Solving 3D Mazes with Machine Learning and Humanoid Robots

Vishnu K. Nath and Stephen E. Levinson
University of Illinois at Urbana-Champaign
405 North Mathews Avenue
Urbana, IL 61801

Abstract

In this paper, we present a system that integrates computer vision with machine learning to enable a humanoid robot to accurately solve any 3 dimensional maze that has not been previously given to it. The robot can construct the optimum path policy based on previous iterations and does not require any specialized programming.

The experimental setup includes a constructed 3D maze with a start and end point. The robot solves the maze using a red-colored ball. The robot can physically tilt the base of the maze with its hand so that the ball can roll into the desired region. The robot would begin tilting the maze only if a path exists between the start and the end point. If none exists, the robot would remain idle.

This work is important and novel for a couple of reasons. The first is to determine if constant repetition of a task leads to gradually increasing performance and eventual mastery of a skill. If yes, can that skill be adapted to a generic ability (Fleishman, 1972)? Also, can a robot's performance match or exceed that of an average human in the acquired ability?

Introduction

Acquiring any new skill requires various forms of learning that involves a mixture of imitation, self-learning and improvisation techniques. Robots that utilize cutting-edge techniques should be able to pick up new skills after being shown the initial steps to perform certain tasks (Begum & Karray, 2011). Moreover, various literature mention that there is a difference between skill and ability (Fleishman, 1972). Ability refers to a more general trait of an individual, like having a keen ability for spatial visualization. This ability is helpful in diverse tasks like aerial navigation, blueprint reading, etc. and has been shown to be the result of learning and development mainly during childhood and adolescence. Skill, on the other hand, refers to the proficiency level for a particular task, like flying a jet, or playing baseball. The core assumption is that skills that comprise complex tasks can be described in terms of simple and basic abilities (Fleishman, 1972).

In this paper, we use an iCub humanoid robot to solve a given 3-dimensional maze using 2 different approaches, a standard graph-based algorithm and learning. We measure the performance change during various iterations of the experiment. The goal is to determine if repetition can develop an ability. Assuming it does, we benchmark the performance of the iCub with that of an average human being for a related skill and see if the iCub can beat the performance of the human. For the experiment, all mazes are built on top of a LEGO board, with the bottom of the board having a handle for easy grasp.

The entire system was designed in a modular fashion and each module has a full-duplex communication channel. The major modules have been compartmentalized, based on functionality, into robotic kinematics, computer vision and algorithms respectively.

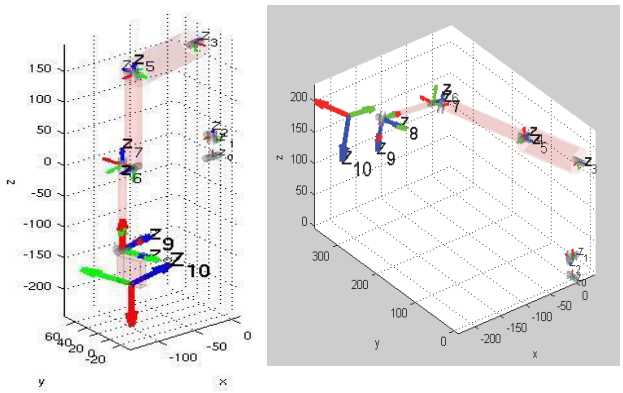
Robot Kinematics

The first step involved getting the iCub's right arm to the home position, and then to the maze solving position, which involves bringing the arm closer to the torso. Furthermore, subsequent movements of the arm are required in order to make adjustments to the maze. Therefore, the DH parameters of the right arm were computed and are given in table 1 below.

