ORIGINAL ARTICLE SPECIAL ISSUE: Intelligent Robotics (2/2) Guest edited by Yi-Hung Liu

Behavior-based Task Learning by Demonstration for Mobile Manipulators

Shu Huang[1,*], Erwin Aertbeliën[2], Herman Bruyninckx[2], and Hendrik Van Brussel[2]

[1]*Division of Intelligent Robotics Technology, Industrial Technology Research Institute, Taiwan*

[2]*Division of Production Engineering, Machine Design and Automation (PMA), Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium*

(**Received** 5 October 2012; **Accepted** 25 October 2012; **Published on line** 1 March 2013)

*Corresponding author: ShuHuang@itri.org.tw

DOI: 10.5875/ausmt.v3i1.161

Abstract: Task learning for behavior-based mobile manipulation is formalized as a behavior recognition problem in this paper. The goal is to generate a behavior diagram through human demonstration, which can be used as a template for execution of a specified task. In order to extract significant and meaningful information from sensory data, various features are defined to convert the sensory data to the feature space. A generic task learning approach based on combined machine learning techniques is used to classify behaviors from the feature space. Not only can this approach recognize already-known behaviors, it can also provide an easy method to add new behaviors to a system by combining new training datasets with the existing ones through retraining a support vector machine kernel (SVM) model. Common household applications, such as door-opening and window-cleaning tasks, are implemented for the evaluation of the proposed method.

Keywords: Behavior-based; Mobile manipulation; Programming by demonstration

Introduction

The purpose of most automation applications is to reduce routine jobs for human workers and to increase precision and efficiency in manufacturing processes. Based on these requirements, engineers design machines and write programs to perform desired tasks. In this case, machines (or robots) that simply execute specific programs are made to replace humans; this functionality is generally categorized as the *task execution* domain. As such, the role of machines in this domain is relatively passive.

Gradually, with the growth of the automation industry, robots with only task execution abilities became insufficient to meet increasingly demanding requirements. Robots with a degree of artificial intelligence were needed in addition to those capable only of performing predefined actions. Eventually, robots with the ability to automatically generate program code for further execution — or even function without code entirely — will be desired. To succeed in this feat, the design of a robot must be approached from a different perspective. In fact, designing a robot to think or act like a human is still a very difficult challenge. However, with the help of machine learning techniques and sensor technology, it is possible to develop a robot capable of learning a simple task under a reasonable set of conditions. *Task execution* consists of applying one or more controls in order to achieve a desired task, while *task learning* is the process of recognizing which control is being demonstrated.

Literature Survey

Most research on behavior-based task learning is focused on mobile robots. In general, task learning on behavior-based systems can be separated into two types: *behavior segmentation and recognition,* and *behavior coordination*. Han *et al.* introduced a novel framework using a Hidden Markov Model (HMM) to represent and recognize strategic behaviors of robotic agents, particularly designed for robot soccer applications [1]. The challenges of high-level behavior recognition from observed low-level state features were addressed. In addition, the perceived signal was characterized in terms of behavior-relevant state features. States in the HMMs correspond to an abstracted decomposition of the robot's behavior. Intermediate state probabilities are an indicator of a behavior that is in progress.

Koenig *et al.* [2] made a reasonable attempt to address task learning by demonstration on behavior-based robots. A method for automatic segmentation of a complex demonstration into an ordered set of simpler behaviors was proposed, with the detection of the segmentation points performed by continuous calculation of a cost function within a sliding time window. In addition, only simple mathematical transformations suitable for runtime conditions were employed and later applied on their mobile robots. These transitions between behaviors were termed *behavior boundaries* by the authors.

Billing *et al.* focused on behavior recognition for *learning from demonstration* (LFD) [3]. They concluded that recent work in LFD often consisted of the identification and selection of *behavior primitives*, or *skills*. Two methods based on a dynamic temporal difference algorithm, *predictive sequence learning* (PSL), were presented and evaluated. The biggest advantage of their method is that the forward computation (controller) and the inverse computation (recognition) are based on two aspects of the same model. They use tele-operation to drive a mobile robot with the aim of recognizing three behaviors — *approaching*, *corridor driving,* and *unloading* — in completing a *load-transport-unload* task. Currently, their method is applicable only for mobile robots. However, it is thought that it can be extended and applied to other domains, such as mobile manipulation.

Little research has addressed task learning of behavior-based mobile manipulators. Corona-Castuera *et al.* claimed to have developed a behavior-based approach for learning peg-in-hole operations from scratch [4]. The robot learns the initial mapping between *contact states* to *motion commands* autonomously, by employing fuzzy rules and creating an acquired-primitive knowledge base (ACQ-PKB). The *adaptive resonance theory* (ART) model was used to train the input patterns and desired output classes.

