*World Automation Congress © 2010 TSI Press.*

# RECURSIVE, HYPERSPHERICAL BEHAVIORAL LEARNING FOR ROBOTIC CONTROL

**SALYER B. REED, TYSON R. C. REED, MONICA NICOLESCU, SERGIU M. DASCALU**
*Department of Computer Science and Engineering*
*University of Nevada, Reno*
*Reno, Nevada 89557, USA*
*http://www.cse.unr.edu*

ABSTRACT— Robots, undoubtedly, are governed by a set of behavioral policies. However, embedding these policies becomes problematic and complex due to the nondeterministic properties of the task and environment. Learning from demonstration, or LFD, alleviates this vexatious conundrum and expedites the mapping process, for the robot implicitly learns the desired objective. This paper presents a novel method for facilitating behavior learning in robots. The algorithm employed, called Recursive, Hyperspherical Behavioral Learning, or RHBL, actively translates the teacher's reactions to various stimuli into a behavioral tree, which defines the robot's current domain knowledge. Once the tree is formulated, the anthropomorphic robot demonstrates proficiency in the observed, complex task, for the tree elicits responses from various stimuli, defining the robot's autonomous behavior. Details of the algorithm and the results of its application are presented in the paper.

Key Words:RHBL; Recursive, Hyperspherical Behavioral Learning; Learning From Demonstration; Robot

## 1. INTRODUCTION

Robot behaviors are increasingly complex and sophisticated, resulting in new, comprehensive strategies and algorithms, endeavoring to increase utility, realism, or functionality. Initially, the architect, or designer, formulates a list of objectives; this involves identifying and characterizing the indicated responses. Ultimately, this behavior schema is imprinted onto the robot. However, transferring this schema, or control policy, from the architect to the robot traditionally yields suboptimal solutions due to an inadequate understanding of the situation, the difficulty of the problem, or time constraints. As such, to accelerate learning, architects leverage cooperative learning techniques and strategies, including learning from demonstration (LFD).

Presently, a multitude of learning techniques exists such as reinforcement learning [5, 8], unsupervised learning [6, 14], and supervised learning [4, 7]. These learning techniques – when coupled with strong, fast algorithms – have produced a plethora of behaviors utilized by many robots. On the other hand, learning from demonstration is, transparently, unique.

Learning from demonstration is a natural, intrinsic behavior for any entity, or being. The persistent exposure to various stimuli yields observable, and often measurable, responses from the entity. By observing the reactions, other entities can infer proper responses to the stimuli, constructing a set of governing, behavioral policies. As such, assimilation of the behavior is assumed to be implicit rather than explicitly defined by the architect [10].

Ideally, learning from demonstration involves two, distinct parties: the teacher and the pupil. The teacher, which is customarily a human being, possesses distinct domain knowledge pertaining to an individual task. It is the objective of the teacher to demonstrate a task to the pupil, articulating proper behavior to various stimuli, and it is the intention of the pupil is to observe and learn. During this process, the teacher must establish a rapport with the pupil, for the pupil fully entrusts the teacher. Eventually, by observing the behavior and the intent, the pupil is able to expand its repertoire by constructing an internal schema, or representation.

During training, the teacher executes a meaningful task, which is observed by the pupil. It is the objective of the pupil to map the intentions of the teacher to meaningful behaviors and actions.

*Recursive, Hyperspherical Behavioral Learning* (RHBL) is a LFD technique derived from the *Recursive Hyperspheric Classification* (RHC) Algorithm [12]. Fundamentally, RHC is a classification algorithm used in identifying and assigning classes to unknown feature vectors by strategically constructing a hierarchical taxonomy of hyperspheres from labeled data. RHBL, like its predecessor, generates a similar

hierarchy of hyperspheres. In this structure, each hypersphere maps to a primitive action, which has previously been branded onto the robot.

