Integration of Maps into the Cognitive Robotic System

Scott D. Hanford* and Lyle N. Long†
The Pennsylvania State University, University Park, PA, 16802, U.S.A.

We have previously developed the Cognitive Robotic System, or CRS, to use the Soar cognitive architecture for the control of unmanned vehicles. The system was able to demonstrate the applicability of Soar for unmanned vehicles by controlling a ground vehicle to navigate to a target location using GPS in the presence of obstacles, including a cul-de-sac. However, only fairly basic information about the environment was available to Soar during this mission. The Soar cognitive architecture will be most interesting for the control of mobile robots when it is coupled to perception systems that are capable of extracting high level information from the environment which Soar can use in its reasoning. Therefore, recent work on the CRS has focused on the development of perceptual systems capable of generating relevant information from the environment for use by Soar agents in the CRS. This paper describes the integration of a system for generating occupancy grid maps in the CRS. The occupancy grid algorithm and a simulation environment used to develop and test the algorithm are described. Occupancy grids of common intersection types generated using the CRS and results from a system that uses fuzzy logic to detect common intersections from occupancy grids are shown.

I. Introduction

The Cognitive Robotic System (CRS) has been previously developed to use the Soar cognitive architecture for the control of unmanned vehicles. The development of this system and its application to the unmanned vehicle problem of navigation to a goal location while avoiding obstacles has been described earlier.[1] This section will discuss the applicability of Soar for unmanned vehicles and the motivation for including a mapping capability into the Cognitive Robotic System for use with Soar.

A. Soar for control of unmanned vehicles

The importance of the characteristics of autonomy and intelligence for unmanned vehicles has been illustrated by programs such as the DARPA Grand Challenges, the DARPA Learning Applied to Ground Vehicles (LAGR) program, and NASA's Mars Exploration Rovers. Computational psychology tools, which describe cognition using computer algorithms and programs, and cognitive architectures in particular have become more popular for the control of unmanned vehicles.[2–7]

The Soar cognitive architecture[8] has been shown to be well-suited for high-level robot activities such as reasoning. Jones et al. have used the Soar architecture to autonomously fly U.S. military fixed-wing aircraft during missions in a simulated environment for the TacAir-Soar project.[9] The agents in TacAir-Soar used 5200 production rules, 450 total operators, and 130 abstract operators to demonstrate autonomous and intelligent behavior in a complex, albeit simulated, environment with real-time constraints. The ability of the Soar architecture to handle large agents is important for intelligent autonomous behavior in complex environments. Soar uses the Rete algorithm, which is efficient for large numbers of working memory elements, to determine which of the agent's production rules have their conditions for firing matched based on the current contents of working memory.[10]

The use of the Soar cognitive architecture to control unmanned vehicles has several benefits. Soar has been shown to be scalable to thousands of rules, allowing sophisticated Soar agents that use detailed knowledge in the forms of rules or sensor information to be developed. Cognitive architectures are good for general

*Graduate Student, sdh187@psu.edu, AIAA Student Member.
†Distinguished Professor, lnl@psu.edu, AIAA Fellow.

decision making, so a Soar agent can be flexible enough to apply to different mobile robot missions. A Soar agent could also be capable of considering multiple approaches to the same problem and deciding which algorithm would be most effective for the agent's current situation. The agent could switch approaches if the current one is ineffective, leading to the potential for robust behavior.

However, Soar is not well-suited to perform a large amount of numerical calculations, perform low level control of motors, process large amounts of low level information from sensors, or solve optimization problems. Combining Soar with other software systems (including systems such as neural networks,[11] robot control algorithms,[12] state estimation techniques,[13] and object recognition methods such as the Scale Invariant Feature Transform (SIFT)[14] that complement the strengths of Soar could increase the usefulness of Soar for the control of unmanned vehicles. The Soar Markup Language (SML) has been developed to allow simplified interaction of a Soar agent with external environments through sensors and motors.[15] Also, since SML can be used with popular programming languages such as Java and C++, integration with other software systems is possible.

A robotic system, the Cognitive Robotic System, has been developed to use the Soar cognitive architecture for the control of unmanned vehicles and was implemented on two types of mobile robots.[1] This multi-threaded software system was written using Java and integrated the Soar cognitive architecture with sensors, motors, and other software systems. A Soar agent was developed to test the CRS on a practical mission of navigation to a goal location while avoiding obstacles. The agent was able to successfully navigate a cul-de-sac type obstacle without the use of a map by using different types of approaches to avoiding obstacles of varying complexity. This paper showed that adding more knowledge to a Soar agent in the form of Soar rules and providing it with more information from sensors can lead to a more capable Soar agent.

Long and Kelley[16] discussed the possibility of conscious (or self-aware) robots. They described components that would be useful for an architecture for a possibly conscious robot: Soar could be used as part of such an intelligent system.

Although the research described in this previous paper[1] demonstrated the usefulness of Soar for intelligent robots on a practical mission, the Soar agent could perform this mission with minimal high level (symbolic) perceptual information about the environment. Incorporating improved perceptual systems (e.g. computer vision object recognition algorithms) or traditional robot capabilities such as map building that could be used to find meaningful symbols from the environment that the Soar agent can reason about could substantially increase the capabilities of the CRS and its Soar agents. The CRS has been developed with the intention of integrating additional software systems within the CRS. The goal of the research described in this paper is to incorporate a mapping capability into the CRS.

B. Maps for use in the CRS

An important problem for mobile robotics is conducting a search within an unmapped building for specific objects. Perhaps the most studied application of this problem is the use of robots for search and rescue missions. The RoboCup (www.robocup.org) is well known for its soccer competitions, but it also has competitions in the domain of urban search and rescue. The RoboCupRescue competition has divisions for both physical and simulated robots. The simulated division emphasizes advanced robotic capabilities such as formulating plans for multiple heterogeneous agents, learning, and completing the mission autonomously, but the physical robots are evaluated on more low level tasks such as map building, identifying simulated victims, and the effectiveness of their human-computer interfaces.[17] The robots compete in three arenas, with each level of arena simulating an increasing level of structural disruption.[18] Teleoperated robots are more common as autonomous robots are allowed to compete only in the least challenging arena. Recently, papers describing more advanced robotic algorithms being applied to search and rescue applications have been written. These papers have included both architecture development for performing search and rescue missions with an attention mechanism to focus the robot's exploration behavior[18] and using an advanced image matching technique to recognize standard hazardous material signs.[19]

Important capabilities for successful robots for the problem of search within an unknown building environment include the ability to create a map of the environment, ideally with symbolic information that could easily be shared with humans such as first responders,[17] and the ability to identify objects of interest and report where they are located within the environment. This problem is a useful challenge to consider for extending the capabilities of the CRS, since these capabilities align well with the possible extensions to the CRS mentioned at the end of Section I.A.