Behavior-based Task Learning

Approaches to robot control vary depending upon the type of robot and its final applications. The behavior-based control methodology, which keeps a balance between the *deliberative control methodology* and the *reactive control methodology*, is especially suitable for operating in diverse, unpredictable, or rapidly changing environments [5]. Generally speaking, a

Dr. **Shu Huang** was born in Taipei, Taiwan, in 1975. He received his M.Sc. in Mechanical Engineering from National Chiao Tung University, Taiwan in 1999. He worked as an Associate Researcher in the Mechanical and Systems Research Laboratories (MSL) at the Industrial Technology Research Institute (ITRI) in Taiwan from 2000 to 2006. From 2006 to 2011, he was a Research Assistant and member of the Mobile Learning Robot group at Katholieke Universiteit Leuven (KULeuven), Belgium. His research interests include behavior-based mobile manipulation, constraint-based task specification, programming by demonstration, and intuitive task learning. He received his Ph.D. in Mechanical Engineering from KULeuven in June 2011. Currently, he is the deputy manager and a full-time researcher in the Control and Sensing Technology Department, Intelligent Robotics Technology Division, Mechanical and Systems Research Laboratories, ITRI. His research work is on developing robot controllers, and human-robot interaction for industrial applications. His objective is to integrate robot technology into Taiwan's industry.

Dr. **Erwin Aertbeliën** is a senior researcher in the faculty of Mechanical Engineering at Katholieke Universiteit Leuven (KULeuven). His research concentrates on different aspects of human-robot interaction: programming by demonstration of process-skills, such as deburring; physical human-robot interaction using impedance and hybrid force-position control; advanced task specification for manipulation tasks; and the interpretation of human motion in clinical gait analysis. His recent research focuses on the control, task specification, and learning of the lower-limb exoskeleton. A second focus of Dr. Aertbeliën's research is the application of advanced robotics in industry for tasks such as deburring, grinding, robot-assisted plate bending, and assembly. A third research interest is software architecture and software support needed for realistic robot control. These open source developments are available at www.orocos.org.

Prof. **Herman Bruyninckx** obtained Masters Degrees in Mathematics (Licentiate, 1984), Computer Science (Burgerlijk Ingenieur, 1987) and Mechatronics (1988), all from the Katholieke Universiteit Leuven, Belgium. In 1995, he obtained his Doctorate Degree in Engineering from the same university, with a thesis entitled 'Kinematic Models for Robot Compliant Motion with Identification of Uncertainties.' He is a full-time Professor at KULeuven, and has held visiting research positions at the Grasp Lab of the University of Pennsylvania, Philadelphia (1996), the Robotics Lab of Stanford University (1999), and the Kungl Tekniska Hogskolan, Stockholm (2002). Since 2007, he has been Coordinator of the European Robotics Research Network EURON. His current research interests are on-line Bayesian estimation of model uncertainties in sensor-based robot tasks, kinematics and dynamics of robots and humans, and the software engineering of large-scale robot control systems. In 2001, he started the Free Software ("open source") project Orocos to support his research interests and to facilitate their industrial exploitation.

Prof. **Hendrik Van Brussel** (1944) is full professor in mechatronics and automation at Katholieke Universiteit Leuven, Belgium. He was a pioneer in robotics research in Europe and an active promoter of mechatronics as a new paradigm in machine design. He has published extensively on robotics, mechatronics, and flexible automation. His present research interests include mechatronics, medical and behavior-based robotics, holonic manufacturing systems, and precision engineering. He is a former president of CIRP (International Academy for Production Engineering) and euspen (European Society for Precision Engineering and Nanotechnology).

task can be divided into a series of subtasks. Accomplishing these subtasks involves the activation of several behaviors within each subtask. For this reason, task learning in a behavior-based system can be transformed into a behavior recognition problem; that is, a problem of recognizing which behaviors are activated within a subtask. To learn a task, useful information must be extracted from the sensory data.

In behavior-based systems, elementary behaviors are the basic encapsulating units for complex tasks. The complex task is achieved by activating one or more behaviors in a time-sequential order. Upon activation of one behavior, the control criteria will produce certain sensory patterns in the sensory data (for instance, the increase of torque or force for a door-opening task). The mission here is to find a generic approach to categorize these patterns, as well as a method to recognize each of them.

In order to generate a behavior diagram from demonstration, a general method is required to recognize the behaviors executed in each subtask. Although systems engineers predefine these behaviors — and thus their patterns are known in advance — a more general method, which can also recognize dynamically generated behaviors where the patterns might be unknown, is required. While these patterns may not be observed or extracted directly from the sensory space, they can be transformed by features and may be recognized in a higher-dimensional space. The proposed method attempts to construct the procedures by using feature vectors to extract significant patterns from the sensory data to the feature space, and static classifiers to identify the mapping from the feature space to specific behaviors. Since the execution of a certain behavior is already predefined, the mapping from sensory data to behaviors allows ambiguities and imprecision to be found. Although the mapping is accomplished by static classifiers based on teaching certain behaviors to a specific robot, the presented method can be applied to various robots with different configurations. It is worth noting that the static classifier proposed in this approach is one of the many methods that can be used to recognize the mappings. Other methods may also be applicable, with different advantages or limitations. However, the static classifier is chosen in this paper as a first attempt at solving the problem.

Feature Extraction

Sensor information, such as force readings or raw encoder data, represents what can be physically measured during a demonstration. These sensory data are the combined results from the user demonstration of a desired task. In order to extract meaningful relationships or hidden information from the data, further processing is required. In this paper, a general set of features is applied as a filter for processing of the raw sensory data.

