A Support-vector Rule Based Method For The Construction Of Motion Controllers Via Natural Language Training

GUANGHONG WANG, PING JIANG
Department of Information and Control Engineering
Tongji University
1239 Siping Road, Shanghai, 200092
P.R. CHINA

Abstract: - This paper presents a support-vector rule based method for the construction of motion controllers via natural language training. It is a two phases process including the motion control information collection from the natural language instructions and the motion information condensation with the aid of Support Vector Machines theory. A self-organizing fuzzy neural network is proposed for the control rules collection and the support vector rules are then extracted from them to form a final controller to achieve any given control accuracy. In this way, the number of control rules is reduced and the structure of controller becomes tidy, which makes the controller constructed by natural language training more applicable in practice and provides a foundation for further development in mobile robot programming via natural language. Experiments are conducted to illustrate the effectiveness of the method.

Key-Words: - Support vector machines, Fuzzy neural networks, Motion primitives, Language instruction based training, Natural language programming

1 Introduction

The Robot Institute of America defines a robot as a programmable, multifunction manipulator designed to move material, parts, tools, or specific devices through variable programmed motions for the performance of a variety of tasks [1]. As yet, robot programming still mostly relies on hard coding. In order to accomplish a specified task, the programmer has to get exact knowledge about the task and detailed particularities of the robot beforehand. The more domestic robots become pervasive, the more users will be involved in interaction and collaboration with them. However, most users are unfamiliar with computer language and unaware of mechanical and control issues and thus cannot personalize robots using standard programming methods. It is also impractical to require these uninitiated users to learn a robot programming language before the robot can really work for them. How a user could instruct a robot to perform tasks, which manufacturers cannot completely program in advance? Various learning methods have been investigated, such as learning from example (see e.g. Billard et al. 1998), learning by reinforcement [2], etc.. But none of these methods has the power that language has for communicating logical rules and procedural knowledge [3]. In some sense, we can think of an utterance as a program, which indirectly causes a set of operations to be carried out within the hearer’s cognitive system. All language use can be thought of as a way of activating procedures within the hearer [4]. Based on these viewpoints, programming by natural language is a key method enabling computer language-analphabetic users to train their domestic robots to accomplish their desired tasks. However, many people are skeptical about the entire concept. In their opinion, large programs that are written explicitly with standard computer interpretable languages such as C++ are almost never perfect, so how can an implicit, ambiguous natural language succeed at all? Notwithstanding these objections, natural language programming, an integration of speech processing, programming and control, absolutely is an irresistible and meaningful try, while challenging.

Fuzzy neural networks have essential similarity to the natural language expression and can be used for understanding natural language and collecting control rules in the robot’s sensor space. The control rules constructed by natural language instruction are
inevitably redundant or even conflict because of the randomness and vagueness of the instructions from users. The complexity of the construction of motion controller depends on the number of control rules as well. Hence, eliminating redundant control rules and extracting really effective ones are necessary and play a significant role practically. Support Vector Machines (SVMs) is applied to resolve this problem, which actually is of classification of the collected fuzzy rules according to control accuracy requirement. SVMs is a new and very promising classification technique developed by Vapnik and his group at AT&T Bell Laboratories. The main idea behind the technique is to separate the classes with a surface that maximizes the margin between them. SVMs, which has been recently introduced as a general alternative to neural networks, has proved highly successful in a number of pattern classification studies. Danny Roobaert employed SVMs to 3D object recognition with cluttered background [5].

At first, this paper demonstrates the overall system structure. Then it introduces a control method that trains robot to learn motion primitives by natural language, adopting fuzzy neural networks as a generic form for any motion primitive. Then, an off-line approach, utilizing SVMs to extract support vector rules from fuzzy rules according to a given control accuracy, is investigated in this paper. Regarding the fuzzy control rules constructed by natural language instruction as the training patterns, we try to look for those samples that really work during the estimation of the regress function, i.e. the motion controller. Hence, the motion controller will be optimized with the reduction of rule base’s redundancy. Finally, experiments to a mobile robot for the language-based training are carried out with the aid of Nuance speech processing software to illustrate the effectiveness of the aforementioned method.

2 System structure
Robot behavior primitives are the basic behavior skills and abilities of a robot, such as “go near to”, “avoid to”, etc.. If adequate motion primitives have been learned, with the aid of dialogue between a supervisor and the robot, the high-level program for a given task can be generated more easily. This is the original motivation of this paper. The overall system structure for the movement primitives training is shown in Fig.1, with an attachment of a dialogue example between the robot and the supervisor as the movement instructions.

