MOBILE ROBOT NAVIGATION USING DYNAMIC FUZZY Q-LEARNING

Tao Meng

School of Electrical & Electronic Engineering
Nanyang Technological University
Hall 6, Bk35-2-670, Nanyang Drive, Singapore 637724
E-mail: 147942513@ntu.edu.sg

Keywords: Fuzzy Inference System, Reinforcement Learning, Dynamic Fuzzy Q-Learning

Abstract

Fuzzy logic is a mathematical approach towards the human way of thinking and learning. Based on if-then rules, we can design fuzzy controllers with the intuitive experience of human beings. However, it is not practical for a designer to find necessary number of rules and determine appropriate parameters by hand. Hence, we incorporate a reinforcement learning method with basic fuzzy rules so that the the controller can be tuned online. In this paper, we present Dynamic Fuzzy Q-Learning (DFQL).

1 Introduction

This project aims to develop a fuzzy controller for a robot so that it can perform wall following tasks in indoor environments that are unknown, perceived inaccurately and partially dynamic. However, the conventional Fuzzy Inference Systems (FIS) is designed based on the past known behavior of a target system. The fuzzy system is then expected to be able to reproduce the behavior of the target system [1]. Since the mobile robot is operated in an unknown environment, we need to implement reinforcement learning into FIS to achieve autonomous navigation.

In common sense, reinforcement learning (RL) means that if an action is followed by a satisfactory state of affairs, or an improvement in the state of affairs, then the tendency to produce the action is strengthened or reinforced (rewarded). Otherwise, that tendency is weakened or inhibited (penalized). Unlike the supervised learning problem where the correct “target” output values are given for each input pattern to instruct the controller’s learning, reinforcement learning problem has only very simple “evaluative” or “critic” information instead of “instructive” information available for learning. One of the basic representative architecture for RL is Q-learning, which estimate the discounted future rewards for taking actions from given states based on temporal-difference (TD) learning [2]. However, it is difficult to deal with continuous states and actions in the real world for ordinary Q-learning because of discrete state and action learning parameter. Therefore, in this paper we introduce Dynamic Fuzzy Q-Learning (DFQL), which is based on Jouffe’s Fuzzy Q-Learning [3]. It is an automatic method allowing self-tuning of FIS based only on reinforcement signals. Continuous states are handled and continuous actions are generated by fuzzy reasoning. Prior knowledge can embedded into the fuzzy rules, which can reduce training significantly.

The robot used here is Khepera II. The DFQL algorithm is then translated into matlab programming language which can be understood by the robot.

2 Algorithm

2.1 Architecture of Dynamic Fuzzy Q-Learning

Fig 1 represents the learner-environment interaction in reinforcement learning. At each time step t, the learner observes the current state St and generates an action Ut from the set of possible actions corresponding to that state At(S). One time step later, as a consequence of the previous action, the learner receives a numerical reward or punishment rt+1 while it also reaches a new state St+1. At each time step, the learner implements a mapping from states to probabilities of selecting each possible action. This mapping is called the learner’s control policy denoted by π.

2.2 Basic Fuzzy Rule

Suppose the number of input variables is r and each input variable xi has n membership functions, which is in the form of Gaussian functions:
\[
\mu_{ij}(x_i) = \exp\left(-\frac{(x_i - c_{ij})}{\sigma_{ij}^2}\right)
\]
(i = 1, 2, ..., r, j = 1, 2, ..., n)  

where \(\mu_{ij}\) is the jth membership function of \(x_i\), \(c_{ij}\) and \(\sigma_{ij}\) are the center and width of the jth Gaussian membership function of \(x_i\) respectively.