Extracting information from sensory data is never straightforward. Sometimes, sophisticated computations are required to elucidate the correlations between sensor data and behaviors. A good example is the detection of the unscrewing of a bottle cap by trajectory monitoring. Treatment of the raw data in this case may require a regression analysis of all recorded data points, or curve-fitting calculations. In the present paper, we propose a method that uses a set of feature vectors, each representing a specific type of interaction with the environment. It is noted that these feature vectors should be as simple as possible, with complex features being decomposed into a combination of more basic features. For the example above, the *unscrewing* behavior can be recognized as a combination of *turning* and *pulling* behaviors.

Features, in the context presented here, contain individual heuristic properties of the phenomena being observed. These features act as virtual sensors, trying to detect specific meaningful information from sensory data. Finding discriminating and independent features are important in the field of pattern recognition. Correct feature vectors increase the efficiency of the learning algorithms in identifying the mappings. However, it is difficult to say which features are important and how many features are sufficient. Finding a general method to cope with the diversity and variety among different behaviors and users is a challenge. For these reasons, a reasonably sized feature pool is defined, from which features are to be selected. Moreover, machine learning techniques are used to select relevant features automatically.

The design of features should satisfy an overarching generality so as to remove their dependence on absolute environmental information. Below are some examples of the general features we use in our system.

Plane feature

One of the basic geometric features that emerge when demonstrating a task is the *plane*. The purpose of this feature is to detect when a demonstrated movement approximately occurs on a plane — that is, when the normal vector of the best-fitting plane for a dataset stays appreciably in the same direction as that of the previous normal vector during a time period. This feature reflects certain environmental relationships when executing a behavior, and provides the classifier with useful information for specific behavior recognition. A point in

space is expressed as $p_i = \begin{bmatrix} x_i & y_i & z_i \end{bmatrix}^T$ where x_i, y_i, and z_i are coordinates of the point p at the time instance i. A plane is fitted through the previous n points, $p_{i-(n-1)},\ldots, p_i$, such that the summed squared distance of the points to the plane is minimized. This plane is parameterized by an origin and a vector normal to the plane. We first calculate the mean of these points (p_m), and express all points relative to this mean: $\tilde{p}_i = p_i - p_m$. Thus, the origin of the plane corresponds to the mean p_m [6]. An $n\times3$ matrix containing the data points is formed as follows:

$$\mathbf{M} = \begin{bmatrix} \tilde{p}_{i-(n-1)}^T & \ldots & \tilde{p}_{i-1}^T & \tilde{p}_i^T \end{bmatrix}^T. \quad (1)$$

With this matrix, the least-squares fitting plane is obtained through the method of [6]. This yields the eigen-problem $(\mathbf{M}^T\mathbf{M})a=\lambda a$. According to [7], the eigenvectors of $\mathbf{M}^T\mathbf{M}$ are equal to the singular vectors of $\mathbf{M}$. Therefore, if we calculate the singular value decomposition (SVD) of $\mathbf{M}$ and the singular vectors corresponding to the smallest singular value, the normal vector of the best-fitting plane among these points can be obtained. This method is equivalent to computing the eigenvalue decomposition of the covariance matrix of data points $\mathbf{M}$. The point matrix $\mathbf{M}$ can be expressed as $\mathbf{M}=\mathbf{UDV}^T$, where $\mathbf{U}$ is an $m\times m$ unitary matrix, $\mathbf{D}$ is an $m\times n$ diagonal matrix with the diagonal entries known as the singular values of $\mathbf{M}$, and $\mathbf{V}^T$ is an $n\times n$ unitary matrix. In fact, the singular value decomposition and eigen-decomposition are closely related. For example, the right singular vectors of $\mathbf{M}$ are the eigenvectors of $\mathbf{M}^T\mathbf{M}$ as mentioned previously, and the non-zero entries of $\mathbf{D}$ (singular values of $\mathbf{M}$) are the square roots of the non-zero eigenvalues of $\mathbf{M}^T\mathbf{M}$. As a result, we define the plane feature in the experiment as:

$$F_{plane,i} = \left| \sigma_{\min}\left(\mathbf{M}_i\right)^T \sigma_{\min}\left(\mathbf{M}_{i-1}\right) \right| \quad (2)$$

where $F_{plane,i}$ is the plane feature at each time instance i and $\sigma_{\min}(\mathbf{M}_i)$ is the singular vector corresponding to the smallest singular value at time instance i. Since $\mathbf{V}$ is a unitary matrix, we can obtain the cosine of the angle between the normal vectors of the current and the previous time instance.

Notice that the *plane feature* here is not limited to the exact movement on a plane. As this feature calculates points within a period of time with some tolerance, the overall points are considered on the same plane as long as the normal vector does not change significantly. For example, moving along the surface of a big cylinder slowly can also be considered as moving on a plane. The purpose is to extract further useful information from raw sensory data, rather than merely to find the best-fitting plane among some points. The exact shape or curvature of the surface is less important in the behavior-based system. During the execution, this shape or curvature will be implicitly accounted for by the interactions with the environment. Therefore, small feature changes will still be considered as the same behavior by the classifier. In addition, the purpose of this feature is not to find the parameters or dimensions of the model, but to indicate to the classifier the particular geometric relationship of the cloud of points. Shakarij describes other geometric fitting applications, such as sphere fitting, cylinder fitting, or cone fitting, but we wish to avoid complex features that require heavy computations in our system [6]. Therefore, the choice of defining features is preferentially kept simple.

