CONTROL OF RANDOMLY SAMPLED ROBOTIC SYSTEMS

Hiroaki Kobayashi,
Xiaoping Yun and
Richard P. Paul

MS-CIS-89-34
GRASP LAB 181

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104

May 1989

Acknowledgements: This research was supported in part by the National Science Foundation under grants ECS-11879, MCS-8219196-CER, IRI84-10413-AO2, DARPA grant NOOO14-85-K-0018, and U.S. Army grants DAA29-84-K-0061, DAA29-84-9-0027.

Abstract

This paper studies control problems of sampled data systems which are subject to random sample rate variations and delays. Due to the rapid growth of the use of computers more and more systems are controlled digitally. Complex systems such as space telerobotic systems require the integration of a number of sub-systems at different hierarchical levels. While many sub-systems may run on a single processor, some sub-systems require their own processor or processors. The sub-systems are integrated into functioning systems through communications. Communication between processes sharing a single processor are also subject to random delays due to memory management and interrupt latency. Communications between processors involve random delays due to network access and to data collisions. Furthermore, all control processes involve delays due to causal factors in measuring devices and to signal processing.

Traditionally, sampling rates are chosen to meet the worst case communication delay. Such a strategy is wasteful as the processors are then idle a great proportion of the time; sample rates are not as high as possible resulting in poor performance or in the over specification of control processors; there is the possibility of missing data no matter how low the sample rate is picked.

Randomly sampled systems have been studied since later 1950's, however, results on this subject are very limited and they are not applicable to practical systems. This paper studies asymptotical stability with probability one for randomly sampled multi-dimensional linear systems. A sufficient condition for the stability is obtained. This condition is so simple that it can be applied to practical systems. A design procedure is also shown.

These results are applied to robot control systems using PD controllers with a feedforward term, computed torque controllers or simple computed torque controllers. The effectiveness of the method is demonstrated by simulations.

AD A218 948

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

MASTER COPY - FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Center of Excellence in AI University of Pennsylvania 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION U. S. Army Research Office
6c. ADDRESS (City, State, and ZIP Code) Dept. of Computer & Information Science 200 S. 33rd Street Philadelphia, PA 19104-6389	7b. ADDRESS (City, State, and ZIP Code) P. O. Box 12211 Research Triangle Park, NC 27709-2211
8a. NAME OF FUNDING/SPONSORING ORGANIZATION U. S. Army Research Office 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) P. O. Box 12211 Research Triangle Park, NC 27709-2211	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)
Control of Randomly Sampled Robotic Systems (MS-CIS-89-34)

12. PERSONAL AUTHOR(S)
Hiroaki Kobayashi, Xiaoping Yun, Richard Paul

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Interim technical	FROM _____ TO _____	May 1989	92

16. SUPPLEMENTARY NOTATION
The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position policy, or decision, unless so designated by other documentation.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Robotics, robot control

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This paper studies control problems of sampled data systems which are subject to random sample rate variations and delays. Due to the rapid growth of the use of computers more and more systems are controlled digitally. Complex systems such as space telerobotic systems require the integration of a number of sub-systems at different hierarchical levels. While many sub-systems may run on a single processor, some sub-systems require their own processor or processors. The sub-systems are integrated into functioning systems through communications. Communication between processes sharing a single processor are also subject to random delays due to memory management and interrupt latency. Communications between processors involve random delays due to causal factors in measuring devices and to signal processing.

Traditionally, sampling rates are chosen to meet the worst case communication delay. Such a strategy is wasteful as the processors are then idle a great proportion of the time; sample rates are not as high as possible resulting in poor performance or in the over specification of control processors; there is the possibility of missing data no matter how low the sample rate is picked.

Randomly sampled systems have been studied since later 1950's, however, results on this subject are very limited and they are not applicable to practical systems. This paper studies asymptotical stability with probability one

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

for randomly sampled multi-dimensional linear systems. A sufficient condition for the stability is obtained. This condition is so simple that it can be applied to practical systems. A design procedure is also shown.

These results are applied to robot control systems using PD controllers with a feedforward term, computed torque controllers or simple computed torque controllers. The effectiveness of the method is demonstrated by simulations.

UNIVERSITY of PENNSYLVANIA

CONTROL OF RANDOMLY SAMPLED ROBOTIC SYSTEMS

Hiroaki Kobayashi,
Xiaoping Yun and
Richard P. Paul

MS-CIS-89-34
GRASP LAB 181

Department of Computer and Information Science
School of Engineering and Applied Science
Philadelphia, PA 19104-6389

PENN

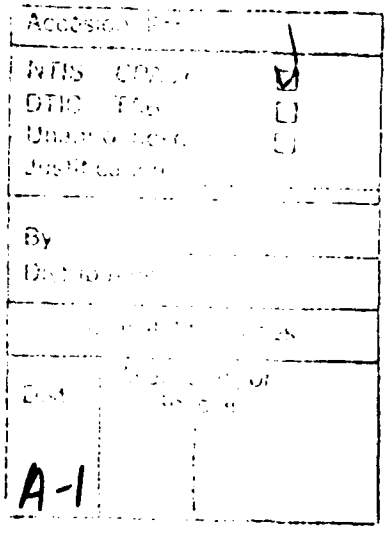
CONTROL OF RANDOMLY SAMPLED ROBOTIC SYSTEMS

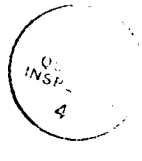
Hiroaki Kobayashi,
Xiaoping Yun and
Richard P. Paul

MS-CIS-89-34
GRASP LAB 181

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104

May 1989





Acknowledgements: This research was supported in part by the National Science Foundation under grants ECS-11879, MCS-8219196-CER, IRI84-10413-AO2, DARPA grant NOOO14-85-K-0018, and U.S. Army grants DAA29-84-K-0061, DAA29-84-9-0027.

Abstract

This paper studies control problems of sampled data systems which are subject to random sample rate variations and delays. Due to the rapid growth of the use of computers more and more systems are controlled digitally. Complex systems such as space telerobotic systems require the integration of a number of sub-systems at different hierarchical levels. While many sub-systems may run on a single processor, some sub-systems require their own processor or processors. The sub-systems are integrated into functioning systems through communications. Communication between processes sharing a single processor are also subject to random delays due to memory management and interrupt latency. Communications between processors involve random delays due to network access and to data collisions. Furthermore, all control processes involve delays due to causal factors in measuring devices and to signal processing.

Traditionally, sampling rates are chosen to meet the worst case communication delay. Such a strategy is wasteful as the processors are then idle a great proportion of the time; sample rates are not as high as possible resulting in poor performance or in the over specification of control processors; there is the possibility of missing data no matter how low the sample rate is picked.

Randomly sampled systems have been studied since later 1950's, however, results on this subject are very limited and they are not applicable to practical systems. This paper studies asymptotical stability with probability one for randomly sampled multi-dimensional linear systems. A sufficient condition for the stability is obtained. This condition is so simple that it can be applied to practical systems. A design procedure is also shown.

These results are applied to robot control systems using PD controllers with a feedforward term, computed torque controllers or simple computed torque controllers. The effectiveness of the method is demonstrated by simulations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robots today have developed beyond the single processor controlled manipulator whose only interaction with the outside world is by means of a small number of binary inputs and outputs. Robots now comprise planning and execution systems, multiple manipulators, stereo vision systems, dense range finders, tactile sensors, etc. These sensors and actuators are typically realized in terms of single or multiple processor systems and their integration into functioning robotic systems is a subject of current research in a number of laboratories. Any form of such integration must be based on communications, both for sending control sequences and collecting measurement data. While the use of large virtual memories have been proposed as a solution, this form of communication is undesirable due to hardware conformity, timing and loading constraints, lack of communications control, the need to link many modules into a single monolithic system, etc. A more attractive communication scheme is based on some form of local area network in which hardware conformity is minimized, timing constraints are virtually eliminated, communications may be monitored and controlled, individual systems may be developed and modified separately, etc. Another important consideration is the replacement of a multi-conductor bus, with severe timing and loading constraints, with a network requiring only a single conductor which may even be disconnected while the system is running.

While all control processes involve delay, causal delay in measurement devices, signal processing, etc., multi-process computer systems involve additional variable delays: interrupt latency, priority scheduling, conditional branching, etc. Local area networks complicate the delay problem with addition delays due to network access and collisions. These delays affect system performance and stability. Designs for such systems must be based on a control theory which includes not only delay elements but elements which have delays with a statistical distribution.

A sampled-data system in which the sample rate is random is called a randomly sampled system. The papers dealing with randomly sampled control systems are not so many. The subject was first addressed by Kalman (1957) [10]. In his paper, bounded-input bounded-output stability problems were discussed for randomly sampled systems in terms of the second moment of the output. Kalman (1962) [11] later discussed optimal regulation problems of randomly varying discrete time linear systems which include randomly sampled systems. Leneman (1968) [15] studied a single-input single-output randomly sampled first order systems and derived a condition for the second moment stability. Kushner and Tobias (1969) [14] studied an autonomous linear system with linear or nonlinear feedback. Using a stochastic Ljapunov function, they obtained conditions for the stability with probability one and the s-th moment stability for one dimensional systems. They also derived a condition which is sufficient for the stability with probability one and is necessary and sufficient for the second moment stability for multi-dimensional systems. Assuming statistical independence among the sampling rates and signals, Dannenberg and Melsa (1975) [5] obtained

the equation of the expectation of the state and the output and investigated the stability in the first moment. In 1982, Koning [12] studied infinite horizon optimal control problems of linear discrete time systems with stochastic parameters. He showed that if the system is second moment stabilizable and second moment observable, then the optimal problem has a unique solution and the closed system is second moment stable. He applied this result to stationary optimal control problems of randomly sampled control systems with long-term average integral criteria and investigated the influence of random sampling on the criterion value by means of simple examples [13]. He pointed out that random sampling may increase or decrease the stability.

Randomness of sampling operation occurs when:

(a) measurements are interrupted randomly by others [7];

(b) the system does not use a constant sampling interval and executes routine tasks such as measuring, calculation, output and pre-calculation repeatedly without any waiting period;

(c) the controllers are interrupted randomly by other controllers or have to wait to access global memory in the network.

For case (a), we can apply a Bernoulli distribution for the probability of the failure of the measurement at each sampling rate [7]. For case (b), the randomness is due to conditional branches, mainly contained in the program, and it is subject to a normal distribution with a small variance or to a shifted waiting distribution[1] with a small average arrival time. For case (c), there are two phases and a combination of two distributions may be reasonable. In a usual operating situation, the sampling interval may be subject to a very narrow distributions such as case (b) but sometimes the controller may have to wait for a rather long time compared to the other case when controllers must exchange their information, access to global memory, and so on. Then the sampling intervals would be subject to a normal distribution with a small variance with probability ε and a shifted waiting distribution with a rather large average arrival time with probability $1-\varepsilon$, where ε is called the hit probability.

