REDUCING THE NUMBER OF LONG-TERM ITEMS: OPTIMAL STRATEGY FOR A BAG-PACKING PROBLEM

Yoshihiro MURAKAMI*, Junichi KURATA*, Hironobu UCHIYAMA*,
and Wolfgang MARQUARDT**

(Received September 15, 2004)
(Accepted November 30, 2004)

Abstract

When several items having different weights are packed into a bag, the sum weight of each packed bag needs to be constant. The problem at packing is how to determine the items to be included in a bag so that the sum weight should be closest to the desired weight. In this paper, the following two algorithms are shown; one is to solve as an optimization problem and the other is to solve as a constraint satisfaction problem. Comparing the two algorithms, their functional rolls are made clear. In order to reduce the number of items remaining in scale hoppers for a long time, a method is proposed in which each of the two algorithms can be turned over one after another. Simulation results seem to prove that the proposed method is preferable.

1. Introduction

A certain items having different weights are packed into a bag so that the sum weight of each packed bag should be constant. Such items as vegetables, snack foods or cereals, processed foods and machinery can be listed as examples. In order to pack such items together, a weighing machine[1)-4)] that can achieve the given desired weight is produced. In the weighing machine, the weight of items is measured in each scale hopper, and several hoppers are opened so that the sum weight should be closest to the desired weight.

In this paper, the upper limit of desired weight is considered. So the sum weight must not be less than the desired weight and it must not be greater than the upper limit. When this constraint cannot be satisfied, there is no solution and the packing operation cannot be executed. So the fewer the number of cases where there is no solution, the better the performance of the machine is. Therefore, the performance can be measured by examining the rate of finding solutions, which is defined as how many times solutions are found. Also the closer the sum weight is to the desired weight, the better the performance is. Therefore, the performance can also be measured by examining how close each sum weight is to the desired weight.

In normal weighing machine usage, both the above two performance measurements must be good. On the other hand, such items as foodstuffs or frozen foods deteriorate if they remain for a long time in scale hoppers. In such cases, it must be considered how long

*Department of Mechanical Systems Engineering
**Department of Mechanical Engineering, RWTH Aachen, Germany

items can remain in a scale hopper. Therefore, the purpose of this paper is to decrease the number of items remaining for a long time, without worsening the other two performance measurements.

2. Outline of a Weighing Machine

A general machine for achieving the desired weight consists of a supplying unit, a diversifiable feeder, a pool hopper, a scale hopper and so on. It carries out a series of operations, which start with supplying items to the diversifiable feeder, and ends with packing them together into a plastic bag. In Fig. 1, a diagram of such a weighing machine is presented.

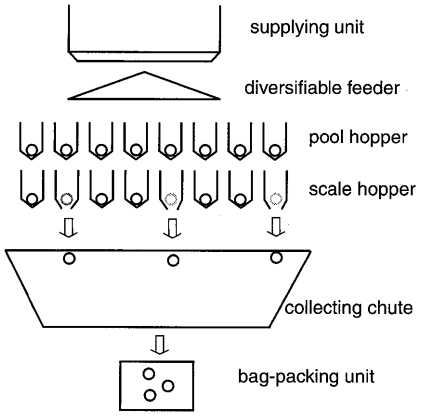

Fig.1 Weighing machine for achieving desired weights

In Fig. 1, items are fed from the supplying unit. In a diversifiable feeder, items are scattered and pour into a pool hopper. The diversifiable feeder can vibrate items and to a certain extent can regulate the quantity of items pouring into pool hoppers. A pool hopper has the function of reducing the vibration influence caused by falling. In a scale hopper, the weight of items is measured by using a load cell as a weighing sensor. After measuring the weight, several hoppers are selected so that the sum weight should be closest to the desired weight. As the result, selected hoppers are opened and several items fall through a collecting chute. Finally, selected items are transferred to an automatic packing machine.

3. How to Solve a Bag-packing Problem (OPT and CSP algorithms)

3.1 Definition of a bag-packing problem

It is assumed that n pairs of scale hoppers are attached to the weighing machine. W_i is defined as the weight of i-th scale hopper. Let 0-1 decision variable be X_i, which is defined as follows; when X_i is 1, the i-th hopper is opened, when X_i is 0, the i-th hopper is not opened.