1. Background

Building a map is a fundamental way in which a mobile robot can interact with and learn about its environment. A robot builds a map by using its perceptual capabilities, such as vision, audio, touch, and range finding, to create a model of its environment. The problem of creating useful representations with noisy sensor data can be very challenging; mapping has been studied in depth. Localization, or the knowledge of where a robot is located with respect to some reference, is important for mapping applications.[20] Often a robot will not know its location or a map of its environment: the problem of creating a map while localizing a robot with respect to that map at the same time is called Simultaneous Localization and Mapping (SLAM). Thrun[21] has called SLAM one of the "key enabling technologies of mobile robot navigation" and said that this "problem is generally regarded as one of the most important problems in the pursuit of building truly autonomous robots." Two types of representations have been widely used in robot mapping: metric and topological. There has also been interest in combining these representations in hybrid metric/topological maps.

Metric Metric maps are quantitative and rely on measurements to locate objects in their environment (for example, an obstacle is located at an (x,y) location) and describe the relationships between objects with physical quantities such as distance and direction. A popular form of metric maps is the occupancy grid.[22] In this approach, the environment to be mapped is discretized into a grid and each cell in the grid contains information about the probability that it contains an obstacle. Occupancy grids work well for environments with arbitrary, dense obstacles (for example, office environments). However, sharing maps between heterogeneous robots can be difficult, since what one robot considers an obstacle may not be an obstacle to a different robot.

Metric maps encode detailed metric information and are a good match to popular robot sensors: laser and sonar range finders are good at detecting whether space is open or occupied by obstacles. A weakness of using metric representations is that they rely on accurate localization of the robot. This can be difficult in environments where the robot's position is not directly observable (no absolute positioning system such as GPS or absolute landmarks). In environments where the accuracy of the robot's localization decreases with time (most indoor settings, for instance), the accuracy of the metric representation degrades for long time frames or large areas. However, there are approaches to maintain good accuracy of localization in the absence of absolute position information. Additionally, there are large memory and processing requirements needed for metric representations in large-scale environments.

Topological Topological maps are qualitative and generally describe how objects in the environment are connected to each other. Typically, topological maps consist of nodes (landmarks, for examples) that are connected by arcs. Topological representations are considered to be good for large-scale environments because of their compact representations. In Thrun,[23] an occupancy grid with 27,000 occupied cells was transformed into a topological graph of 67 nodes. Two challenging aspects of constructing topological maps for robots are the identification of important landmarks that are suitable to be used as nodes and the correspondence problem of whether landmarks that appear similar correspond to the same or different nodes in the map. Tomatis[24] stated that "the perception required by a topological approach has to permit the distinction between places." This requirement describes one reason why metric maps have been more common in robotics than topological maps. The most popular types of robot sensors, laser and sonar range finders are good at identifying whether space is open or not, but have difficulty distinguishing between places. However, if the places represented by nodes in the graph can be distinguished from each other, sharing information between robots or between robots and humans may be easier with topological maps than metric maps.

Hybrid Hybrid metric-topological maps have also been used in an attempt to combine the strengths of the two representations. For example, Tomatis[24] developed a methodology to build a global topological map for an office environment. The topological map was used to connect local metric maps of individual rooms. A robot with a laser range finder was used to identify corners and openings (such as intersections or doorways) that could be used as both landmarks in the topological maps and reference points for the metric maps.

2. Occupancy grids in the CRS

Initially, the focus of adding a mapping capability to the CRS will be the generation of occupancy grids. Occupancy grids will be useful for the mission of search within an unmapped building as this representation is well suited for the types of obstacles encountered in building environments and can be used to report the locations of objects of interest. In addition to its importance for the successful completion of the search mission, the generation of occupancy grids for the CRS will demonstrate the integration of a traditional robotic algorithm into the CRS. The occupancy grid could be used within the CRS for things such as obstacle avoidance,[25] grid-based path planners, or exploring the use of spatial reasoning capabilities of a Soar agent to reason about maps. Additionally, other researchers have been able to extract symbolic information from occupancy grids.[26,27] This capability would be very interesting for the CRS as it could provide symbolic information about the environment for Soar agents to use in reasoning.

The rest of this paper discusses the integration of occupancy grids in the CRS. Section II gives an overview of the algorithm that is used to create the occupancy grids and Section III describes a simulation that was used to develop and test our implementation of that algorithm. Section IV discusses the hardware and software that was used to generate occupancy grids and detect intersections in the CRS and results for common types of intersections are shown.

II. Occupancy grid algorithm

The problem of creating occupancy grids has been well studied, so not all the details are included here. A more detailed derivation is available in chapters four and nine of *Probabilistic Robotics* by Thrun, Burgard, and Fox.[28] The goal of a mapping algorithm is to estimate the probability, or likelihood, of a map, m, given the measurements for times 1 through t, $z_{1:t}$, and robot states for times 1 through t, $x_{1:t}$:

$$p(m|z_{1:t}, x_{1:t}). \tag{1}$$

The occupancy grid approach divides the environment into cells:

$$m = m_i. \tag{2}$$

The probability that the i^{th} grid cell is occupied (i. e. the probability that the grid cell has a value of 1) is denoted by: $p(m_i = 1)$ (or equivalently, $p(m_i)$). Generally, occupancy grid algorithms assume that the probability of each grid cell m_i being occupied can be considered independently and that the map is static. With these assumptions, the map can be calculated by calculating the probability that each grid cell m_i is occupied, given the measurements for times 1 through t, $z_{1:t}$, and robot states for times 1 through t, $x_{1:t}$:

$$p(m_i|z_{1:t}, x_{1:t}). \tag{3}$$

For computational reasons, the log odds ratio of Eq. (3) is often used for implementations of occupancy grids. The odds ratio is defined as:

$$\text{Odds}(m_i) = \frac{p(m_i|z_{1:t}, x_{1:t})}{p(\neg m_i|z_{1:t}, x_{1:t})} = \frac{p(m_i|z_{1:t}, x_{1:t})}{1 - p(m_i|z_{1:t}, x_{1:t})}. \tag{4}$$

Then, taking the log of both sides of Eq. (4) results in:

$$\log\left(\text{Odds}(m_i)\right) = l(m_i) = \log \frac{p(m_i|z_{1:t}, x_{1:t})}{1 - p(m_i|z_{1:t}, x_{1:t})}. \tag{5}$$

Several sources have shown how Eq. (5) can be used to calculate $l_{t,i}$, the log odds ratio of cell i at time t:

$$l_{t,i} = \log \frac{p(m_i|z_t, x_t)}{1 - p(m_i|z_t, x_t)} + \log \frac{p(m_i|z_{1:t-1}, x_{1:t-1})}{1 - p(m_i|z_{1:t-1}, x_{1:t-1})} + \log \frac{p(m_i)}{1 - p(m_i)}. \tag{6}$$

The term $p(m_i|z_t, x_t)$ in Eq. (6) is the inverse sensor model, which calculates the probability that a grid cell i is occupied given the current (at time t) measurements and robot state.

