Cognitive Robotics: Analysis of Preconditions and Implementation of a Cognitive Robotic System for Navigation Tasks

Stefano Bennati [benna86@gmail.com]
Università degli Studi di Brescia
Brescia, Italy

Marco Ragni [ragni@cognition.uni-freiburg.de]
Department of Cognitive Science, Friedrichstrasse 50
Freiburg 79098, Germany

Abstract

Cognitive robotics is a fascinating field in its own right and comprises both key features of autonomicity and cognitive skills like learning behavior. Cognitive architectures aim at mirroring human memory and assumptions about mental processes. A robot does not only extend the cognitive architecture regarding real-world interaction but brings new and important challenges regarding perception and action processes. Cognitive robotics is a step towards embodied cognition and may have influences not only on cognitive science but on robotics as well. This paper presents an integration of the cognitive architecture ACT-R on a simple but programmable robotic system. This system is evaluated on a navigation experiment.

Keywords: Cognitive Science; Robotics; Mindstorms; ACT-R; Navigation

Introduction

From the early beginning of robotics one line of research has tried to bring human cognition and robotics closer together Brooks et al., 1999. Nowadays, technological progress in the field of robotics and the development of cognitive architectures allows for a leap forward: A robot able to navigate an environment, with the ability to learn and a human-like attention shift.

This new and exciting field is sometimes referred to as *Cognitive Robotics* [1]. This combination of two fields leads to a number of important research questions: What are the immediate advantages of cognitive robotics (a term we will use in the following for a robot controlled by a cognitive architecture) over classical robotics? Is the cognitive architecture (which partially is able to simulate human learning processes) restricting or improving navigation skills? In cognitive science new research focuses on *embodied cognition*.

Embodied cognition claims that understanding (especially of spatial problems) is derived from the environment Anderson, 2003. In other words, cognition is not independent on its physical realization. The study described in Buechner et al., 2009 used a virtual reality environment. Participants had to navigate through a labyrinth in the ego-perspective and had to find an initially specified goal (red dot). The study identified recurrent navigation strategies (which we introduce later) used by the subjects.

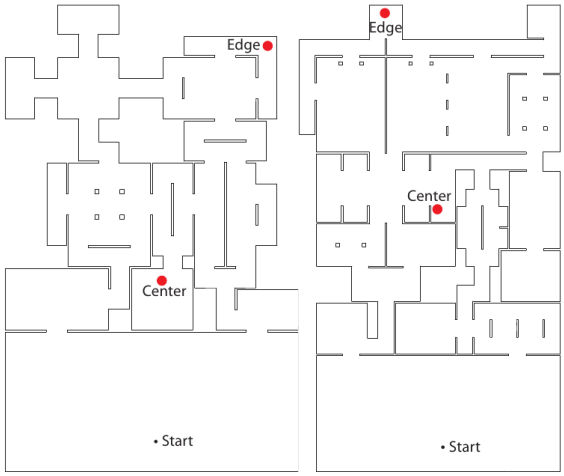

Figure 1: Layouts of two mazes: The task is to find the target object (red dot) at the edge or center Buechner et al., 2009.

Modeling navigation tasks, for instance in labyrinths, still poses a challenge for cognitive architectures: Although they can model decision processes they typically abstract from metrical details of the world, from sensor-input, and from the integration processes of environmental input to actions (like move operations). Robotics, on the other hand, has typically captured all of these aspects, but does not necessarily make use of human-like learning and reasoning processes or even try to explain human errors or strategies.

Compared to humans a robotic agent has limited perceptions and capabilities. On the other hand, it can teach us something about the relevant perceptions that are already sufficient for the robotic agent to perform successfully. For instance, in the navigation task above (e.g., cp. Fig. 1), the distance from the wall (in the direction it is facing) and the color of the floor the robot is standing on are sufficient.