The control block of every motion primitive is a fuzzy neural controller trained by the instruction of natural language. For any given primitive, the robot is controlled by the rule base in a form of fuzzy neural networks and has been waiting for the language instructions from a supervisor. The human supervisor observes the performance of the robot. If necessary, he may give the robot his advice about the right movement action. Explained by the block of instruction interpretation, the language instructions are transferred into quantitative values and the stored networks would be adjusted or even new neurons would be generated accordingly. The updated fuzzy rules are then used for further motion control. During training, the supervisor may give evaluations about the performance of the robot movement. With a “good” evaluation, the related rules at the instant would be given more reliability and the rules would be selected as the control action with higher probability in further control. Of course, the robot can also ask the supervisor for some more information through the block of speech synthesis, when it can not understand an instruction.

The block of the rule base for a given movement primitive is implemented by a three layers’ neural network. It is used for establishing a feedback controller between sensor space and action space as

---

**Dialogue Example:**
Robot: *I am waiting for your command.*
Supervisor: *Go forward.* (Dim sound)
Robot: *Pardon. Please tell me again.*
Supervisor: *Go forward.*
…
Supervisor: *Turn right a little more.*
…
Supervisor: *Very good.*

---

![Fig.1 System Structure](image-url)
shown in Fig.2. The fuzzy rules are generated and updated based on linguistic instructions from the supervisor. On the input layer, each neuron represents a sensor input of the robot system. The second layer is a rule layer. Each neuron represents a fuzzy rule for a given primitive. The input layer and the rule layer are implemented by a self-organizing network and are trained based on competitive learning scheme when the supervisor speaks to the robot. The first two layers realize the antecedents of fuzzy rules for matching sampled data. The third layer is an output layer to generate defuzzified control output. For each rule, the weights \( w \) and \( w_o \) can be adjusted according to the linguistic instructions and performance evaluations.

The block of SVMs is clarified in section 4.

3 Fuzzy Neural Networks Training by Natural Language

In order to implement a general controller for different movement primitives, the fuzzy neural networks are used as a controller prototype. For any given movement behavior, generate a new network and let the robot make trial initially. A supervisor observes the performances of its movement and tells robot how to accomplish it correctly. Based on the language instructions and evaluations of a supervisor, new neurons are generated and network is trained accordingly. The training algorithm can be referred to paper [6] (Xianli Nie and Ping Jiang).

4 Synthesis of control rules by SVMs

Due to the preference and randomness of a supervisor, the rules constructed through fuzzy neural networks are inevitably redundant. Here, we employ the SVMs method to extract the support vector rules from the original fuzzy rules.

Suppose the trained fuzzy neural networks consists of a set of If-Then fuzzy rules:

\[ \text{If } x = x_j = w_{ij} \text{ then } y = y_j = w_{oj}, j = 1, \ldots, k. \]

They can be described by a mapping from the input space to the output space as \( \{(x_i, y_i)\}_{i=1}^{k} \). The SVMs method [7][8] is introduced in the following.

Begin by describing the mapping of a linear function:

\[ f(x) = w \cdot x + b, \; w, x \in \mathbb{R}^n, b \in \mathbb{R} \]  

where “\( \cdot \)” denotes the dot product in the space of input patterns. Flatness in the case of (1) means that one seeks a small \( w \). One way to ensure this is to minimize the Euclidean norm \( \|w\|^2 \). Formally this problem can be written as a convex optimization problem by requiring:

\[ \min \frac{1}{2} \|w\|^2 \text{ s.t. } \begin{cases} y_i - w \cdot x_i - b \leq \varepsilon \\ w \cdot x_i + b - y_i \leq \varepsilon \end{cases} \]  

The tacit assumption in (2) was that such a function \( f \) actually exists that approximates all pairs \( (x_i, y_i) \) with \( \varepsilon \) precision, or in other words, that the convex optimization problem is feasible. Sometimes, however, this may not always be the case. Allowing for some errors, one can introduce slack variables \( \xi, \xi^* \) to cope with otherwise infeasible constraints of the optimization problem (2). Hence the formulation can be stated in:

\[ \min \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{k} (\xi_i + \xi_i^*) \text{ s.t. } \begin{cases} y_i - w \cdot x_i - b \leq \varepsilon + \xi_i \\ w \cdot x_i + b - y_i \leq \varepsilon + \xi_i^* \\ \xi, \xi^* \geq 0 \end{cases} \]  

where the user-specified constant \( C > 0 \) determines the tradeoff between the flatness of \( f \) and the tolerated amount larger than \( \varepsilon \).