Line feature

Another useful geometrical feature is the *line feature*. This feature indicates when the cloud points approximately follow a line in space. The implementation of the line feature is similar to the plane feature described above. According to [6], the same equations $(\mathbf{M}^T\mathbf{M})a=\lambda a$ can be used as in the case of plane fitting. A point on the line is given by p_m, and the direction of the line is given by the eigenvector corresponding to the largest eigenvalue [6]. Therefore, the line feature in the following experiments is defined as:

$$F_{line,i} = \left| \sigma_{\max}\left(\mathbf{M}_i\right)^T \sigma_{\max}\left(\mathbf{M}_{i-1}\right) \right| \quad (3)$$

where $F_{line,i}$ is the line feature at each time instance i, and $\sigma_{\max}$ $(\mathbf{M}_i)$ is the singular vector corresponding to the largest singular value at time instance i. This feature provides information about the angle or the direction of the best-fitting line between the current and previous time instance.

Velocity-force feature

In addition to these and other geometrical features, relationships between other kinds of sensory data may also be considered features. These represent certain degrees of correlation between sensors. In our experiment, for example, the trainer uses a six degrees-of-freedom (DOF) spacemouse to demonstrate a desired task. The spacemouse commands are transformed as velocity commands to the end-effector of the robot. During the guided movements, the interaction forces between the end-effector and the environment are measured and recorded. The command velocity in the above scenario indicates the intention of the user and the interaction force reflects the condition of the

environment. The variance of user intentions with respect to changes in the environmental interaction force is considered useful information. Thus, the relationship between the command velocity and the resulting force can be defined as a feature.

Instead of investigating the exact correlations between these two factors, velocity commands and force readings are defined as two separate potential feature candidates in the feature pool:

$$F_{velocity,i} = \dot{p}_i = \begin{bmatrix} \dot{x}_i & \dot{y}_i & \dot{z}_i \end{bmatrix}^T, \quad (4)$$

$$F_{force,i} = f_i = \begin{bmatrix} f_{x,i} & f_{y,i} & z_{z,i} \end{bmatrix}^T, \quad (5)$$

where $F_{force,i}$ is the force feature, $F_{velocity,i}$ is the velocity feature, and f_i and $\dot{p}_i$ are the force and velocity of the end-effector at time instance i, respectively.

Feature Mapping

Decision tree learning

A *decision tree* is a decision support tool with a tree-like model of decisions and their possible consequences [8]. In data mining, a decision tree describes data but not decisions; instead, the resulting classification tree can be an input for decision making. A tree can be learned by splitting the dataset into subsets based on an attribute value test. This method can therefore be used to select relevant features from the feature pool. The primary advantage of a decision tree is that it assigns exact values to the outcomes of different actions. It has few requirements for data preparation, and possesses the ability to handle both numerical and categorical data. Moreover, it is possible to evaluate the model by common statistical tests. The algorithms that are used for constructing decision trees usually work by choosing a variable at each step that represents the next best variable to use in splitting the set of items into subsets.

In data mining, trees can also be described as a combination of mathematical and computational techniques to aid the categorization and generalization of a given set of data. Data comes in records of the form: $(x, Y) = (x_1, x_2, x_3, \ldots, x_k, Y)$. The dependent variable, Y, is the target variable that must be classified. The vector x is composed of the input variables, $x_1, x_2, x_3, \ldots, x_k$, that are used for the performed task.

The algorithm used for the decision tree is C4.5, which generates a decision tree developed by Ross Quinlan [9]. It is an extension of Quinlan's earlier ID3 algorithm. At each node of the tree, C4.5 chooses one attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. The decision trees generated by C4.5 are classifiers, which is the reason C4.5 is often referred as a statistical classifier.

Support vector machines

A *support vector machine* (SVM) is a set of related supervised learning algorithms that analyze datasets and recognize patterns [10]. As such, an SVM is a classifier, which, when given a training dataset, will generate a model to predict whether a new sample falls into one category or another. Supposing some given data points belong to one of two classes, and the goal is to decide which class a new data point should fall in, if a linear classifier is first used, the separation will be achieved by high-dimensional hyperplanes. However, there are many hyperplanes that might classify the data. One reasonable choice for the best hyperplane is the one that represents the largest separation between the two classes. In other words, the hyperplane whose distance to the nearest data point on each side is maximized would be an appropriate hyperplane to choose.