This paper studies the stability of randomly sampled systems in relation to the random sampling processes and addresses the communication network requirement in terms of system control performance. Though Kalman [10], Kushner *et al.* [14] and Koning [12] have obtained necessary and sufficient conditions for the stability in the second moment, it is not so easy to apply these conditions to practical systems. This paper studies asymptotic stability with probability one and gives a necessary and sufficient condition for one-dimensional systems and a sufficient condition for multi-dimensional systems. These conditions are easy to verify for given sampling distributions and are thus applicable to practical systems.

It is clear from the study of randomly sampled systems that the distribution of random sampling processes, the characteristic of communication networks, has a decisive influence on the stability and performance of robotic systems. In addition, the results of this paper provide a theoretical basis for designing communication networks for hierarchically distributed control systems such as the NASA/NBS Telerobot Control System Architecture [1].

In the next chapter, we discuss the stability problem for linear time invariant systems. First of all, the asymptotical stability with probability one is defined and a necessary and sufficient condition is given for one-dimensional linear time-invariant randomly sampled systems. Next, the results are extended to multi-dimensional linear time-invariant randomly sampled systems and a sufficient condition is also obtained. In Chapter 3, the stability of a randomly sampled robot control system is considered. By linerizing the system equation and the controller equation along the desired trajectory, linear time-variant randomly sampled systems are derived and the stability is discussed under a PD controller with a feedforward term, a

[1] here a *shifted waiting distribution* means that the sampling interval = constant interval + a random variable which is subject to a waiting distribution.

computed torque controller, and a simple computed torque controller, respectively. In Chater 4, the results given in Chater 3 are applied to PUMA 260. First of all, the linerlized dynamic equation is obtained by a Lisp program. Next, the stability is discussed for the cases of a computed torque controller and a simple computed torque controller. Conclusions are given in Chapter 5.

Chapter 2

Linear Time Invariant Control Systems

In this section, we define the asymptotically stability with probability one for a linear randomly sampled controls systems, and obtain a necessary and sufficient condition for one-dimensional systems. A sufficient condition is given for multi-dimensional systems and a design procedure of feedback gains is also discussed.

2.1 Definition of Stability

Consider a linear time-invariant control system

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2.1}$$

where x is an n-dimensional state vector, u an r-dimensional control vector, and A and B are $n \times n$ and $n \times r$ matrices, respectively. For this system, we apply a constant state feedback input

$$u(t) = Kx(t_k) \tag{2.2}$$

from $t = t_k$ to $t = t_{k+1}(= t_k + \Delta_k)$, where K is an $r \times n$ matrix. Then $x(t_{k+1})$ is given as follows.

$$x(t_{k+1}) = (\Phi(\Delta_k) + \Psi(\Delta_k)K)x(t_k) \tag{2.3}$$

where

$$\Phi(\Delta_k) = \exp(A\Delta_k)$$

and

$$\Psi(\Delta_k) = \int_0^{\Delta_k} \exp(A\tau)d\tau B.$$

Sampling interval Δ_k is assumed to be subject to a same distribution for all k and, Δ_i and $\Delta_j (i \neq j)$ are statistically independent of each other. For simplicity, we write Eq. (2.3) as follows

$$x_{k+1} = \Gamma(\Delta_k)x_k. \tag{2.4}$$

The stability of a randomly sampled control system Eq. (2.4) is defined as follows.

Definition 1 (Stability) *The randomly sampled system Eq. (2.4) is asymptotically stable if for any positive ε and ρ, there exits some integer N such that*

$$Prob[\|x_k\| > \varepsilon] < \rho, \qquad \text{for } k > N$$

for any x_0, or alternatively

$$\lim_{k-\infty} x_k = 0, \qquad \text{with probability one for any } x_0.$$

2.2 One-Dimensional System

In this section, the asymptotic stability defined above is discussed for one-dimensional systems

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2.5}$$

where $n = 1$ and $r = 1$. We define following notations:

$E[\omega]$	:	expectation of random variable ω
$V[\omega]$	:	variance of random variable ω

and assume that

$$V[\log(|\Gamma(\Delta)|)] < \infty \tag{2.6}$$

and

$$E[(\log(|\Gamma(\Delta)|))^3] < \infty \tag{2.7}$$

The following theorem is used in the proof of the next proposition.

Theorem 1 (Berry-Esseen Theorem) *[18] Consider a sequence $x_1, x_2, \cdots$ of independent random variables such that*

$$E[x_i] = 0, \qquad E[x_i^2] = \sigma_i^2, \qquad E[x_i^3] < c\sigma_i^2.$$

We form the sum

$$x = \frac{1}{\sigma}\sum_i x_i, \qquad \sigma^2 = \sum_i \sigma_i^2.$$

Then the distribution of x converses to the normal distribution in the following sense:

$$|f(x) - g(x)| < 4\frac{c}{\sigma}$$

where $f(x)$ and $g(z)$ are the distribution functions of x and the normal random variable z with zero mean and unit variance.

Then the following proposition holds.

Proposition 1 (One-Dimensional System) *Under Assumptions (2.6) and (2.7), one-dimensional randomly sampled system (2.5) is asymptotically stable with probability one if and only if*

$$E[\log(|\Gamma(\Delta)|)] < 0.$$

< proof >

Assuming $x_0 \neq 0$ without loss of generality, from Eq. (2.4) we have

$$\log(|x_k|/|x_0|) = \sum_{i=0}^{k-1} \log(|\Gamma(\Delta_i)|). \tag{2.8}$$

From the statistical independence of Δ_i's and Thebysheff's inequality [19], the following equation is deduced.

$$Prob[|\frac{\log(|x_k/x_0|)}{k} - E| > \varepsilon] \leq \frac{V}{k\varepsilon^2} \tag{2.9}$$

for any ε and any k, where E and V stand for $E[\log(|\Gamma(\Delta)|)]$ and $V[\log(|\Gamma(\Delta)|)]$ respectively.

(i) Sufficiency. Now suppose that $E < 0$ and set ε equal to $-E/2$. Then we have

$$\log(|x_k/x_0|) > -kE/2, \qquad \text{for any } k \text{ with probability less that or equal to } \frac{4V}{E^2k}$$

This shows that for any positive numbers ε and ρ,

$$Prob[|x_k| > \varepsilon] < \rho, \qquad \text{for } k > \max\{-\frac{2}{E}\log(\frac{\varepsilon}{|x_0|}), \frac{\rho E^2}{4V}\}$$

or

$$\lim_{k\to\infty} x_k = 0, \qquad \text{with probability one.}$$

(ii) Necessity. For the case of $E > 0$, if we use $\varepsilon = E/2$, then we have

$$\log(|x_k/x_0|) > kE/2, \qquad \text{for any } k \text{ with probability larger than } 1 - \frac{4V}{E^2k}$$

This means that

$$\lim_{k\to\infty} |x_k| = \infty, \qquad \text{with probability one.}$$

For the case of $E = 0$, from Theorem 1, Assumptions (2.6), and (2.7), we have

$$|f(x) - g(x)| < 4\frac{c}{\sigma}$$

where

$$x = \frac{\log(|x_k/x_0|)}{\sigma}, \qquad \sigma = \sqrt{kV}$$

Therefore we have

$$Prob[|x_0| < |x_k| < |x_0|\exp\{\sqrt{kV}\}] = Prob[0 < x < 1] > \int_0^1 g(x)dx - \frac{4c}{\sqrt{kV}}$$

Since

$$\int_0^1 g(x)dx = 0.34134$$

if we use $\varepsilon = |x_0|$ and $\rho = 0.3$, then we can find out $k > N$ for any N such that

$$Prob[|x_k| > \varepsilon] > \rho$$

□

Note that Assumption (2.7) was used ony in the case of $E = 0$ of Necessity part.

Now defining

$$\gamma(\Delta) = \log(\|\Gamma(\Delta)\|) \tag{2.10}$$

$$g(\Delta) = \int_0^\Delta \gamma(\tau)d\tau \tag{2.11}$$

then we have the following corollary.

Corollary 1 (Bernoulli, Uniform, and Mixed Uniform Distributions)

i. *If the sampling rate Δ is subject to a Bernoulli distribution where $\Delta = \alpha$ with probability p and $\Delta = \beta$ with probability $q = 1-p$, then the one-dimensional randomly sampled system (2.5) is asymptotically stable with probability one, if and only if*

$$p\gamma(\alpha) + q\gamma(\beta) < 0.$$

ii. *If the sampling rate Δ is subject to a uniform distribution $\mathcal{U}[\alpha, \beta]$, then the system is asymptotically stable with probability one, if*

$$g(\beta) - g(\alpha) < 0.$$

iii. *If the sampling rate Δ is subject to $\mathcal{U}[\alpha, \beta]$ with probability ε and to $\mathcal{U}[\gamma, \delta]$ with probability $1-\varepsilon$, then the system is asymptotically stable with probability one, if*

$$\varepsilon\frac{g(\beta) - g(\alpha)}{\beta - \alpha} + (1-\varepsilon)\frac{g(\delta) - g(\gamma)}{\delta - \gamma} < 0.$$

< proof >

i. For the Bernoulli distribution, the expectation is given as follows.

$$E[\log(|\Gamma(\Delta)|)] = p\gamma(\alpha) + q\beta.$$

Assumptions (2.6) and (2.7) are not satisfied only if $\Gamma(\alpha) = 0$ or $\Gamma(\beta) = 0$. Here we assume that $\Gamma(\alpha) = 0$. In this case, we have always $E[\log(|\Gamma(\Delta)|)] < 0$ and

$$Prob[|x_k| \neq 0] = q^k, \qquad \text{for any } x_0 \neq 0.$$

Hence the system is asymptotically stable, and the condition is necessary and sufficient.

ii. Next we consider of the uniform distribution. For this case, the expectation is given by the following equation.

$$E[\log(|\Gamma(\Delta)|)] = \frac{g(\beta) - g(\alpha)}{\beta - \alpha}.$$

Therefore the condition in the corollary implies that the expectation is negative.

From Eq. (2.3), we have

$$\Gamma(\Delta) = \exp\{A\Delta\} + \frac{K}{A}(\exp\{A\Delta\} - 1).$$

Therefore $\log(|\Gamma(\Delta)|)$ is upper bounded for the uniform distribution. If $\Gamma(\Delta) \neq 0$ for all $\Delta \in [\alpha, \beta]$, then the logarithm is also lower bounded and Assumptions (2.6) and (2.7) are satisfied. Hence the corollary is true.