When a demonstration is performed, it can be considered a collection of labeled feature vectors. As such, because each feature vector is labeled, the robot can construct a behavior policy utilizing various classification techniques, including RHBL. When learning a demonstrated task using the RHBL algorithm, the robot constructs a set of overlapping hyperspheres in a n-dimensional space from labeled feature vectors, which are gathered through observation. After mapping, or partitioning, the dimensional space, the robot is said to be "learned;" that is, the control policy is embedded in the robot.

Following the training process, the robot "goes online" and is placed in the environment. During this period, the hierarchical spheres, when recursively traversed, dictate the robot's responses to unlabeled vectors, which are extracted from the environment, as they are presented to the robot. In classifying the unlabeled vector, a label, or response, is inferred from the current domain knowledge, which is encapsulated by the collection of hyperspheres. Ultimately, the RHBL algorithm will ensure fast responses to stimuli as well as believable reactions from environmental forces.

RHBL has been used to create an autonomous robot capable of performing a straightforward collection of tasks described by the architect. The robot, once it has learned the proper behavior through observation, is able to perform a desired task without intervention by the overseer. In this paper, a Pioneer 3DX robot is endowed with the ability to classify unlabeled vectors by means of the RHBL algorithm. In training the robot, the robot's sensors will be utilized, which will tell the robot of its current state. In the following arrangement, the robot is first demonstrated a specific task; afterwards, the robot's understanding is affirmed when the robot performs the desired task. Should additional examples, or vectors, be needed to correct a behavioral deficiency, the robot, again, observes a demonstration and appends and augments its current hierarchy of hyperspheres; as such, the RHBL algorithm is extensible.

The remaining sections describe the RHBL algorithm and its applications. Section 2 discusses related work in the realm of LFD. In Sections 3, 4, and 5 the RHBL algorithm is discussed in detail, including the characteristics, production of hyperspheres, and the classification process. Section 6 describes the training processes and results obtained from the robot that utilizes the RHBL algorithm. Finally, a brief discussion about the RRHL algorithm is included in Section 7, followed by conclusions and future work in Section 8.

## 2. RELATED WORK

A problem plaguing the robotic community is the excessive number of behaviors, or responses, a robot must possess. Many of these behaviors – learned or imparted – are utilized when the robot goes online; however, many unforeseen scenarios exist and creating behaviors for every minute scenario is fruitless. As such, learning from demonstration can expedite the learning process and assist in substantiating and formulating solutions for problematic scenarios.

In many learning techniques, the robot is endowed with a set of primitive actions. These actions, when combined, will form complex behaviors, or a policy. This policy – once compiled – dictates the robot's responses to stimuli in the environment. In [2], the authors create such an action policy by actively controlling the robot; the robot is a passive observer. Ultimately, the policy is the set of Gaussian mixture models for each primitive behavior where each Gaussian probability distribution function is weighted. The mixture model actively governs the behavior of the robot by deterministically selecting the primitive action.

On the other hand, state space boundaries are not always linearly separable; labeled data often overlap. As such, utilizing support vector machines and optimizing a Lagrangian function, one is able to classify unlabeled data [9, 13]. However, learning from demonstration can also assist by removing ambiguity and defining the appropriate action. In [3], a robot performs a specific action with respect to its current state and domain knowledge. However, because the space is not linearly separable, actions – or labels – overlap; as such, the authors create option classes, which are more likely to select actions that have been previously demonstrated.

Also, a strong proponent supporting LFD is the ability to generalize. In [11], the authors utilize behavior networks and create a generalized topology, endeavoring to accomplish a desired task. Demonstrating the task to the robot, the teacher provides audio cues to the robot, and the robot creates a behavior network, or graph. From the graph, a generalized topology is produced. In performing the desired task, should the teacher deem the action inappropriate or incomplete, the teacher will intervene, correcting the robot, indicating the correct response. As such, the topology is extensible, for it can be appended.

Finally, in [10], behaviors are defined to be a linear combination of behaviors and weights. Demonstrating an action to a robot, the robot, by utilizing a particle filter, ascertains the weights. Once estimated, a resultant vector is molded from the combination of weights and predictive vectors. Ultimately, the advantage is that the robot assumes that behaviors run concurrently and the observed action is a composite of these synchronous behaviors, for the robot is striving to estimate the behavior by modifying the composite weights.

