Control using knowledge-based models

Previously, we have examined a lot of knowledge-based models. Now it is time to look at how we can control systems with them. First, we investigate the working principle of inverse control. After that, we also look at other types of knowledge-based control.

1 Inverse control

1.1 Basics of inverse control

Let's examine a system. For simplicity, we will examine a **single-input single-output** (SISO) system. This system can be described by a **model** of the system

$$y(k+1) = f(\mathbf{x}(k), u(k)). \tag{1.1}$$

Let's suppose that we know the function f and that we can find an inverse function f^{-1} such that

$$u(k) = f^{-1}(\mathbf{x}(k), y(k+1)). \tag{1.2}$$

Now let's say that we want to reach a **desired state** $r(k+1)$. Then we simply replace $y(k+1)$ by $r(k+1)$ in the inverse function. The resulting value of $u(k)$ will be the input that makes sure that $r(k+1)$ is reached. This method of controlling a system is called **inverse control**.

There are various ways in which we can implement inverse control. Which one we use depends on whether we know the state $\mathbf{x}$. In **open-loop feedback control** we use the output of the system to determine $\mathbf{x}$. This state is then inserted into the inverse function f^{-1} to find the required input $u(k)$.

However, sometimes we can't use the output $y(k)$ of the system to find the state $\mathbf{x}$. In this case, **open-loop feedforward control** is an alternative. Now we use the model f of the system to keep track of $\mathbf{x}(k)$. This method has as a downside that the error between the model and the actual system can grow over time. So, an accurate model needs to be available.

The question remains how you can find the inverse f^{-1}. This is mostly done using numerical methods. In these methods, you try to minimize an objective function like

$$J(u(k)) = (r(k+1) - f(\mathbf{x}(k), u(k)))^2. \tag{1.3}$$

For some models, the inverse f^{-1} can be computed analytically. Let's examine a few of such cases.

1.2 The inverse of an affine TS fuzzy model

Let's examine an affine TS fuzzy model in which the rules do not contain the input $u(k)$ in the antecedent. Instead, $u(k)$ only occurs in the consequent. So, the K rules will have the form

$$\mathcal{R}_i: \ \textbf{if } \mathbf{x}(k) \text{ is } X_i \textbf{ then } y(k+1) = \mathbf{a_i}^T\mathbf{x}(k) + b_i u(k) + c_i. \tag{1.4}$$

To find the actual value of $y(k+1)$ we have to know the degree of fulfillment $\beta_i(\mathbf{x}(k))$ or the corresponding normalized degree of fulfillment $\lambda_i(\mathbf{x}(k))$. If we do, then we can find $y(k+1)$ using

$$y(k+1) = \frac{\sum_{i=1}^K \beta_i(\mathbf{x}(k)) \left(\mathbf{a_i}^T\mathbf{x}(k) + b_i u(k) + c_i\right)}{\sum_{i=1}^K \beta_i(\mathbf{x}(k))} = \sum_{i=1}^K \lambda_i(\mathbf{x}(k)) \left(\mathbf{a_i}^T\mathbf{x}(k) + b_i u(k) + c_i\right). \tag{1.5}$$

It can be noted that this equation is linear in $u(k)$. So, it can easily be solved for $u(k)$. If we also replace the output $y(k+1)$ by the desired output $r(k+1)$, we will find that

$$u(k) = \frac{r(k+1) - \sum_{i=1}^K \lambda_i(\mathbf{x}(k)) \left(\mathbf{a_i}^T\mathbf{x}(k) + c_i\right)}{\sum_{i=1}^K \lambda_i(\mathbf{x}(k)) b_i}. \tag{1.6}$$

This is the inverse function f^{-1} of the system model f.

1.3 The inverse of a singleton model

We now examine a similar case: the singleton model. However, there is an important difference. The input $u(k)$ now isn't in the consequent anymore, but in the antecedent. So, the rules have the form

$$\mathcal{R}_{ij}: \ \textbf{if } \mathbf{x}(k) \text{ is } X_i \textbf{ and } u(k) \text{ is } U_j \textbf{ then } y(k+1) = c_{ij}. \tag{1.7}$$

The problem now is that the degree of fulfillment $\beta_{ij}(k)$ of rule ij at time k not only depends on the (known) state $\mathbf{x}(k)$, but on the (to-be-determined) input $u(k)$ as well. Luckily, this can be solved, if we use the product t-norm operator. We then have

$$\beta_{ij}(k) = \mu_{X_i}(\mathbf{x}(k)) \cdot \mu_{B_j}(u(k)). \tag{1.8}$$

We also assume that the antecedent membership functions $\mu_{B_j}(u(k))$ form a partition. That is,

$$\sum_{j=1}^{N} \mu_{B_j}(u(k)) = 1. \tag{1.9}$$

If this is the case, and if the state $\mathbf{x}(k)$ is known, then we can simplify the rule base. To do this, we first define

$$c_j(k) = \sum_{i=1}^{M} \lambda_i(\mathbf{x}(k)) \cdot c_{ij}. \tag{1.10}$$

The rules can now be simplified to

$$\mathcal{R}_{j}: \ \textbf{if } u(k) \text{ is } U_j \textbf{ then } y(k+1) = c_{j}. \tag{1.11}$$

The main trick to invert the singleton model is to invert the rule base. There is just one problem: $c_j(k)$ is not a fuzzy set. As a solution, we use the fuzzy sets $C_j(k)$. All rules are thus rewritten as

$$\mathcal{R}_{j}: \ \textbf{if } r(k+1) \text{ is } C_j(k) \textbf{ then } u(k) = U_j. \tag{1.12}$$

The fuzzy sets $C_j(k)$ are defined as to have triangular membership functions. Also, all membership functions add up to one. So,

$$\mu_{C_j(k)}(r) = \begin{cases} \max\left(0, \min\left(1, \frac{c_2 - r}{c_2 - c_1}\right)\right) & \text{if } j = 1, \\ \max\left(0, \min\left(\frac{r - c_{j-1}}{c_j - c_{j-1}}, \frac{c_{j+1} - r}{c_{j+1} - c_j}\right)\right) & \text{if } 1 < j < N, \\ \max\left(0, \min\left(\frac{r - c_{N-1}}{c_N - c_{N-1}}, 1\right)\right) & \text{if } j = N. \end{cases} \tag{1.13}$$

In the above equation, N is the number of fuzzy sets U_j corresponding to the input $u(k)$. By the way, sometimes it may occur that a rule base is not invertible. In this case, you first need to split up the rule base into invertible parts, and afterwards connect them again. However, we won't go any further into detail on that here.

1.4 Other types of inverse control

Some models have an **input delay** n_d. The system model is then given by $y(k+1) = f(\mathbf{x}(k), u(k-n_d))$. We cannot invert this function directly, since $u(k)$ can't affect $y(k+1)$. The first output which is affected by $u(k)$ is $y(k+n_d+1)$. We thus use as inverse function

$$u(k) = f^{-1}(r(k+n_d+1), \mathbf{x}(k+n_d)). \tag{1.14}$$

The only problem is finding $\mathbf{x}(k+n_d)$. Luckily, it does not depend on $u(k)$ or any of the later inputs. And all the previous inputs $u(m)$ with $m < k$ are already known. So, $\mathbf{x}(k+n_d)$ can simply be predicted using our known model.

Another problem occurs when the system is subject to (output) noise. This causes the system output $y(k)$ to be a bit unreliable. In this case, **internal model control** (IMC) offers a solution. When IMC is applied, we use a model of the system without noise. This system predicts the output $y_m(k)$ of the system without noise. The difference $y(k) - y_m(k)$ is then used to change the desired input $r(k+1)$, such that it is more accurately reached. In this way, the effects of the noise are significantly reduced.

2 Other types of knowledge-based control

2.1 Model-based predictive control

Let's suppose that we have a system, of which we have a model f. We now need to decide on a series of H_c inputs $\mathbf{u}$. Based on these inputs, the following H_p predicted outputs $\hat{\mathbf{y}}$ are determined. (To find all these outputs, we usually assume that, after a time $k + H_c$, the input u stays constant.) How do we decide on these inputs?

Usually, the inputs $\mathbf{u}$ are chosen such that a cost function is minimized. This cost function usually looks like

$$J = \sum_{i=1}^{H_p} ||\mathbf{r}(k+i) - \hat{\mathbf{y}}(k+i)||^2_{\mathbf{P_i}} + \sum_{i=1}^{H_c} ||\Delta\mathbf{u}(k+i-1)||^2_{\mathbf{Q_i}}. \tag{2.1}$$

The first parts adds a 'penalty' if the output deviates from the desired output. The second part adds a penalty if the input changes a lot. (That is, if the control effort is high.) The matrices $\mathbf{P_i}$ and $\mathbf{Q_i}$ should be chosen such that the right parts of the input and the output are prioritized/penalized.

When a set of H_c inputs has been decided in this way, only the first of these inputs $\mathbf{u}(k)$ is executed. The resulting output $y(k+1)$ is then examined. Based on this data, a new series of inputs $\mathbf{u}$ is determined, after which again only the first one is executed. This is called the **receding horizon** principle.

You may think that the receding horizon principle is silly: why determine a whole set of future input values, when only the first one of them is executed? The reason behind this is that possibilities for future inputs are also taken into account. If, on the other hand, you only look at the input $\mathbf{u}(k+1)$ at time $k+1$, it might occur that you select an input $\mathbf{u}$ with very good short-term effects, but which does put the system into trouble after that.

2.2 Adaptive control

Sometimes, we encounter a process of which the behavior changes over time. A controller with fixed parameters won't work anymore. Instead, **adaptive control** is required.

We can make a distinction between indirect and direct adaptive control. In **indirect adaptive control**, we use a model of the system which is continuously adapted. (For example, by comparing the predicted output $\mathbf{y_m}$ of the model by the actual output $\mathbf{y}$ of the system.) This model is then used to determine the controller parameters. (For example, the model can be inverted at every time step to find the controller.) On the other hand, in **direct adaptive control**, we don't use a model. We then simply directly adapt the controller parameters.