If nothing is known about the map of the environment at $t = 0$, as will be the case in our mission, $p(m_i) = 0.5$ and the last term in Eq. (6) will be equal to zero. Also, noting that:

$$\log \frac{p(m_i|z_{1:t-1}, x_{1:t-1})}{1 - p(m_i|z_{1:t-1}, x_{1:t-1})} = l_{t-1,i}, \tag{7}$$

where $l_{t-1,i}$ is the the log odds ratio of cell i at time $t-1$, Eq. (6) can be simplified to:

$$l_{t,i} = \log \frac{p(m_i|z_t, x_t)}{1 - p(m_i|z_t, x_t)} + l_{t-1,i}. \tag{8}$$

The probability that a cell i is occupied can be calculated from the log odds ratio for that cell:

$$p(m_i|z_{1:t}, x_{1:t}) = 1 - \frac{1}{1 + \exp(l_{t,i})}. \tag{9}$$

III. Simulation for algorithm testing

A simulation has been developed for Matlab to test the implementation of the occupancy grid algorithm. This section describes the robot, environment, sensor, and inverse sensor models that have been developed for the simulation and shows an example occupancy grid generated using the simulation.

A. Robot model

The robot for the simulation is a non-holonomic ground vehicle, similar to the SuperDroid, our mobile robot platform that will be described in Section IV.A. The vehicle state, x, consists of the robot's x and y positions x_p and y_p, its orientation θ, and its speed v:

$$x = [\ x_p \quad y_p \quad \theta \quad v\ ]^T \tag{10}$$

The dynamics of the simulated robot are given by

$$\dot{x} = f(x, u), \tag{11}$$

where f is the set of nonlinear equations that describe the motion of the robot and u are the control inputs (u_1 and u_2 defined in Eqs. (14) and (15).

The robot state at time $k+1$ is given by:

$$x_{k+1} = x_k + v_k \cdot \cos\theta_k \Delta t \tag{12}$$
$$y_{k+1} = y_k + v_k \cdot \sin\theta_k \Delta t \tag{13}$$
$$\theta_{k+1} = \theta_k + u_{1,k+1}\Delta t \tag{14}$$
$$v_{k+1} = v_k + u_{2,k+1}\Delta t \tag{15}$$

The robot's measurement vector, z, includes information from simulated encoders (change in orientation, $\Delta\theta$, and speed, v, and ranges from the simulated sonar sensors, z_i, where i refers to the i^{th} sonar sensor, and can be written as:

$$z = [\ \Delta\theta \quad v \quad z_1 \quad \cdots \quad z_N\ ]^T, \tag{16}$$

where N is the number of sonar sensors used by the robot.

We will initially assume the measurements from encoders, $\Delta\theta$ and v, are exact (free of noise) to test the occupancy grid algorithm. The measurements from the sonar sensors are governed by the model described below in Section B.

B. Simulation environment and sensor models

The simulation will use a model similar to the arenas developed by the National Institute of Standards and Technology (http://robotarenas.nist.gov/) for the RoboCupRescue competition. In Matlab, the environment is considered to be a two-dimensional plane and is represented by a two-dimensional matrix. Free space in the environment is represented by 0's in this 2-D matrix and obstacles are represented as 1's. A model of the environment is shown in Fig. 2a.

A model of a sonar distance sensor was created to sense the distance to obstacles in the simulation environment. The sonar sensor models were created to represent the sonar sensors used in the CRS (described in section IV.A). The following parameters used for the simulated sonar model were chosen to correspond to the parameters of the physical sonar sensors:

- Angle of perceptual field of view = 50 degrees
- Minimum sensor range = 0.03 meters
- Maximum sensor range = 4 meters
- Standard deviation of measurements, σ, = 0.01 meters

The distances measured by the sonar sensors are assumed to be corrupted by Gaussian noise. Each measurement from the simulated sonar sensor is selected from a Gaussian distribution with a mean corresponding to the correct measurement and standard deviation σ.

C. Inverse Sensor Model

The last item to define for the simulation is the inverse sensor model in its log odds form: $\log \frac{p(m_i|z_t,x_t)}{1-p(m_i|z_t,x_t)}$ from Eq. (8). Thrun, Burgard, and Fox[28] give an overview of a simple inverse sensor model for range finders that we have implemented for the sonar sensor in the simulation. This simple model increases the probability that the grid cells within the sensor beam that have ranges approximately equal to the measured range are occupied. The model decreases the probability that the grid cells within the sensor beam that have ranges less than the measured range are occupied. For cells which are outside the sensor beam or with ranges greater than the measured range, the model returns the same probability as before the measurement.

The inverse sensor model used for the sonar sensors is described below and calculated for each occupancy grid cell m_i within the perceptual field of the sonar sensor, which is approximated by a fifty degree wide cone.

- First, r_i, the distance between the current cell m_i and the sensor is calculated.
- If the distance r_i is less than the current sensor measurement, z_i, then the current cell m_i is likely to be unoccupied, and the inverse sensor model returns a value l_{free} that will decrease the probability that the cell m_i is occupied.
- If the distance r_i is greater than or equal to the current sensor measurement, z_i, and the difference between r_i and z_i is less than the expected obstacle thickness t, the current cell is likely to be occupied by an obstacle and the inverse sensor model returns a value l_{occ} that will increase the probability that the cell m_i is occupied.
- If the distance r_i is greater than or equal to the current sensor measurement, z_i, and the difference between r_i and z_i is greater than the expected obstacle thickness t, the current sensor reading does not give any new information about the contents of the current cell and the inverse sensor model returns a value l_0 that will not change the probability that the cell m_i is occupied.

Fig. 1 shows an example result of the inverse sensor model, demonstrating the shape of the sonar sensor's perceptual field. More sophisticated inverse sensor models are available: for example, some models use statistical distributions to weigh cells in the middle of the sensor's perceptual field more than cells at the edge of the perceptual field. A more complex inverse sensor model could be implemented if the performance of this simple sensor model is inadequate.