## 3. THE ALGORITHM

In demonstrating these behaviors, direct teleoperation of the robot is employed. In this mode the teacher controls the robot, and the robot maintains a passive role in the interaction. However, during this process, the robot observes the task, creating a behavioral tree of the observed actions.

### *3.1 Algorithm Terminology*

The hypersphere is the quintessential geometric entity utilized in RHBL. Every hypersphere innately possesses two geometric properties: a center of gravity, or COG, and a radius. The radius and COG define the locus for all points in the space that compose the hypersphere. Moreover, every hypersphere carries two additional properties. First, every hypersphere possess a label; the label corresponds to an element in the primitive behavior in the set. Second, the hypersphere maintains a set of spawned hyperspheres. During training the hypersphere will – possibly – spawn additional hyperspheres; as such, it maintains a list of the spawned children. This spawning process generates a hierarchical structure of hyperspheres, or a tree.

### *3.2 Initialization*

Creating complex behaviors from local primitives involves spawning hyperspheres from an initial hypersphere and monitoring the teacher's responses to stimuli with the aid of sensors. In many instances, ranges for sensors are crisp; specifically, they have definitive operating values. As the limits are known, one may scale the values, defining the boundary of the action space. In creating hyperspheres, during the demonstration, sensor values are sampled and merged, creating an input vector, which is then presented to the system. When introduced to the system, the vector – potentially – can spawn many new hyperspheres.

### *3.3 Creating the First Hypersphere*

If the space can be scaled in the range [0, 1], the first hypersphere, intrinsically, is positioned in the center of the space. As such, the center of gravity for the hypersphere in $\mathbb{R}^n$ is determined to be:

$$\overline{COG} = \left[0.5_1, 0.5_2, 0.5_3, \cdots 0.5_{n-1}, 0.5_n\right]$$

where $\overline{COG}$ is the center of gravity, and n is the dimensionality of the space. Also, the radius of this first hypersphere, too, should be large enough to encompass all vectors that are introduced to the system. Therefore, if all vectors are scaled, the radius is distance from the COG to the furthest possible vector, for no vector can exceed the boundaries of the hypersphere [Figure 1].

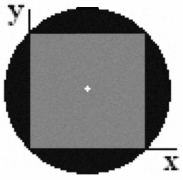


**Figure 1. Creating a radius about the first COG in a two-dimensional space. The gray depicts the dimensional boundaries. Some parts of the hypersphere, which are black, are void and cannot contain a valid vector. The white cross is the COG. The axes are shown.**

Finally, a class designation, or default behavior, must be assigned to the hypersphere. The default behavior of the hypersphere is, in fact, a primitive action from the set $a \in A$, where $A$ is the set of all primitive actions.

### *3.4 Creating Additional Hyperspheres*

As was mentioned, spawning hyperspheres creates a taxonomic hierarchy of behaviors. This tree, when traversed, is capable of producing complex behaviors. To spawn additional hyperspheres the robot surveys the current environment by monitoring sensor inputs that produce quantitative readings. The amassed sensor data produces a solitary input vector, which is scaled. The label for the vector is the observed reaction from the teacher to environment stimuli.

Starting with the root hypersphere, or node, the algorithm parses the tree using standard tree traversal algorithms. For every hypersphere that can encapsulate the input vector, RHBL continues to parse its children. If, in the graph search, a hypersphere has no children that can encapsulate the vector, and a confliction between classes, or observed behaviors, is sensed, a new hypersphere is spawned.

Unlike the first hypersphere, the center of gravity for a spawned child is, in fact, the input vector. However, the radius of the new hypersphere is defined to be:

$$r_{child} = r_{parent} - \left\| COG_{child} - COG_{parent} \right\|^2,$$

where $r_{child}$ is the radius of the spawned child, $r_{parent}$ is the radius of the parent, and $\|COG_{child} - COG_{parent}\|^2$ is the Squared Euclidean Distance between the center of gravities of the two hyperspheres. Finally, the spawned hypersphere also assumes the behavior, or class, of the input vector, which mimics the action of the teacher. It is this spawning process that truly extends the behavior of the robot, creating complex behaviors.