If each rule's firing strength is computed by multiplication, the output of the jth rule \(R_j\) is

\[
\Phi_j(x_1, x_2, \ldots, x_r) = \exp\left(-\frac{1}{\sigma_{ij}^2} \sum_{i=1}^{r} (x_i - c_{ij})^2\right)
\]
(j = 1, 2, ..., n)

Hence, the weight \(\alpha_j\) of each rule is

\[
\alpha_j = \frac{\Phi_j}{\sum_{j=1}^{n} \Phi_j}
\]
(j = 1, 2, ..., n)

Using the center-of-gravity method, the output variable as the weighted summation of incoming signals

\[
y = \sum_{j=1}^{n} \alpha_j o_j
\]

where \(y\) is the value of the output variable and \(o_j\) is the consequent parameters of the jth rule.

The firing strength of each rule shown in Eq(2) can be regarded as a function of regularized Mahalanobis distance (M-distance), i.e.

\[
\Phi_j(x_1, x_2, \ldots, x_r) = \exp\left(-\frac{1}{\sigma_{ij}^2} \sum_{i=1}^{r} (x_i - c_{ij})^2\right)
\]
(5)

where

\[
md(j) = \sqrt{(X - C_j)^T \sum_{j=1}^{n} (X - C_j)}
\]
(6)

is the M-distance, \(X = (x_1, x_2, \ldots, x_r)^T \in \mathbb{R}^r\), \(C_j = (c_{1j}, c_{2j}, \ldots, c_{nj})^T \in \mathbb{R}^r\) and \(\sum_{j=1}^{n} \) is defined as following:

\[
\sum_{j=1}^{n} = \begin{bmatrix}
\frac{1}{\sigma_{ij}^2} & 0 & \cdots & 0 \\
0 & \frac{1}{\sigma_{2j}^2} & 0 & 0 \\
0 & 0 & \ddots & 0 \\
0 & \cdots & 0 & \frac{1}{\sigma_{nj}^2}
\end{bmatrix}
\]
(7)

(j = 1, 2, ..., n)

### 2.3 Generation of Continuous Action

The function of control policy is shown below

\[
\pi_A(q) = \arg\max_{a \in A} (q(a) + \eta(a))
\]
(8)

The undirected term of exploration \(\eta\) stems from a vector of exponential random value \(\Psi\).

\[
s_f = \begin{cases}
1, & \text{if } \max(q) = \min(q) \\
\frac{s_p (\max(q) - \min(q))}{\max(\Psi)} & \text{otherwise}
\end{cases}
\]
(9)

\[
\eta = s_f \Psi
\]
(10)

where \(s_p\) is the noise size, with respect to the range of qualities, and \(s_f\) is the corresponding scaling factor.

Assume each rule \(R_i\) has \(M\) possible discrete actions \(A_i\). With reference to Eq(8), local actions can be selected from \(A_i\) base on their Q-values. Subsequently, the winning local action \(a_i\) cooperates with the rule’s normalized firing strength \(\alpha_i\) to produce the global action, which is given by

\[
U_i = \sum_{i=1}^{n} \pi_{A(i)}(q_i) \alpha_i = \sum_{i=1}^{n} a_i' \alpha_i'
\]
(11)

where \(a_i'\) is the selected action of rule \(R_i\) at time step \(t\). The Q-value of the selected discrete action \(a\) is initialized to a fixed value \(kq\) while all the others are given random values according to a uniform distribution in \([0, kq/2]\).

### 2.4 Update of Q-values

Since the global action is generated according to the Q-values, it is necessary to update Q-values in order to achieve optimal actions. Thus, we define a function \(Q\), which gives the action quality at time step \(t\), as depicted below:

\[
Q_i(U_i) = \sum_{i=1}^{n} q_i'(a_i') \alpha_i'
\]
(12)

where \(q_i'\) is the associated Q-value of the action \(a_i'\).