To learn the mapping from the feature space to manipulator behavior, the traditional SVM should be extended to a *multi-class* SVM, since the number of behaviors is usually more than two [11]. Consider an M-class problem, where we have N training samples: $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$. For this case, $x_i \in R^m$ is an m-dimensional feature vector and $y_i \in \{1, 2, \ldots, M\}$ is the corresponding class label. The *one-against-all* approach constructs M binary SVM classifiers, each of which separates one class from the rest. The ith SVM is trained with all the training examples of the ith class with positive labels, and all the others with negative labels. Mathematically, the ith SVM solves the following problem that yields the ith decision function $f_i(x) = w^T \phi(x) + b$:

$$\begin{aligned} &\underset{w,\xi_j^i}{\text{minimize}}: \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_j^i \\ &\text{Subject to: } \tilde{y}_j\left(w_i^T\phi\left(x_j\right)+b_i\right) \geq 1-\xi_j^i, \xi_j^i \geq 0, \end{aligned} \quad (6)$$

where $\tilde{y}_j = 1$ if $y_j = i$ and $\tilde{y}_j = -1$ otherwise. At the classification phase, a sample x is classified as being in class:

$$i^* = \underset{i=1,\ldots,M}{\arg\max}\, f_i(x) = \underset{i=1,\ldots,M}{\arg\max}\left(w_i^T\phi(x)+b_i\right). \quad (7)$$

In our experiment, the *multi-class support vector machine* described in [12] was used as the foundation; it

was then optimized with an algorithm that is faster in the linear SVM. In order to solve this problem, an algorithm based on structural SVMs described in [13] was used in our experiments to find the mapping between the feature space and specific behaviors. In general, x_i represents feature vectors and y_i represents the index of elementary behaviors in the system.

The performance of the multi-class SVM varies depending on both the kernel type and the related parameters. Different types of kernels can be selected for the training process. For our experiments, we tried both the linear type kernel and the radial type kernel $\left(\exp^{-\gamma\|a-b\|^2}\right)$.

Post-processing

After performing classifications at each time instance, the demonstrated behaviors can be recognized through a series of *post-processing* calculations. The classifications are highly dependent on the sensory data generated by the demonstrator. Delays, hesitations, and noise are common disturbances for the recognition process at each time instance, and potentially result in misclassified predictions. The purpose of post-processing is to minimize the amount of misclassified predictions, and to generate a reasonable subtask sequence for a demonstration. This is done by assuming that activated behaviors do not change rapidly from one behavior to another within a certain period of time.

Post-processing is realized by a combination of a *sliding window* and a *voting mechanism*. The term *raw prediction* indicates the direct results from the SVM classifier, and the term *final prediction* (p_k) the results after post-processing.

Sliding window

The concept of the *sliding window* is similar to that of the *moving average,* but replaces the calculation of an average with vote counting. The sliding window takes n sequential samples of the classifications from the SVM. The voting vector is defined as $\mathbf{c_k} = \left[c_k^{(1)} \quad c_k^{(2)} \quad \ldots \quad c_k^{(b)}\right]^T$, where $\mathbf{c_k}$ is the vector containing the votes for each elementary behavior at time instant k, and b is the number of behaviors. The votes for each behavior are obtained by

$$c_k^{(j)} \triangleq \sum_{i=0}^{n-1} x_{k-i}^{(j)}, \tag{8}$$

where $c_k^{(j)}$ denotes the vote for the elementary behavior j, n is the window size of the sliding window, and $x_k^{(j)}$ is the classification of behavior j at time k ($x_k^{(j)}$ is either 0 or 1). The votes can then be calculated by the following recursive formula:

$$c_{k+1}^{(j)} = c_k^{(j)} + x_{k+1}^{(j)} - x_{k+1-n}^{(j)}, \tag{9}$$

From $c_k^{(j)}$, the behavior prediction with the highest number of votes during a period of time is obtained. This predicted behavior is the candidate which best matches the actual behavior in the demonstration. However, the highest voted candidate may have been only marginally preferred over the others, due to noise or demonstrator error. For this reason, an additional voting mechanism is introduced.

Voting mechanism

The voting mechanism ensures that the final prediction (p_k) will only change its state when the vote counts within the sliding window exceed a certain threshold number m. Otherwise, the final prediction remains unchanged, as shown in (10).

$$p_k = \begin{cases} 1, & \text{if } c_k^{(1)} \geq m \\ 2, & \text{if } c_k^{(2)} \geq m \\ \quad\vdots & \\ b, & \text{if } c_k^{(b)} \geq m \\ p_k, & \text{otherwise} \end{cases} \tag{10}$$

This voting mechanism, however, is not merely a counting mechanism. The key assumption here is that behaviors will remain unchanged before another activated behavior is successfully recognized. This mechanism clearly improves stability and increases the noise tolerance.

System Overview

The experiments were performed on our mobile manipulator "LiAS" (the *Leuven intelligent autonomous system*, as shown in Figure 1). The manipulator is equipped with a JR3 wrist force/torque sensor and a gripper at the end-effector. It has 6 DOFs for the arm and 1 DOF for the gripper. The non-holonomic platform has 2.5 DOFs and is equipped with a SICK laser scanner and wheel encoders.