## 4. COMPLEX BEHAVIORS

The structure of the hyperspheric tree, a collection of primitive actions, creates complex behaviors from a rather mundane set of actions. When the robot is unleashed in an environment, it utilizes standard tree traversal algorithms to select a primitive behavior from the tree nodes.

### *4.1 Behavioral Selection*

The current schema assumes the robot receives quasi-, or near-, perfect information from the sensors; that is, the sensors contain very little noise. From the sensors, a scaled vector is created and is introduced to the system for identification.

Tree traversal algorithms are employed that examine the hierarchical structure of the tree and selects the appropriate behavior given the current state of the sensors. Starting with the first hypersphere, or root, the system compiles a list of hyperspheres, or nodes, that can encircle, or describe, the unlabeled vector. If a hypersphere can encompass the vector, the system inspects that hypersphere's descendants, determining if they, too, can encapsulate the vector. As such, if a child encapsulates the vector, continue performing a graph search on the node and its descendants. When probing a sphere's children and no child can enclose the vector, that sphere is marked as a descriptive candidate. On the other hand, for each hypersphere that cannot enclose the vector, prune that child and its direct descendants.

When the search is complete, the system selects from the potential candidates the sphere with the smallest radius, for this sphere is most descriptive. The robot then performs the primitive action associated with this hypersphere. Ultimately, as the input vector varies due to the dynamic nature of the robot and its environment, new primitive actions are selected from the behavioral tree; the complex behavior of the robot is, in fact, the aggregate of all the primitive behaviors in the behavioral tree.

## 5. SQUARED EUCLIDEAN DISTANCE

It was mentioned that RHBL utilizes the Squared Euclidean distance when spawning the set of hyperspheres. In fact, many distances could be utilized in spawning additional hyperspheres; however, using the squared Euclidean distance has a distinct advantage.

As opposed to the Euclidean distance, the squared Euclidean distance has a smaller contraction rate when spawning. For example, consider the points (0, 0) and (5, 0). The Euclidean distance is 5 units, and the squared Euclidean distance is 25 units. Now, if these distances are the radii, and another vector is found at (2.5, 0), then the contraction rate for the Euclidean distance is 50% while the squared Euclidean distance contracts by just 25%. The smaller contraction rate leads to better classification results, for the volume that is truncated in the space is decreased, reducing overshoot.

### *5.2 Convexity*

To use the squared Euclidean distance in classification, however, one must first prove that this set is, in fact, convex. Assume that $x$ and $y$ are vectors in $\mathbb{R}^n$ and $a \in [0, 1]$, then…

$$g(a) = \sum_{i=1}^{n} [ax_i + (1-a)y_i]^2 \leq \max\left\{\sum_{i=1}^{n} x_i^{\,2}, \sum_{i=1}^{n} y_i^{\,2}\right\}.$$

In other words, the following set must be convex.

$$B_r = \left\{x \in R^n : \sum_{i=1}^{n} x_i^{\,2} \leq r\right\}$$

**Proof**: It is noted that…

$$0 \leq \sum_{i=1}^{n} (x_i - y_i)^2 = \sum_{i=1}^{n} \left(x_i^{\,2} + y_i^{\,2} - 2x_i y_i\right)$$

$$= \sum_{i=1}^{n} x_i^{\,2} + \sum_{i=1}^{n} y_i^{\,2} - 2\sum_{i=1}^{n} x_i y_i$$

for the sum of nonnegative numbers is, indeed, nonnegative, and any number squared is nonnegative. Next,

$$g(a) = \sum_{i=1}^{n} [ax_i + (1-a)y_i]^2$$

$$g(a) = \sum_{i=1}^{n} \left[a^2 x_i^{\,2} + (1-a)^2 y_i^{\,2} + 2a(1-a)x_i y_i\right]$$

$$g(a) = a^2 \sum_{i=1}^{n} \left[x_i^{\,2} + y_i^{\,2} - 2x_i y_i\right] + 2a\sum_{i=1}^{n} \left[x_i y_i - y_i^{\,2}\right] + \sum_{i=1}^{n} y_i^{\,2}$$