The sum weight of packed items should satisfy the following constraints; it must not be less than the desired weight, W_{set}, and it must not be greater than the maximum weight,

W_{max}. Here W_{set} and W_{max} are predetermined constants, and the inequality $W_{set} \leqq W_{max}$ must be satisfied. X_i must be determined so that the sum weight should satisfy the upper and lower limit constraints. The following are the constraints that must be satisfied in solving a bag-packing problem ;

$$\sum_{i=1}^{n} W_i X_i \geqq W_{set} \tag{1}$$

$$\sum_{i=1}^{n} W_i X_i \leqq W_{max} \tag{2}$$

$$X_i \in \{0, 1\}, i=1, 2, \cdots, n \tag{3}$$

Here the pair of X_i that satisfies Eq. (1) to (3) is called a solution. In searching for a solution, the pair of X_i that satisfies only Eq. (3) is called a candidate solution.

In order that there should be a candidate solution that satisfies Eq. (1), at least W_i must satisfy Eq. (4).

$$\sum_{i=1}^{n} W_i \geqq W_{set} \tag{4}$$

If $W_i > W_{max}$, W_i cannot be selected as a solution. Therefore, W_i must satisfy Eq. (5), in order that all n pairs of W_i could be selected as a premise for candidate solution.

$$W_i \leqq W_{max}, i = 1, 2, \cdots, n \tag{5}$$

For a specific pair of W_i, no solution can be found even when every candidate solution is enumerated[5]. In this case, the following operation is executed; items are added into one hopper or all the contents of scale hoppers are eliminated. Detailed procedure of this operation is shown in section 3.2.

3.2 Simulation algorithm

Based on the problem definition shown in the previous section, a simulation algorithm for executing a bag-packing operation is shown as follows;

Step 1 $K \leftarrow 0$. $N_b \leftarrow 0$. The value of N_a is given a priori.

Step 2 One item is supplied into each empty hopper. $S_i \leftarrow 0$ for each number of i if the item is newly supplied. $S_i \leftarrow S_i+1$ for every number of i if it is not supplied.

Step 3 It must be checked to ascertain whether W_i can satisfy Eq. (4) and Eq. (5). If W_i cannot satisfy Eq. (4), go to Step 5. If W_i cannot satisfy Eq. (5), go to Step 8.

Step 4 Each of the candidate solutions is enumerated. If a solution (or optimal solution) is found, go to Step 9.

Step 5 If every hopper has two items, go to Step 6. Otherwise, go to Step 7.

Step 6 All the contents in every scale hopper are eliminated. $N_b \leftarrow N_b+1$. Go to Step 2.

Step 7 One item is added only once to a scale hopper that has one item. $S_i \leftarrow S_i+1$ for i=1 to n. Go to Step 3.

Step 8 If $W_i > W_{max}$, the contents of W_i in the i-th hopper are eliminated. Go to Step 2.

Step 9 Open the hoppers that are selected as a solution (or optimal solution). $K \leftarrow K+1$.

Step 10 $S^* \leftarrow S_i$ for each i-th hopper that is selected. If $K < N_a$, go to step 2, otherwise, stop.

In this paper, the above simulation has finished when the packed items tatal 5,000. Therefore N_a is set at 5,000. N_b is the number of cases where there is no solution even if items are added into hoppers. The rate of finding solutions is defined as follows;

$$\alpha = \frac{N_a}{N_a + N_b} \tag{6}$$

The execution of Steps 2, 5, and 7 determines how many items can be held in each hopper. In the above algorithm, it is assumed that one or two items can be held. On the other hand, it is possible to manipulate the number of holding items in each hopper so that the rate of finding solutions is high. In this paper, it is assumed that the average weight of input materials is a constant 27.5g. In this case, it has been examined in advance that the rate of finding solutions is high when one or two items are held in each hopper[6].

In Steps 4 and 9, a solution or optimal solution is selected, depending on the algorithm shown in section 3.3 or 3.4.

S_i and S^* are the variables that tell us how long the items have remained in each hopper. Here, items remaining in a hopper are called 'residues'. The residues have remained continuously S_i steps in i-th hopper. If a new item is supplied into an empty hopper, S_i is renewed as 0. S^* is the final value of S_i when items are eliminated from the scale hopper. When a new item is supplied and it is extracted immediately, then S^*=0. For example, when items in i-th hopper are extracted after Step 4 is executed 10 times, S^*=9. In other words, the items in the i-th hopper have remained continuously 9 times. Note that when a new item is added in Step 7, S_i is not renewed as 0. That is because S^* is calculated for the items that remain in the hopper.

3.3 Problem definition as an optimization problem and OPT algorithm

In this section, a bag-packing problem, in which sum weight is closest to the desired

weight, is defined as an optimization problem[7]. In this case, every solution is enumerated and among them an optimal solution is selected so that the objective function shown in Eq. (7) is minimized.

$$Z = \sum_{i=1}^{n} W_i X_i \tag{7}$$

Such a searching algorithm is defined as an OPT one. For the OPT algorithm, an optimal solution is selected in both Step 4 and Step 9, as shown in section 3.2.

3.4 Problem definition as a constraint satisfaction problem and CSP algorithm

In this section, the bag-packing problem is defined as a constraint satisfaction problem. A candidate solution is enumerated and when it satisfies both Eq. (1) and Eq. (2), the enumeration is terminated. So the solution that is found first is selected in Step 4 and Step 9, as shown in section 3.2. Therefore, it is important to determine how to enumerate a candidate solution. In this paper, because the number of residues needs to be decreased, a method is proposed in which the higher the number of steps remaining in the i-th hopper, the higher the priority for selection is.

In Step 4 of the simulation algorithm, a process enabling enumeration of candidate solutions is illustrated using an example. Let us think of a case where $n = 10$ and $\Sigma_{i=1}^{n} X_i = 6$. Table 1 shows the process of enumerating candidate solutions. The index i shows the hopper number. The position shown as 1 stands for $X_i = 1$. Candidate solutions are enumerated in order from the top line down to the bottom. We then examine whether the selected pair of W_i for each column can satisfy Eq. (1) and Eq. (2).

When candidate solutions are enumerated in the order shown in Table 1, the i-th hopper that has the smaller i, has a higher priority in searching for a solution. However the hopper number i does not necessarily correspond to the actual hopper number i. Therefore, a priority can easily be introduced, which permits the hopper with the highest priority to have the lowest number i.

As the number of residues needs to be decreased, W_i is sorted in the order of hopper with the largest S_i. That is, the larger the steps remaining in the hopper are, the higher the priority is. Using the characteristics of the enumeration process shown in Table 1, such an algorithm is defined as a CSP one, by which we can decrease the number of residues.

Table 1 Enumeration steps in a searching algorithm

	i									
j	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1				
2	1	1	1	1	1		1			
3	1	1	1	1	1			1		
4	1	1	1	1	1				1	
5	1	1	1	1	1					1
6	1	1	1	1		1	1			
7	1	1	1	1		1		1		
8	1	1	1	1		1			1	
9	1	1	1	1		1				1
10	1	1	1	1			1	1		
11	1	1	1	1			1		1	
12	1	1	1	1			1			1
13	1	1	1	1				1	1	
14	1	1	1	1				1		1
15	1	1	1	1					1	1
16	1	1	1		1	1	1			
17	1	1	1		1	1		1		
18	1	1	1		1	1			1	
19	1	1	1		1	1				1
:			:					:		

3.5 Comparison between OPT and CSP algorithms

By executing a bag-packing simulation, the solution characteristics obtained by OPT and CSP algorithms are examined.

It is assumed that $n=10$, $W_{set}=150.0$g, $W_{max}=156.0$g, the weight of input items is generated from a normal distribution that obeys the average $\mu=27.5$g, and the standard deviation $\sigma=3.3$g. The maximum weight of input items is $\mu+3\sigma$, and minimum is $\mu-3\sigma$. Under these parameter settings, there is no case where n pairs of W_i cannot satisfy Eq. (4) and Eq. (5).

In the OPT and CSP algorithms, we examine how many residues there are in the hoppers. The number of residues is compared with the case of OPT and CSP algorithms. Fig. 2 shows the result of OPT algorithm and Fig. 3 shows that of CSP algorithm. The X-axis shows S^*, which is the number of steps remaining. The Y-axis shows the number of items for each value of S^* as a logarithm. The number of residues in Figs. 2 and 3 is the average for executing the simulation 3 times where N_a = 5,000. But as the number of cases where S^* = 0 is very large, they are omitted from the figures.