The software architecture of the arm controller is implemented based on the OROCOS environment. OROCOS is an open-source robotic control system developed by Katholieke Universiteit Leuven [14]. It provides a hard real-time environment called *real-time toolkit* (RTT) and basic components for programmers to develop final applications, such as *reporters*, *data ports*, *property files,* and a *script language* interface. Another useful library is the kinematics and dynamics library (KDL), which can perform kinematic calculations. It also

provides mathematical calculation libraries. Within the framework of OROCOS, we create our *generic behavior* classes and a fusion component, realizing the *constraint-based behavior fusion mechanism* (CBFM) on LiAS [15].

Experiment

The performance of the entire learning process depends on the feedback of the simulation process and the modification of parameters. The detailed block diagram of the training phase is shown in Figure 2.

Household experiments

The behavior diagram of a household task was created by a human demonstration through a spacemouse. Without prior knowledge, the robot was taught to open a door and clean a window by demonstration of the exact task sequences on the robot. These household experiments also indicate the applicability of the current research in our daily lives.

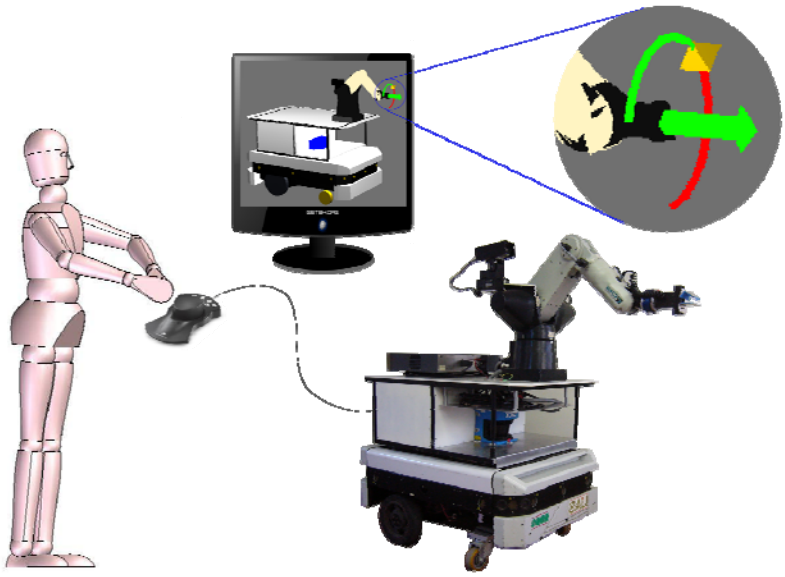

Figure 1. A semi-haptic visualization interface and spacemouse are used to assist the task learning process for the trainer.

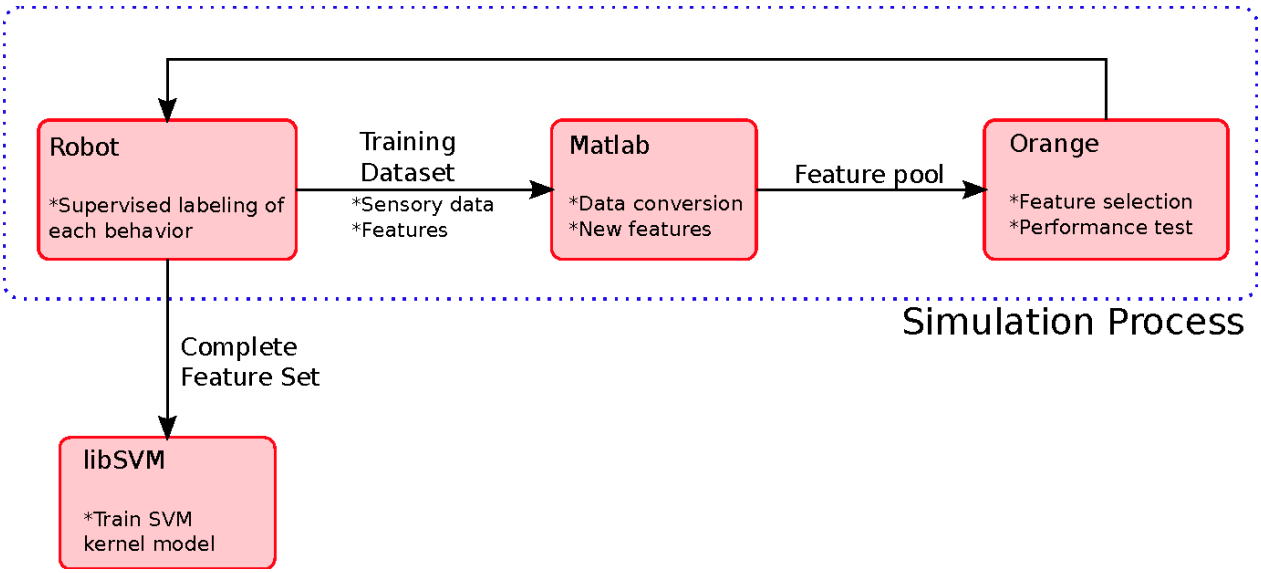

Figure 2. Detailed block diagram of behavior-based task learning. The upper loop forms a simulation process in which new features can be analyzed and evaluated before being adding to the feature set.

Door opening

The behaviors involved in the door-opening sequence are defined as follows: *stand-still* behavior (B_1: $v_{x,\{w\}} = v_{y,\{w\}} = v_{z,\{w\}} = 0$, $\omega_{x,\{w\}} = \omega_{y,\{w\}} = \omega_{z,\{w\}} = 0$), *force-following* behavior (B_2: $f_{x,\{w\}} = f_{y,\{w\}} = f_{z,\{w\}} = 0$, $\omega_{x,\{w\}} = \omega_{y,\{w\}} = \omega_{z,\{w\}} = 0$), *turning* behavior ($B_3$: $t_{z,\{ee\}}$ = const., $v_{z,\{ee\}} = 0$) and *pulling* behavior (B_4: $f_{z,\{ee\}}$ = const., $\omega_{x,\{ee\}} = 0$), where v is velocity, ω is angular velocity, f is force, and t is torque. A demonstrator guides the robot through the door-opening sequence by performing the behaviors as follows: $B_1 \rightarrow B_2 \rightarrow B_3 \rightarrow B_4 \rightarrow B_1$. This dataset is used to train the SVM kernel. With this learned kernel, the demonstrator re-demonstrates the above sequence to evaluate the task learning performance of the robot.

Window cleaning

Window cleaning was implemented as another household task for the experiment. This task involves the following behaviors: *stand-still* behavior (B_1), *force-following* behavior (B_2), *plane move* behavior (B_5: $v_{z,\{ee\}}$ = const., $f_{x,\{ee\}} = f_{y,\{ee\}} = 0$) and *moving back* behavior (B_6). The *moving back* behavior is identical to the *force-following* behavior.

Rather than retraining the robot from scratch for the window cleaning task, we decided to investigate the addition of new behaviors to an existing system. A demonstrator guides the robot to perform the window-cleaning sequence as follows: $B_1 \rightarrow B_2 \rightarrow B_5 \rightarrow B_6 \rightarrow B_1$. The obtained dataset is combined with the dataset from the *door-opening* task, giving six behaviors in total for the system. The corresponding SVM kernel model was trained by this combined dataset. A demonstrator then demonstrated a new sequence representing a household task; this included a combined window-cleaning and door-opening procedure: $B_1 \rightarrow B_2 \rightarrow B_5 \rightarrow B_6 \rightarrow B_2 \rightarrow B_3 \rightarrow B_4 \rightarrow B_1$. The classification results are shown in Figure 3, in which *behavior* is the demonstrated behavior index, *raw classification* is the direct output from the trained SVM model, and *final classification* is the final result after post-processing. This figure illustrates the correct recognition of the demonstrated sequence, and a successful retrain of the kernel for a newly-added dataset. It is evident that the correct behavior sequence was recognized.

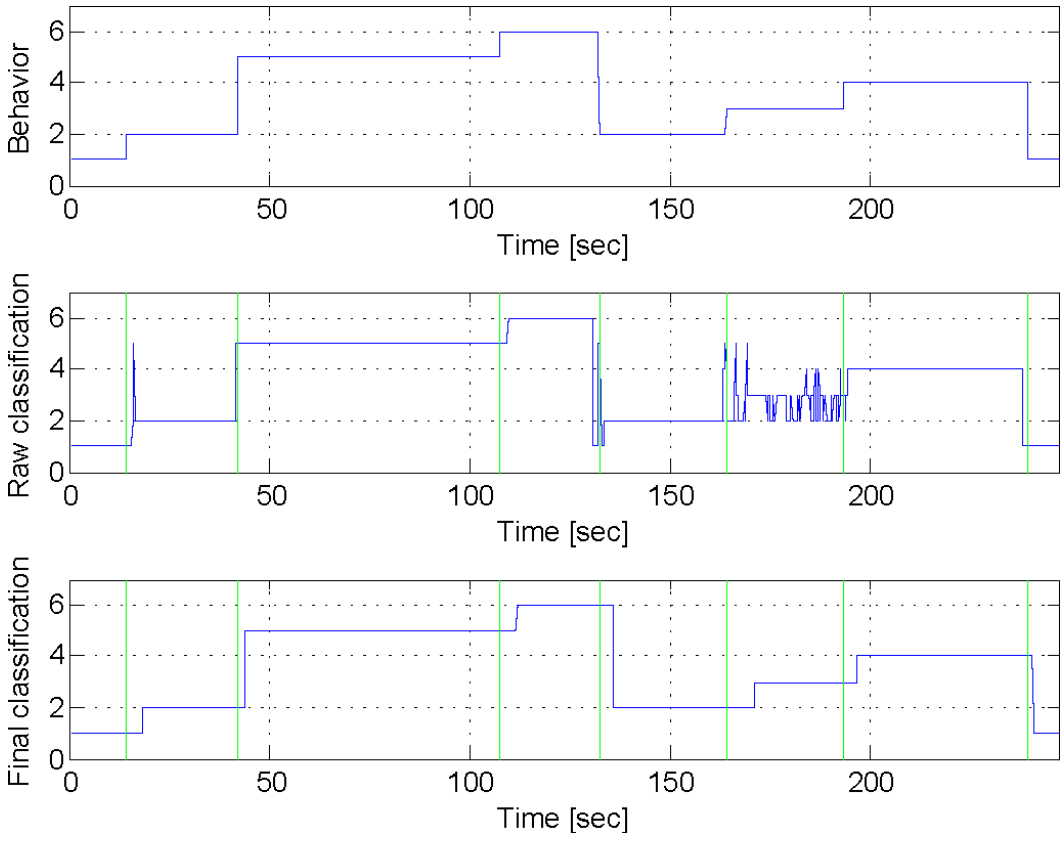

Figure 3. Behavior recognition results for the household tasks application. (Top) the demonstrated behavior index; (middle) raw SVM classification at the runtime; (bottom) final classification after the post-processing with window size = 15 and hit count = 13.

