Task Learning Using Graphical Programming and Human Demonstrations

Staffan Ekvall, Daniel Aarno and Danica Kragic*
Computational Vision and Active Perception and Centre for Autonomous Systems
Royal Institute of Technology, Stockholm, Sweden
{ekvall, bishop, danik}@nada.kth.se

Abstract— **The next generation of robots will have to learn new tasks or refine the existing ones through direct interaction with the environment or through a teaching/coaching process in Programming by Demonstration (PbD) and Learning by Instruction frameworks. In this paper, we propose to extend the classical PbD approach with a graphical language that makes robot coaching easier. The main idea is based on graphical programming where the user designs complex robot tasks by using a set of low-level action primitives. Different to other systems, our action primitives are made general and flexible so that the user can train them online and therefore easily design high level tasks.**

I. Introduction

With robots moving out of factories to home and office environments, it is unfeasible to preprogram them for all tasks they are required to perform. Furthermore, the end-user cannot be expected to have a deep knowledge of programming or robotics algorithms and can thus not extend the robot's capabilities by classical programming implementation. It has been recognized that for successful deployment of autonomous systems in unstructured environments there is a need for online task learning and problem solving with and without interaction with a human, [1]–[5]. This also applies to medical and traditional factory settings where reprogramming of robots can still be a bottleneck in times of increased outsourcing and just-in-time production requirements. If the reprogramming is performed by anyone of the regular factory staff, robots can be used to solve a more diverse set of tasks and increase the throughput. One of the examples close to robot users is the Roomba vacuuming robot from iRobot, [6] which is designed for a single application. Its control panel allows a user to partially reprogram the robot by selecting different room sizes. Robotic systems that are able to perform more complex tasks commonly use text-based programming languages with few high-level abstractions, [7].

In cognitive psychology, it is well known that there are at least three different ways to learn how to solve problems, [8]. One way is by *discovery*, or exploration. A second way is by *instruction*, and a third way is by *observation*. The robotics community has been trying to replicate these three teaching modalities in different ways. For example, the Programming by Demonstration (PbD) paradigm [1], [9]–[11] is inspired by the observation whereas learning by discovery is related to reinforcement learning (RL) and similar methods. For teaching a robot by instruction, a *language* for human-robot communication is required. Low-level programming languages such as C++ have significant flexibility, but are the most difficult to use in case of regular users. In addition, they require a lot of tedious work when even simple tasks are considered. Spoken (high-level) languages like English are easy to use, but require complex translation to low-level programming languages before the robot understands and executes the task.

Related to the above, three scientific problems currently investigated in our research are: (1) **Synthesis:** the development of systems tools necessary for describing and implementing a PbD system; (2) **Modeling:** given sensor traces of a human performing a task, segmenting those traces into logical task components and/or measuring the compatibility of a given PbD structure to that sequence of components; and (3) **Validation:** measuring system's performance. In this paper, we study the synthesis and modeling issues by considering the problem of teaching a robot household and office tasks given a set of action primitives ranging from robot localization to object manipulation.

For similar applications, there have been examples where a set of simple behaviors (follow, stop, turn etc.) are pre-programmed and, during teaching, the right behavior is identified through dialog with the teacher, [12]. Conditional behaviors are however more difficult to model. It is in general hard to keep track of all instructions in a sentence such as "If X do A and B, otherwise if Y do C otherwise do D". Motivated by this, we propose to use a mid-level, graphical language for robot instruction. This way, the drawbacks of high-level languages are avoided and the user is given a detailed overview of what the goals of different steps in a task are. It is straightforward to monitor how the program branches at conditional behaviors and the user (teacher) is also free to add and change behaviors, something that is hard to achieve just using verbal communication.

Similar systems use only one of the three teaching modalities mentioned above when teaching robots new tasks. This often limits the range of the tasks that can be performed by the robot. We propose an architecture where graphical programming, PbD and skill refinement through practice can be used together with reasoning on different levels to allow a non-expert user to teach the robot a large set of new and meaningful skills in both autonomous and human-machine

This work has been supported by EU through the project PACO-PLUS, FP6-2004-IST-4-27657 and Swedish Research Council.

collaborative environments. One of the main contributions of this paper is the integration of graphical programming and PbD in a behavior based control framework.

This paper is organized as follows. In Section II, we shortly review the related work and present our system design. We continue by presenting the task description system in Section III and reasoning mechanisms in Section IV and give details on various PbD solutions in V. Experiments are described in Section VI and the paper is concluded in Section VII.