Due to Temporal Difference (TD) learning, the Q-values
corresponding to the optimal actions are used to estimate the TD error, which is defined as following
\[
Q_t^* = \sum_{i=1}^{n} \left( \max_{a \in A(i)} q_t^i(a) \right) \alpha_t^i
\]  
(13)

And the TD error is then obtained as
\[
\tilde{e}_{t+1} = r_{t+1} + \gamma Q_t^* - Q_t(U_t)
\]  
(14)

where rt+1 is the reinforcement signal received at time t+1 and \( \gamma \) is the discount factor used to weight the short and long term. Note that we have to estimate this error term only with quantities available at time step t+1.

Let \( e_t(i, m) \) be the trace associated with discrete action \( A(i, m) \) of rule \( R_i \) at time step t
\[
e_t(i, m) = \begin{cases} 
\gamma \alpha_t(i, m) + \alpha_t(i, m) & \text{if } A(i, m) = \alpha_t \\
\gamma \alpha_t(i, m) & \text{otherwise}
\end{cases}
\]  
(15)

where the eligibility rate \( \lambda \) is used to weight time steps.

Finally, Q-values can be updated as following
\[
q_{t+1}(i, m) = q_t(i, m) + \rho \tilde{e}_t(i, m)
\]  
(16)

\((i = 1, 2, \ldots, n, m = 1, 2, \ldots, M)\)

\(2.5 \, \varepsilon\)-Completeness of Fuzzy Rules

\(\varepsilon\)-Completeness of Fuzzy Rules means that for any input in the operating range, there exists at least one fuzzy rule whose firing strength is no less \( \varepsilon \) \[4\]. Normally, the minimum value of \( \varepsilon \) is chosen as 0.5 in fuzzy applications.

If a new pattern satisfies \( \varepsilon \)-Completeness, the DFQL will not generate a new rule but accommodate the new sample by updating the parameters of the existing rules. According to \( \varepsilon \)-Completeness, we firstly calculate the M-distance \( md(j) \) between the observed input vector \( X \in R^r \) and the centers \( C_j (j = 1, 2, \ldots, n) \) based on Eqs(6) and (7). Next, we need to find the minimum value of \( md(j) \)
\[
J = \arg \min_{i \in \{1, 2, \ldots, n\}} (md(j))
\]  
(17)

if
\[
md_{\min} = md(J) > k_d
\]  
(18)

it implies that the existing system is not satisfied with \( \varepsilon \)-Completeness and a new rule should be considered. Here, \( k_d \) is a pre-specified threshold, which is given below
\[
k_d = \sqrt{\sum_{i=1}^{r} \ln(1/ \varepsilon_i)}
\]  
(19)

where \( \varepsilon_i \) is the corresponding \( \varepsilon \)-Completeness for each input variable \( x_i (i = 1, 2, \ldots, r) \).

\(2.6 \, \text{TD Error Criteria for Rule Generation}\)

It is not sufficient to only use \( \varepsilon \)-Completeness of fuzzy rules as the criteria of rule generation. New rules need to be generated in regions of input fuzzy spaces the approximation performance of the DFQL is unsatisfactory. Hence, we introduce a separate performance index, \( \xi_i \), for each fuzzy subspace to enable the discovery of “problematic” regions in the input space.
\[
\xi_i = \left[ (K - \alpha_i^t) \xi_i^t + \alpha_i^t (\tilde{e}_{t+1} - \xi_i) ^2 \right] / K
\]  
(20)

where \( K \) is a predefined constant.

If \( \xi_i \) is lower than a certain threshold, further segmentation should be considered for this fuzzy subspace.