D. Simulation Results

The occupancy grid algorithm was tested by having the simulated robot move through the simulated maze environment and generate an occupancy grid. Three sonar sensors were used to sense obstacles in the maze: one facing straight ahead, one facing to the robot's right (90 degrees to the right of straight ahead), and the third facing to the robot's left (90 degrees to the left of straight ahead). The simulated robot was given a series of waypoints, which were used to generate a path through the maze. Fig. 2a shows the simulated

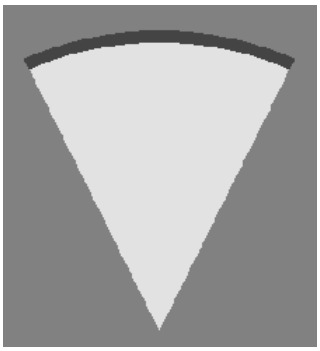


Figure 1: Example result of inverse sensor model for simulated sonar sensor. The sonar sensor is located at the bottom point of the light cone. The lighter region represents cells whose probability of being occupied have been decreased. The darker band at the top of the cone represents cells whose probability of being occupied have been increased. The gray regions outside the cone represent cells whose probabilities of being occupied were unchanged for the given sensor reading.

robot's path through the maze and Fig. 2b shows the occupancy grid that was generated as the robot moved through the maze. In most areas of the maze, especially where the robot was able to travel in a straight path (for example from (x=400, y=400) to (x=400, y=900)), the occupancy grid algorithm was able to localize walls well. However, the occupancy grid also displays some of the challenges in using only sonar sensors: for example, the sensors can have difficulty sensing through small gaps in the maze walls. Also, the perceptual field of the three sonar sensors did not completely cover the area around the robot: in areas of the maze where the robot had to turn quickly, there are some regions of free space that do not have as high probability of being free (they are not as white in the occupancy grid) as other regions of free space. This issue could be improved by having the robot turn more slowly or by adding more sensors to increase the area that the robot could perceive. Additionally, the map does not contain any information about the upper right part of maze, as the robot's path did not allow the sensors to measure any information about that area of the maze. Several frontier regions, areas at the boundary between free and unexplored space, can be seen in the upper right part of the maze. The simulation was useful for testing the occupancy grid algorithm, allowing the effects of using different sensor configurations and speeds at which the robot moved through the maze to be easily tested without complications from motor and encoder noise often associated with robot hardware.

IV. Implementation of occupancy grids on the SuperDroid

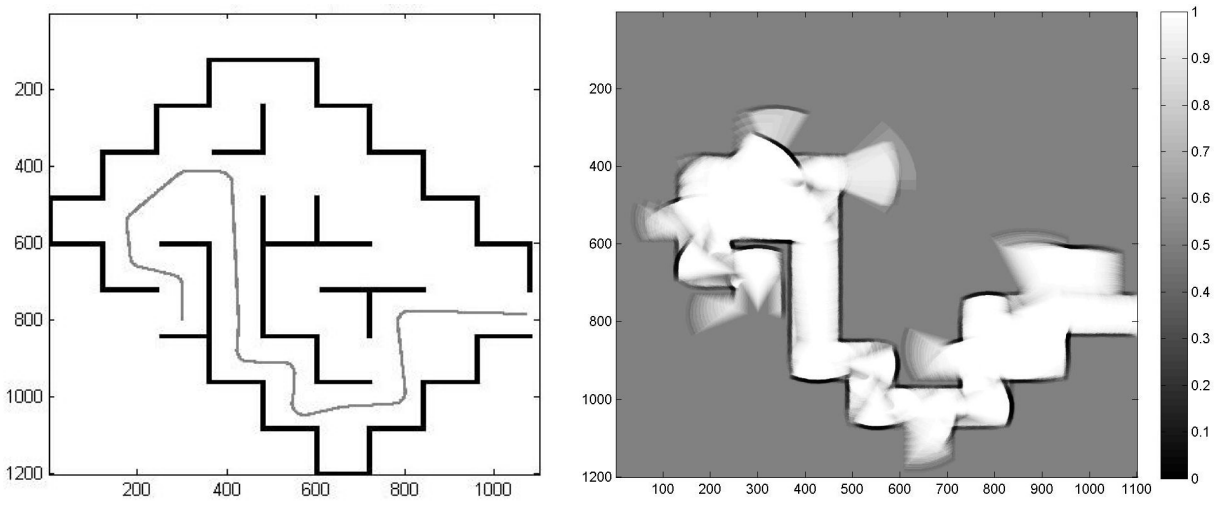
The occupancy grid algorithm has also been implemented within the Cognitive Robotic System on the SuperDroid (Fig. 3). The details of the hardware and software components used in the CRS will be described in this section.

A. Hardware

The hardware used on the SuperDroid is shown in Fig. 4 and includes two BrainStem microcontrollers, an onboard laptop, a motor controller board, and several sensors.

The BrainStem modules are used to communicate with several sensors, the motor driver board, and the onboard PC. Each BrainStem module is a small (2.5" by 2.5") microcontroller board that has a 40 MHz RISC processor (PIC 18C252 microcontroller) with multiple input/output capabilities, including servo motor outputs, analog inputs, digital input/output pins, a serial port, and I2C ports and is capable of running up to four Tiny Embedded Application (TEA, a subset of the C programming language) programs concurrently.

The BrainStem modules are designed to make use of multiple modules simple. Only one BrainStem module was used previously in the CRS, but to accommodate the digital input/output requirements of the quadrature wheel encoders, a second BrainStem module was added to the CRS. The Java program on the PC can communicate with the second BrainStem in the same manner as the first BrainStem, except that it sends messages to and receives messages from a different module address. The first BrainStem module sends information from three sonar distance sensors and two infrared distance sensors to the onboard PC



(a) (b)

Figure 2: The left figure shows the simulated maze environment, with free space as white and obstacles as black. The simulated robot's path through the maze is shown in gray. The right figure shows the cccupancy grid generated as the robot moved through the maze. Areas of lighter color have a higher probability of being free space and areas of darker color are more likely to be occupied. No information about the contents of the gray areas of the map is known.

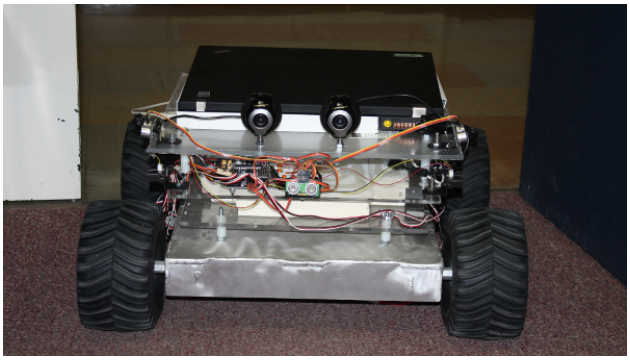


Figure 3: A picture of the SuperDroid robot, shown in an office doorway.