The key idea is to construct a Lagrange function from both the objective function and the corresponding constraints, by introducing a dual set of variables \( a_i, a_i^*, \eta, \eta_i^* \geq 0 \). Hence it appears to be:

\[ L(w, b, \xi, \xi^*, a_i, a_i^*, \eta, \eta_i^*) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{k} (\xi_i + \xi_i^*) \]

\[ - \sum_{i=1}^{k} a_i (\varepsilon + \xi_i - y_i + w \cdot x_i + b) - \sum_{i=1}^{k} (\eta_i \xi_i + \eta_i^* \xi_i^*) \]

\[ - \sum_{i=1}^{k} a_i^* (\varepsilon + \xi_i^* + y_i - w \cdot x_i - b) \]  

It follows from the saddle point condition that the partial derivatives of \( L \) with respect to the primal variables \( (w, b, \xi, \xi^*) \) have to vanish for optimality.

\[ \frac{\partial}{\partial b} L = \sum_{i=1}^{k} (a_i^* - a_i) = 0 \]  

\[ \frac{\partial}{\partial w} L = w - \sum_{i=1}^{k} (a_i - a_i^*) x_i = 0 \]  

\[ \frac{\partial}{\partial \xi_i} L = C - a_i^* - \eta_i = 0 \]  

\[ \frac{\partial}{\partial \xi_i^*} L = C - a_i^* - \eta_i^* = 0 \]
Substituting (5), (6), and (7) into (4) yields the dual optimization problem:

$$\max -\frac{1}{2} \sum_{i,j=1}^{k} (a_i - a_i^*) (a_j - a_j^*)(x_i \cdot x_j) - \varepsilon \sum_{i=1}^{k} (a_i + a_i^*) + \sum_{i=1}^{k} y_i (a_i - a_i^*)$$

$$\text{s.t. } \sum_{i=1}^{k} (a_i - a_i^*) = 0, a_i, a_i^* \in [0, C]$$

Eq.6 can be rewritten as:

$$w = \sum_{i=1}^{k} (a_i - a_i^*) x_i, f(x) = \sum_{i=1}^{k} (a_i - a_i^*) (x_i \cdot x) + b \quad (9)$$

This is the so-called Support Vector expansion, i.e. $w$ can be completely described as a linear combination of the training patterns $x_i$.

The so-called Karush-Kuhn-Tucker (KKT) conditions show that at the optimal solution the product between the dual variables and the constraints has to vanish. In this case it means:

$$a_i (\varepsilon + \xi_i - y_i + w \cdot x_i) + b = 0 \quad (10)$$

$$a_i^* (\varepsilon + \xi_i^* + y_i - w \cdot x_i) - b = 0$$

$$\eta_i \xi_i = (C - a_i) \xi_i = 0 \quad (11)$$

$$\eta_i^* \xi_i^* = (C - a_i^*) \xi_i^* = 0$$

Hence $b$ can be computed as follows:

$$b = y_i - w \cdot x_i - \varepsilon \text{ for } a_i \in (0, C)$$

$$b = y_i - w \cdot x_i + \varepsilon \text{ for } a_i^* \in (0, C) \quad (12)$$

From Eq.10 it can be inferred that if $|f(x_i) - y_i| < \varepsilon$, then the Lagrange multipliers vanish, $a_i, a_i^* = 0$. Therefore there is no need to involve all $x_i$ to describe $w$ and $f$. The samples that come with non-vanishing coefficients are called Support Vectors (SVs). In a sense, the complexity of a function’s representation by SVs depends only on the number of SVs, and is independent of the dimensionality of the input space. Moreover, even it isn’t necessary to compute $w$ explicitly. These observations will become handy for the formulation of a nonlinear extension.

As for a nonlinear regression, it can be achieved by preprocessing the training patterns by a map $\Phi : \mathbb{R}^n \rightarrow F$ into some feature space $F$, in which the problem turns into linear. If there exists a kernel function satisfying $k(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$, then Eq.8 can be rewritten as:

$$\max -\frac{1}{2} \sum_{i,j=1}^{k} (a_i - a_i^*) (a_j - a_j^*) k(x_i, x_j) - \varepsilon \sum_{i=1}^{k} (a_i + a_i^*) + \sum_{i=1}^{k} y_i (a_i - a_i^*)$$

$$\text{s.t. } \sum_{i=1}^{k} (a_i - a_i^*) = 0, a_i, a_i^* \in [0, C] \quad (13)$$

The expansion of $f$ in (9) will be written as:

$$w = \sum_{i=1}^{k} (a_i - a_i^*) \Phi(x_i)$$

$$f(x) = \sum_{i=1}^{k} (a_i - a_i^*) k(x_i, x) + b \quad (14)$$

The difference to the linear case is that $w$ is no longer explicitly given.

**Fig.3 Architecture of synthesis of control rules by SVMs**

Fig.3 contains a graphical overview over the different steps in the regression stage mentioned above. The process described here is very similar to Fig.2, with the difference, that in the SVMs case the weights in the input layer are predetermined by the training patterns.