Now we assume that $\Gamma(\Delta) = 0$ for some $\Delta \in [\alpha, \beta]$. Since $\Gamma(\Delta)$ is a monotone function, it has only one root Δ^* at most. We approximate $\Gamma(\Delta)$ by the following $\bar{\Gamma}_\xi(\Delta)$ for a small positive number ξ.

$$\bar{\Gamma}_\xi(\Delta) = \begin{cases} |\Gamma(\Delta)| & \Delta \in [\alpha, \Delta_1] \cup [\Delta_2, \beta] \\ |\Gamma(\Delta_1)| & \Delta \in [\Delta_1, \Delta_2] \end{cases}$$

where

$$\begin{aligned} \Delta_1 &= \Delta^* - \xi \\ \Delta_2 &= \Delta^* - \frac{1}{A}\log(2 - \exp\{A\xi\}) \end{aligned}$$

and consider a autonomous random coefficient system:

$$z_{k+1} = \bar{\Gamma}_\xi(\Delta_k) z_k, \qquad z_0 = x_0.$$

Then we can find out some $\xi > 0$ such that $E[\log(\bar{\Gamma}_\xi(\Delta))] < 0$, because of the continuity of $\Gamma(\Delta)$. Assumptions (2.6) and (2.7) are satisfied for this ξ and the autonomous system is asymptotically stable with probability one. At the same time, we have

$$|x_k| \le |z_k|, \qquad \text{for any } k = 0, 1, 2, \cdots.$$

Therefore the one-dimensional system is also asymptotically stable with probability one.

iii. For the mixed uniform distribution, the proof is trivial from the above proof for the uniform distribution.

□

Example 1 (Bernoulli Distribution) *We consider the stability of the following simple system:*

$$\dot{x} = u \tag{2.12}$$

$$u = -x. \tag{2.13}$$

Then Γ is given as follows:

$$\Gamma = 1 - \Delta.$$

First of all, we assume that the sampling interval Δ subjects to a Bernoulli probability distribution as follows;

$$\Delta = \begin{cases} \alpha & \textit{with probability } p \\ \beta & \textit{with probalility } q \end{cases}$$

where $q = 1 - p$. Then the stability condition is

$$E[\log(\Gamma)] = p\log(|1 - \alpha|) + q\log(|1 - \beta|) < 0$$

or

$$|1 - \alpha|^p |1 - \beta|^q < 1. \tag{2.14}$$

Remarks:

- Eq. (2.14) gives intuitive understanding of the proposition. Namely, for large k, we can expect $k \times p$ times occurrences of $\Delta_i = \alpha$ and $k \times q$ times of $\Delta_i = \beta$. Therefore Eq. (2.14) is a necessary and sufficient condition for the asymptotic stability.

- Kushner and Tobias [14] obtained a sufficient condition for $E[|x(k)|^s] \to 0$ and $|x(k)| \to 0$ with probability one as follows:

$$E[|1-\Delta|^s] < 1. \tag{2.15}$$

This equation implies if $-0.5 < \beta < 2.5$ for $\alpha = p = q = 0.5$, then $|x(k)| \to 0$ with probability one. On the other hand, Eq. (2.14) obtains $-1 < \beta < 3$ for the same situation.

Example 2 (Uniform Distribution) *Next we assume that Δ subjects to a uniform probability distribution $U[\alpha, \beta]$. The probability density function is given as follows:*

$$f(\Delta) = \begin{cases} 1/(\beta-\alpha) & \text{if } \Delta \in [\alpha, \beta], \\ 0 & \text{if } \Delta < \alpha \text{ or } \Delta > \beta. \end{cases}$$

We define a function $g(\Delta)$ as follows:

$$\begin{aligned} g(\Delta) = & \int_0^\Delta \log(|1-\tau|)d\tau \\ = & (\Delta-1)\log(|\Delta-1|) - \Delta, \end{aligned} \tag{2.16}$$

then the asymptotic stability condition is given as follows:

$$E[\log(|1-\Delta|)] = \frac{g(\beta)-g(\alpha)}{\beta-\alpha} < 0 \tag{2.17}$$

or

$$g(\beta) < g(\alpha). \tag{2.18}$$

The graph of $g(\Delta)$ is shown in Fig. 2.1. From this graph, we can obtain the maximum β for given α that makes the system stable. For example, if $\alpha = 0.5$ then β must be less than 3.86 to maintain the stability. Fig. 2.2 shows a simulation for $U[0.5,\ 3.5]$.

Remarks:

- Γ is stable if and only if $\Delta \leq 2$ in usual sense. Therefore the unstability for $\Delta \in (2, 3.5]$ is compensated by the stability for $\Delta \in [0.5,\ 2]$.
- There are upper bounds for both α and β, respectively. Namely, if β is larger than 4.59, then the system is unstable for any α,and if α becomes larger than 2, System (2.12) is always unstable for any β.
- From Eq. (2.15), we define $g_K(\Delta)$ as follows:

$$\begin{aligned} g_K(\Delta) &= \int_0^\Delta |1-\tau|d\tau - \Delta, \\ &= \begin{cases} -\Delta^2/2 & \text{if } 0 \leq \Delta \leq 1, \\ \Delta^2/2 - 2\Delta + 1 & \text{if } 1 < \Delta. \end{cases} \end{aligned} \tag{2.19}$$

$g_K(\Delta)$ is also drawn in Fig. 2.1. It shows that the system is asymptotically stable if $\Delta \in U[0.5, 3.32]$, while this system is asymptotically stable if $\Delta \in U[0.5, 3.86]$.

- If System (2.12) is asymptotically stable for $[\alpha, \beta]$ for $u = -x$, then the system is also asymptotically stable for $[\alpha/K, \beta/K]$ when we use $u = -Kx$ instead of $u = -x$. This implies that there exists K for any α and β which stabilizes the system.

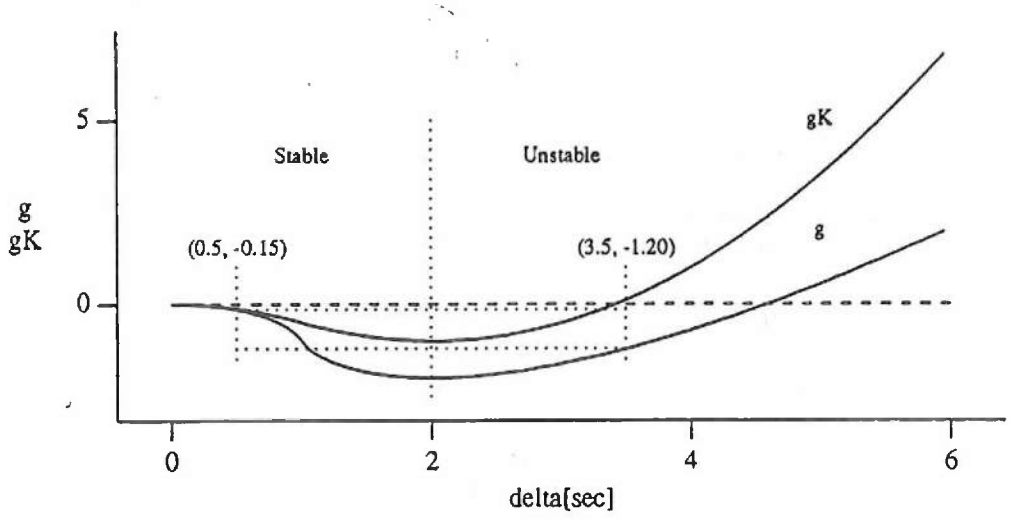


Figure 2.1: function g

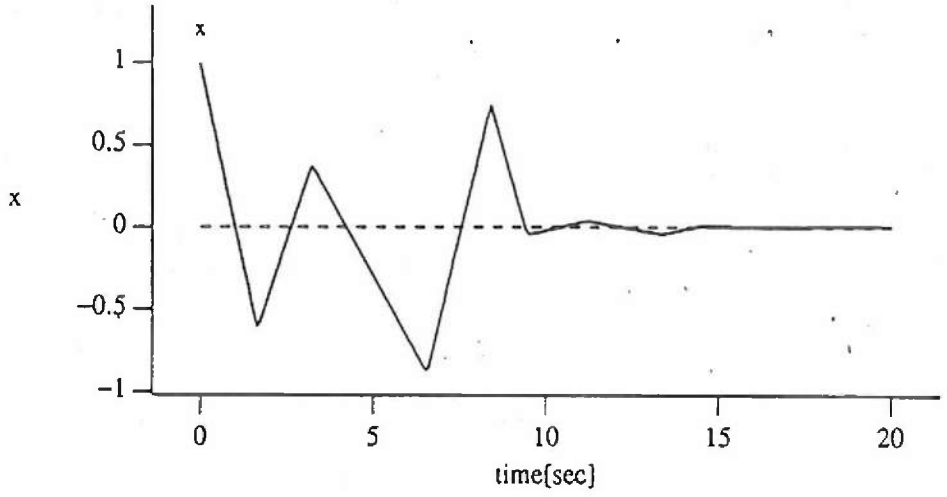


Figure 2.2: Simulation

2.3 Multi-Dimensional Systems

In this subsection, we derive a sufficient condition for the multi-dimensional randomly sampled system Eq. (2.1) under the input Eq. (2.2).

A sufficient condition for the stability can be obtained from Prop. 1 using an appropriate matrix norm instead of the absolute value operation, for example

$$\|\Gamma\| = [\lambda_{\max}(\Gamma^*\Gamma)]^{1/2} \tag{2.20}$$

where Γ^* is the conjugate transformed matrix and $\lambda_{\max}(\Gamma)$ denotes the maximum eigenvalue of the matrix Γ. Note, however, that while the stability of the system (2.1) or (2.3) is invariant under the equivalent transformation of the state variables, the matrix norm depends on the transformation

$$\|\Gamma\| \neq \|T^{-1}\Gamma T\|.$$

This fact is taken into account in the following proposition which gives a sufficient condition for the asymptotical stability with probability one of multi-dimensional systems.

Proposition 2 (Sufficient Condition) *A randomly sampled control system (2.1) is asymptotically stable if there exists a non-singular matrix T such that*

$$E[\log(\|T^{-1}\Gamma(\Delta)T\|)] < 0 \tag{2.21}$$

and

$$V[\log(\|T^{-1}\Gamma(\Delta)T\|)] < \infty. \tag{2.22}$$

The proof is similar to the Sufficiency part of the proof of Prop. 1 except that we must use

$$\|x_k\|/\|x_0\| \leq \|T^{-1}\Gamma(\Delta_k)T\| \cdot \|T^{-1}\Gamma(\Delta_{k-1})T\| \cdots \|T^{-1}\Gamma(\Delta_0)T\| \tag{2.23}$$

instead of Eq. (2.8) in the proof of Prop. 1. Therefore we will omit it here. The next example shows that Conditions (2.21) and (2.22) are not necessary.

Note that for any $n \times n$ matrix Γ and any $\epsilon > 0$, there exits an $n \times n$ non-singular matrix T such that [1]

$$\|T^{-1}\Gamma T\| < |\lambda_{max}(\Gamma)| + \epsilon \tag{2.24}$$

while we always have

$$\|T^{-1}\Gamma T\| \geq |\lambda_{max}(\Gamma)|. \tag{2.25}$$

Therefore if the sampling interval is constant, Prop. 2 gives a necessary and sufficient condition for the asymptotical stability of the system.

Example 3 *Let's consider the stability of the random varying autonomous system as follows:*

$$x_{k+1} = \Gamma x_k$$

and assume $\Gamma = \Gamma_0$ with probability p or Γ_1 with probability $q = 1 - p$, where

$$\Gamma_0 = \begin{bmatrix} 0 & a \\ 0 & 0 \end{bmatrix}$$

[1] After transforming Γ into a Jordan canonical form using T_1, use $T_2 = \text{diag.}\{1, \delta, \ldots, \delta^{n-1}\}$ where δ is a sufficient small positive number depending on ϵ. T is given as $T_1 T_2$

and

$$\Gamma_1 = \begin{bmatrix} 0 & 0 \\ a & 0 \end{bmatrix}$$

and $a > 0$.