Link	a	d	α	δ
1	32	0	π/2	0
2	0	-5.5	π/2	- π/2
3	-23.467	-143.3	π/2	- π/2
4	0	-107.74	π/2	- π/2
5	0	0	- π/2	- π/2
6	-15	-152.28	- π/2	- π/2
7	15	0	π/2	π/2
8	0	-137.3	π/2	- π/2
9	0	0	π/2	π/2
10	62.5	16	0	π

Table 1: DH parameters of right hand at home position

A MATLAB simulation was performed to verify if the position vectors of all 10 links of the right arm were correctly aligned to be at the home position. The result of the simulation is shown in figure 1 below. The next step is to get the arm to the maze solving position to enable the greatest field of view for the robot while solving the maze. This required the use of transformation matrices to move all the joints to the appropriate position. Figure 2 shows the position vectors of the joints in the maze solving position, after the transformation matrices have been applied.



Figures 1 and 2: Position vectors of joints of the right arm

Computer Vision

The constructed maze is 3 dimensional in nature. This means that the maze can contain not just multiple paths and dead ends, but also cliffs and troughs that need depth perception as well. There are two algorithmic approaches being considered in this experiment. The first one is to use Dijkrasta's graph traversal algorithm, and the other is to use learning. They are discussed in detail in the algorithms section. The given maze must be studied in order to perform a graph-traversal based maze solving algorithm, as well as obtain an accurate control policy. However, it would be difficult to obtain a perfect orthographic view of the maze at all times, especially when the maze is tilted away along the line of axis of the view of the robot. In order to overcome this difficulty, an inverse homography must be performed so that the iCub will have an orthographic image of the maze at all times, for analysis purposes (Forsyth & Ponce, 2011). Given the rectangular shape of the base of the maze, the minimum number of identifiable geometric features with a known relationship to each other is four. The easiest way to accomplish this was to place high contrast color markers on each corner of the maze. The color red was selected, and so a resulting condition while building the maze was that no block must be red in color. The ball was also chosen to be red in color. While inverse homography would remove any depth information, the dual cameras of the iCub were used to maintain depth variations as metadata, which would be fed to the algorithm. Although multiple mazes were used in the experimental analysis, for the description of this module, only 1 maze is taken to be analyzed in depth. Figure 3 shows the unprojected coordinates of the maze after computing the inverse homography.

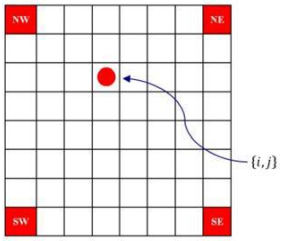


Figure 3: Unprojected coordinates of the maze after inverse homography

In the initial top down analysis of the maze, no ball was put on the board. To begin the entire experiment, the video feed that needs to be analyzed requires preparatory work. Thresholding by color would provide a binary image indicating where high concentrations of red are present. We decided to determine the RGB values of various points in the image to determine if raw RGB values would be suitable for thresholding. However, the variation of unmapped RGB values was too large for seemingly similar colors to be of any practical use, and we decided to use HSV values instead (Forsyth & Ponce, 2011). The resultant image is shown in figure 4 below.

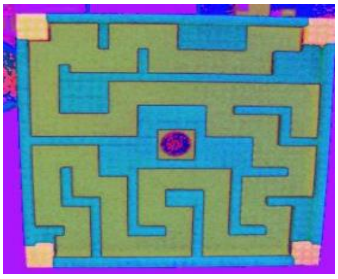


Figure 4: The maze (with markers) after HSV

A segmentation algorithm, like RasterScan, was used to label contiguous regions and sort them by size. The four largest regions are expected to be the four corner markers. The markers are assumed to be related in a square manner (from the shape of the base of the maze board) from a top down perspective. Using this information, all video information, outside the maze, can be cropped off after the inverse homography. As a result, the maze and its external maze walls are the only things that remain after this step. At this step, performing a RasterScan once again will provide the open contiguous path of the maze (Forsyth & Ponce, 2011). Figure 5 shows the resultant path that is obtained after the second RasterScan operation has been performed.