Much research is being done in the field of the cognitive robotics; some prototypes of cognitive robots have already been built [2]. This research concentrates on the human-robot and robot-environment interaction, allowing the robots to rec-

[1] "Towards a Cognitive Robotics", Clark, Andy `https://www.era.lib.ed.ac.uk/handle/1842/1297`

[2] e.g., Cognitive Robotics with the architecture ACT-R/E `http://www.nrl.navy.mil/aic/iss/aas/CognitiveRobots.php`

ognize and interact with objects and people through their visual and auditory modules Sofge et al., 2004. The architecture proposed contains *Path Planning and Navigation routines* based on the Vector Field Histogram Borenstein and Koren, 1991 that allow the robot to navigate avoiding obstacles and explore the environment. Unlike the architecture proposed, the objective of this paper is to implement a *Cognitive Navigation System* which is completely based on ACT-R and takes advantage of its cognitive features like *Utility Learning*, that allows Reinforcement Learning Russell and Norvig, 2003, p.771 on productions. No other softwares than ACT-R will be used to control the robot. That experiment has the merit of have taken the first steps towards interfacing ACT-R with a mobile robot, but the data is still incomplete and tries to combine as many different abilities as possible (from natural language processing, to parallel computing) in a manner that is likely more complex than necessary. Our approach starts at the other end, taking only those aspects and sensor data into account that are necessary to perform the task – the most simple robot. Other research studied the possibility of interfacing ACT-R with a robot and giving it direct control over the robot's actions. That effort produced an extension of ACT-R called *ACT-R/E(mbodied)* Trafton, 2009; Harrison and Trafton, 2010. ACT-R/E contains some new modules that act as an interface between the cognitive model and the *Mobile-Dexterous-Social (MDS)* robot Breazeal et al., 2008, allowing it to perceive the physical world through a video camera. However, no navigation was investigated, as the robot did not navigate an environment. In both this and our implementation ACT-R has been extended. The vast difference between the two is the smaller amount of changes made to the standard ACT-R by our implementation, due to the sensors' higher complexity in the MDS.

Consequently this article investigates, first, how to control a robot through ACT-R and, second, if this *cognitive robot* shows human-like behavior while navigating and searching for a goal, e.g., an exit from the maze. The paper is structured in three parts: The first part – The Elements – contains a brief description of ACT-R, the robot and its features. The second part – the Integration – describes the software that connects ACT-R with the robot, through which perceptions can reach the cognitive architecture and actions the actuators. The third part – the Evaluation – tests the robot on a navigation task and analyzes the results.

The Elements

The Robotic System

Our device of choice is a standard Mindstorms[3] class robot (cf. Fig. 2). It consists of a central "brick" that contains the processor and the batteries. At this core component several peripherals can be attached. It supports up to three step-to-step engines and up to four sensors.

[3]This type of robot is produced by The Lego Group

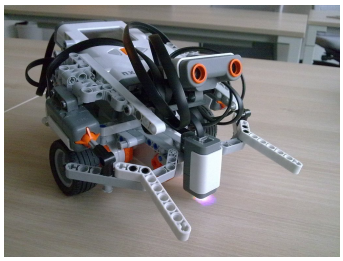

Figure 2: The Mindstorms class robot. It is equipped with a color, ultrasonic and two touch sensors.

Chassis design. The chassis is the structure to which the central brick, the motors and the sensors are attached. To limit odometry errors two fixed, independent driving wheels have been used, instead of caterpillar tracks. A third central fixed idle wheel allows the robot to keep its balance. The structure resembles a differential drive robot, with the exception that the third wheel cannot pivot. Removing the tire from the wheel allows the robot to turn without too many difficulties.

Sensors. A robot can be equipped with several kinds of sensors: from a simple sound or light sensor, to more complex ones like a compass or webcam. Our design makes use only of the most basic sensors: a *touch sensor* activated by pressure, a *color sensor* capable of recognizing light intensity or six tonalities of color, and an *ultrasonic sensor* for distance measurements. The color sensor and the ultrasonic sensor are fixed to the front, while a touch sensor is placed on each side. Both touch sensors are linked to a structure that covers the front of the robot on that side. They are not used during navigation but to stop the robot when it touches a wall.

Integrating Architecture and Robot

The first step towards a *cognitive robot* is to create a working interface between the ACT-R framework and the NXT platform. This interface allows an ACT-R model to control the robot and to receive sensor inputs from the robot. Due to the modular structure of ACT-R, this interface is composed of modules.

Low level lisp function. On the basis of these modules there is a library called nxt-lsp Hiraishi, 2007 that provides low-level lisp functions to execute simple tasks like interrogate a sensor or move a motor. The capabilities of this library are the following: it can read information about the environment from the light sensor, the distance sensor and the touch sensor; it can make the robot perform some actions, like move its motors, turn on and off the light and play some sounds; it can also check the robot's internal state, querying the battery or the motors about their states. The library did not have the support for the color sensor, so it has been extended to support that sensor, necessary for our purposes, as well.