Note that $\Gamma_0^2 = \Gamma_1^2 = 0$. *This means that* $x_k \neq 0$ *for some* x_0 *if and only if*

$$x_k = \Gamma_0\Gamma_1\Gamma_0\Gamma_1 \cdots x_0$$

or

$$x_k = \Gamma_1\Gamma_0\Gamma_1\Gamma_0 \cdots x_0.$$

Therefore

$$x_k = 0 \text{ for any } x_0 \text{ with probability } \begin{cases} 1 - 2(pq)^{k/2} & \text{if } k \text{ is even,} \\ 1 - (pq)^{(k-1)/2} & \text{if } k \text{ is odd.} \end{cases}$$

Clearly the system is asymptotically stable for any p, q *and* a.

Now we apply Prop. 2 to this system. If we use the following matrix:

$$T = \begin{bmatrix} 1 & 0 \\ 0 & \delta \end{bmatrix}$$

then we have

$$E[\log(\|\Gamma\|)] = \log a + (p - q)\log\delta.$$

This equation implies that when $a < 1$, *the system is asymptotically stable for any* p *and* q *by setting* $\delta = 1$ *and when* $a \geq 1$, *we can select an appropriate* δ *for which the expectation becomes negative if* $p \neq q$. *Hence the system is asymptotically stable if* $p \neq q$ *or* $p \neq 1/2$.

It is easily to show that the system is asymptotically stable in the first moment if and only if $a^2pq < 1$ *[5] and in the second moment if and only if* $a^4pq < 1$ *[11]. Therefore neither the stability in the first moment nor the one in the second moment implies the condition of Prop. 2, while it gives the nearest condition among of them for this example.*

We have the following corollary.

Corollary 2 *The multi-dimensional randomly sampled system (2.1) with* $n \neq r$ *is asymptotically stable with probability one, if there exist a positive number* c *such that* $f(\Delta) = 0$ *for* $\Delta \geq c$ *and*

$$E[\log(|\Gamma(\Delta)|)] < 0$$

where $f(\Delta)$ *is the distribution function of sampling intervals.*

< proof >

For the multi-dimensional system, Condition (2.22) is not satisfied only if

$$\Gamma(\Delta) = \Phi(\Delta) + \Psi(\Delta)K = 0.$$

However this is impossible because $\Phi(\Delta)$ is a nonsingular matrix, while $\Psi(\Delta)K$ is not so for any Δ if $n \neq r$. Therefore Condition (2.22) is always satisfied. < end of proof >

Also defining

$$\gamma(\Delta) = \log(\|T^{-1}\Gamma(\Delta)T\|) \tag{2.26}$$

$$g(\Delta) = \int_0^\Delta \gamma(\tau)d\tau \tag{2.27}$$

we have the following corollary from Coro. 2.

Corollary 3 (Bernoulli, Uniform, and Mixed Uniform Distributions)

i. *If the sampling rate Δ is subject to a Bernoulli distribution where $\Delta = \alpha$ with probability p and $\Delta = \beta$ with probability $q = 1 - p$, then the multi-dimensional randomly sampled system (2.1) with $n \neq r$ is asymptotically stable with probability one, if*

$$p\gamma(\alpha) + q\gamma(\beta) < 0.$$

ii. *If the sampling rate Δ is subject to a uniform distribution $\mathcal{U}[\alpha, \beta]$, then the system is asymptotically stable with probability one, if*

$$g(\beta) - g(\alpha) < 0.$$

iii. *If the sampling rate Δ is subject to $\mathcal{U}[\alpha, \beta]$ with probability ε and to $\mathcal{U}[\gamma, \delta]$ with probability $1 - \varepsilon$, then the system is asymptotically stable with probability one, if*

$$\varepsilon \frac{g(\beta) - g(\alpha)}{\beta - \alpha} + (1 - \varepsilon) \frac{g(\delta) - g(\gamma)}{\delta - \gamma} < 0.$$

This corollary is trivial from Prop. 2 and Coro. 2.

2.4 Design of Feedback Gains

In the above, we discussed the stability of randomly sampled control systems assuming that the feedback gain K was given. In this section, we discuss the design procedure to select the feedback gain K and matrix T to make the system stable.

If a continuous time system

$$\dot{x}(t) = Ax(t) + B\tilde{u}(t) \tag{2.28}$$

is controllable, it is well known that the discretized system

$$x_{k+1} = \Phi(\Delta)x_k + \Psi(\Delta)u_k \tag{2.29}$$

is also controllable for almost any sampling interval Δ [3]. We can then assign poles $\{\lambda_i, i = 1, 2, \ldots, n\}$ to the system (2.29) using an appropriate state feedback with probability one if the poles $\{\lambda_i\}$ are symmetric with respect to the real axis. So far, a lot of pole assignment algorithms are proposed. Among them, Hikita's pole assignment algorithm[8] is very convenient for us because it gives us not only the feedback gain K but also the transformation matrix T which we can use to check the stability by using Prop. 2. Hence we modify it as follows:

[Algorithm]

step (i) *For a given $\{\lambda_i\}$, find the r-dimensional vectors $\{\xi_i\}$ where $i = 1, 2, \ldots, n$ which make the matrix $T(\hat{\Delta}) = [v_1 : \cdots : v_n]$ non-singular with v_i's being defined by the following equations, in which $\Phi = \Phi(\hat{\Delta})$ and $\Psi = \Psi(\hat{\Delta})$.*

- *if λ_i is a real number, then*

$$v_i = (\Phi - \lambda_i I_n)^{-1} \Psi \xi_i. \tag{2.30}$$

- *if λ_i and λ_{i+1} are conjugate complex numbers $\alpha_i \pm j\beta_i$, then*

$$\begin{aligned} v_i &= V_{1i}\xi_i - V_{2i}\xi_{i+1} \\ v_{i+1} &= V_{1i}\xi_i + V_{2i}\xi_{i+1} \end{aligned} \tag{2.31}$$

where

$$V_{1i} = \{(\Phi - \alpha_i I_n)^2 + \beta_i^2 I_n\}^{-1}(\Phi - \alpha_i I_n)\Psi$$
$$V_{2i} = \{(\Phi - \alpha_i I_n)^2 + \beta_i^2 I_n\}^{-1}\beta_i \Psi.$$

step (ii) *Feedback gain K is given as follows.*

$$K(\hat{\Delta}) = -[\xi_1 : \cdots : \xi_n]T(\hat{\Delta})^{-1}. \tag{2.32}$$

It is easy to show that if λ_i is a real number

$$(\Phi + \Psi K)v_i = \lambda_i v_i \tag{2.33}$$

and if $\lambda_i, \lambda_{i+1} = \alpha_i \pm j\beta_i$,

$$(\Phi + \Psi K)v_i = \alpha_i v_i - \beta_i v_{i+1} \tag{2.34}$$
$$(\Phi + \Psi K)v_{i+1} = \beta_i v_i + \alpha_i v_{i+1}. \tag{2.35}$$

This implies

$$||T^{-1}(\hat{\Delta})\Gamma(\hat{\Delta})T(\hat{\Delta})|| = \sigma(\Gamma(\hat{\Delta})). \tag{2.36}$$

Hence we can use matrices $T(\hat{\Delta})$ and $K(\hat{\Delta})$ to check the stability.

2.5 Two Dimensional Systems

We view a robot manipulator as a component of a large system, such as a space station. The robot controller communicates with the other components of the system to achieve cooperative actions. Communication between components is considered to have a longer delay than that within a component. We assume that robot controller has an inner feedback loop which compensates the nonlinearity of manipulator dynamics and operates independently of the other part of the system. The robot dynamic system together with the inner feedback loop becomes a linear system. It is feasible to treat the robot manipulator subsystem as a linear system when integrating and communicating with the other components. For example, if we use the nonlinear feedback controller developed in [2], we have r (=DOF of manipulator) decoupled two-dimensional linear systems

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \tag{2.37}$$

where $x(t) = (e_i(t), \dot{e}_i(t))$ is the error vector for the i-th component of outputs and $u(t)$ is the corresponding input for this component of outputs. If the task is specified in joint space (the joint space control), the i-th component of output is simply the displacement of the i-th joint and the error vector is composed of the joint position error and joint velocity error. This system also can be used as an approximation subsystem of a robot manipulator which is controlled by a randomly sampled computed torque controller or a randomly sampled simple computed torque controller as shown in the next chapter, where the i-th component of out put is the joint position error and joint error, precisely.

We now study the asymptotical stability of this system under the random sampling rate. The corresponding discrete time system is easily obtained for a sampling interval Δ as follows.

$$x_{k+1} = \begin{bmatrix} 1 & \Delta_k \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} \Delta_k^2/2 \\ \Delta_k \end{bmatrix} u_k. \tag{2.38}$$

We apply the algorithm given above to this system directly. Then we have the following proposition.

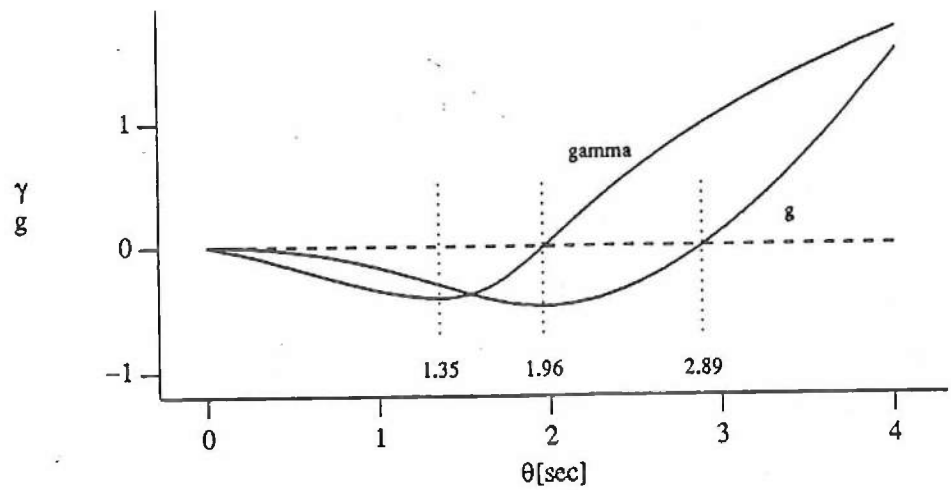


Figure 2.3: function $\gamma(\theta, 1)$ and $g(\theta, 1)$

Proposition 3 (PD Controller) *Assume that* $\{\lambda_i\} = \{\lambda_1, \lambda_2\}$ *where* $\lambda_1 \neq \lambda_2$, *then we have*

$$K(\hat{\Delta}) = ((\lambda_1 + \lambda_2 - \lambda_1\lambda_2 - 1)/\hat{\Delta}^2, (\lambda_1 + \lambda_2 + \lambda_1\lambda_2 - 2)/(2\hat{\Delta})),$$

$$T(\hat{\Delta}) = \begin{bmatrix} -\xi_1\hat{\Delta}(1+\lambda_1) & -\xi_2\hat{\Delta}(1+\lambda_2) \\ 2\xi_1(1-\lambda_1) & 2\xi_2(1-\lambda_1) \end{bmatrix},$$

and

$$\gamma(\Delta, \hat{\Delta}) = \gamma(\theta, 1),$$

where $\theta = \Delta/\hat{\Delta}$.