A notion that is central to the construction of the support vector learning algorithm is the inner-product kernel between a support vector $x_i$ and the vector $x$ drawn from the input space. The requirement on the kernel function $k(x_i, x)$ is to satisfy Mercer’s theorem. Within this requirement there is some freedom in how it is chosen. In this paper, we employ the second RBF:
\[ k(x_i, x_j) = \exp[-\|x_i - x_j\|^2/(2\sigma^2)] \]  
(15)

where the width \( \sigma^2 \) is specified a priori by the user.

Ordinarily, the training of a support vector machine, as shown in Eq.13, is of a quadratic programming problem, stated as:

\[
\begin{align*}
\min & \quad \frac{1}{2}z^TQz + p^Tz \\
\text{s.t.} & \quad A_{eq} \cdot z \leq b_{eq}, A_{eq} \cdot z = b_{eq}, l_b \leq z \leq u_b 
\end{align*}
\]  
(16)

Here let:

\[
\begin{align*}
\alpha & = [\alpha_1, \ldots, \alpha_k]^T, \\
\alpha^* & = [\alpha^*_1, \ldots, \alpha^*_k]^T, \\
\beta & = [\beta_1, \ldots, \beta_k]^T, \\
\beta^* & = [\beta^*_1, \ldots, \beta^*_k]^T, \\
\theta & = [\theta_1, \ldots, \theta_k]^T, \\
\end{align*}
\]

Eq.13 and Eq.16 are equivalent. For Eq.16, function \textit{quadprog} in the Optimal Toolbox of Matlab can be applied to solve the quadratic programming problem. The result will be with only a fraction of \( \alpha_i, \alpha^*_i \neq 0 \), and the primal fuzzy rules corresponding to them are just the support vector rules. Then, as show in Fig.3, the support vector rules can be involved in the synthesis with the input vector to construct the motion controller.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{Fig4.png}
\caption{Three Marks (two on robot)}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{Fig5.png}
\caption{Two Trained Path From Different States}
\end{figure}

5 Experiments and results

To illustrate the effectiveness of the proposed method, experiments are carried out on a real mobile robot, which looks like a spherical cap, with radius of 100mm. Nuance Speech Recognition System [9] is employed for natural language interpretation in the system. The processing of vision information is simplified by tagging marks with different color on the robot and the target object. In order to specify the two parameters \( \alpha \) and \( D \) shown in Fig.4, two marks with different color are tagged on the robot. \( \alpha \) and \( D \) can be calculated directly from the image coordinate space transferred from a global video camera. The input vector is \( x = [\alpha, D] \in \mathbb{R}^2 \), and the output \( y \) is a quantified linguistic instruction. The controllers based on those primal fuzzy rules and on the support vector rules are called controller 1 and 2 respectively.

Two trajectories are trained from two different start states to the target object, as shown in Fig.5. There are 33 rules acquired from the training. Select the width \( \sigma^2 = 160^2 \) and adjust the user-specified parameter C to 48. Then the relationship between the number of the support vectors \( s \) and \( \varepsilon \) is indicated in Fig.6. The number of the support vectors decreases with \( \varepsilon \) increasing. Therefore, for a given motion primitive, appropriate \( \varepsilon \) can be set to reduce the number of the support vectors, and the structure of controller will become tidy correspondingly. Fig.7 demonstrates the support vector rules selected from the primal rules and the corresponding controller with \( \varepsilon = 10 \) and \( \varepsilon = 18 \). A set of results is gained, as shown in Fig.8. Both controllers are competent for those trained start states. However, for those untrained start states, which are also far from the region formed by the trained start states, controller 1 is incapable to drive the robot to close the target, while controller 2 performs quite well. This embodies both the effectiveness and expansion ability of SVM, which represents a stable characteristic of the training patterns and plays an important role in the synthesis of the controller.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{Fig6.png}
\caption{Relationship between \( s \) and \( \varepsilon \)}
\end{figure}
6 Conclusion
This paper presents a support-vector rule based method for the construction of motion controllers via natural language training. There are two stages to achieve this aim. At first stage, we employ fuzzy neural networks driven by the natural language instructions to obtain the primal control rules. Thereafter, SVMs are utilized to extract the expected support vector rules from the primal rules to achieve any given control accuracy. In this way, the number of control rules is reduced and the structure of controller becomes tidy, which makes the controller constructed by natural language training more applicable in practice and provides a foundation for further development in mobile robot programming via natural language. Experiments are conducted to illustrate the effectiveness of the method. The results show that SVMs has the ability to explore some untouched states, due to its representation of a stable characteristic of the training patterns.

References: