l_side, r_side, back minimum ranges = *Not Close* (NC) with desired translational speed = *Positive Slow* (PS), and desired rotational speed = *Zero* (ZE).

An important lesson learned with this type of obstacle avoidance is that some rules need to be adjusted depending on the robot's previous behavior in order to keep the robot from oscillating in situations such as corners. In that case, as the robot turns in one direction, the rules would naturally tell it to turn in the other direction because the minimum ranges in the side regions of the robot in the direction of the turn will become smaller than those on the other. The solution is to have the rules that trigger in these instances switch their preferred direction of rotational speed to match the direction of the current robot rotational speed.

2.4.3.4 Steps of Obstacle Avoidance – Defuzzification

After the rules have been evaluated, the resulting decision levels for the outputs are defuzzified using the singleton method. Using this method saves computational time and produces easily adjustable behavior[12].

Overall, the use of fuzzy logic results in very smooth, robust, and predictable behavior. We have developed a solution to a multi-dimensional control problem by decomposing it into human language. The net benefit is ease of tuning, debugging, and a high level of portability.

3. SUMMARY

The system described in this paper provides a modular, robust, and platform-independent intelligence package designed to allow easier integration of new mobile robot capabilities. Multiple communications protocols and user input devices can easily be supported, allowing for the robot to fit the needs of multiple mission scenarios. The state machine architecture allows the composition of multiple behaviors to produce more refined and intelligent modes of operation. Finally, the fuzzy logic based arbitration of behaviors and fuzzy logic based obstacle avoidance produce smooth, robust, and predictable movement of the robot.

This system has already been demonstrated for and received initial positive feedback from representatives of NAVEODTECHDIV, DARPA, and FCS. Current work includes development of new modules to support the integration and test efforts, funded by the Center of Commercialization of Advanced Technology (CCAT), of industry products being evaluated for military robotic applications by the Technology Transfer Project. The entire system is also undergoing optimization and verification testing to support the Autonomous Robotic Mapping System (ARMS) project. In order to refine the algorithms more efficiently, future plans include applying genetic algorithms to the fuzzy logic arbiters and obstacle avoidance.

REFERENCES

1. Laird, R.T. and H.R. Everett, "Reflexive Teleoperated Control," Association For Unmanned Vehicle Systems, 17th Annual Technical Symposium and Exhibition (AUVS '90), Dayton, OH, pp. 280-292, July-August, 1990.
2. Everett, H.R., G.A. Gilbreath, and T.T. Tran, "Modeling the Environment of a Mobile Security Robot," NOSC Technical Document 1835, Naval Oceans Systems Center, August, 1990.
3. Everett, H.R., G.A. Gilbreath, R.T. Laird, and T.A. Heath, "Multiple Robot Host Architecture," NCCOSC Technical Note 1710, Revision 1, Naval Command Control and Ocean Surveillance Center, RDT&E Division, San Diego, CA, November, 1992.
4. http://spawar.navy.mil/robots/research/TechXfer/TechXfer.html
5. D. J. Bruemmer, J. L. Marble, D. D. Dudenhoeffer, M. O. Anderson, and M. D. McKay. Mixed-Initiative Control for Remote Characterization of Hazardous Environments, *HICSS 2003*, Waikoloa Village, Hawaii, January 2003.
6. http://cmusphinx.sourceforge.net/html/cmusphinx.php
7. Powell, D. N., Gilbreath, G., Bruch, M. H., "Multi-robot operator control unit," SPIE Proc. 6230: Unmanned Systems Technology VIII, Defense Security Symposium, Orlando, FL, April 17-20, 2006
8. D. J. Bruemmer, D. A. Few, C. W. Nielsen. 2006. "Spatial Reasoning for Human-Robot Teams," in Emerging Spatial Information Systems and Applications, ed. Brian Hilton, Idea Group Inc. Hershey, PA, pp. 350 – 372
9. http://www-personal.umich.edu/~johannb/vff&vfh.htm
10. Brooks, R.A. *Cambrian Intelligence: The Early History of the New AI*, MIT Press, Cambridge, MA, 1999.
11. Melin, P. and Castillo, O. *Hybrid Intelligent Systems*, Springer-Verlag Berlin Heidelberg, New York, NY, 2005.
12. B. Kosko, *Neural Networks and Fuzzy Systems*, New Jersey: Prentice-Hall Inc. Pub., 1992.
13. Sights, B., Everett, H.R., Pacis, E.B., and G. Kogut, "Integrated Control Strategies Supporting Autonomous Functionalities in Mobile Robots," Computing, Communications, and Control Technologies Conference, Austin, TX, July 24-27, 2005.
14. Yen, J. and Pfluger, N. 1995, "A Fuzzy Logic Based Extension To Payton and Rosenblatt's Command Fusion Method For Mobile Robot Navigation", *Systems, Man and Cybernetics, IEEE Transactions on* 25(6), 971-978.