The proof is obtained by direct calculation. This proposition implies that the function $\gamma(\Delta, \hat{\Delta})$ is the same as the function $\gamma(\theta, 1)$ if we use $K(\hat{\Delta}) = (k_p/\hat{\Delta}^2, k_v/\hat{\Delta})$ instead of $K(1) = (k_p, k_v)$. Therefore we have $g(\Delta, \hat{\Delta}) = \hat{\Delta}g(\theta, 1)$ for the same $K(\hat{\Delta})$. This fact is very useful to design the feedback gain. This will be shown by examples.

Fig. 2.3 shows $\gamma(\theta, 1)$ and $g(\theta, 1)$ for $\lambda_1 = 0.4$ and $\lambda_2 = 0.7$, where we have

$$K(1) = -(0.18, 0.81), \text{ and } T(1) = \begin{bmatrix} -0.759 & -0.943 \\ 0.651 & 0.333 \end{bmatrix}, \tag{2.39}$$

and ξ_i was used to make the norm of column vectors of T matrix be equal to one.

2.5.1 Bernoulli Distribution

Let's assume that the sampling interval is subject to Bernoulli distribution, i.e. $\Delta = \alpha$ with probability p and $\Delta = \beta$ with probalility q, where $\alpha < \beta$, $0 \leq p \leq 1$, and $q = 1 - p$. The sufficient stability condition is given as follows.

$$p\gamma(\alpha/\hat{\Delta}, 1) + q\gamma(\beta/\hat{\Delta}, 1) < 0. \tag{2.40}$$

Note that if $\hat{\Delta} \geq \beta/1.96 (= \hat{\Delta}^*)$ then the system is asymptotically stable for any α because $\gamma(\theta, 1) < 0$ for any $\theta \leq 1.96$. But we are generally interested in the smallest $\hat{\Delta}$ because it gives us the fastest response.

Fig. 2.3 shows that the function $\gamma(\theta, 1)$ reaches the minimum value -0.417 at $\theta = 1.35$. Let θ^* be the point which satisfies the following equation.

$$\gamma(\theta^*, 1) = \frac{p}{q} \times 0.417.$$

Then it is clear that $\hat{\Delta}$ must be greater than $\hat{\Delta}_{min}(= \beta/\theta^*)$ for Eq. (2.40).

A suitable value of $\hat{\Delta}$ can be found from the range $\hat{\Delta}_{min} < \hat{\Delta} < \hat{\Delta}^*$ by a trial-and-error method using Fig. 2.3 or Table 2.1 which gives pairs of $\{\theta_1, \theta_2\}$ such that $\gamma(\theta_1, 1) = \gamma(\theta_2, 1)$.

(i) Calculate $a = -(q/p)\gamma(\beta/\hat{\Delta}, 1)$.

(ii) Find $\{\theta_1, \theta_2\}$ such that $\gamma(\theta_1, 1) = \gamma(\theta_2, 1) \leq a$ using Fig. 2.3 or Table 2.1.

(iii) Check $\theta_1 < \alpha/\hat{\Delta} < \theta_2$. If so, calculate $K(\hat{\Delta})$. If not so, go back to step (i) with another $\hat{\Delta}$.

For example, if $\alpha = 10\ msec$, $\beta = 30\ msec$, and $p = 0.75$, then θ^* is about 3.64 and $\hat{\Delta}_{min} = 8.24\ msec$, while $\hat{\Delta}^* = 15.3\ msec$. If we select $\hat{\Delta} = 11\ msec$ then $\frac{q}{p}\gamma(\beta/\hat{\Delta}, 1) = -0.278$ and $\alpha/\hat{\Delta} = 0.91$. Therefore we can try the 6-th row of Table. 2.1, and we have $\theta_1 = 0.84 < 0.91 < \theta_2 = 1.68$. Hence the system is asymptotically stable for $K = -(1488, 73.64)$.

2.5.2 Uniform Distribution

Now assume that Δ is subject to a uniform distribution $\mathcal{U}[\alpha, \beta]$. The sufficient condition of the asymptotical stability with probability one is given as follows:

$$g(\alpha/\hat{\Delta}, 1) > g(\beta/\hat{\Delta}, 1).$$

The function $g(\theta, 1)$ has its minimum value at $\theta = 1.96$. Now we define $\hat{\Delta}^* = \beta/19.6$ and $\hat{\Delta}_{min} = \beta/2.89$. If $\hat{\Delta} \geq \hat{\Delta}^*$, then the above sufficient condition is satisfied for any α. Therefore the system is asymptotically stable if $\hat{\Delta} \geq \hat{\Delta}^*$. On the other hand, if $\hat{\Delta} \leq \hat{\Delta}_{min}$, then the above condition is not satisfied for any α.

Table 2.1 also gives pairs of $\{\theta_3, \theta_4\}$ and the ratio θ_3/θ_4 such that $g(\theta_3, 1) = g(\theta_4, 1)$. If there is a pair $\{\theta_3, \theta_4\}$ such that $\alpha/\beta > \theta_3/\theta_4$, then the system is asymptotically stable for the $K(\hat{\Delta})$ where $\hat{\Delta} = \alpha/\theta_3$. Therefor we can determine $\hat{\Delta}$ easily using this table as follows:

(i) Calculate $a = \alpha/\beta$.

(ii) Find a pair $\{\theta_3, \theta_4\}$ in the Table 2.1 such that $a > \theta_3/\theta_4$.

(ii) Calculate $\hat{\Delta} = \alpha/\theta_3$ and $K(\hat{\Delta})$.

Now assume that $\alpha = 10\ msec$ and $\beta = 30\ msec$, then we have $\hat{\Delta}^* = 15.3\ msec$, $\hat{\Delta}_{min} = 10.38\ msec$, and $\alpha/\beta = 1/3 > 0.273$ in the Table 2.1. Therefore we can use $\alpha/\hat{\Delta} = 0.75$ and $\hat{\Delta} = 13.33\ msec$. Hence the system is asymptotically stable with $K = -(1065, 62.31)$ if $\beta < 36.7\ msec$. Table 2.2 shows the IAE (Integration of Absolute value of the Error) for fifty random streams with the initial condition $x(0) = (1.0, 0)^T$. The table shows that when $\beta \geq 40 msec$, the STD (STanderd Deviation) and the maximum values of IAE for the velocity error $\dot{e}_i(t)$ become very large compared to the cases where $\beta \leq 35 msec$. This means that the system is still stable but there is a large vibration in the response for $\hat{\Delta} \geq 40\ msec$. It is interesting since $\hat{\Delta}$ selected above assures the asymptotically stability for $\beta < 36.7\ msec$.

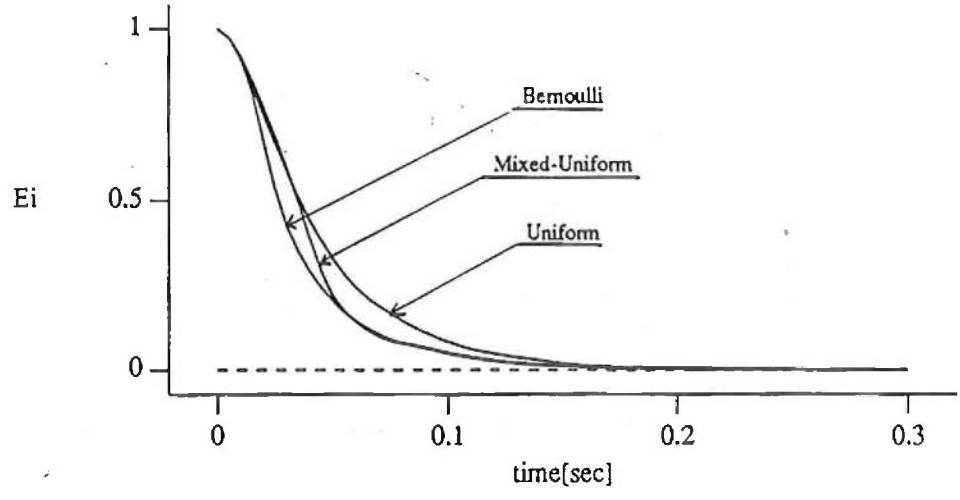


Figure 2.4: Simulations for Bernoulli Distribution, Uniform Distribution and Mixed Uniform Distribution

2.5.3 Mixed Uniform Distribution

Next we assume that Δ is subject to a uniform distribution $\mathcal{U}[\alpha, \beta]$ with probability ε and to $\mathcal{U}[\mu, \nu]$ with probability $1 - \varepsilon$. The sufficient condition is given as follows:

$$E = \varepsilon \frac{g(\beta/\hat{\Delta}, 1) - g(\alpha/\hat{\Delta}, 1)}{\beta/\hat{\Delta} - \alpha/\hat{\Delta}} + (1-\varepsilon)\frac{g(\nu/\hat{\Delta}, 1) - g(\mu/\hat{\Delta}, 1)}{\nu/\hat{\Delta} - \mu/\hat{\Delta}} < 0.$$

Though the selection of $\hat{\Delta}$ becomes a little difficult, we can use the following procedure to estimate an appropriate $\hat{\Delta}$:

(i) Define $\bar{\alpha} = (\alpha + \beta)/2.0$, $\bar{\beta} = (\mu + \nu)/2.0$, $p = \varepsilon$, and $q = 1 - p$.

(ii) Determine $\hat{\Delta}$ using the procedure in Exam. 1 for $\alpha = \bar{\alpha}$ and $\beta = \bar{\beta}$.

(iii) Check the condition. If satisfied, calculate $K(\hat{\Delta})$. If not, try another value for $\hat{\Delta}$.

Now assume that Δ is subject to $\mathcal{U}[5\,msec, 15\,msec]$ with probability $\varepsilon = 0.75$ and to $\mathcal{U}[20\,msec, 40\,msec]$ with probability 0.25. Then we have $\bar{\alpha} = 10\ msec$, $\bar{\beta} = 30\ msec$, $p = 0.75$, and $q = 0.25$. If we use $\hat{\Delta} = 11\ msec$ from the result of Exam. 1, then we have $E = -0.04 < 0$. Therefore the system is asymptotically stable for the same $K = -(1488, 73.64)$.

Fig. 2.4 shows the simulations of $x(t)$ for three cases discussed above where $x(0) = (1.0, 0)^T$.

2.6 PID Controller and PD Controller with One Step Delay

Here a PID controller and a PD controller with one step delay for a random sampled system are discussed.