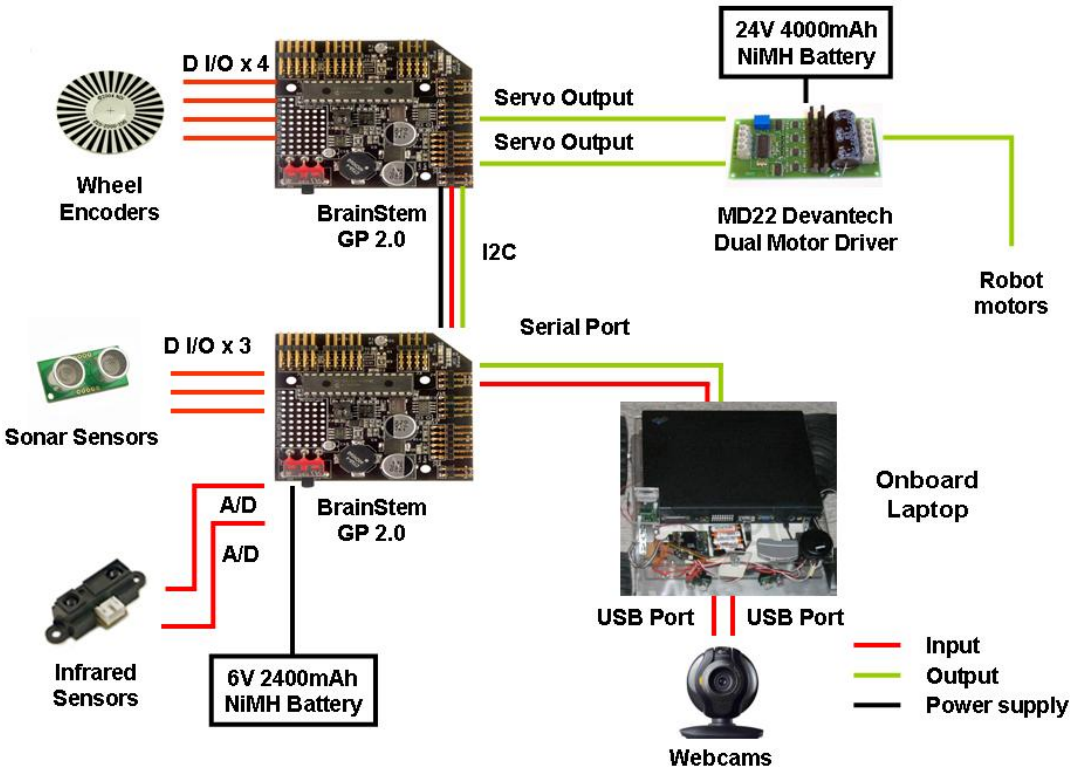


Figure 4: A schematic of the hardware used onboard the SuperDroid and their connections. Inputs to the BrainStem modules and laptop are shown as red lines. Outputs from the laptop, BrainStem modules, and Devantech Motor Driver are shown as green lines. The black lines represent power supplied to the BrainStem module and the Devantech Motor Driver. The second BrainStem GP 2.0 module communicates with and receives power from the first BrainStem GP 2.0 module using its I2C ports.

and acts as a router for messages between the second BrainStem module and the onboard PC. The second BrainStem module senses wheel rotation by monitoring the wheel encoders and uses the information in two ways. First, the encoder information is sent to the onboard PC so that it can be used to estimate the robot's state. Second, it is used, along with the desired motor speeds (sent from the onboard PC), by a motor control program on the second BrainStem module to calculate the motor input needed to achieve the desired speed of the SuperDroid. The addition of the wheel encoders and second BrainStem module allows the Java program on the PC to command the robot to either move with a desired velocity and angular velocity or to complete a translation and rotation.

A Lenovo ThinkPad laptop with an Intel Core2 Duo 2.0 GHz processor and 2 GB of RAM running Windows XP communicates with the webcams and the two BrainStem modules.

The MD22 Motor Controller from Devantech (www.robot-electronics.co.uk) receives a steer command and a throttle command from the servo motor outputs on the second BrainStem module and converts these signals into voltages to control the SuperDroid's motors.

The sensors shown in Figure 4 are described below:

- Incremental quadrature wheel encoders, the Nubotics WheelWatchers (www.nubotics.com), have been installed on the two front wheels of the SuperDroid. Each encoder sends two signals to digital input/output pins on the second BrainStem module. Previously, the SuperDroid had relied on GPS for estimates of its position. The addition of wheel encoders allows the SuperDroid to obtain estimates of its position in environments where GPS does not work (such as indoor environments) and also provides the potential to estimate the position of the SuperDroid in environments where GPS does work by combining information from both GPS and odometry.

- Three sonar sensors, Devantech SRF05 Ultrasonic Rangers (www.robot-electronics.co.uk), can measure the distance to the closest object in its beam pattern with a range of 3 cm to 4 m and send that distance to the BrainStem using the digital I/O pins. These sonar sensor can detect obstacles in about a fifty degree wide cone shaped field of view. One sonar sensor was placed at the front of the robot facing straight ahead. The other two sensors were placed on each side of the robot and face to the robot's sides.

- Two infrared distance sensors (model number GP2Y0A02YK0F) from Sharp were placed on each side of the SuperDroid. These sensors send an analog voltage proportional to the measured distance to the closest obstacle within the sensor's range of 20-150 cm to the BrainStem module's analog input pins. These infrared sensors detect obstacles in a very narrow (about 8 cm wide) rectangular area.

- Two webcams (the Logitech QuickCam Communicate Deluxe, www.logitech.com) have been installed on the SuperDroid. These webcams have 1.3-megapixel sensors and are capable of sending video up to 1280 pixels by 1024 pixels to the PC using USB ports. The cameras have not been used in the current occupancy grid algorithm, but will be in the future.

B. Software for occupancy grid algorithm in the CRS

The occupancy grid algorithm described in section II and tested in simulation was implemented within the CRS using Java. Three sonar and two infrared distance sensors were used to detect obstacles in the SuperDroid's environment and the wheel encoders were used for the odometry calculations that were used to estimate the SuperDroid's position and orientation. A Soar agent with rules about how to navigate in indoor environments (for example, by following walls or following the center of a hallway) was used to command the SuperDroid to move while the CRS collected information needed to build an occupancy grid. While the SuperDroid moves through its environment, cells in the occupancy grid that are within the field of view of the sonar and infrared sensors are updated using Eq. (8), the inverse sensor models for the sonar and infrared sensors, and the SuperDroid's current state estimate.