From this, it is true that g is a second degree polynomial with respect to $a$, and $g$ is, certainly, concave up, for the coefficient preceding $a^2$ is positive. Simple Calculus dictates that the minima and maxima occur at points where derivatives are equal to zero or the endpoints of the interval. By definition, if the minimum occurs at one endpoint, the maximum must fall on the other endpoint. On the other hand, if the minimum falls within the interval, the maximum occurs at either endpoint. As such, either…

$$g(1) = \sum_{i=1}^{n} x_i^{\,2} \quad or \quad g(0) = \sum_{i=1}^{n} y_i^{\,2}$$

will be the maximum of $g$, and the proof is complete, showing that the Squared Euclidean distance is, actually, convex.

## 6. ROBOT

The robot was developed in Player, which is an open source program used in creating robots with robust behaviors and dynamic environments. The robot possesses a solitary laser rangefinder. The laser rangefinder has 360 laser "pings" in a 180° arc. Of the 360 laser pings, five were used in the creating the complex hierarchy of hyperspheres [Figure 2].

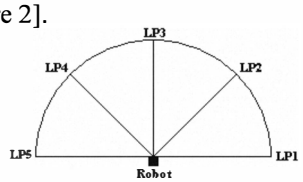


**Figure 2. A depiction of the robot with the five laser pings used in training and simulation.**

The robot was also endowed with a position sensor. This sensor enables the robot to determine its position and orientation with respect to the environment. In developing the behaviors, only the rotational measurement of the position sensor was used.

### *6.1 Primitive Behaviors*

In developing behaviors, the robot possesses a small, discrete set of behaviors, including forward, forward-left, rotate-left, forward-right, and rotate-right.

$$A = \{\text{Fwd, Fwd-Left, Rot-Left, Fwd-Right, Rot-Right}\}$$

It is the objective of RHBL to create complex behaviors from this set of primitive behaviors. One should note that at no time can the robot remain stationary, for the robot would never leave this state in a static environment.

## 7. EXPERIMENTAL RESULTS

The robot learned a plethora of behaviors, including wall following and maze navigation. During wall following, the robot maintains a desired distance from the wall, following the contours and jagged geometry of the environment yet avoiding obstacles. In maze navigating, the robot maneuvers in a simplified maze, endeavoring to be free of the confining walls, or screens.

### *7.1 Wall Following*

In wall following, it is the intent of the robot to actively pilot a course, maintaining close proximity to a wall yet avoiding direct collision with the wall or other obtrusive obstacles. Initially, during this task, the robot only utilized the laser rangefinder. Consequently, as the pings from the rangefinder can reach, or sense, very far distances, a threshold was applied to each ping. In this scenario, each ping was constrained to 2.0 units. If the ping exceeded this threshold, it was tapered to 2.0 units. The readings from the pings were then scaled by a factor of 2.0, for, as stated, inputs are scaled in the range [0, 1].

Because five pings were used to create an input vector, the initial hypersphere is located at [0.5, 0.5, 0.5, 0.5, 0.5, 0.5]. This implies the radius is 1.25 units, for it is large enough to encompass all foreseeable vectors. Finally, the behavior, or label, of the hypersphere is assigned to be "forward."

Once the first hypersphere is initialized, the experiment is started, and the teacher governs the robot's movements. While navigating, the robot reads the values of the five laser pings, creates a labeled feature vector, scales this vector, and sends it to the system for classification. The system will potentially spawn one or more children from this vector.

Initially, the robot's trajectory was parallel with the wall; however, because of the wall's protrusions, the robot would inevitably collide with the wall unless a corrective action were performed by the teacher. As the robot grew closer to the wall, the observer's reaction to the impending wall collision instructed the algorithm to create a new hypersphere, averting a collision. On the other hand, if the robot retreated, or ventured, too far from a wall, again, the teacher's reaction spawned a hypersphere, indicating the robot should turn toward the wall.