2.6.1 PID Controller

We consider a linear time-constant control system

$$\dot{x}(t) = Ax(t) + Bu(t) + d, \tag{2.41}$$
$$y(t) = Cx(t), \tag{2.42}$$

$\gamma(\theta_1,1)=\gamma(\theta_2,1)$			$g(\theta_3,1)=g(\theta_4,1)$			
$\gamma(\theta,1)$	θ_1	$\theta 2$	$g(\theta,1)$	θ_3	θ_4	θ_3/θ_4
0.00	0.00	1.96	0.000	0.00	2.88	0.000
-0.05	0.18	1.92	-0.009	0.25	2.87	0.087
-0.10	0.33	1.89	-0.039	0.50	2.84	0.176
-0.15	0.46	1.83	-0.094	0.75	2.78	0.270
-0.20	0.58	1.79	-0.173	1.00	2.69	0.372
-0.25	0.71	1.73	-0.270	1.25	2.56	0.488
-0.30	0.84	1.68	-0.373	1.50	2.39	0.628
-0.35	0.98	1.60	-0.456	1.75	2.17	0.806
-0.40	1.18	1.48	-0.467	1.80	2.12	0.849

Table 2.1: θ_1, θ_2 ,θ_3, and θ_4

	MEAN		STD		MAX	
β	$e_i(t)$	$\dot{e}_i(t)$	$e_i(t)$	$\dot{e}_i(t)$	$e_i(t)$	$\dot{e}_i(t)$
25	0.0531	0.9999	0.0022	0.0011	0.0572	1.0020
30	0.0511	1.0039	0.0043	0.0200	0.0561	1.1310
35	0.0498	1.0198	0.0060	0.0515	0.0589	1.2370
40	0.0509	1.2418	0.0077	0.4305	0.0717	2.7051
45	0.0709	2.6492	0.0638	4.4592	0.4354	30.276

Table 2.2: IAE for $U[10\ msec, \beta\ msec]$

where d and $y(t)$ are an n-dimensional constant disturbance and an r-dimensional output vector, respectively. We assume that this system is controllable and observable. Then a continuous-time PID controller is given as follows:

$$\dot{z}(t) = \hat{y} - y(t) \tag{2.43}$$

$$u(t) = K_1 x(t) + K_2 z(t), \tag{2.44}$$

where $\hat{y}$ is an r-dimensional constant reference vector. It is well known that if and only if

$$\det \begin{bmatrix} A & B \\ C & 0 \end{bmatrix} \neq 0, \tag{2.45}$$

there exist feedback gains K_1 and K_2 such that $x(t) \to \hat{y}$ as $t \to \infty$ for any $x(0)$, $\hat{y}$ and d [3].

Now we consider a randomly sampled system controlled by a PID controller as follows:

$$x_{k+1} = \Phi(\Delta_k)x_k + \Psi(\Delta_k)u_k + d \tag{2.46}$$

$$y_k = Cx_k \tag{2.47}$$

$$z_{k+1} = z_k + (\hat{y} - y_k), \tag{2.48}$$

$$u_k = K_1 x_k + K_2 z_k, \tag{2.49}$$

Then we have

$$\begin{pmatrix} x_{k+1} \\ z_{k+1} \end{pmatrix} = \bar{\Phi}(\Delta_k) \begin{pmatrix} x_k \\ z_k \end{pmatrix} + \bar{\Psi}(\Delta_k)u_k + \begin{pmatrix} d \\ \hat{y} \end{pmatrix}, \tag{2.50}$$

$$u_k = \bar{K} \begin{pmatrix} x_k \\ z_k \end{pmatrix}, \tag{2.51}$$

where

$$\bar{\Phi}(\Delta_k) = \begin{bmatrix} \Phi(\Delta_k) & 0 \\ -C & I_r \end{bmatrix}, \qquad \bar{\Psi}(\Delta_k) = \begin{bmatrix} \Psi(\Delta_k) \\ 0 \end{bmatrix}, \qquad \bar{K} = [K_1\ K_2].$$

Now we assume that Condition (2.45) is satisfied, then it is easy to show that for almost all Δ_k,

$$\det \begin{bmatrix} \Phi(\Delta_k) - I_n & \Psi(\Delta_k) \\ -C & 0 \end{bmatrix} \neq 0 \tag{2.52}$$

is satisfied, too. This implies that System (2.50) is controllable for almost all Δ_k. Hence we can apply the algorithm described in Section 2.4 to choose feedback gain $\bar{K}$ and transformation matrix $\bar{T}$ for which the randomly sampled system becomes asymptotically stable with probability one. Then we have

$$\|x_{k+1} - x_k\| \to 0, \text{ or } x_k \to \hat{y} \text{ as } k \to \infty \text{ with probability one.} \tag{2.53}$$

For two-dimensional system (2.37), we have

$$\bar{\Phi}(\Delta_k) = \begin{bmatrix} 1 & \Delta_k & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}, \qquad \bar{\Psi}(\Delta_k) = \begin{bmatrix} \Delta_k^2/2 \\ \Delta_k \\ 0 \end{bmatrix}, \tag{2.54}$$

where we assumed $C = [1\ 0]$. By applying the algorithm in Section 2.4, we have following proposition:

Proposition 4 (PID Controller) *Let $T(\bar{1}.0)$ and $K(\bar{1}.0)$ be given by the algorithm in Section 2.4 for extended system (2.50) by using $\hat{\Delta} = 1$. If we use the algorithm for a $\hat{\Delta}$, then we have*

$$\bar{T}(\hat{\Delta}) = \begin{bmatrix} \hat{\Delta} t_1 \\ t_2 \\ \hat{\Delta} t_3 \end{bmatrix}, \qquad \bar{K}(\hat{\Delta}) = [k_1/\hat{\Delta}^2\ k_2/\hat{\Delta}\ k_3/\hat{\Delta}^2],$$

and

$$\bar{\gamma}(\Delta, \hat{\Delta}) = \bar{\gamma}(\Delta/\hat{\Delta}, 1.0)$$

where

$$\bar{\gamma}(\Delta, \hat{\Delta}) = \log(\|\bar{T}^{-1}(\hat{\Delta})\{\bar{\Phi}(\Delta) + \bar{\Psi}(\Delta)\bar{K}(\hat{\Delta})\}\bar{T}(\hat{\Delta})\|),$$

t_i and k_i are the i-th row vector of $\bar{T}(1.0)$ and the i-th element of $\bar{K}(1.0)$, respectively.

The proof is given by direct calculation. Therefore we can use the similar procedures to ones described in above section to determine the appropriate $\hat{\Delta}$ for given sampling interval distribution.

Example 4 *Fig. 2.5 shows the $\gamma(\Delta, 1.0)$ and $g(\Delta, 1.0)$ for $\{\lambda_i\} = \{0.95, 0.7, 0.4\}$ and*

$$T(1.0) = \begin{bmatrix} -0.0499 & -0.0715 & -0.0294 \\ 0.0026 & 0.0252 & 0.0252 \\ -0.9986 & -0.2382 & -0.0491 \end{bmatrix}.$$

The feedback gain is $K(1.0) = [-0.221, -0.840, 0.009]$. Note that $\gamma(\Delta, 1.0)$ becomes positive for small Δ. This is due to the change of the direction of eigen vectors. For $\Delta \in U[10\ msec, 30\ msec]$, we have $\hat{\Delta} = 14.29\ msec$ and

$$\begin{aligned} z_{k+1} &= z_k + (\hat{y} - y_k), \\ u_k &= -[1083, 58.80]x_k + 44.10 z_k. \end{aligned}$$

Fig. 2.6 shows a simulation for $x_0 = 0$ and $\hat{y} = 1.0$. Disturbance $d = (0.0, 1.0)^T$ is applied to continuous time system (2.42). The result shows a smooth convergence into $\hat{y}$.

2.6.2 PD Controller with One Step Delay

If u_k is calculated using not x_k but x_{k-1}, then we have a one step delay controller. This is common situation for Robot controllers. A simple way to compensate the delay for constant sampling time system is to use a controller as follows:

$$u_k = K(\Phi(\Delta)x_{k-1} + \Psi(\Delta)u_{k-1}). \tag{2.55}$$

Here we consider the stability of the system controlled by the above controller under random sampling intervals. Namely, we use a PD controller with one step delay as follows:

$$u_k = K(\hat{\Delta})(\Phi(\hat{\Delta})x_{k-1} + \Psi(\hat{\Delta})u_{k-1}), \tag{2.56}$$

where $K(\hat{\Delta})$ is determined using the algorithm in Section 2.4 for a $\hat{\Delta}$. Of course, $T(\hat{\Delta})$ is also determined at the same time. Then we have following extended equation:

$$\begin{pmatrix} x_{k+1} \\ u_{k+1} \end{pmatrix} = \begin{bmatrix} \Phi(\Delta_k) & \Psi(\Delta_k) \\ K(\hat{\Delta})\Phi(\hat{\Delta}) & K(\hat{\Delta})\Psi(\hat{\Delta}) \end{bmatrix} \begin{pmatrix} x_k \\ u_k \end{pmatrix}. \tag{2.57}$$

If we use

$$\bar{T}(\hat{\Delta}) = \begin{bmatrix} T(\hat{\Delta}) & 0 \\ K(\hat{\Delta})T(\hat{\Delta}) & \delta I_r \end{bmatrix}, \tag{2.58}$$

where δ is a small real number, then we have

$$\bar{T}^{-1}(\hat{\Delta}) \begin{bmatrix} \Phi & \Psi \\ \hat{K}\Phi & \hat{K}\Psi \end{bmatrix} \bar{T}(\hat{\Delta}) = \begin{bmatrix} \hat{T}^{-1}\Gamma\hat{T} & \delta\hat{T}^{-1}\Psi \\ -\hat{K}(\hat{\Gamma} - \Gamma)\hat{T}/\delta & \hat{K}(\hat{\Psi} - \Psi) \end{bmatrix}, \tag{2.59}$$

where $\Phi = \Phi(\Delta), \Psi = \Psi(\Delta), \hat{K} = K(\hat{\Delta}), \hat{T} = T(\hat{\Delta})$ and $\Gamma = \Phi + \Psi\hat{K}$. $\hat{\Phi}, \hat{\Psi}$, and $\hat{\Gamma}$ are corresponding matrices for $\Delta = \hat{\Delta}$, respectively. Therefore we can check the stability of the system using matrix $\bar{T}(\hat{\Delta})$.

For the two-dimensional systems such as Eq. (2.37), we have proposition as follows:

Proposition 5 (PD Controller with One Step Delay) *For two-dimensional system (2.37), we have*

$$\bar{\gamma}(\Delta, \hat{\Delta}) = \bar{\gamma}(\Delta/\hat{\Delta}, 1.0),$$

where

$$\bar{\gamma}(\Delta, \hat{\Delta}) = \log(\| \begin{bmatrix} \hat{T}^{-1}\Gamma\hat{T} & \delta\hat{T}^{-1}\Psi \\ \hat{K}(\hat{\Gamma} - \Gamma)\hat{T}/\delta & \hat{K}(\hat{\Psi} - \Psi) \end{bmatrix} \|).$$

The proof is also straightforward.