The inverse sensor models for the sonar and infrared sensors are similar to the model described in section III.C. Figure 5 shows example results from the inverse sensor models for the two sensors. These illustrate the complementary features of the sensors. The sonar sensors are good for sensing free space, but can not tell where the closest obstacle it senses is located within its wide perceptual field. The infrared sensors have a much narrower perceptual field and are useful for sensing openings, such as doors, that can be more narrow than the sonar sensor's perceptual field.

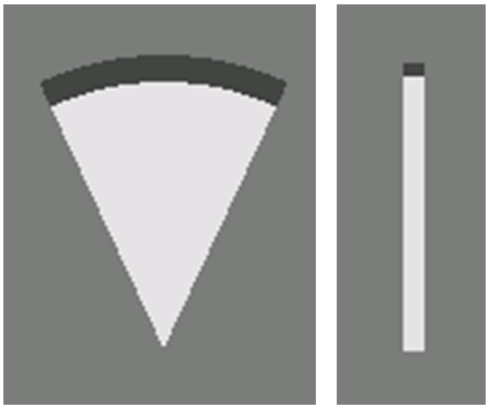


Figure 5: A comparison of IR and sonar perceptual fields.

The results of the inverse sensor models ($\log \frac{p(m_i|z_t,x_t)}{1-p(m_i|z_t,x_t)}$ from Eq. (8)) for the three sonar and two infrared sensors are combined into a single occupancy grid. By using different values of the parameters l_{free} and l_{occ} (described in section II.C), the measurements from the sonar and infrared sensors can be weighted differently. This also allows different measurements from the infrared sensors to be weighted differently: when an infrared sensor returns a measurement equal to its maximum sensor range, this measurement is weighted heavily because it is possible that the measurement represents information that a sonar sensor is not capable of measuring. For example, while a sonar sensor facing a door may return a measurement of the distance to one side of the door because its perceptual field is wider than the door opening, an infrared sensor facing the same door would likely return a measurement equal to its maximum sensor range since its perceptual field is narrower than the door opening. However, if an infrared measurement senses an obstacle at a distance less than its maximum range, this measurement is weighted less heavily than measurements from a sonar sensor because this measurement is likely to be duplicating information from a sonar sensor and infrared measurements have higher standard deviations than sonar measurements.

C. Software for detecting intersections from occupancy grids

Once occupancy grid maps are generated in the CRS, we are interested in using them to detect common types of intersections. Information about the presence of intersections will be useful for generating symbolic information about the environment to be used by Soar agents and could be used to generate topological representations of the environment. Previous work has used a laser rangefinder and logic rules to allow a mobile robot to detect intersections in its environment.[6] Common types of intersections found in indoor environments are shown and labeled in Fig. 6. For example, a T intersection with its top to the right as shown in Fig. 6a, is labeled as Tr.

A fuzzy logic algorithm has been developed that considers information from an occupancy grid and attempts to detect the nine types of intersections from Fig. 6. A virtual rangefinder was developed that uses an occupancy grid as a virtual sensor (similar to Martínez Mozos and Burgard[27]). The virtual rangefinder divides the 180 degrees in front of the rangefinder's position on the occupancy grid into 60 arc regions. Within each of the 60 arc regions, the closest cell that contains an obstacle is located and the distance from the simulated rangefinder to this closest cell is calculated. This process results in 60 ranges, each of which is the distance to the closest obstacle in each arc region. The 60 ranges from the virtual rangefinder are used to calculate an average range in three areas: straight ahead of and to the left and right of the rangefinder's position on the occupancy grid.

The three average ranges are input into a fuzzy logic system that detects the presence of the intersections types shown in Fig. 6. Several membership functions are defined that generate confidence values about the presence of a wall in front of and to the left and right of the rangefinder and classify the degree to which the average ranges to the left and right of the rangefinder are small or large. Various combinations of these fuzzy variables are input into fuzzy set functions to determine the confidence values that a particular type of intersection is present. For example, the confidence value for a dead end intersection (labeled as D in Fig. 6e) is calculated by performing a fuzzy AND on the fuzzy variables that contain the confidence values of

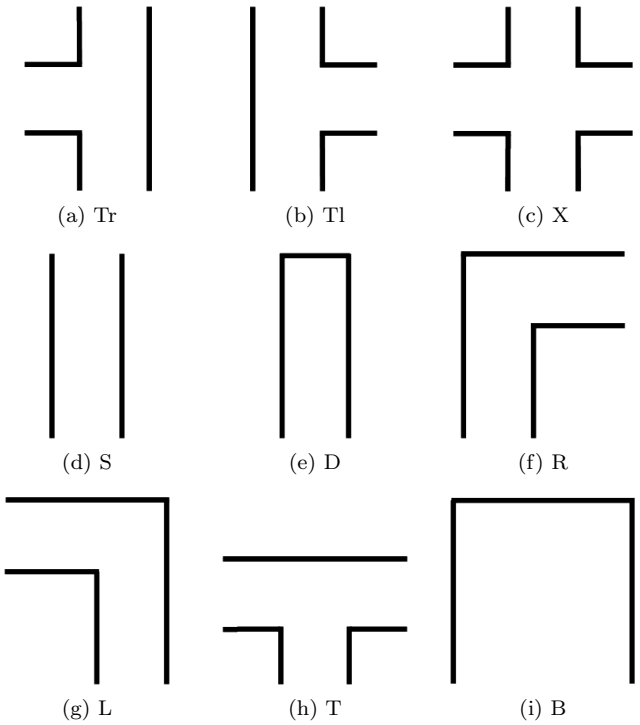


Figure 6: Common types of intersections

there being a wall in front, a wall on the right, and a wall on the left of the virtual rangefinder.