Conclusions and Future Works

Conclusions

The proposed task learning method simplifies the learning procedure of a behavior-based mobile manipulator to a behavior recognition problem. Elementary behaviors intuitively encapsulate robot control, as well as their interactions with the environment. The fusion component, which is based on CBFM, allows more independent and autonomous design of behaviors by optimizing their coordination. The complexity of logical conditions in behavior diagrams can therefore be reduced, and such reduced-complexity behavior diagrams can then be recognized from sensory data. The expected result of the task learning process is a behavior diagram. A general methodology based on machine learning techniques, such as decision trees and SVMs, was proposed to recognize behaviors from human demonstrations. In this methodology, relevant features are selected by decision trees and used to train the kernel of the SVM classifier. The generation of a correct behavior sequence was demonstrated by a household example.

Future works

Proposed in this paper is a scheme for a systems framework ranging from motion controls to task learning. Components in each level can be improved independently by incorporating more advanced techniques. Additionally, during the task learning process, an assisting device (a spacemouse) was required for users to demonstrate a desired task; it is acknowledged that this method can be rather unintuitive for new users. An intuitive and easy-to-use device is still difficult to acquire. Ideally, the guiding device would be another robot with haptic feedback, in a similar configuration to the learning robot. Users could then demonstrate the task on the guiding robot with relative ease, and still be able to sense the environment from the target robot. However, this ideal scenario requires more research and attention.

References

[1] K. Han and M. Veloso, "Automated robot behavior recognition," in *The 9th International Symposium of Robotics Research*, London, 2000, pp. 249-256. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.680

[2] N. Koenig and M. J. Matarić, "Behavior-based segmentation of demonstrated tasks," in *The 5th International Conference on Development and Learning (ICDL)*, Bloomington, US, 2006. Available: http://cres.usc.edu/pubdb_html/files_upload/492.pdf

[3] E. A. Billing, T. Hellström, and L. E. Janlert, "Behavior recognition for learning from demonstration," in *IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, Alaska, US, 2010, pp. 866-872. doi: 10.1109/ROBOT.2010.5509912

[4] J. Corona-Castuera and I. Lopez-Juarez, "Behaviour-based approach for skill acquisition during assembly operations, starting from scratch," *Robotica,* vol. 24, pp. 657-671, 2006. doi: 10.1017/S0263574706002839

[5] M. J. Mataric, "Integration of representation into goal-driven behavior-based robots," *IEEE Transactions on Robotics and Automation,* vol. 8, no. 3, pp. 304-312, 1992. doi: 10.1109/70.143349

[6] C. M. Shakarji, "Least-squares fitting algorithms of the NIST algorithm testing system," *Journal of Research of the National Institute of Standards and Technology,* vol. 103, no. 6, pp. 633-641, 1998. Available: http://nvlpubs.nist.gov/nistpubs/jres/103/6/j36sha.pdf

[7] G. H. Golub and C. F. Van Loan, *Matrix computations*, 3 ed. Baltimore: Johns Hopkins University Press, 1996.

[8] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Transactions on Systems, Man and Cybernetics,* vol. 21, no. 3, pp. 660-674, 1991. doi: 10.1109/21.97458

[9] J. R. Quinlan, *C4.5 : Programs for machine learning*, 1 ed. San Mateo, Calif.: Morgan Kaufmann Publishers, 1993.

[10] C. M. Bishop, *Pattern recognition and machine learning*, 1 ed. New York: Springer, 2006.

[11] L. Yi and Y. F. Zheng, "One-against-all multi-class svm classification using reliability measures," in *IEEE International Joint Conference on Neural Networks (IJCNN)*, Montreal, Canada, 2005, pp. 849-854. doi: 10.1109/IJCNN.2005.1555963

[12] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector machines," *Journal of Machine Learning Research,* vol. 2, pp. 265-292, 2002. Available: http://dl.acm.org/citation.cfm?id=944813

[13] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun, "Support vector machine learning for interdependent and structured output spaces," in *The twenty-first international conference on Machine learning*, Banff, Alberta, Canada, 2004, pp. 104-111.
doi: 10.1145/1015330.1015341

[14] H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the Orocos project," in *IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, 2003, vol.2, pp. 2766-2771.
doi: 10.1109/ROBOT.2003.1242011

[15] H. Shu, E. Aertbelien, and H. Van Brussel, "A constraint-based behavior fusion mechanism on mobile manipulator," in *ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems (LAB-RS)*, Edinburgh, Scotland, 2008, pp. 83-88.
doi: 10.1109/LAB-RS.2008.10