Example 5 *Fig. 2.7 shows $\gamma(\Delta, 1.0)$ and $g(\Delta, 1.0)$ for $\{\lambda_i\} = \{0.95, 0.90\}$ and $\delta = 0.005$. Then we have $K(1.0) = -[0.0050, 0.1478]$ and*

$$T(1.0) = \begin{bmatrix} -0.9986 & -0.9945 & 0.000 \\ 0.0513 & 0.1047 & 0.000 \\ -0.0026 & -0.0010 & 0.005 \end{bmatrix}.$$

We must use eigenvalues near to $1 + j0$ in order to get a wide minus zone of function $\gamma(\Delta, 1.0)$ and this results in rather small feedback gain $K(1.0)$. Now assume that usually the controller works every 5 msec

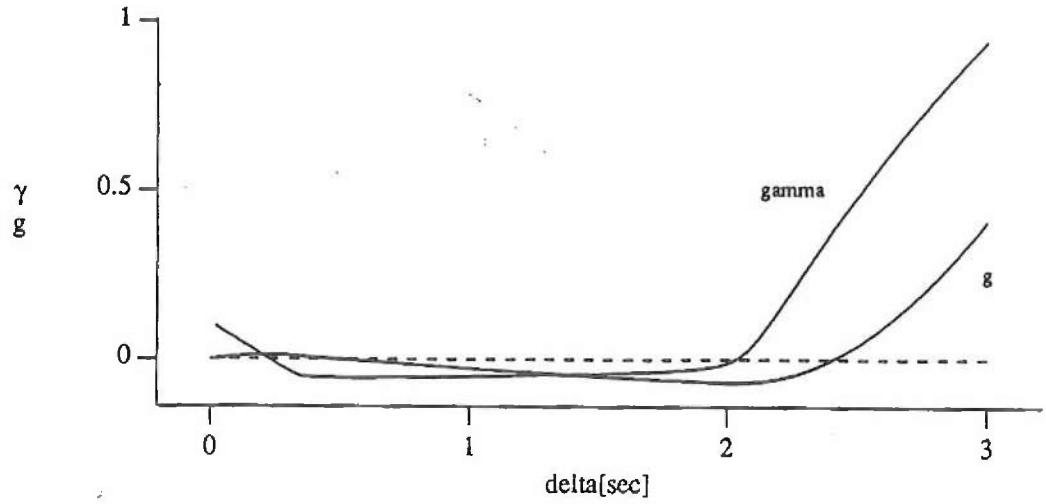


Figure 2.5: $\gamma(\Delta, 1.0)$ and $g(\Delta, 1.0)$ for PID Controller

with probability 0.9 *but it must sometimes wait for from* 10 *msec to* 20 *msec with probability* 0.1. *Then the system is asymptotically stable if*

$$0.9 * \gamma(5\ msec, \hat{\Delta}) + 0.1 * \frac{g(20\ msec, \hat{\Delta}) - g(10\ msec, \hat{\Delta})}{10\ msec} < 0,$$

or

$$0.9 * \gamma(5\ msec/\hat{\Delta}, 1.0) + \frac{0.1 * \hat{\Delta}}{10\ msec}\{g(20\ msec/\hat{\Delta}, 1.0) - g(10\ msec/\hat{\Delta}, 1.0)\}.$$

If we use $\hat{\Delta} = 2.5\ msec$, *it is shown that the above inequality is satisfied and we have*

$$u_k = -(800.0, 61.21)x_{k-1} - 0.1503u_{k-1}.$$

Fig. 2.8 shows a simulation for $x_0 = (1, 0)^T$. *This figure also shows the smooth and rapid convergence to zero state.*

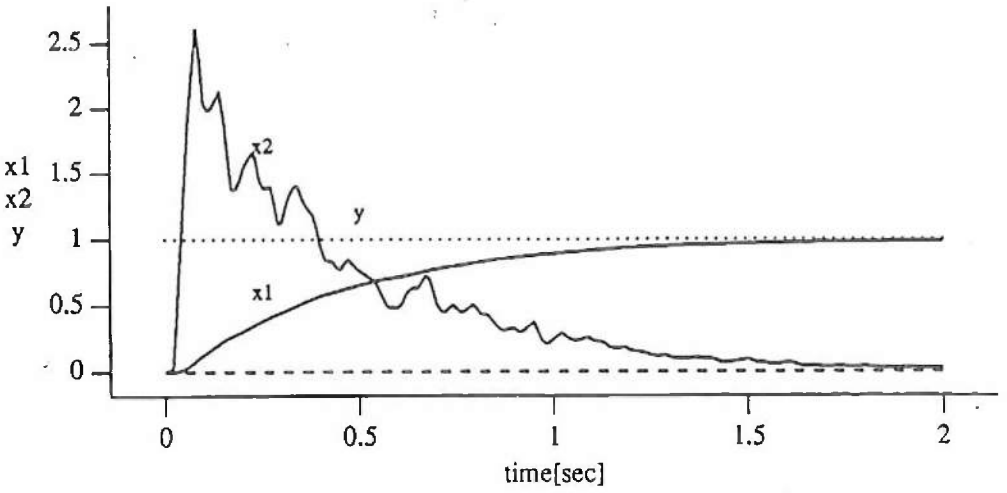


Figure 2.6: Simulation for PID Controller

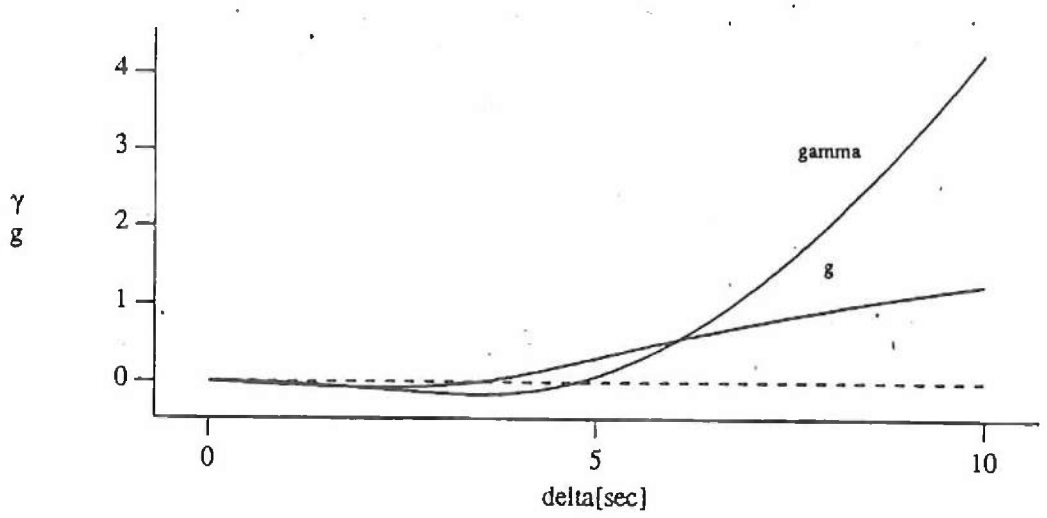


Figure 2.7: $\gamma(\Delta, 1.0)$ and $g(\Delta, 1.0)$ for PD Controller with One Step Delay

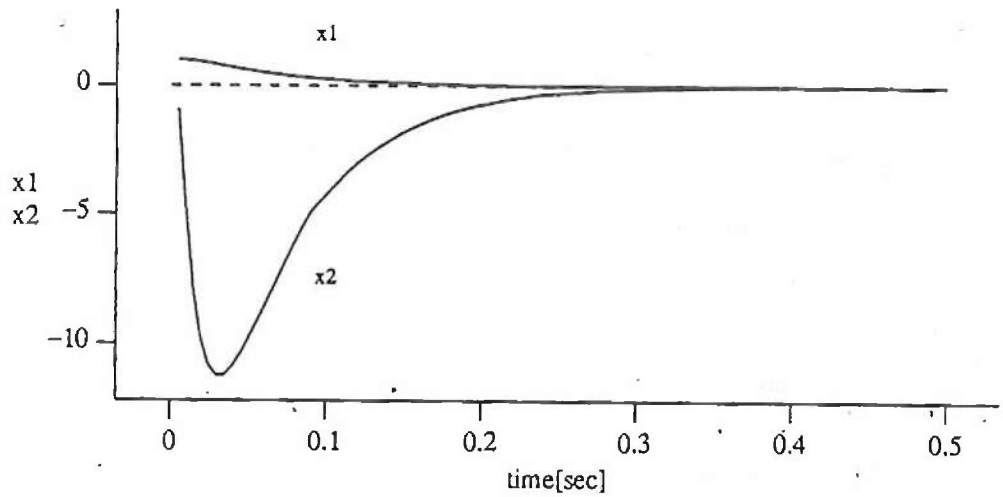


Figure 2.8: Simulation for PD Controller with One Step Delay

Chapter 3

Robot Manipulators

In this chapter, we consider the asymptotical stability with probability one of randomly sampled robot systems. Dynamic equations of robot systems are usually described by non-linear functions and controllers also have non-linearity. Therefore, first of all, these equations are linearized along desired trajectories to time-variant multi-dimensional linear systems and, next, the results given above are applied to them.

Here we consider three kinds of controllers:

- PD controllers with feedforward terms [16],
- Conputed Torque Controllers [20] [17],
- Simple Computed Torque Controller,

and discuss their effectiveness under the random sampling situation.

3.1 Stability Condition

In this section we will consider the stability of randomly sampled robot control systems. The dynamic equation of a robot and the equation of a controller are given as follows.

$$\dot{x}(t) = f(x(t), u_k) = f_1(x(t)) + f_2(x(t))u_k \tag{3.1}$$

$$u_k = h(x(t_k)), \qquad t_k \leq t < t_{k+1} = (t_k + \Delta_k) \tag{3.2}$$

where $x(t)$ and u_k are the state vector and the input vector of the robot, respectively. f and h are smooth non-linear functions. We assume that a planned trajectory $\{\bar{x}(t) : t_0 \leq t\}$ is given and also assume that

$$\dot{\bar{x}}(t_k) = f(\bar{x}(t_k), \bar{u}_k) \tag{3.3}$$

$$\bar{u}_k = h(\bar{x}(t_k)). \tag{3.4}$$

Then using the following variables

$$x(t) = \bar{x}(t) + \delta x(t) \tag{3.5}$$

$$u_k = \bar{u}_k + \delta u_k \tag{3.6}$$

and linearizing the above equations around the trajectory, we have

$$\delta\dot{x}(t) = A(t)\delta x(t) + B(t)\delta u_k + d(t) \tag{3.7}$$
$$\delta u_k = C_k \delta x(t_k) \tag{3.8}$$

where

$$\begin{aligned} A(t) &= [\partial f/\partial x](\bar{x}(t), \bar{u}_k) \\ B(t) &= f_2(\bar{x}(t)) \\ C_k &= [\partial h/\partial x](\bar{x}(t_k)) \\ d(t) &= f(\bar{x}(t), \bar{u}_k) - \dot{\bar{x}}(t) \\ \partial f/\partial x &= [\partial f/\partial x_1 : \cdots : \partial f/\partial x_n]. \end{aligned}$$

$d(t)$ is a disturbance due to using sampling controllers for the system and it is omitted from the system equation in the following discussion of the stability.