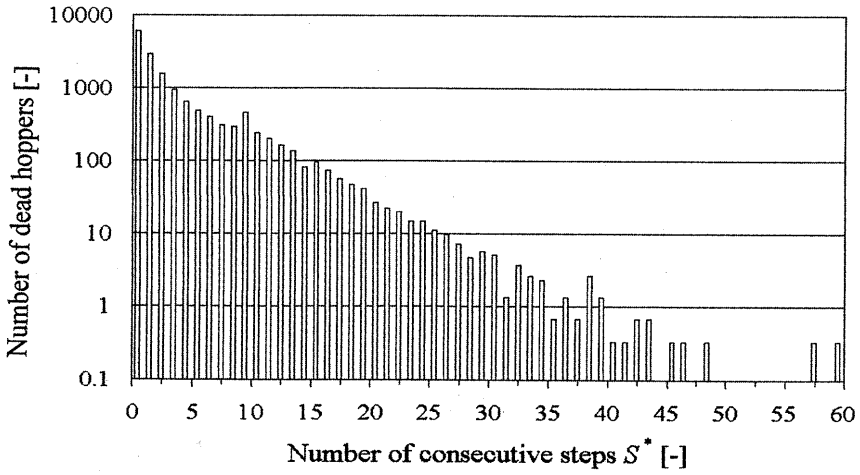

Fig.2 Distribution of the number of residues for S^* (OPT)

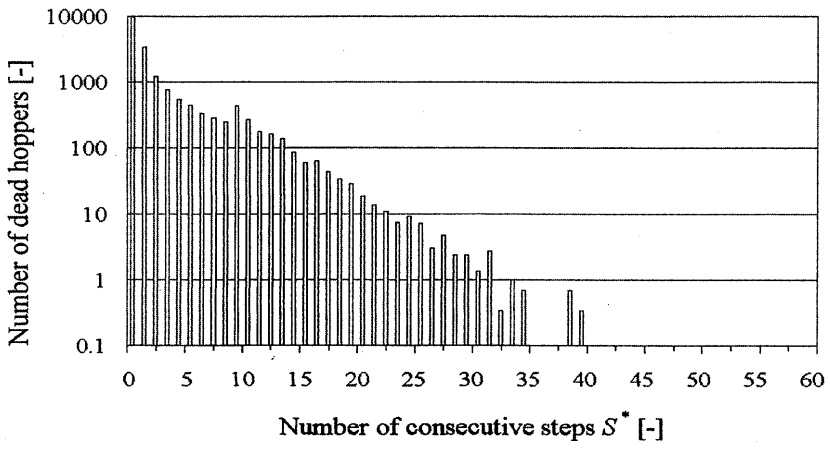

Fig.3 Distribution of the number of residues for S^* (CSP)

Comparing the two results, we see that there are some items remaining for a long time in the OPT algorithm result. On the other hand, there is no item that satisfies $S^* \geqq 41$ in the CSP algorithm result. The CSP one has a marked effect on decreasing the number of items remaining for a long time.

In comparing the two results, the following three performance indices are considered:

1) $\bar{Z}$, the average value of objective function Z
2) α, the rate of finding solutions
3) β, the average number of items remaining for a long time

In 1), $\bar{Z}$ is defined as the average of Z only when a solution is found. It is desirable that $\bar{Z}$ is also as close as possible to the desired value W_{set}. In 2), it is better for α to be higher, which is defined in Eq. (6). In 3), β is defined as the average number of residues that satisfy $S^* \geqq 30$. It is better for β to be smaller, because the number of items remaining for a long time should be minimized.

Table 2 shows the results when the OPT and CSP algorithms are compared for the above three performance indices. The values shown are averaged 3 times of bag-packing simulation where N_a=5,000. The used parameters are the same as those in section 3.5.

Table 2 Comparison of the performance indices of OPT and CSP

	$\bar{Z}$ [g]	α [%]	β [-]
OPT	151.2	96.7	31.0
CSP	152.9	97.4	11.7

In Table 2, the following results are made clear; $\bar{Z}$ obtained by OPT algorithm is the lowest possible value that can be closest to W_{set}. $\bar{Z}$ obtained by CSP algorithm is larger than that by OPT algorithm. α of CSP algorithm is better than that of OPT algorithm. This is because by extracting long-term items with priority, a solution can easily be found in the next search. β of CSP algorithm is much smaller than that of OPT algorithm, which is clear from the results shown in Figs. 2 and 3.