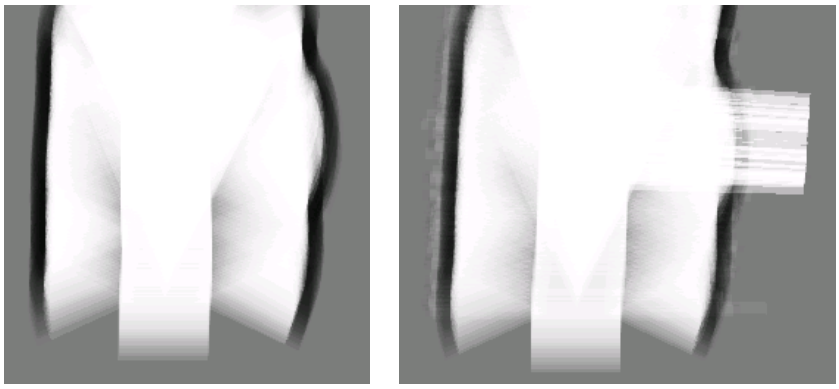
D. Results

To demonstrate the importance of using both sonar and infrared distance sensors to generate our occupancy grids, occupancy grids were created while the SuperDroid navigated down a hallway with an open narrow door to the right. The SuperDroid was in the middle of the hallway with its right sonar and infrared sensors approximately 70 cm from the right wall and open door. Figure 7a shows the occupancy grid generated using only the three sonar sensors on the SuperDroid. The black vertical regions represent the walls of the hallway. The slight bump to the right on the right wall is at the location of the open door. Figure 7b shows the occupancy grid generated using the three sonar sensors and the two infrared sensors on the SuperDroid. In this occupancy grid, the infrared distance sensor on the right of the SuperDroid was able to sense the free space associated with the door and the open door in the right wall is clearly seen. It is also possible to see the effects of noisy infrared distance measurements near the left wall of Fig. 7b.

Figure 8 shows occupancy grids generated of the nine common intersection types (from Fig. 6) using both sonar and infrared sensors. The occupancy grids generated by the CRS are able to capture key features of the intersections.

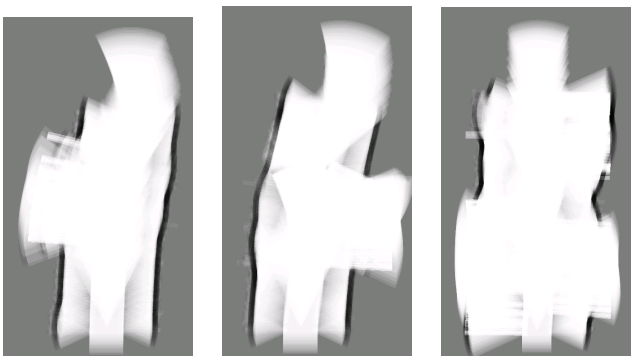
The fuzzy logic algorithm for intersection detection was used for each of the nine occupancy grids shown in Fig. 8. A virtual rangefinder was placed in the center of each intersection, facing towards the top of each occupancy grid. The results of intersection detection for each occupancy grid are given in Table 1. Each row of Table 1 shows an occupancy grid and the confidence value that each type of intersection is present in that occupancy grid. Even in the presence of noisy occupancy grids, the fuzzy logic algorithm is able to assign the correct type of intersection the highest confidence value for each occupancy grid.

There is some evidence of odometry errors in the occupancy grids shown in Fig. 8 and these errors will be more evident as we generate occupancy grids of larger areas. Despite the presence of some error associated with odometry, the occupancy grids were accurate enough for the fuzzy logic intersection detection algorithm to successfully detect the correct intersection types. Map algorithms that take into account uncertainty in robot position (for example, FastSLAM[29] or the continuous localization approach by Schultz, Adams, and Yamauchi[30]) or use additional sensors such as gyroscopes could be used to correct for drift. Alternatively,

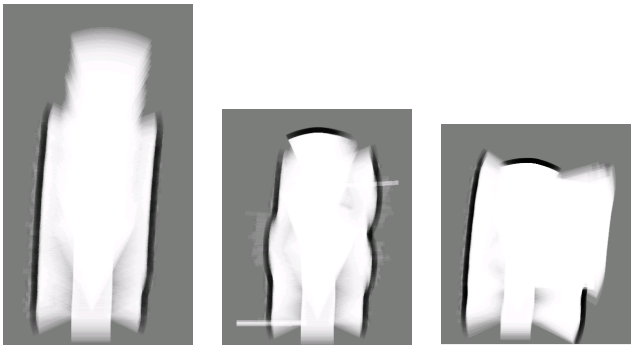


(a) Using only sonar sensors (b) Using sonar and infrared sensors

Figure 7: Occupancy grid maps of a hallway with an open office door in the right wall



(a) Tr (b) Tl (c) X

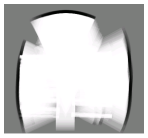


(d) S (e) D (f) R

(g) L (h) T (i) B

Figure 8: Occupancy grid maps of different intersection types

Table 1: Confidence values for the presence of common intersection types, detected from occupancy grids

Occupancy grid	Confidence values for each intersection type								
	R	L	S	Tr	Tl	T	X	D	B
	0.94	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.00
	0.00	0.75	0.00	0.03	0.00	0.25	0.00	0.00	0.00
	0.00	0.00	0.86	0.00	0.00	0.00	0.00	0.14	0.00
	0.00	0.00	0.00	0.73	0.00	0.00	0.00	0.00	0.00
	0.00	0.00	0.00	0.00	0.72	0.00	0.00	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.74	0.00	0.00	0.26
	0.00	0.00	0.00	0.05	0.00	0.00	0.55	0.00	0.00
	0.03	0.03	0.21	0.00	0.03	0.00	0.00	0.80	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.95

the creation of topological maps using the intersection detection capabilities described above could be used to connect local metric maps, minimizing the accumulation of odometry errors.[24]

To generate the results shown in Table 1, the intersection detection was performed after the occupancy grid was generated. However, it will be more useful to be able to detect intersections as the SuperDroid is generating the occupancy grid. The presence and location of walls in front of the robot is an important cue for intersection detection, but this information is most accurate after the robot is able to sense these walls with sonar and infrared sensors located at its sides. To detect intersections using occupancy grids as they are generated with the current sensor configuration, it may be most useful to perform intersection detection with a virtual rangefinder that has a position behind the robot to take advantage of an occupancy grid that has integrated information from all of the robot's sensors. Also, additional sensors such as a laser rangefinder or stereo cameras could be used along with the current sonar and infrared sensors to generate occupancy grids that can more accurately localize obstacles in front of the robot.

V. Summary

This paper has described the initial integration of occupancy grids in the Cognitive Robotic System. A simulation used to help develop and test the occupancy grid algorithm and the hardware components used to develop occupancy grids in the CRS were described. Occupancy grids were generated for several common types of intersections and a system that is able to detect these types of common intersections using occupancy grids and fuzzy logic has been developed and tested. In the future, the incorporation of stereo vision data into the occupancy grid algorithm to complement information available from sonar and infrared distance sensors has the potential to enhance the quality of the occupancy grids.