Extending ACT-R

The interface is composed by several modules with different functions, this step has been taken to keep things simple and to follow the general design of ACT-R. The lisp library has been bundled in the code and its functions are called by these modules to control the robot.

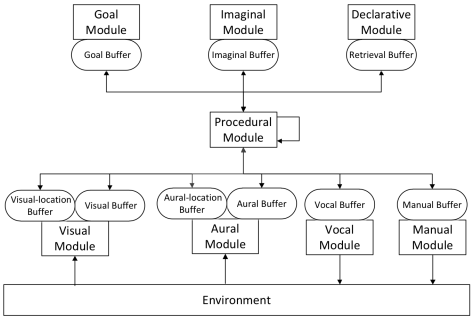

Figure 3: The ACT-R modules.

Motor module. This module controls the engines, when a request is made to its buffer the module responds, activating or deactivating the specified motors.

Touch module. This module controls the touch sensors, it can control up to four sensors but the default value is two. This module has a buffer called nxt-touched, but it is only used to query the module. If the nxt-touched buffer receives a query on the touched slot, with value true, the query returns true if the sensor reports a pressure.

Vision module. This module controls the color sensor. It has a buffer called nxt-visual, but it does not have any other purpose than querying the module. If the nxt-vision buffer receives a query on the touched light, with value on, it will turn the sensor on and schedule a polling of the sensor every 50ms. If the value is off, the sensor will be turned off and the polling stopped. When the sensor identifies a change in color, or the color blue, it draws on the visicon, the standard ACT-R visual input, a letter of the same color. This action triggers a standard ACT-R procedure that will allow the model to realize that a change has happened.

Distance module. This module controls the ultrasonic sensor. It has a buffer called nxt-distance. When a request is made to this buffer, with a chunk of type obstacle, the module reads the distance from the sensor and updates the chunk in its buffer with the read value. It uses two productions to obtain the value, the first production fires a request and the second reads the results from the nxt-distance buffer.

Evaluation: Navigation in a Labyrinth

We decided to test the cognitive robot on several self-built labyrinths. The environment is perceived by the robot through its sensors, the robot can perceive the presence of an obstacle, the color of the ground that is used to discriminate between obstacles (a wall, a junction, the goal, or a clear way), and the distance to the next wall. It is important to keep in mind that the robot has no a priori knowledge of the environment it is operating in. It will explore the labyrinth like humans would – this is a classical example of contingency problem Russell and Norvig, 2003, p.80. The only two types of information that the robot can obtain from its sensors are the presence/absence of an obstacle close by, and its distance from the next wall. With this information the robot must avoid walls and be able to choose in which direction to turn when it finds a junction. A human being can approximately measure this distance in two ways: looking at the walls a person can guess its distance from it, like the robot does with its ultrasonic sensors, and through his sense of touch, the same as the touch sensors equipped on the robot.

The Model

The model implements the *Active Reinforcement Learning* algorithm Russell and Norvig, 2003, p.771, that uses the feedback on its performance to learn what actions are to be preferred. In case the available information is not enough to decide what is the best way, the model follows the perimeter strategy to navigate. An aleatory component can favour a known way over the entrance in a not jet explored branch. Thanks to its internal representation of the environment, the model is able to recognize the junctions it has visited before and remember their performance.

The internal representation

While exploring, the cognitive robot develops an internal representation of the environment in declarative memory . This representation contains all of the information that it knows about the already explored environment. For every step or turn the robot takes, information is stored in declarative memory in a chunk of type “movement,” those chunks have a slot called “direction” that encodes the direction of the very movement: This slot has the value “forwards”, unless the robot turns itself. In that case the slot will contain the name of the new orientation. Every movement chunk has another slot called “counter”, where a progressive number is stored. The numbers detail the sequence of movements of the current run. Through this trace, it is always possible to retrieve the direction it is currently facing. This is the only purpose of using these chunks during navigation. The maze is represented in declarative memory as a graph, whose nodes are the junctions. The model does not care about how long and how twisted the road from JunctionA to JunctionB is, that is unimportant during the decision process because it does not involve decision making. The only relevant information is the expected performance in taking a specific way, which is represented in declarative memory by a chunk of type “junction.”

This chunk identifies a specific junction by the values in four of its slots (north, east, south and west) and a direction by its "turn" slot. For every turn the model must know its goodness, this value is encoded in the "performance" slot. A smaller number implies a better performance, the number "-1" means that this direction was taken but not yet rated. Recognizing a junction allows the model to remember which routes it has already explored and the current best. With such a complete knowledge the model is able to find a shortest way to the goal. These two chunk types form the model's *Knowledge Base*, through which the previous explorations can be reconstructed. The "movement" chunks contain the path history: They save, for every step, the direction in which the model was going; while the "junction" chunks compose the decisional history and save the order in which the junctions were taken and the performance value's updates of every junction.