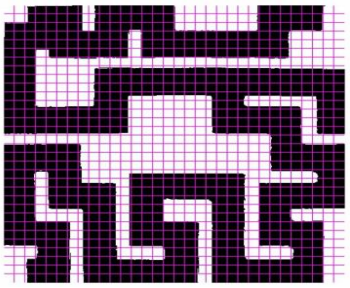


Figure 5: Resultant path after RasterScan

Once a path has been obtained, it can be discretized into a grid. This discretization is important for two reasons, depending on the algorithm being followed. For learning, this discretization would make it easier to derive the optimum rules for controlling the motion of the ball, on a regional basis. For graph traversal, the Dijkstra's algorithm was used. For this algorithm, each corner of the maze was identified and labelled a vertex. The cost of each path between two vertices was determined by a scaled factor of the distance between them. If a 3-dimensional path was involved, the cost was computed in the same way, no extra points/cost was added. The robot could be run in one or the other mode.

It is imperative to find the correct resolution for sampling the video feed. Sampling below the threshold would cause degradation in the maze and may result in open segments of the maze when there might be none in reality. As a result, the resulting learned policy would fail. Vertices might be missed and therefore, possible edges, thus leading to an incomplete maze. On the other hand, sampling above the threshold would produce an extremely fine resolution which would cause an exponential increase in the time taken by the learning algorithm to converge upon a solution. This issue is referred to as the 'curse of dimensionality' in literature and is present in all uniformed dynamic programming schemes (Begum & Karray, 2011).

In order to solve this problem, we experimented with several resolutions and eliminated several of them on a trial-and-error basis. Eventually, we decided to choose from amongst three resolutions, namely 16x16, 32x32 and 50x50. The resultant images are shown below in figure 6.

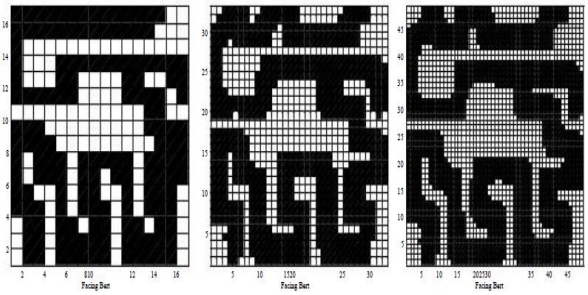


Figure 6: 16x16, 32x32 and 50x50 grids respectively

From figure 6, we inferred that the 16x16 resolution is inadequate since information about the ball on the grid would be lost. The iCub would be able to identify the location of the ball only in terms of 4 quadrants which would result in several errors. Emperically, we determined that the 50x50 resolution will definitely determine the live location of the ball at any instant. However, the convergence of the learning algorithm takes an unacceptable amount of time to take place in a live manner. We also determined that the 32x32 resolution strikes a compromise between the two resolutions. It can be used to determine the location of the ball with respect to the grid with sufficient accuracy and converges within a satisfactory running time.

Algorithms

There were two approaches used to enable the iCub to solve mazes. One was the graph traversal based approach, using Dijkstra's algorithm. The other was the usage of learning to obtain an optimum control policy. The implementation of Dijkstra's algorithm was done using a min-priority queue, using a Fibonacci heap. The reason for using this method instead of the original algorithm was because of performance issues. The Fibonacci heap gives a worst –case run time of $O(|E| + |V|\log|V|)$, where $|E|$ is the number of edges and $|V|$ is the number of vertices. This was a major improvement of the original worst-case run time of $O(|V|^2)$,since the number of vertices would increase rapidly based on the complexity of the maze.

It needs to be noted that the original algorithm had to be modified because of the problem of cyclic graphs. During our test runs, we observed in certain cases that the ball would keep moving in an infinite loop, because the resultant graph would be cyclic. If the weights of the edges that would enable the ball to continue along its path is equal to the weight that would take it back to a previously visited vertex, the algorithm favors the latter in this race condition. So, we had to add logic for a tie-break scenario wherein the ball would take the unexplored territory, over the familiar, making it adventurous! The results are given in the section for experimental results.

For the learning approach, the fundamental approach that we used to enable the iCub to determine the shortest path of any maze that was presented to it was to use reinforcement learning, drawing upon previous experiences (Nath & Levinson, 2013).The update equation for temporal difference Q-Learning is given by equation (1) below.

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)) \qquad (1)$$

where Q(s,a) is the Q-value of action 'a' at state 's', R(s) is the reward function of the state 's', α is the learning rate and γ is the discount factor. Each state is defined as the state of the maze when the ball has come to a stop at one of the corners of the maze. Each action is defined as the tilting of the maze along one of the principal axes. An examination of (1) shows that Q-learning backs up the best Q-value from

the state reached in the observed transition. It pays no attention to the actual policy being followed, being an off-policy learning algorithm (Russell & Norvig, 2010). Clearly, there was a need to come up with a learning algorithm that utilizes a policy that would maximize the probability of the robot solving the maze. Such an algorithm is of the on-policy type algorithm and is called the SARSA algorithm. SARSA stands for State-Action-Reward-State-Action and utilizes the optimum policy for updating the Q-values. The update equation for SARSA is given by

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,(R(s) + \gamma\, Q(s',a') - Q(s,a)) \qquad (2)$$

The difference between (1) and (2) is the omission of the '*max'* term of the new Q-value. This means that SARSA actually waits until an action is taken and backs up the Q-value for that action (Nath & Levinson, 2013), making it excellent for exploration. For the objective of finding the shortest path to solve a maze, heavy exploration (or at least the consideration) of all possible paths is mandatory. Q-learning is more flexible than SARSA, i.e. an agent that learns by Q-learning can behave well even when guided by a random or adversarial exploration policy. However, SARSA is more realistic than Q-learning. For example, if the overall policy is even partly controlled by other agents, it is better to learn a Q-value function for what will actually happen rather than what the agent would like to happen. This is possible in certain scenarios like the ball getting stuck in a particular nook. So, we felt it is better to use the SARSA approach. The optimum policy for SARSA is given by equation (3).

$$\pi^* = \text{argmax}_\pi \sum_h P(h|e) u_h^\pi \qquad (3)$$

In equation (3), the posterior probability P(h|e) is obtained in the standard way, by applying Bayes' rule on the observations till date, where u_h^π is the expected utility averaged over all possible start states. This is how the feedback loop is created that would allow constant improvisation.

Furthermore, we performed a value iteration of a discrete state-action space, and the algorithm used a sample based quality space. Here, φ is an index of discretized space and θ is the value at that index. The control space was U = {0,1,2,3,4} where 0 is a random action and {1,2,3,4} is a wrist tilt in the direction {North East, North West, South West, South East} respectively. The state space corresponds to the location in the n x n discretized path space of the maze. The value of α and γ were set to 0.99 and an exploration function of $\varepsilon = Q_{visist}^{-0.01}$ was used.

Q-Learning with Algorithm2.3:

\\----------

Input: discount factor γ,

exploration schedule $\{\epsilon_k\}_{k=0}^{\infty}$ s.t. $\sum_{k=0}^{\infty} \epsilon_k = \infty$,

learning rate schedule $\{\alpha_k\}_{k=0}^{\infty} \in [0,1]$

BFs $\phi_{[j]}: X \times U \rightarrow \mathbb{R}^{p(j)}$,

Initialize parameter vector, e.g., $\theta_0 \leftarrow 0$

Measure initial state x_0

For every time step $k = 0,1,2,\ldots$ **do**

- $u_k \leftarrow \begin{cases} u \in \arg\max_{u'} \phi^T(x_k,u')\theta_\ell & P = 1-\epsilon_k \\ \text{random action in } U & P = \epsilon_k \end{cases}$
- Apply u_k, measure next state x_{k+1} and reward r_{k+1}
- $\theta_{\ell+1} \leftarrow \theta_\ell + \alpha_k[r_{k+1} + \gamma \max_{u'} \phi^T(x_{k+1},u')\theta_\ell - \phi^T(x_k,u_k)\theta_\ell]\phi(x_k,u_k)$