**Figure 3. (a) The path taken by the teacher during wall training. (b) The path taken by the robot in the environment after learning the desired behavior.**

### *7.2 Maze Navigating*

In maze navigating, the robot traversed a simplified, tiered maze. At each tier there is a single inlet and a single outlet. The robot is to navigate each tier and reach the goal while avoiding wall collisions and previously visited tiers.

Like wall following, the robot employs the use of its laser rangefinder, but, during maze navigating, it also utilizes its position sensor. The position sensor is used to determine the robot's orientation. However, unlike a laser ping where the value is continuous, the orientation of the robot is considered discrete because the value can assume only two values: left-directional or right-directional. It is important to determine the

directional orientation of the robot as the robot may inadvertently revisit a tier. Adding this additional feature to the vector, coupled with proper training, will prevent this scenario. Subsequently, the discrete values left-directional and right-directional are assigned the numeric values 0.33 and 0.66, respectively, for the values are spaced equidistance on the interval [0, 1].

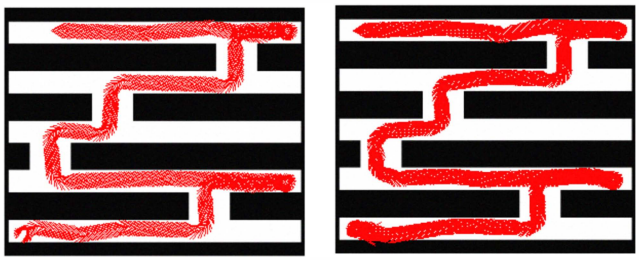

**Figure 4. (a) The path taken by the teacher during maze training. (b) The path taken by the robot in the maze after learning the desired behavior.**

### *7.3 Discussion*

In both scenarios the robot was able to traverse the environment after implicitly learning the task by monitoring the reactions of the teacher. In the first scenario, wall following, the robot easily navigated the course with no additional training. On the other hand, in one instance the robot did stumble and entered an infinite loop. The robot continually bounced between two hyperspheres, or states, that were labelled rotate-left and rotate-right. In this case, user intervention was needed by demonstrating an action that could put the robot in a new state, which altered the behavioral tree. Once altered, the scenario was reset, and the robot could successfully pilot the course. Lastly, the average node count of the spawned tree was 53.4, and the average tree height was 12.

In the second scenario, maze navigating, the robot – on all attempts – successfully navigated the terrain after just one training session. The success of the robot can be attributed to the many vectors introduced to the robot for classification, for the terrain has many tiers. Moreover, the average node count of the generated tree was 412.5 and had an average tree height of 13.

**Table I. Spawned tree properties.**

| Scenario | Average Node Count | Average Tree Height |
|---|---|---|
| Wall Following | 53.4 | 12 |
| Maze Navigating | 412.5 | 13 |

## 8. FUTURE WORK AND CONCLUSIONS

### *8.1 Future Work*

Currently, learning from demonstration is the instrument employed in learning behaviors. Future modifications are likely to utilize reinforcement learning with the aid of fitness functions. With a fitness function, the behavior will be learned by inputting the vector into a fitness function. It will then be the fitness function that determines if the learned behavior is acceptable. As such, the role of the teacher would diminish; in fact, the role of the teacher would be reduced to correcting erroneous actions.

Also, presently, RHBL has no conception of state; as such, the algorithm is purely reactive; that is, it cannot solve a temporal set of goals. For example, if the goal is for the robot to first pick up a red ball and then a green ball, RHBL would fail. In many architectures and algorithms, robots maintain an internal state [1]; each state dictates and regulates the instantaneous action of the robot. Improved, future alterations to the RHBL algorithm would allow for the system to emulate a state machine, allowing the robot to achieve a set of sequential goals.

### *8.2 Conclusions*

Because of the nature of RHBL – like many classification algorithms – a multitude of samples, or labeled vectors, must be presented to the system; this explains the rationale for RHBL continually

monitoring the reactions of the teacher during training. RHBL is attempting to discover the separable boundaries between classes, or actions, by methodically partitioning the space into hyperspheres, allowing it to quickly adapt and learn the desired behavior. Once divided, the algorithm exhibits unparalleled prowess in classifying unlabeled vectors.

Moreover, RHBL is a strong, yet simplistic, classification algorithm that – when harnessed – invokes legitimate responses to environmental forces. Initially, when observing the actions of the teacher, the robot actively parses its current domain knowledge, which is comprised of hyperspheres. If the action of the teacher disagrees with the action, or class, of the hypersphere, a new hypersphere is spawned, which augments and enhances the behavior of the robot, creating a hierarchy of hyperspheres.

Following the construction of the behavioral tree, the robot is considered to be "learned," and its perceived behavior is affirmed by examining its responses to various stimuli. Sampling its sensors, the robot produces an input vector and proposes it to the system for classification. Standard tree traversal algorithms are employed, and a list of possible candidates are gathered. From the list, RHBL selects the hypersphere with the smallest radius because this hypersphere is the most descriptive, and the robot executes the primitive action. It is the conglomeration and traversal of hyperspheres that defines the robot's complex behaviors.

Finally, RHBL, too, is extensible. Should the learned behavior of the robot warrant a refinement, the robot can observe additional demonstrations, broadening and honing its internal schema of the demonstrated behavior, for the additional examples also solidify state boundaries as a result of spawning additional hyperspheres.

## REFERENCES


1. B. Argall, B. Browning, and M. Veloso, "Automatic Weight Learning for Multiple Data Sources when Learning from Demonstration," *IEEE International Conference on Robotics and Automation.* Kobe, 2009, pp. 226 – 31.
2. S. Chernova and M. Veloso, "Confidence-Based Policy Learning from Demonstration Using Gaussian Mixture Models," *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*. 2007, pp. 1 – 8.
3. S. Chernova and M. Veloso, "Learning Equivalent Action Choices from Demonstration," *IEEE International Conference on Intelligence Robots and Systems*. Nice, 2008, pp. 1216 – 1221.
4. M. J. Er and C. Deng, "Obstacle Avoidance of a Mobile Robot Using Hybrid Learning Approach," *IEEE Transactions on Industrial Electronics*. 2005, pp. 164-9.
5. C. Gaskett, et al, "Reinforcement Learning for Visual Servoing of a Mobile Robot," *Proceedings of the Australian Conference on Robotics and Automation.* 2009.
6. P. Giguere and G. Dudek, "Clustering Sensor Data for Autonomous Terrain Identification Using Time-Dependency," *Autonomous Robots*. 2009, pp. 171-86.
7. M. Harb, et al, "Neural Control System of a Mobile Robot," *IEEE International Joint Conference on Neural Networks*. 2008, pp. 2825-32.
8. H. Kim, et al, "A Robotic Model of the Development of Gaze Following," *IEEE International Conference on Development and Learning*. 2008, pp. 238-43.
9. O. L. Mangasarian and D. R. Musicant, "Lagrangian Support Vector Machines," *The Journal of Machine Learning Research*. 2001, pp. 161-77.
10. M. Nicolescu, O. C. Jenkins, and A. Olenderski, "Behavior Fusion Estimation for Robot Learning from Demonstration," *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications*. 2007, pp. 31-6.
11. M. Nicolescu and M. J. Mataric, "Natural Methods for Robot Task Learning: Instructive Demonstrations, Generalization and Practice," *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*. 2003, pp. 241-8.
12. S. B. Reed, C. G. Looney, and S. M. Dascalu, "A Recursive Hyperspheric Classification Algorithm," *Computer and their Applications in Industry and Engineering*. 2008, pp. 156 – 60.
13. V. Vapnik, "Statistical Learning Theory," Wiley-Interscience, New York, 1998.
14. S. Yamada and M. Morimichi, "Unsupervised Learning to Recognize Environments from Behavior Sequences in a Mobile Robot," *IEEE International Conference on Robotics and Automation.* 1998, pp. 1871-6.