\(2.7 \, \text{Estimation of Premise Parameters}\)

Once there is necessity to generate a new rule, we should determine the centers and widths of the corresponding membership function for the new rule. Assume that n fuzzy rules have been already generated. A new rule will be formed when criteria of rules generation is not satisfied. Initially, the incoming multidimensional input vector \( X \) is projected to the corresponding one-dimensional membership for each input variable \( i = 1, 2, \ldots, r \) and the Euclidean distance \( ed_i(j) \) between the data \( x_i \) and the boundary set \( \Omega_i \) is computed as below:
\[
ed_i(j) = |x_i - \Omega_i(j)|
\]  
(21)

\((j = 1, 2, \ldots, n+2)\)

where \( \Omega_i \in \{x_{i_{\min}}, c_{i1}, c_{i2}, \ldots, c_{in}, x_{i_{\max}}\} \). Next, find
\[
j_k = \arg \min_{j=1, 2, \ldots, n+2} (ed_i(j))
\]  
(22)

if \( ed_i(j_k) \) is not bigger than \( kmf \), a predefined constant that controls the similarity of neighboring membership function, \( x_i \) can be completely represented by the existing fuzzy set \( E_{jk}(c_{jk}, \sigma_{jk}) \) without generating a new membership function. Otherwise, a new Gaussian membership function should be allocated whose center is
\[
c_{(i+1)} = x_i
\]  
(23)

and whose width is
\[
\sigma_{(i+1)} = \max \left\{ \frac{|c_{(i+1)} - c_{(i+1)}}{\sqrt{\ln(1/ \varepsilon_i)}} \right\}
\]  
(24)
where $c_i(n+1)$ and $c_j(n+1)$ are the two centers of neighboring membership function of the new membership function. By this approach, the fuzzy sets of input variables can satisfy $\varepsilon$-Completeness of fuzzy rules [5].

2.8 Generation of New Rules

Combining the $\varepsilon$-Completeness criteria, TD error criteria and estimation of premise parameters together, we obtain the procedure of generating a new rule. First, check whether the existing system is satisfied with $\varepsilon$-Completeness. If not, we then estimate premise parameters and obtain the number, $J$ of the rule which has the minimum M-distance. Meanwhile the initial parameter $q_{n+1}$ are set to zero. Next, we check whether $\tilde{g}^J$ is larger than $ke$. If so, the fuzzy rule $R_J$ is not satisfied with TD error criteria and a new rule should also be considered. And the initial parameter vector $q_{n+1}$ are set to the same value of $q_J$.

3 Experimentation

3.1 Assumption

Based on the algorithm above, we can develop fuzzy controllers using matlab programming language, which has great advantages in matrix operation. To simplify the problem, we assume that the robot moves in clockwise direction. Therefore, the fuzzy controller only needs 4 input variables, which are the values of reflected light of sensor $S_i$ ($i = 0, 1, \ldots, 3$). The output of the controller is the steering angle of the robot. As we are dealing with wall following task, the more parallel the robot moves against the wall, the better the performance is. Moreover, the robot should move along the wall as close as possible. Hence, we can generate a reinforce signal when evaluating the performance of the robot. The value $d$ of sensor $S_0$ can be regarded as the distance between the wall and the robot. And we assign a minimum distance $d-$ and a maximum distance $d+$, which define a range of distance that is allowable for the robot. The reward function is shown below

$$r = \begin{cases} 
0.1, & \text{if } (d_\text{m} \leq d < d_\text{m}) \text{and } U \in [-8^\circ, +8^\circ] \\
-3.0, & \text{if } (d \leq d_\text{m}) \text{or } (d \geq d_\text{m}) \\
0.0, & \text{otherwise}
\end{cases}$$

(24)

if an action brings the robot outside $[d_-, d_+]$, the robot receives a punishment. Here, $d_+ = 0.85$, $d_- = 0.15$.

3.2 Basic Fuzzy and DFQL

Because basic fuzzy rule serves as the basis of DFQL, we divided the project into two stages. Firstly, basic fuzzy rule was implemented into the controller. After we demonstrated basic fuzzy rule was applicable, we extended it to DFQL.

For each input variable, we can separate it into two linguistic subspaces: Small and Big, of which the membership functions cover the region of the input space evenly with the value of $\varepsilon$-Completeness is 0.5. As a result, there are total 16 ($2^4$) fuzzy rules. Through trial and error, we can get the 16 fuzzy rules as shown in Table 1.