II. Motivation and System Design

Our system is based on four different levels of control, shown in Fig. 1. At the bottom level, there are robot platform dependent *action primitives*, implemented in C++. One or more primitives can be combined into a *behavior*, executed at the *execution level* which also deals with fault detection and error recovery. At the top, there is a *task level* where several behaviors are combined to reach the goal state. Variations of the approach with different levels have been widely and successfully used in systems such as ISR, DAMN, Saphira and others, [13]–[16] and favors platform portability. What makes our system unique is the training of the action primitives. Each action primitive comes as an untrained PbD problem, capable of learning from human demonstration within its domain. Then, the primitives are combined in the graphical framework in order to unite several domains and perform advanced tasks. Section V describes in detail how some of our action primitives are trained.

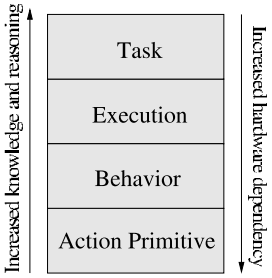

Fig. 1. Different control levels of the system.

The PbD paradigm has been successfully applied in several areas, ranging from the typical pick-and-place type tasks [17] to fine manipulation [10] and games such as air hockey [11] and marble mazes [18]. In classical approaches, it has only been possible to program variations of the same task. For example, a PbD system written to handle pick-and-place tasks can only handle pick-and-place tasks, where as a system that can be taught how to play air-hockey can only be used to play air-hockey. By combining PbD with a graphical task specification interface our system is able to support the PbD paradigm for a much broader set of tasks.

Systems that rely on graphical programming or task description have been proposed previously. As an example, MissionLab included a graphical editor that allowed users to construct programs in the configuration description language (CDL) [19]. The CDL can be used to describe a set of agents, the channels between them and the data-flow graph. In [20] a program called RoboGlyph was used to program various tasks for a PUMA 560 manipulator. These examples show that graphical programming can be successfully used by a regular end user who is only familiar with typical computer software such as, for example, word processors. The main difference to our work is that the low-level items, corresponding to our action primitives, cannot be trained by human demonstration.

Another example is the Behavior Composer in ERSP, [21], which is a graphical programming environment that allows the user to connect several behavior blocks and build a behavior based robot task. Compared to our work, the behaviors operate at a much lower level which means that more blocks are needed for accomplishing the same task. Again, the blocks cannot be trained as in our system. Instead, different parameters and thresholds have to be set. All behaviors are active in parallel, in opposite to our system which operates sequentially. Consequently, our system is more useful for teaching the robot sequential tasks, while Behavior Composer could be useful for constructing the low-level action primitives.

Most work presented in the areas of PbD, graphical programming, RL and behavior based robotics has focused mainly on one, or perhaps two of the teaching modalities. Our work is different in the way it integrates all teaching modalities to form a complete environment where appropriate methods can be applied at different levels in the architecture to solve more complex problems.

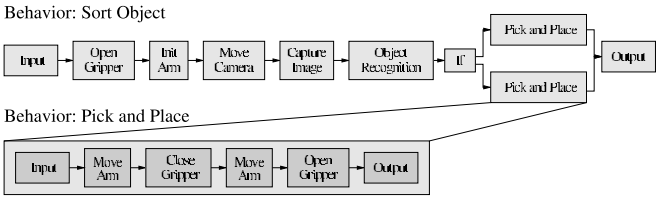

Fig. 2. A sorting behavior: sorting the object in front of the robot depending on the object type, and Pick-and-place behavior used by the sorting behavior.

III. Building a Task Description

In our system, there are three different teaching modalities together with a reasoning level that allows a robot to learn new tasks. In the system, the user is able to i) instruct the robot through a graphical programming interface (GPI), effectively providing an outline of a new behavior, ii) the user may then, by demonstration, train parts of this new behavior that need further specification, and finally, iii) the robot may carry out the task under supervision of the human, where the human has the possibility of giving feedback, such as corrective motions or penalizing/rewarding the robot for its decisions.

A. Action Primitives

The basic foundation in our framework are the action primitives. An action primitive is a flexible "black box"

that is implemented in C++ towards a specific hardware platform. It is flexible in the sense that it is possible to train it and black in the sense that the user has no detailed knowledge of how it works. This also supports the general idea that users do not have to know the details of a specific algorithm to be able to use it. An action primitive must be simple enough so that it can be combined with other primitives and reused in a variety of tasks. On the other hand, it also has to be expressive enough so that it is able to achieve the goal of the task. Example action primitives are `MOVE_ARM` or `CLOSE_GRIPPER`. Their implementations are platform dependent and therefore low-level.

B. Behaviors

Different action primitives can be connected using a GPI to form a behavior, as it will be described in Section III-D. Once a set of action primitives has been defined, it constitutes a behavior that can be reused. For example, the behavior `PICK_UP_OBJECT` consists of the primitives `MOVE_ARM ; VISUAL_SERVO ; MOVE_ARM ; CLOSE_GRIPPER`. The action can be stored and reused in another task. If training data for the primitives is present when storing it, the *trained* behavior is stored, i.e., if we have trained `PICK_UP_OBJECT` on the object `MILK`, we can store the behavior as `PICK_UP_MILK`, and the action will be executable without training it in some other task. Fig. 2 shows an example of a sorting behavior. The behavior contains another behavior, pick-and-place. Note that the two instances of the pick-and-place behavior do not share the same training data, so the robot will move the object to different positions dependent on its type.

C. Programming by Demonstration

In our framework, a GPI is used to determine the domain in which PbD is to take place. Using the GPI requires only moderate knowledge of the capabilities of the platform and no programming experience. Given the GPI, it is possible to use PbD to solve a larger set of tasks. For example, in the GPI it is possible to define a pick-and-place behavior by specifying the following sequence of action primitives: `PP_BHVR` = `LOCATE_OBJECT`; `VISUAL_SERV_ARM`; `GRASP_OBJECT`; `MOVE_ARM`; `RELEASE_OBJECT`.

The actions in `PP_BHVR` can now be trained to pick up an object and place it to a certain position. PbD is used primarily at the action primitive level, but can also be used at the behavior and task level. The majority of action primitives must be programmed before they can be executed, e.g., an object recognition primitive must first be demonstrated the object to recognize before it can actually recognize it. PbD can also be performed at the behavior level. By specifying that a number of `PP_BHVR` are to be executed in a sequence it is possible to demonstrate, for example, a *set table* behavior, [22].

By combining PbD with a GPI, it is therefore possible to solve a much larger set of tasks. The final step towards a more autonomous learning is to incorporate a learning by practice step, where the robot performs a task under the supervision of a human in such a way that the human can provide useful feedback. Feedback can be in many forms such as simple “good robot”/”bad robot”, by supplying further demonstrations or modifying the robot’s plan.

D. Graphical User Interface

The GUI used to instruct the robot is shown in Fig. 3. From a menu, the user can select from a wide variety of actions and behaviors, both trained and untrained. When connecting components, the GUI uses visual cues to match input types with the data provided by the active output. This can be seen as a highlighted (green) input on the `OBJECT_SIMILARITY` and `IMAGE_SIMILARITY` behaviors in Fig. 3, since they both require an image as input and that is the data that is provided by the `CAPTURE_HAND_IMAGE` action. The `MOVE_ARM` action on the other hand does not accept an image as input and its input connector is therefore disabled (gray). By use of tooltips the user can also easily investigate the different inputs and outputs of each action.

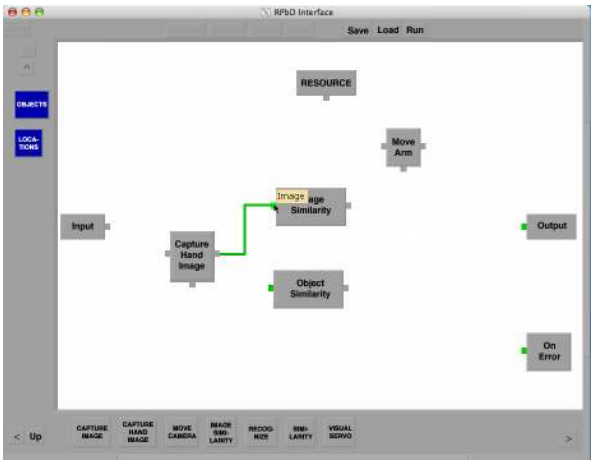

Fig. 3. The graphical programming interface.

IV. Planning

The behaviors are stored in the behavior database. In addition to the name of the behavior, inspired by the STRIPS planner [23], the *preconditions* and *effects* are also stored. As an example, the behavior `PICK_UP_OBJECT` is represented in the XML behavior database as following:

```
<BEHAVIOR> PICK_UP_OBJECT
  <VARIABLES> x </VARIABLES>
  <PRECOND> OBJECT_IN_FRONT
    <VARIABLES> x </VARIABLES>
  </PRECOND>
  <EFFECT> HOLDING
    <VARIABLES> x </VARIABLES>
  </EFFECT>
  <EFFECT> !OBJECT_IN_FRONT
    <VARIABLES> x </VARIABLES>
  </EFFECT>
  <FILE> pick_up_object.bhr </FILE>
</BEHAVIOR>
```

In this case, before an object can be picked up, it has to be in front of the robot. The effect is that the object will be held by the gripper and no longer in front of the robot. By specifying the preconditions and effects for each

behavior, the robot can search the behavior database and automatically plan a sequence of behaviors that eventually fulfills the end conditions. When planning for the behavior PICK_UP_OBJECT, the robot would find that either the behavior MOVE_TO_OBJECT or PLACE_OBJECT has to come immediately before, as they are the only behaviors that fulfill the precondition. The user is also allowed to add new behaviors to the behavior database. New behaviors are automatically taken into account by the planner and next time a task is executed, new behaviors will be considered, e.g., DROP_OBJECT.

This approach relieves the user of some tedious task description work. Instead of having to specify all the behaviors for, e.g., FETCH_MILK it is enough to just describe the behavior POUR_MILK with the precondition that the robot is already holding the milk. The steps for acquiring the milk are automatically generated by the planning algorithm. When there are several ways to satisfy preconditions the robot must learn from experience and decide which one to select. For example, if the robot chooses to go to the store instead of the kitchen to fetch milk, the user might penalize it through the skill refinement system so that the most preferred place (kitchen) is always considered first.

V. Action Primitive Training

One of the main contributions of this work is the design of flexible action primitives that can be trained by human demonstrations. In this section we provide examples of how training is done for specific behaviors. Several action primitives have been implemented in the system, some of them are listed below. We avoid a detailed description and refer to our previous work instead, [24], [25].

MOVE_ARM Robot arm movement along a prerecorded trajectory. Stops if colliding with an obstacle. Trained by the user by physically dragging the gripper to the desired position and orientation. A force/torque sensor is used together with inverse kinematics to calculate the desired pose.

CLOSE/OPEN_GRIPPER Two different primitives for controlling the gripper. No training necessary.

IMAGE_SIMILARITY Calculates the similarity between a stored image and a given image using Receptive Field Cooccurrence Histograms [24].

OBJECT_SIMILARITY Similar to the previous, but in the training phase a specific object and not the entire image is learned. Object segmentation is performed using image differencing and morphological operations so that a segmented training image is automatically generated.

OBJECT_RECOGITION Evaluates if the object in front of the camera is exactly the same as the training object using SIFT-features [26]. Same training procedure as with OBJECT_SIMILARITY. The two primitives OBJECT_RECOGNITION and OBJECT_SIMILARITY complement each other. The former is used for recognizing a specific object, rich in local features and the latter is used for recognizing object categories, or for recognizing a specific object with few or no local features.

MOVE_TO A navigation primitive that moves the robot to a specific place. The navigation is SLAM-based using the SICK laser scanner and sonars to avoid unexpected local obstacles.

A. Learning Object Recognition

In our framework, objects are learned from human demonstrations using two steps. First, the robot observes the background. Then the operator places an object in front of it and the object is segmented using image differencing, relieving the user of manually extracting the object. Image differencing is followed by a number of incremental morphological operations to achieve better segmentation (errode - dilate - errode), [27]. These operations are performed using information from the original image, i.e., a growing effect (covering holes) will not add black pixels but pixels from the original image. The result of this step can be seen in Fig. 4.

Fig. 4. Left: The original image. Center: The result after image differencing. Right: The result after morphological operations

A problem with image differencing is the choice of a threshold θ that determines if a pixel is part of the background or not. If θ is set too high, too much of the background will remain. If θ is set too low, significant parts of the object may be omitted. In our work, we use an automatic adjustment of θ based on the result of the differencing performance. If image differencing was successful, the remaining pixels should be concentrated to a single area where the object has moved. If the differencing has failed, the pixels are mostly scattered around the entire image. Thus, the success is measured in terms of detection variance. In addition, a penalty that is linearly proportional to the number of pixels remaining is added to cover the case of very few remaining pixels that have a low variance but are not sufficient for the object representation. The algorithm tests every θ from 1 to 150, to find the optimal setting with the lowest score.

B. Learning Arm Movements

Many tasks require the user to guide the robot arm, either for teaching the robot how to move the arm along a specific trajectory or to position the camera mounted on the end-effector relative to the object. This can be achieved using a keyboard or a joystick but these devices are not intuitive for controlling a 6-DOF robot arm. Instead, we use the force/torque sensor attached to the end-effector and an arm movement is taught by simply dragging the arm to the desired pose. The compliant control of the arm is briefly described below.

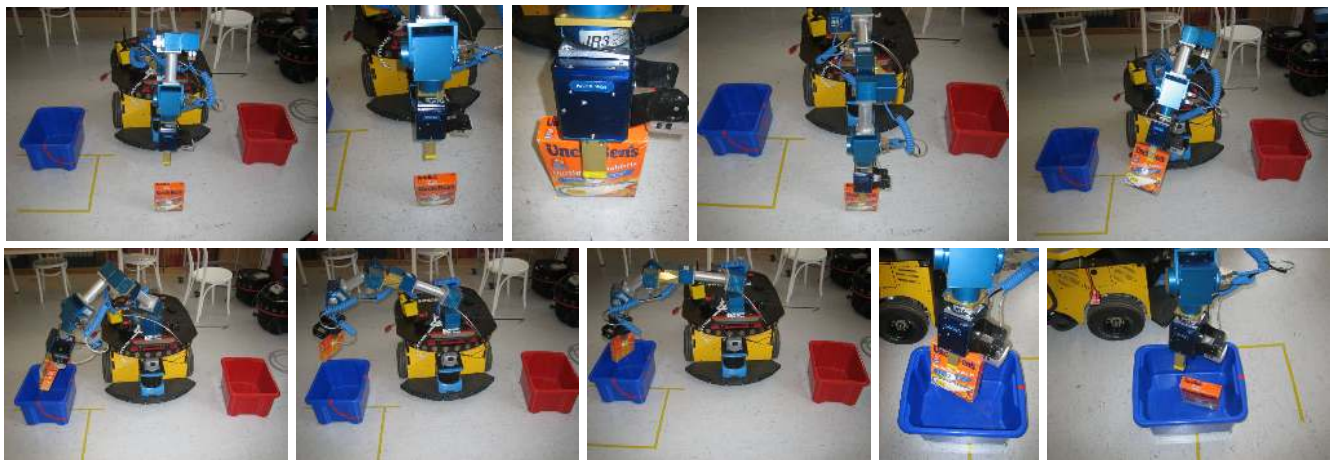

Fig. 6. The robot executing the task. The rice package has been recognized and moved to the left bin as demonstrated by the user.

1) Arm Control: The manipulator is equipped with a JR3 force/torque sensor mounted between the end-effector and the last link, providing 6 DOF force/torque measurements. It provides decoupled data at 8 kHz per channel, which is low-pass filtered with the bandwidth 30 Hz (-3 dB) by a DSP. The data is first read from the DSP and the current arm configuration is then used to subtract the influence of gravity on the end-effector. The force/torque vector is then transformed to the base frame attached to the base of the mobile platform. If the magnitudes of the force and torque are both below a threshold the velocity of all joints are set to zero. Otherwise, the Cartesian velocity of the arm is set to be proportional to the force. The same applies to the torque. The Cartesian velocities are then transformed to joint velocities of the arm using the inverse kinematics.

C. Training the Navigation Primitive

When the program flow reaches an untrained `MOVE_TO` primitive, it will stop and ask for advice. The user can then teach the system where to move by i) asking the robot to follow him/her, or ii) by controlling the robot directly with a joystick. The `FOLLOW` behavior is implemented by tracking the user's legs with the laser scanner. However, it is easier to teach more precise locations using the joystick. As the robot moves, it drops virtual roadmap nodes in its map and automatically connects these. The nodes indicate free space and the roadmap tree enables the robot to quickly find the fastest way to a specific location. For more details, see [25].

Fig. 5. The user drags the robot arm to show it how to grasp the object, and move the object to the bin.