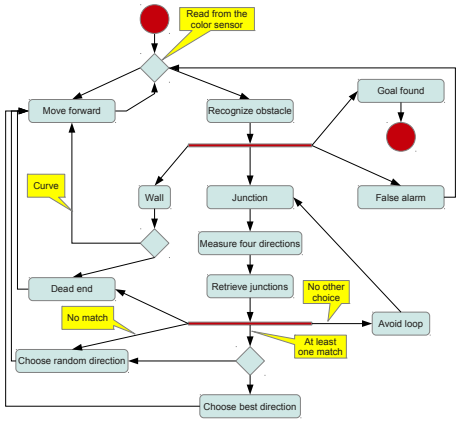

Figure 4: The local search algorithm implemented in ACT-R.

Algorithm

The robot uses a local search algorithm (cf. Fig. 4). The robot moves forward until a chunk is present in the visual-location buffer, then it reads the color and discriminates if it is an obstacle, a junction, the goal or a false alarm. In case of a false alarm it keeps going forwards. If a wall is detected there could be a curve right or left, or a dead-end. Special productions are called for a quicker response and the robot will follow the curve or, in case of a dead-end, go back. If a dead-end is encountered, the last junction is marked with a low performance value, so that in future the model will avoid that direction. When a junction is detected, with or without a wall on the front, the robot turns itself in the four directions and measures the distance with its ultrasonic sensor. Once the distances are retrieved, it tries to recognize this junction among the ones it has seen before. Many retrieval requests to the declarative memory are issued, by calling the same production many times, until the retrieval process fails. For every direction that is not retrieved, the model checks the corresponding distance. If it is less than a certain threshold it detects a wall and marks that direction as not selectable. Now, if there is still at least one direction that has not been tried yet, the model has two possibilities: The first possibility is to select that direction and explore a new branch of the labyrinth; if there is more than one, the model follows the perimeter strategy. In this way the maze, in the first stage of the exploration, is covered following the perimeter strategy. The second possibility is to select the way with the best performance among the directions that matched in declarative memory and go on a safe path. The decision between an enterprising and a conservative approach is taken randomly, with a probability of taking a conservative path proportional to the performance rating: The better the performance, the more likely the conservative approach is to be selected. This solution incentivizes the exploration in the first stage, when it is more likely that a shorter way can be found; and is more conservative in the end, when exploration will bring a minimal increment of performance. If all the selectable directions matched in declarative memory, the model chooses the way with the best performance. If the junction does not have any selectable direction, because there are only walls or dead-ends, the branch is purged: the last junction, that brought the model to this dead-end, is selected and marked with a very poor performance value. In the particular case in which the only selectable direction has already been taken during the current run, the robot retraces its steps, a loop is detected. The last unrated junction that the robot took is marked as a dead-end, so that the next time it will not turn in that direction again, interrupting the loop. If a goal is detected all the junction chunks used in the last run are rated by a rating function that implements the *Policy-Iteration* routine Russell and Norvig, 2003, p.624. For every junction the rating function calculates the new performance value as the time difference between the goal discovery and the last use of the junction's representation in declarative memory. If the old performance value is higher, or that junction had not been rated yet, the function updates the performance with the current value. After that the model starts again from the beginning, but this time it uses the accumulated experience during the previous explorations leading to better choices.

Distance computation

The Act-Rientierung Project Dietrich et al., 2011 implemented an algorithm that reconstructs the distance between non-adjacent waypoints, using the information about adjacent waypoints gleaned from exploration. It has been integrated in our model. We used junctions as waypoint because the environment and the perceptions did not leave other choice. The computation is made using chunks that represent numbers, adjacent numbers have high similarity and that leads to counting errors (reproducing human behavior). The model, during exploration, stores distance in steps of adjacent junctions in specific "waypoint" chunks. When the model finds the goal, it goes through the list of known junctions and calculates the distance to the goal for every pair of junctions. With consecutive runs the distance values may change, because of the retrieval errors in the counting process.

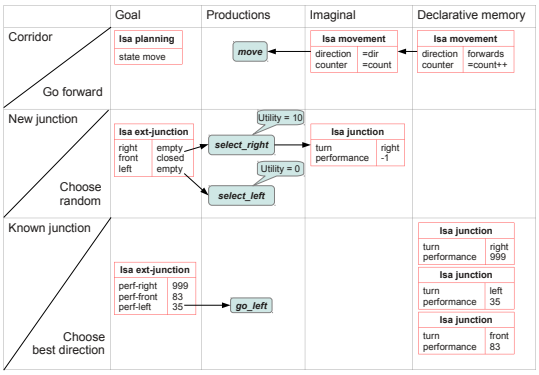

Figure 5: **Left:** An example of a test labyrinth. **Right:** Decision process for corridor, new and known junctions.

Evaluation

Several tests have been executed to evaluate the differences in performance that this new implementation brings. Those tests have been run in a simulated environment that reproduces exactly a real robot going through a real maze, with the advantages of being faster and not having to deal with odometry or measurement errors. To be able to compare the results the test has been set up as a replica of the Büchner experiment Buechner et al., 2009, recreating the same labyrinths inside the simulator (cf. Fig. 1); in both the configuration with the goal on the edge and in the center. The performance was measured in PAO (Percentage Above Optimum).

$$PAO = ((d_{walked} - d_{shortest})/d_{shortest}) \times 100$$

The same unit of measurement was used in this experiment as well, so that the results of both can be easily compared. During every simulation, for each of the four environments, the model had a time limit of 4000 units of ACT-R's virtual time. Within this time constraint the model could find a stable suboptimal solution was found 84% of the time. The diagram in Fig. 6 shows the mean quality of the consecutive solutions.

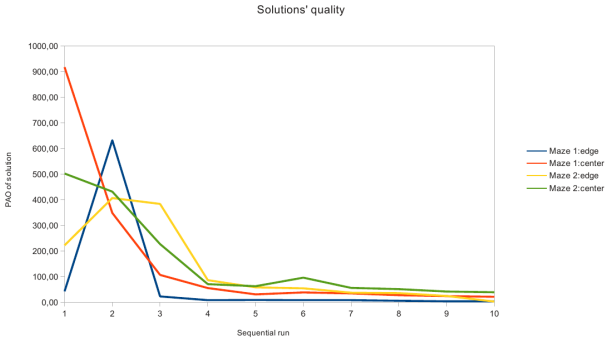

Figure 6: Behaviour of the model, showing a learning curve similar to the humans' Buechner et al., 2009.

The curves in Fig. 6 show similar learning behavior for all four labyrinths: The model starts with an uninformed exploration, according to the actual strategy, until it finds the goal. Then it starts an exploration phase that will complete when all possibilities have been tried and rated. During this phase the model tries unexplored ways, needing in average more time to find the goal than the human counterpart. After less than 4 runs, on average, the model has gathered enough information about the environment and stabilizes on a suboptimal solution that is covered until the time elapses. In the experiment done by Büchner the participants shown a preference for the perimeter strategy to navigate in the maze and search for the goal. In the current experiment different strategies were tested:

Deterministic perimeter strategy. The first strategy uses a right-preference perimeter strategy. Each application of turn-right production rules yields a utility bonus, a smaller bonus is given for go-straight production while for turn-left no bonus is assigned. The test showed that the first run is completely deterministic and in every run the agent always take the same path, that implies a much smaller degree of choice during the rest of the exploration. Even if some differences can be seen during the exploration phase, every run leads to the same suboptimal solution. In conclusion the tests demonstrated that this strategy is too deterministic and dependent on the specific maze to be suitable for a real exploration task.

Random walk. Another strategy was a random strategy with utility learning, in the beginning all the productions have the same utility, which can change during the run according to the utility rewards gathered during the exploration. A positive utility reward is given by the action of finding the goal, with a negative reward for each dead-end. As expected from a random walk strategy, the performance is nearly identical for all four environments.

Gaussian perimeter strategy. The last strategy uses a random gaussian utility bonus, instead of fixed one. The more desirable is the action to perform, the higher the center of the gaussian will be. This bonus is added to the standard production's utility by the mean of the *:utility-offsets* function Bothell, 2004, p.187. The gaussian functions are calculated by the *act-r-noise* function Bothell, 2004, p.138 with parameter $s = 0.5$, that correspond to a variance $\sigma^2 = 0,82246$.