<table>
<thead>
<tr>
<th>Rule</th>
<th>$S_0$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>Steering Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Small</td>
<td>Small</td>
<td>Small</td>
<td>Small</td>
<td>-15</td>
</tr>
<tr>
<td>2</td>
<td>Small</td>
<td>Small</td>
<td>Small</td>
<td>Big</td>
<td>-15</td>
</tr>
<tr>
<td>3</td>
<td>Small</td>
<td>Small</td>
<td>Big</td>
<td>Small</td>
<td>-15</td>
</tr>
<tr>
<td>4</td>
<td>Small</td>
<td>Small</td>
<td>Big</td>
<td>Big</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>Small</td>
<td>Big</td>
<td>Small</td>
<td>Small</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>Small</td>
<td>Big</td>
<td>Small</td>
<td>Big</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>Small</td>
<td>Big</td>
<td>Big</td>
<td>Small</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>Small</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>30</td>
</tr>
<tr>
<td>9</td>
<td>Big</td>
<td>Small</td>
<td>Small</td>
<td>Small</td>
<td>15</td>
</tr>
<tr>
<td>10</td>
<td>Big</td>
<td>Small</td>
<td>Small</td>
<td>Big</td>
<td>15</td>
</tr>
<tr>
<td>11</td>
<td>Big</td>
<td>Small</td>
<td>Big</td>
<td>Small</td>
<td>15</td>
</tr>
<tr>
<td>12</td>
<td>Big</td>
<td>Small</td>
<td>Big</td>
<td>Big</td>
<td>30</td>
</tr>
<tr>
<td>13</td>
<td>Big</td>
<td>Big</td>
<td>Small</td>
<td>Small</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>Big</td>
<td>Big</td>
<td>Small</td>
<td>Big</td>
<td>30</td>
</tr>
<tr>
<td>15</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Small</td>
<td>30</td>
</tr>
<tr>
<td>16</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 1: 16 Fuzzy Rules

The steering angles, which are also the consequent parameters, are fixed. The centers are 0 for all Small subspaces and 1 for all Big subspaces. The width of all the membership functions is set to 0.6. The performance of basic fuzzy rule is actually quite acceptable. However, the controller can’t get rid of disturbances like ambient light and colors of the wall. If disturbances occurred, we would need to readjust the consequent parameters by hand, which is impractical and time consuming. Hence, to overcome these limitations, DFQL is considered to achieve autonomous navigation.

The basic fuzzy rules from intuitive experience are used as starting point of DFQL. The values of parameters described in learning algorithm are: Initial Q-value $k_q = 3.0$, Initial E-trace $E_0(i, m) = 0$, Initial performance index $\xi = 0$, Exploration rate $Sp = 0.001$, Discounted factor $\gamma = 0.95$, Trace-decay factor $\lambda = 0.7$, TD learning rate $\beta = 0.05$, $\varepsilon$-Completeness $\varepsilon = 0.5$, Similarity of membership function $kmf = 0.3$, TD error factor $K = 50$, TD error criteria $ke = 1$, and Set of discrete actions $A = [-30, -25, -20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30]$. The performance of DFQL is more optimal than basic fuzzy rules. Furthermore, it is shown that DFQL is tolerant to disturbance.
4 Conclusion

In this project, we prove that DFQL is a suitable algorithm for autonomous navigation of robots. However, due to the time constraint, we can’t compare DFQL to other algorithms so that the superiority of DFQL remains unknown. The performance of DFQL is only evaluated qualitatively, which seems lack of persuasion. Further improvement needs to be done in future to prove that DFQL is a more optimal algorithm than others in both qualitative and quantitative measurement.

Acknowledgements

Last but not least, I wish to thank A/P Er Meng Joo for his guidance throughout the project. I also wish to thank Mr Deng Chang and Tian Ping. Without their help, this project could not be finished successfully in such a short time.

Reference