VI. Example Tasks

The experimental platform is an ActivMedia PowerBot, Fig. 6. It is a non-holonomic differential drive platform with a 6 DOF robotic manipulator on the top. It has a SICK LMS200 laser scanner, 28 Polaroid sonar sensors, a Canon VC-C4 pan-tilt-zoom camera and a Firewire camera on the last joint of the arm.

One of the behavior that was designed using the system is an object sorting task. The main goal of the task is to sort two objects shown in the small image in Fig. 5. The rice package should be sorted to the left bin, and the raisins package to the right bin, shown in the larger image in Fig. 5. First, if the behavior is not constructed, the user drags the blocks according to Fig. 2 and then presses the run button. The program starts by opening the robot gripper, then initializes the arm for movement but stops at the `MOVE_CAMERA` primitive because this primitive has not yet been trained. The user is the asked to rotate the camera to the desired direction using the keyboard. When satisfied, training of this primitive is complete and the value is stored for future use. The robot then captures an image of the object, which requires no training. The program then stops again, at the object recognition block which is untrained. The training proceeds according to Section V-A and the robot learns to recognize the rice box. Then, the robot executes the learned primitive and realizes that the rice box is indeed present in the image. Thus, the upper `PICK_AND_PLACE` behavior is executed. However, this behavior is also untrained and the individual arm movements are shown by the user which drags the robot arm to the correct positions, see Fig. 5. The user then executes the task again, this time with the rice package not in the image. Thus, the rice package is not recognized and the program chooses the lower branch in Fig. 2, and once again asks for instruction. After the second movement have been shown, the program is complete. A few example images taken during robot performing the learned

sorting task are shown in Fig. 6. Currently, the approach requires the object to be in a precise pose. We plan to incorporate a VISUAL_SERVO primitive that will allow the object to be just roughly at the same position. The robot executed the task 10 times, with 100 % success rate. This was expected, as it is easy to separate the two objects visually and the grasping action cannot fail as long as the object is placed at the correct position. Although this experiment is not particularly challenging, we believe that it demonstrates the concept of our approach. The human is able to teach the robot a combined behavior that is not within any of the PbD domains of each individual primitive.

Another example of a behavior is the STATUS_CHECK behavior. It involves both navigation, camera movement and computer vision, and can be trained for many different tasks, e.g., checking if a certain button is pressed, or if a door is open or closed. The behavior uses two image similarity actions and picks the training image that is most similar to the test image. This behavior can be used by higher order behaviors, for example if the robot takes the elevator, it can use the STATUS_CHECK to verify that the button has been pressed successfully. Naturally, the success rate of this behavior is highly dependent on what it is used for.

VII. Conclusions

Robots that are to operate in everyday, dynamic environments such as homes and offices need learning mechanisms that allow for adaptation to the surrounding. As the reasoning capabilities of robots are still quite limited, most of the high level knowledge is acquired through interaction with humans. In this paper, we have presented a mid-level communication tool for teaching robots different tasks in a Programming by Demonstration framework. The tool is based on graphical programming which does not require any programming skills making it suitable for regular users.

One of the main contributions of this paper is the integration of graphical programming and Programming by Demonstration in a behavior based control framework. The proposed system allows the user to specify the task structure, and then, using demonstrations, instruct the robot exactly how to perform the task following the task structure. Using this method, we have successfully taught a mobile robot several behaviors, such as SORT and STATUS_CHECK. An important issue we are dealing with is fault handling and error recovery. Currently, all action primitives can report success or failure but to achieve more robust and flexible performance, the system has to have the support so that the user can instruct the system what to do in case of failure. This will be the strongest topic for our future research.

References

[1] M. J. Matarić, "Getting humanoids to move and imitate," in *IEEE Intelligen Systems*, pp. 18–24, jul 2000.

[2] T. Fong, I. Nourbakhsh, and K. Dautenhahn, "A survey of socially interactive robots," 2002. citeseer.ist.psu.edu/article/fong02survey.html.

[3] M. Kleinehagenbrock, J. Fritsch, and G. Sagerer, "Supporting advanced interaction capabilities on a mobile robot with a flexible control system," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, vol. 4, pp. 3469–3655, Oct. 2004.

[4] F. Michaud, Y. Brosseau, C. Cote, D. Letourneau, P. Moisan, A. Ponchon, C. Raievsky, J.-M. Valin, E. Beaudryy, and F. Kabanza, "Modularity and integration in the design of a socially interactive robot," in *IEEE International Workshop on Robot and Human Interactive Communication, 2005. ROMAN*, pp. 172–177, Aug. 2005.

[5] A. M. Steinfeld, T. W. Fong, D. Kaber, M. Lewis, J. Scholtz, A. Schultz, and M. Goodrich, "Common metrics for human-robot interaction," in *2006 Human-Robot Interaction Conference*, ACM, March 2006.

[6] iRobot, "Roomba robotic vacuum cleaner," in *http://www.roombavac.com*, 2003.

[7] G. Biggs and B. MacDonald, "A survey of robot programming systems." citeseer.ist.psu.edu/biggs03survey.html.

[8] J. R. Anderson, *Cognitive Psychology and its Implications*. 41 Madison Avenue, New York, NY 10010: Worth Publishers, 5th ed., 2002.

[9] C. Atkeson and S. Schaal, "Robot learning from demonstration," in *In Machine Learning: Proceedings of the Fourteenth International Conference (ICML '97) (ed. D. H. Fisher Jr.)*, pp. 12–20, July 1997.

[10] R. Zöllner, O. Rogalla, R. Dillmann, and M. Zöllner, "Understanding Users Intention: Programming Fine Manipulation Tasks by Demonstration," in *IEEE International Conference on Intelligent Robots and Systems*, 2002.

[11] D. C. Bentivegna and C. G. Atkeson, "Learning From Observation Using Primitives," in *IEEE International Conference on Robots and Automation*, 2001.

[12] R. Arkin, *Behavior-based Robotics*. Intelligent Robotics and Autonomous Agents series, Cambridge, MA: MIT Press, 1998.

[13] H. I. Christensen and P. Pirjanian, "Theoretical methods for planning and control in mobile robotics," in *1st International Conference on Conventional and Knowledge Based Intelligent Electronic Systems (KES-97)*, vol. 1, pp. 81–86, IEEE Computer Society, April 1997.

[14] K. Konolige, K. L. Myers, E. H. Ruspini, and A. Saffiotti, "The Saphira architecture: A design for autonomy," *Journal of experimental & theoretical artificial intelligence: JETAI*, vol. 9, no. 1, pp. 215–235, 1997.

[15] J. K. Rosenblatt, *DAMN: A Distributed Architecture for Mobile Navigation*. PhD thesis, Carnegie Mellon University Robotics Institute, 1997.

[16] M. Lindström, A. Orebäck, and H. Christensen, "BERRA:A research architecture for service robots," in *IEEE ICRA*, vol. 4, pp. 3278–3283, 2000.

[17] R. Zöllner, M. Pardowitz, S. Knoop, and R. Dillmann, "Towards Cognitive Robots: Building Hierarchical Task Representations of Manipulations from Human Demonstration," in *IEEE ICRA*, 2005.

[18] D. C. Bentivegna, C. G. Atkeson, and G. Cheng, "Learning From Observation and Practice Using Primitives," in *AAAI Fall Symposium Series, Symposium on Real-life Reinforcement Learning, October 22-24*, 2004.

[19] D. C. MacKenzie, J. M. Cameron, and R. C. Arkin, "Specification and Execution of Multiagent Missions," 1995.

[20] L. D. Spencer, *Graphical Programming Language for Service Robots in Semi-Structured Environments*. PhD thesis, Stanford University, 1994.

[21] M. Munich, J. Ostrowski, and P. Pirjanian, "ERSP: A software platform and architecture for the service robotics industry," in *Proc. IEEE International Conference on Intelligent Robots and Systems*, 2005.

[22] S. Ekvall and D. Kragic, "Integrating object and grasp recognition for dynamic scene interpretation," in *IEEE International Conference on Advanced Robotics, ICAR'05*, 2005.

[23] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence 2*, pp. 189–205, 1971.

[24] S. Ekvall and D. Kragic, "Receptive field cooccurrence histograms for object detection," in *IEEE/RSJ IROS*, 2005.

[25] P. Jensfelt, S. Ekvall, D. Kragic, and D. Aarno, "Integrating slam and object detection for service robot tasks," in *IROS 2005 Workshop on Mobile Manipulators: Basic Techniques, New Trends and Applications*, Edmonton, Canada: IEEE/RSJ, 2005.

[26] D. Lowe, "Object recognition from local scale-invariant features," in *International Conference on Computer Vision*, pp. 1150–1157, 1999.

[27] R. Gonzalez and R. Woods, *Digital Image Processing*. Addison Wesley Publishing Company, 1992.