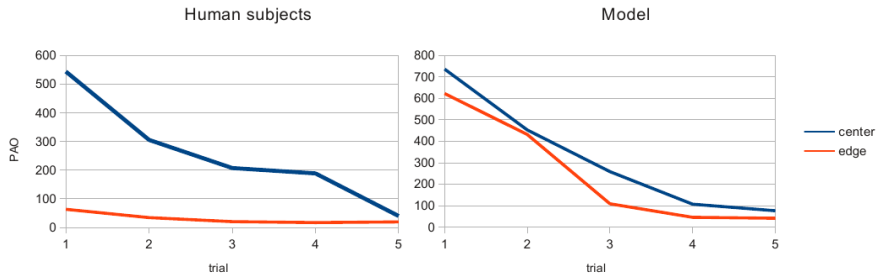

Figure 7: Human (left) and model performance (right) with random walk, the curves have a correlation of 0,962 and 0,954.

The three curves, identifying the action of turning left, going straight on and turning right, are centered respectively in 0, 2 and 4, the higher the center the more likely to happen. This method models well the human concept of perimeter strategy, not mere deterministic but a good compromise between following the rules and taking own initiative. The tests report that the perimeter strategy is chosen 47% of the time in the first maze and the 43% in the second, and that initial value and solution quality are close to the human's. Some differences are present in the execution, due to the model's intensive exploration, unlike the human subjects who prefer to travel along a safe path.

Conclusions & Future work

This experiment demonstrated that ACT-R can be successfully used in conjunction with a Mindstorms robot. We applied a bottom-up method starting with a restricted number of sensors and computational power, relying more on the cognitive aspects of ACT-R. Our cognitive model, implemented in ACT-R, successfully simulates the human behavior and learning strategies while navigating in a labyrinth, and shows human performances. The cognitive model could be further enhanced by implementing a better simulation of human memory, for example adding a utility threshold under which a chunk is not retrieved. If the model is able to forget the less used junctions it would behave more like a human being. Some tests in this direction have been executed, but to get valuable results more work is needed on the parameter's tuning. At the moment the model has a perfect memory and the only source of error is the ambiguity between junctions. Another interesting enhancement is to allow the model to randomly forget or switch junction performances, as well as the user direction, like humans tend to do when the quantity of information they have to remember is too great. This feature could be implemented using the similarity function embedded in ACT-R.

Acknowledgments

The authors are grateful to Dan Bothell for extensive discussions during the coding phase, Tasuku Hiraishi who developed the nxt-lsp library, and Simon Büchner for discussions about his study and human navigation in general. This work has been partially supported by the SFB/TR 8 "Spatial Cognition" within project R8-[CSPACE] funded by the DFG.

References

Anderson, M. (2003). Embodied cognition: A field guide. *Artificial intelligence*, 149(1):91–130.

Borenstein, J. and Koren, Y. (1991). The vector field histogram and fast obstacle-avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7:278–288.

Bothell, D. (2004). ACT-R 6.0 reference manual. Technical report.

Breazeal, C., Siegel, M., Berlin, M., Gray, J., Grupen, R. A., Deegan, P., Weber, J., Narendran, K., and McBean, J. (2008). Mobile, dexterous, social robots for mobile manipulation and human-robot interaction. In *SIGGRAPH New Tech Demos*, page 27. ACM.

Brooks, R., Breazeal, C., Marjanović, M., Scassellati, B., and Williamson, M. (1999). The cog project: Building a humanoid robot. In *Computation for metaphors, analogy, and agents*, pages 52–87. Springer-Verlag.

Buechner, S., Hölscher, C., and Wiener, J. (2009). Search strategies and their success in a virtual maze. In *Proceedings of the 31st Annual Conference of the Cognitive Science Society*, pages 1066–1071.

Dietrich, F., Kan, J., et al. (2011). ACT-Rientierung. `http://sourceforge.net/projects/act-rientierung/`.

Harrison, A. and Trafton, J. (2010). Cognition for action: an architectural account for "grounded interaction". In *Proceedings of the 32nd Annual Conference of the Cognitive Science Society*, pages 246–251.

Hiraishi, T. (2007). Nxt controller in lisp. `http://super.para.media.kyoto-u.ac.jp/~tasuku/index-e.html`.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: a modern approach*. Prentice Hall, 2nd international edition edition.

Sofge, D., Trafton, J. G., et al. (2004). Human-robot collaboration and cognition with an autonomous mobile robot. *Intelligent autonomous systems 8*, page 80.

Trafton, J. (2009). An embodied model of infant gaze-following. Technical report, DTIC Document.