end

\\----------

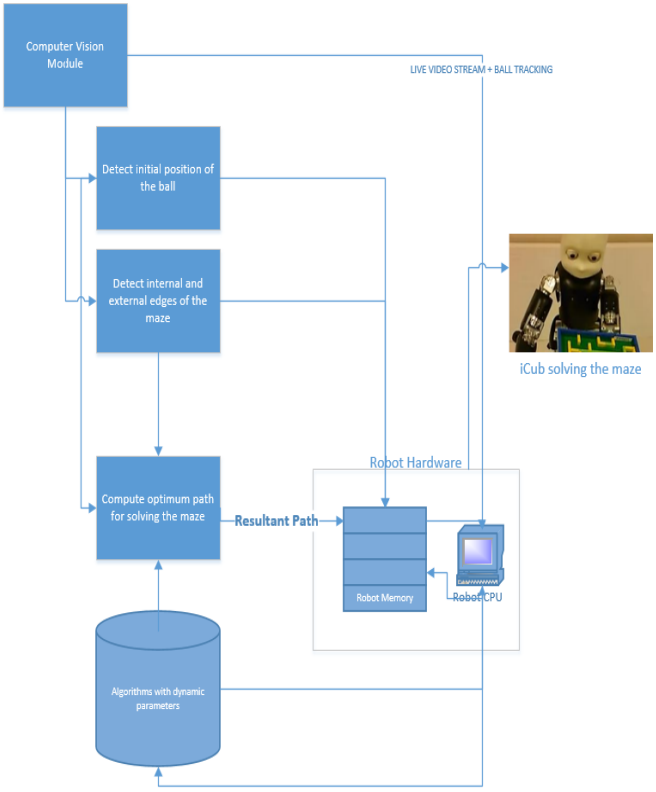


Figure 7: Interaction of Modules

Interaction of Modules

Figure 7 above shows the interaction of all the various modules, both hardware and software, that take place in order to successfully solve any 3 dimensional maze given to the iCub robot.

Importance and Experimental Results

Abilities are defined by empirically determined relations among observed separate performances. For example, that individuals do not do well on task A, as well as, B and C, but not on tasks D, E and F implies that there is some common process involved in performing the first three tasks (Fleishman, 1972). In our study, we first wanted to determine the change in performance of similar tasks over time, and then determine if a learned ability in a robot can beat that of a human. In order to achieve this, we setup the experiment so that the iCub would first solve 23 different mazes, using both graph traversal and learning. Whichever method performs empirically better would then be used to perform a similar task in conjunction with a human. For our experiment, the similar task was to solve a variation of PacMan in the shortest amount of time possible. In this version, there are no spiders, so PacMan simply has to eat all dots in the smallest amount of time. The PacMan is controlled by a joystick, whose handle is very similar to the handle at the base of the maze board, and the game had to be slowed down to a rate of eating 1 dot per second in order to allow the iCub sufficient time to move the joystick. This was required because the motor capabilities of the iCub do not allow it to operate a joystick, or any other machinery, with the same dexterity as an average human. The same type of joystick was used by both the iCub and the human. In order to do this, the goal had to be changed in the program, and an additional round of training was required. Also, the grasping of the stick for the robot was performed manually, and should not have any impact in the outcome of the experiment. The details of the experiment are given below.

Initially, we decided to benchmark the two algorithms. We used a variety of mazes, some of which had a 2 dimensional path and some had a 3 dimensional path. Also, some mazes have a cyclical path in them that may or may not be part of the actual shortest path of the maze. Therefore, getting stuck in one of these loops would affect the total run time of the maze for that approach. The results are summarized in table 2 below. Note that those iterations that failed do not feature in the calculation of the average time to solve the type of maze.