4. Decreasing the Number of Items Remaining for a Long Time

From the results shown in the previous sections, it is clear that, in the OPT algorithm, the sum weight can be made closer to the desired weight, but the number of items remaining for a long time increases. On the other hand, in the CSP algorithm, while the number of items remaining for a long time can be decreased, it is more difficult for the sum weight to be close to the desired weight. Therefore, a method is proposed in which the OPT and CSP algorithms are combined. In this way, we hope the sum weight will be closer to the desired weight, and the number of items remaining for a long time will decrease.

The OPT algorithm is used as an initial step for searching, so that the sum weight should be closest to the desired weight. The number of steps remaining is counted for each item. The CSP algorithm replaces the OPT one if $\max_i\{S_i\}$ equals or exceeds the critical value R. Therefore, even if only one item in a hopper has been remaining R times, the algorithm is

replaced. When the OPT one is replaced with the CSP one, if the number of long-term items has decreased, we reverted to the OPT one. This is because by using only the CSP one we cannot make sum weight closer to the desired weight. The criterion, $\max_i\{S_i\} < R$, is applied for the timing of replacing CSP with OPT. In this paper, such a method is defined as OPT+CSP, in which each of the two searching algorithms is turned over, one after another, according to the number of residues.

The following simulation is executed with the parameter R fixed to constant. In this section R is set to 21 as the items that satisfy $S_i \geqq 21$ seem to be remaining for a long time. Other parameters are the same as for section 3.5. The number of residues is shown in Fig. 4.

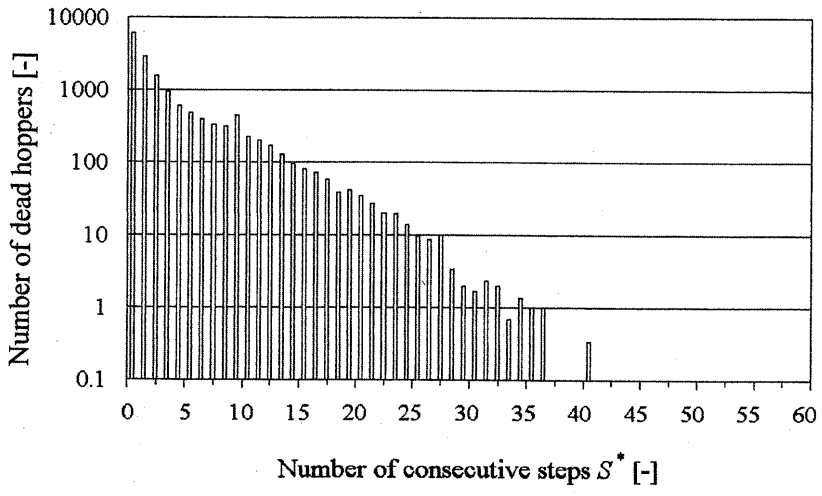

Fig.4 Distribution of the number of residues for S^* (OPT+CSP)

From Fig. 4, we can see that the number of residues that satisfy $S^* \geqq 21$ is decreased, as compared with the OPT result (Fig. 2). But still it is larger than the CSP result (Fig. 3). Even though $R = 21$, there are still residues that satisfy $S^* \geqq 21$. This is because in OPT+CSP the residues that satisfy $S^* \geqq 21$ are not completely eliminated. $N_c = 132.3$ and is defined as the number of times the CSP algorithm is executed when both OPT and CSP algorithms are executed a total of N_a times. And N_c is the average of 3 times where N_a=5,000.

Next, we examined how the three performance indices, α, β, $\bar{Z}$, change when the parameter R is varied. The parameter R is specified as X-axis, and α, β, $\bar{Z}$ as Y-axis. In Fig. 5, results within the range $R = 10$ to 30 are plotted. $N_c = 744.7$ when $R = 10$, and $N_c = 30.3$ when $R = 30$. When R is large, the results are closer to that of the OPT one, but, when R is small, they are closer to that of the CSP one.

Fig. 5 shows that $\bar{Z}$ decreases when R is increased from 10 to 30. But the range of its change is very small. Notably, when $R \geqq 19$, it becomes almost constant. As compared with the OPT result, shown in Table 2, it converges to the lower limit of $\bar{Z}$, which is 151.2g. $\bar{Z}$ of OPT+CSP, as shown in Fig. 5, is small enough when compared with the CSP result shown in Table 2. This is because the OPT algorithm is used more frequently than the CSP one (N_c =

744.7 when R = 10).