The map generating and intersection detection capabilities described in this paper should be useful for applying the CRS to the mission of search within an unmapped building for specific objects. The ability to detect common types of intersections will permit the generation of symbolic information and topological representations of the environment. This symbolic and topological information will allow Soar agents to interact with and reason about their environment in more interesting and useful ways. Additionally, the generation of both metric (occupancy grids) and topological (intersection types) map information could be used to create hybrid representations of the robot's environment that leverage the strengths of each representation.

Acknowledgments

The authors thank Gregory L. Sinsley for his help with the sensor model for the simulation and with the fuzzy logic algorithm for intersection detection.

References

[1]Hanford, S. D., Janrathitikarn, O., and Long, L. N., "Control of Mobile Robots Using the Soar Cognitive Architecture," *Journal of Aerospace Computing, Information, and Communication*, Vol. 6, No. 2, 2009.

[2]Trafton, J. G., Schultz, A. C., Cassimatis, N. L., Hiatt, L. M., Perzanowski, D., Brock, D. P., Bugajska, M., and Adams, W., "Communicating and Collaborating with Robotic Agents," *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation*, edited by R. Sun, Cambridge University Press, Cambridge, 2006, pp. 252–278.

[3]Kennedy, W., Bugajska, M., Marge, M., Adams, W., Fransen, B., Perzanowski, D., Schultz, A., and Trafton, J., "Spatial Representation and Reasoning for Human-Robot Collaboration," *Proceedings of the Twenty-Second Conference on Artificial Intelligence*, AAAI, 2007, pp. 1554–1559.

[4]Kelley, T. D., "Developing a Psychologically Inspired Cognitive Architecture for Robotic Control: The Symbolic and Subsymbolic Robotic Intelligence Control System (SS-RICS)," *International Journal of Advanced Robotic Systems*, Vol. 3, No. 3, 2006, pp. 219–222.

[5]Kelley, T. D., "Using a Cognitive Architecture to Solve Simultaneous Localization and Mapping (SLAM) Problems," ARL-MR-0639, Aberdeen Proving Ground, Aberdeen, MD, May 2006.

[6]Kelley, T. D., Avery, E., Long, L. N., and Dimperio, E., "A Hybrid Symbolic and Sub-Symbolic Intelligent System for Mobile Robots," *AIAA InfoTech@Aerospace Conference*, AIAA, Wahington, DC, 2009, AIAA Paper No. 2009-1976.

[7]Benjamin, P., Lyons, D., and Lonsdale, "Designing a Robot Cognitive Architecture with Concurrency and Active Perception," *Proceedings of the AAAI Fall Symposium on the Intersection of Cognitive Science and Robotics*, AAAI, October 2004.

[8]Laird, J., Newell, A., and Rosenbloom, P., "Soar: An Architecture for General Intelligence," *Artificial Intelligence*, Vol. 33, No. 3, 1987, pp. 1–64.

[9]Jones, R., Laird, J., Nielsen, R., Coulter, K., Kenny, R., and Koss, F., "Automated Intelligent Pilots for Combat Flight Simulation," *AI Magazine*, Spring 1999, pp. 27–41.

[10]Forgy, C., "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artificial Intelligence*, Vol. 19, No. 1, 1982, pp. 17–37.

[11]Gupta, A. and Long, L. N., "Hebbian Learning with Winner Take All for Spiking Neural Networks," *Proceedings of the 2009 IEEE International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2009.

[12]Kuo, B. C. and Golnaraghi, F., *Automatic Control Systems*, John Wiley and Sons, Inc., Hoboken, NJ, 8th ed., 2003.

[13]Simon, D., *Optimal State Estimation: Kalman, H_∞, and Nonlinear Approaches*, John Wiley and Sons, Inc., Hoboken, NJ, 2006.

[14]Lowe, D., "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, Vol. 60, No. 2, 2004, pp. 91–110.

[15]ThreePenny Software LLC., *SML Quick Start Guide*, 2005.

[16]Long, L. N. and Kelley, T. D., "A Review of Consciousness and the Possibility of Conscious Robots," *Journal of Aerospace Computing, Information, and Communication*, Vol. 7, No. 2, 2010.

[17]Balaguer, B., Balakirsky, S., Carpin, S., and Visser, A., "Evaluating maps produced by urban search and rescue robots: lessons learned from RoboCup," *Autonomous Robots*, 2009, published online August 28, 2009.

[18]Carbone, C., Finzi, A., Orlandini, A., and Pirri, F., "Model-based control architecture for attentive robots in rescue scenarios," *Autonomous Robots*, Vol. 24, No. 1, 2008, pp. 87–120.

[19]Gossow, D., Pellenz, J., and Paulus, D., "Danger Sign Detection Using Color Histograms and SURF Matching," *Proceedings of the 2008 IEEE International Workshop on Safety, Security and Rescue Robotics*, IEEE, October 2008.

[20]Bekey, G., *Autonomous Robots: From Biological Inspiration to Implementation and Control*, MIT Press, Cambridge, MA, 2005.

[21]Thrun, S., "Simultaneous Localization and Mapping," *Robot and Cognitive Approaches to Spatial Mapping*, edited by M. E. Jefferies and W. K. Yeap, Springer, Berlin/Heidelberg, 2008, pp. 13–41.

[22]Moravec, H. and Elfes, A., "High-resolution maps from wide-angle sonar," *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, IEEE, 1985, pp. 116–121.

[23]Thrun, S., "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, Vol. 99, No. 1, 1998, pp. 21–71.

[24]Tomatis, N., "Hybrid, Metric-Topological Representation for Localization and Mapping," *Robot and Cognitive Approaches to Spatial Mapping*, edited by M. E. Jefferies and W. K. Yeap, Springer, Berlin/Heidelberg, 2008, pp. 43–63.

[25]Borenstein, J. and Koren, Y., "The Vector Field Histogram-Fast Obstacle Avoidance for Mobile Robots," *IEEE Transactions on Robotics and Automation*, Vol. 7, No. 3, 1991, pp. 278–288.

[26]Galindo, C., Saffiotti, A., Coradeschi, S., Buschka, P., Fernandez-Madrigal, J., and Gonzalez, J., "Multi-hierarchical semantic maps for mobile robotics," *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2005.

[27]Martínez Mozos, O. and Burgard, W., "Supervised Learning of Topological Maps using Semantic Information Extracted from Range Data," *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2006.

[28]Thrun, S., Bugard, W., and Fox, D., *Probabilistic Robotics*, MIT Press, 2006.

[29]Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B., "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem," *Proceedings of the AAAI National Conference on Artificial Intelligence*, AAAI, 2002.

[30]Schultz, A., Adams, W., and Yamauchi, B., "Integrating exploration, localization, navigation, and planning with a common representation," *Autonomous Robots*, Vol. 6, 1999, pp. 293–308.