Approach	Maze Type	Iterations			Average Time
		Pass	Fail	Pass %	(secs)
Dijkrista algorithm	Simple 2-D	23	0	100 %	43
	2-D with cycles	14	9	61%	62
	Simple 3-D	12	11	52%	81
	3-D with cycles	6	17	26%	95
Learning	Simple 2-D	23	0	100 %	67
	2-D with cycles	22	1	96%	71
	Simple 3-D	18	5	78%	74
	3-D with cycles	16	7	70%	75

Table 2*: Time to solve successful mazes*

From table 2, we can see that the learning algorithm is a lot more resilient to variations in the maze and the variation in the response time is much smaller than that of Dijkstra's algorithm. Also, the pass% is significantly higher for the learning algorithm, rather than the graph traversal algorithm. Therefore, we decided to benchmark the performance of the iCub with an average human using the learning algorithm discussed above. Figures 8 and 9 show the normalized log value function of the maze during one iteration, and the optimal control policy respectively.

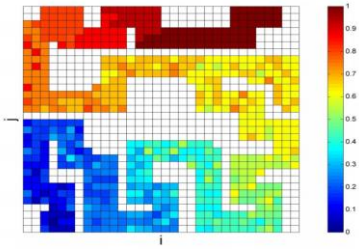


Figure 8: Normalized log value function of maze

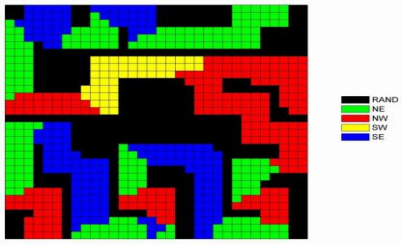


Figure 9: After application of optimal control policy

Based on prior work (Fleishman, 1972), there are eleven types of perceptual-motor functions and nine factors in the area of physical proficiency. Of these, the ones that are at play in this experiment are (i) Control Precision (ii) Response Orientation (iii) Reaction Time. Studies indicate that systematic changes in the abilities required to perform a task are observed when the display-control relations in the task were varied systematically. Furthermore, the frequent repetition of a task leads to a skill being developed, thus leading to a variation in the abilities being measured (Fleishman, 1972). These research works were the

motivation for us to make the iCub solve both 2-dimensional and 3-dimensional mazes. In the benchmark test, we perform a test on the same configurations after the iCub has performed the maze solving and has all the associated info in memory, and when it performs a fresh explorative iteration. The goal for this experiment was set to consume all the dots in the maze, and the reward function was for the smallest amount of time possible. In all cases, both the robot and the human were allowed one explorative iteration of the game, before the benchmarking test took place. The results are tabulated below, in table 3.

Agent	Maze Size	Time (seconds)	
		After maze training	No maze training
Robot	10 x 10	61	87
	20 x 20	84	117
	30 x 30	101	161
Human	10 x 10	n/a	96
	20 x 20		119
	30 x 30		173

Table 3*: Benchmarking results*

From the benchmark test, there are a couple of observations that can be made. Firstly, there is a marked difference in the response time before and after the maze training phase. This implies that by learning a common ability, different skills that make use of the ability are improved, as aforementioned. Furthermore, skills are developed by repetition of tasks, not exactly similar but more of an assorted fashion. This validates the claims made by Fleishman (Fleishman, 1972), not just in humans, but also in robots. In humans, the ability to move objects and observe its consequences are part of the developing neuro-motor skills of the human as an infant. More importantly, it shows that humans are better at predicting a maze path when the resolutions are small, but as the complexity increases, we essentially follow a random walk approach to solving problems. That is perhaps why the solving time of the maze of the human is in closer proximity to the "no maze training" set, than the "after training" set.

Conclusion and Future Work

While standard maze solving algorithms involve a graph traversal approach, the performance of such algorithms was found to be lacking in comparison to a learning algorithm. Furthermore, it was empirically observed that improving a particular skill has a direct impact on the ability to perform other tasks that make use of the same skillset. In other words, the result of several psychological studies that claim that complex tasks are a subset of simpler tasks hold true not just for neural models created by the human brain, but also hold true for computer generated models. In other words, it would not be a stretch to say that skills can be quantified and can be numerically analyzed each step along the way.

There is still a lot more work to be done in this field of work, and within this experiment in general. One area would be to benchmark other tasks that make use of learned skillsets and confirm if the trend takes place across all eleven types of perceptual-motor functions (Fleishman, 1972) and how much cross skillset development can take place in a robot (Begum & Karray, 2011). Furthermore, the sample size representing humans was too small (just 1). This needs to be vastly expanded to see how much variations take place amongst humans with varying skill sets and abilities. There are certain expected variables like learning ability, motor dexterity, etc. that will impact the results of these experiments. However, there are bound to be hidden random variables, "hidden" meaning that they are not immediately intuitive to have an effect on performance. Uncovering these hidden random variables could be the key to enabling human beings to achieve the mathematically maximum possible level of efficiency. It would also allow researchers the ability to consider these random variables while designing future experiments, as well as while incorporating learning or other dynamic algorithms to achieve required tasks.

References

Barber, D. (2012). *Bayesian Reasoning and Machine Learning*. Cambridge: University Press.

Begum, M., & Karray, F. (2011). Visual Attention for Robotic Cognition: A Survey. *IEEE Transactions on Autonomous Mental Development*.

Breazeal, C., Wang, A., & Picard, R. (2007). Experiments with a Robotic Computer: Body, Affect and Cognition Interactions. *HRI'07* (pp. 153-160). Arlington, Virginia: ACM.

Buşoniu, L., Babuška, R., De Schutter, B., & Ernst, D. (2010). *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press.

Fleishman, Edwin A. Little, Kenneth B. (1972). On the relation between abilities, learning, and human performance: American Psychologist.

Forsyth, D., & Ponce. (2011). *Computer Vision: A Modern Approach*. Prentice Hall.

Harnad, S. (1995). *Grounding Symbolic Capacity in Robotic Capacity*. New Haven: Lawrence Erlbaum.

iCub Robot Wiki. (n.d.). Retrieved 02 2012

Kormushev, P., Calinon, S., Saegusa, R., & Metta, G. (2010). Learning the skill of archery by a humanoid iCub. *2010 IEEE-RAS International Conference on Humanoid Robotics*. Nashville.

Metta, G., Sandini, G., Vernon, D., & Natale, L. (2008). The iCub humanoid robot: an open platform for research in embodied

cognition. *8th Workshop on performance metrics for intelligent systems*. ACM.

Michalski, Carbonell, & Mitchell, T. (1983). *Machine Learning*. Palo Alto: Tioga Publishing Company.

Michie, D. (1986). *On Machine Intelligence*. New York: John Wiley & Sons.

Nath, V., & Levinson, S. (2013). Learning to Fire at Targets by an iCub Humanoid Robot. *AAAI Spring Symposium*. Palo Alto: AAAI.

Nath, V.(2013).Usage of Computer Vision and Machine Learning to Solve 3D Mazes, MS thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL.

Nath, V. & Levinson, S. (2014). *Autonomous Military Robotics*. New York: Springer New York.

Nath, V. & Levinson, S. (2014). *Autonomous Robotics and Deep Learning*. New York: Springer New York.

Russell, S., & Norvig, P. (2010). *Artificial Intelligence, A Modern Approach*. Ner Jersey: Prentice Hall.

Sandini, G., Metta, G., & Vernon, G. (2007). The iCub Cognitive Humanoid Robot: An Open-System Research Platform for Enactive Cognition. *In 50 years of artificial intelligence* (pp. 358-369). Berlin Heidelburg: Springer Berlin Heidelberg.

Sigaud, O., & Buffet, O. (2010). *Markov Decision Processes in Artificial Intelligence*. Wiley.

Spong, M. W., Hutchinson, S., & Vidyasagar, M. (2006). *Robot Modelling and Control*. New Jersey: John Wiley & Sons.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An Introduction*. Cambridge: MIT Press.

Tsagarakis, N., Metta, G., & Vernon, D. (2007). iCUb: The design and realization of an open humanoid platform for cognitive and neuroscience research. *Advanced Robots 21.10*, (pp. 1151-1175).