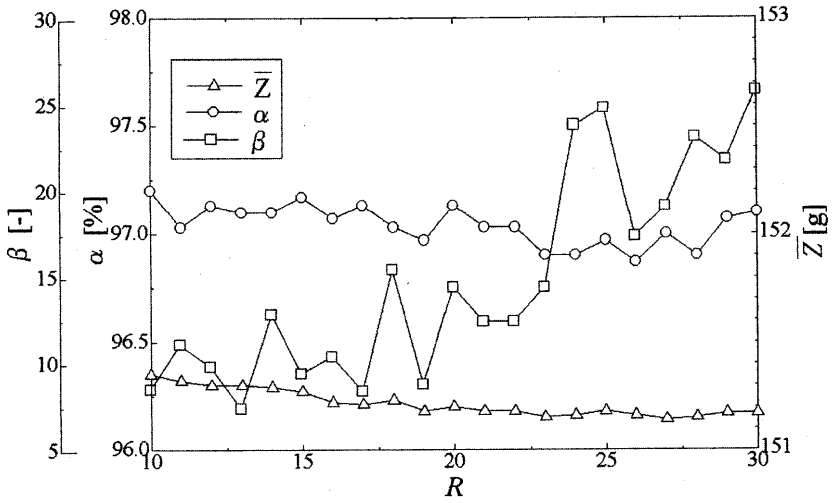

Fig.5 Sensitivity of the parameter R to performance indices (OPT+CSP)

In Fig. 5, α , the rate of finding solutions, is changed in a range of 96.8 − 97.2 %. α does not deteriorate drastically with varying R. α in OPT+CSP is better than that of the OPT algorithm shown in Table 2.

In Fig. 5, we see that β is low when R = 10 to 23. But when R is large, β increases, and it becomes difficult to decrease the number of residues. Therefore, it is not desirable that R is set to a large value.

From the above-mentioned results, it is found that α is high, β is low, and $\bar{Z}$ is closer to W_{set}, in the range of R = 10 − 23. Under the given simulation parameters, we find that OPT+CSP is superior to the result of using only the OPT algorithm or only the CSP one (see Table 2).

5. Conclusion

When items having different weights are packed together, the sum weight should be as close as possible to the desired weight. The following two algorithms were considered: the OPT algorithm for solving the problem by optimization, and the CSP algorithm for solving the problem by constraint satisfaction.

Using the OPT one, the sum weight of packed items is minimized. With the CSP one, however, without considering optimization, solutions that satisfy upper and lower limit constraints are searched. A further function is introduced so that long-term items are reduced.

The simulation results showed that, by using the OPT algorithm, the sum weight can be made closer to the desired weight. However, many items are left remaining for a long time. On the other hand, using the CSP one, number of items remaining for a long time can be decreased, but the sum weight is a little bit too far from the desired weight. Therefore, it is

proposed that, according to the number of steps remaining, each of the two algorithms is turned over one after another (OPT+CSP). Simulation results seem to prove that the proposed OPT+CSP is preferable.

In OPT+CSP, the key parameter to be determined is when each of two algorithms should be turned over. Taking the viewpoint of a maximum time items may be retained in a hopper, the timing of turning over can be determined by simulation a priori.

Acknowledgement

A part of this research was supported by Kansai University's Overseas Program for 2003 and was also supported by the Kansai University HRC project, "Development and Characterization of Sensors and Actuators for Micro Robots".

References

1) http://www.ishida.co.jp
2) K. Kameoka, M. Nakatani and N. Inui, *SICE*, **36**, 366 (2000), (in Japanese)
3) K. Kameoka and M. Nakatani, *SICE*, **37**, 911 (2001) , (in Japanese)
4) Nihon Keiryou Kiki Kougyou Rengoukai (Ed.), "Hakari Handbook", Nikkan Kougyou Shinbunsha, Japan, 1999, p. 115-126 , (in Japanese)
5) Y. Murakami, J. Kuratra, H. Uchiyama, and T. Ueno, *SICE*, **38**, 784 (2002), (in Japanese)
6) M. Kawai, Y. Murakami, J. Kuratra, and H. Uchiyama, Proceedings of Nihon Kikai Gakkai Kansai Sibu Teiji Soukai, **044-1**, 4-11, (2004), (in Japanese)
7) J. Furuya, Y. Murakami, J. Kuratra, and H. Uchiyama, Proceedings of SICE Kansai Sibu Symposium, 155, (1999), (in Japanese)