$\delta x(t_{k+1})$ can be calculated for a given $\delta x(t_k)$ and δu_k as follows:

$$\delta x(t_{k+1}) = \Phi(t_{k+1}, t_k)\delta x(t_k) + \Psi(t_{k+1}, t_k)\delta u_k. \tag{3.9}$$

where $\Phi(t_{k+1}, t_k)$ is the unique solution of

$$\frac{\partial}{\partial t}\Phi(t, t_k) = A(t)\Phi(t, t_k) \qquad \Phi(t_k, t_k) = I_n$$

and $\Psi(t_{k+1}, t_k)$ is given by

$$\Psi(t_{k+1}, t_k) = \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, t)B(t)dt.$$

By substituting δu_k of Eq. (3.8) into the above equation, we have

$$\delta x(t_{k+1}) = \Gamma(t_k, \Delta_k)\delta x(t_k) \tag{3.10}$$

where

$$\Gamma(t_k, \Delta_k) = \Phi(t_{k+1}, t_k) + \Psi(t_{k+1}, t_k)C_k.$$

We now define a random variable γ for any non-singular matrix T as follows.

$$\gamma(t_k, \Delta_k) = \log(\|T^{-1}\Gamma(t_k, \Delta_k)T\|). \tag{3.11}$$

Note that $\gamma(t_k, \Delta_k)$ is a function of $\Delta_0, \cdots, \Delta_k$, therefore $\gamma(t_k, \Delta_k)$ and $\gamma(t_{k+l}, \Delta_{k+l})$ are not independent of each other, even though we have assumed that Δ_k's are in this paper. On the other hand, we have

$$\gamma(t_k, 0) = 0 \tag{3.12}$$

for any $t_k \geq t_0$. This denotes that the effects of $\Delta_i, i = 0, \cdots, k-1$ are not accumulated on $\gamma(t_k, \Delta_k)$, and we can expect $\gamma(t_k, \Delta_k)$ and $\gamma(t_{k+l}, \Delta_l)$ to become independent statistically as l increases.

We now introduce some definitions.

Definition 2 (weak correlation) *The robot system is weakly correlated along the trajectory* $\{\bar{x}(t) : t_0 \leq t\}$, *if there exists an integer* L *such that*

$$Cor[\gamma(t_k, \Delta_k), \gamma(t_{k+l}, \Delta_{k+l})] = 0$$

for $k = 0, 1, 2, \cdots$ *and integer* $l > L$, *where* $Cor[a, b]$ *means the correlation coefficient between random variables* a *and* b *and* $t_{k+l} = t_k + \Delta_k + \cdots + \Delta_{k+l-1}$.

Definition 3 *The randomly sampled robot system is asymptotically stable with probability one along the trajectory* $\{\bar{x}(t) : t_0 \leq t\}$, *if*

$$Prob[\lim_{k\to\infty} ||\delta x(t_k)|| = 0] = 1$$

for any $\delta x(t_0)$.

We then have the following proposition.

Proposition 6 (Stability of Robot Manipulator) *Assume that the robot system is weakly correlated along the trajectory* $\{\bar{x}(t) : t_0 \leq t\}$. *The robot system is asymptotically stable with probability one along the trajectory , if there exist two numbers* σ *and* $\mu < 0$ *such that*

$$\begin{aligned} E[\gamma(t, \Delta)|t] &\leq \mu < 0 \\ E[\{\gamma(t, \Delta)\}^2|t] &\leq \sigma^2 + \mu^2 < \infty \end{aligned}$$

for any $t \geq t_0$, *where* $E[a|b]$ *is the conditional expectation of* a *given* b.

< proof >

From Eq. (3.10), we have

$$\delta x(t_k) = \Gamma(t_{k-1}, \Delta_{k-1}) \cdots \Gamma(t_0, \Delta_0)\delta x(t_0).$$

Therefore

$$log(||T^{-1}\delta x(t_k)||) \leq \log(||T^{-1}\delta x(t_0)||) + \sum_{i=0}^{k-1} \gamma(t_i, \Delta_i).$$

Note that

$$\begin{aligned} E[\gamma(t_i, \Delta_i)] &= E[E[\gamma(t_i, \Delta_i)|t_i]] \leq \mu \\ V[\gamma(t_i, \Delta_i)] &= E[E[\{\gamma(t_i, \Delta_i)\}^2|t_i]] - \{E[\gamma(t_i, \Delta_i)]\}^2 \leq \sigma^2 \end{aligned}$$

for $i < j$. Now define z_k as follows.

$$z_k = \frac{1}{k}\sum_{i=0}^{k-1} \gamma(t_i, \Delta_i)$$

then we have

$$\begin{aligned} E[z_k] &\leq \mu \\ V[z_k] &\leq \sigma^2 \frac{(1+2L)k - L - 1}{k^2} \end{aligned} \tag{3.13}$$

where $\gamma_i = \gamma(t_i, \Delta_i)$ and we used $|Cov[\gamma_i, \gamma_j]| \leq \sqrt{Cov[\gamma_i]Cov[\gamma_j]} \leq \sigma^2$ where $Cov[\gamma_i, \gamma_j]$ is the covariance of γ_i and γ_j. This means that $V[z_k] \to 0$ as $k \to \infty$, and by Tchebycheff's inequality we have

$$Prob[z_k < \mu/2] \geq Prob[|z_k - E[Z_k]| < \mu/2] > 1 - \frac{4V[z_k]}{\mu^2}.$$

Hence we have

$$\log(\|T^{-1}\delta x(t_k)\|) \to -\infty$$

with probability one for any x_0 as $k \to \infty$. □

Note that the condition of the above proposition does not depend on t_0 at all. Therefore if the proposition holds, we have

$$\lim_{k\to\infty} \|\delta x(t'_k)\| = 0$$

with probability one for any $\delta x(t'_0), (t'_0 \geq t_0)$, where $t'_k = t'_0 + \Delta'_0 + \cdots + \Delta'_{k-1}$.

We now define

$$g(t, \Delta) = \int_0^{\Delta} \gamma(t, \tau) d\tau \tag{3.14}$$

then we have the following proposition.

Proposition 7 (Bernoulli, Uniform, and Mixed Uniform Distributions) *Assume that the robot system is weakly correlated along the trajectory $\{\bar{x}(t) : t_0 \leq t\}$, and*

$$E[\{\gamma(t, \Delta)\}^2 | t] < \sigma^2 < \infty$$

for any $t \geq t_0$.

i. *If the sampling rate Δ is subject to a Bernoulli distribution where $\Delta = \alpha$ with probability p and $\Delta = \beta$ with probability $q = 1 - p$, then the randomly sampled robot system is asymptotically stable along the trajectory with probability one, if there exists a negative number μ such that*

$$p\gamma(t, \alpha) + q\gamma(t, \beta) < \mu < 0$$

for any $t \geq t_0$.

ii. *If the sampling rate Δ is subject to a uniform distribution $U[\alpha, \beta]$, then the system is asymptotically stable with probability one, if there exists a negative number μ such that*

$$g(t, \beta) - g(t, \alpha) < \mu < 0$$

for any $t \geq t_0$.

iii. *If the sampling rate Δ is subject to $U[\alpha, \beta]$ with probability ε and to $U[\gamma, \delta]$ with probability $1 - \varepsilon$, then the system is asymptotically stable with probability one, if there exists a negative number μ such that*

$$\varepsilon \frac{g(t, \beta) - g(t, \alpha)}{\beta - \alpha} + (1 - \varepsilon) \frac{g(t, \delta) - g(t, \gamma)}{\delta - \gamma} < \mu < 0$$

for any $t \geq t_0$.

In practical situation, the dimensions of the state vector and the input vector of robot systems are different ($n = 2r$, in general), the trajectory is defined in a finite time interval $[t_0, t_f] t_f - t_0 < \infty$ and the sampling intervals have a finite upper limit. We have the following corollary for such case.

Corollary 4 *If the dimensions of the state vector and the input vector of the robot system are different, the trajectory is defined in a finite time interval* $[t_0, t_f]$ $(t_f - t_0 < \infty)$ *and there is a positive number* c *such that* $f(\Delta) = 0$ *for ant* $\Delta > c$, *then there exists a finite number* σ *such that*

$$E[\{\gamma(t,\Delta)\}^2|t] < \sigma^2 < \infty$$

where for any $t \in [t_0, t_f]$.

< proof >

We denote set $[t_0, t_f] \times [0, c]$ as $\mathcal{T}$ in this proof. Clearly $\mathcal{T}$ is a closed set in a usual phase. Therefore there is an upper bound a such that $\|T^{-1}\Gamma(t,\Delta)T\| < a$ for all $(t,\Delta) \in \mathcal{T}$ because of continuity of the function.

Just same as multi-dimensional time-invariant system, $\gamma(t,\Delta)$ becomes infinite only if

$$\Gamma(t,\Delta) = \Phi(t+\Delta,t) + \Psi(t+\Delta,t)K = 0.$$

$\Phi(t+\Delta,t)$ is nonsingular for any (t,Δ) while $\Psi(t+\Delta,t)K$ is not so for any (t,Δ) if $n \neq r$. Therefore $\|T^{-1}\Gamma(t,\Delta)T\| \neq 0$ for any $(t,\Delta) \in \mathcal{T} = [t_0,t_f] \times [0,c]$. If there is a sequence of pair $(t_i,\Delta_i), i = 1,2,3\cdots$ such that $\|T^{-1}\Gamma(t_i,\Delta_i)T\| \to 0$ as $i \to \infty$, there is a accumulating point (t_a,Δ_a) which belongs to $\mathcal{T}$ because the set is closed. This implies $\Gamma(t_a,\Delta_a) = 0$ and contradicts the fact shown above. Hence there is a lower bound $b > 0$ such that $\|T^{-1}\Gamma(t,\Delta)T\| > b$. Therefore we have

$$E[\{\gamma(t,\Delta)\}^2|t] < \max\{\{\log|a|\}^2, \{\log|b|\}^2\} < \infty.$$

□

3.2 Linearized Systems, Controllers and Matrix T

The typical dynamic equation of a robot manipulator is given as follows:

$$\begin{aligned} \tau &= M(q)\ddot{q} + C(q,\dot{q}) + B(\dot{q}) + G(q) \\ &= R(q,\dot{q},\ddot{q}) \end{aligned} \tag{3.15}$$

where τ is the joint force/torque vector, q the joint variable vector, and M, C, B, and G correspond to the inertia tensor, the centrifugal and Coriolis force, the viscous friction, and the gravitational force, respectively. Coulomb forces are omitted because we consider the linearization of the equation here.

Then the robot dynamic equation (3.15) is linearized along a trajectory $\{\bar{q}(t), \dot{\bar{q}}(t), \ddot{\bar{q}}(t)\}$ at t_k as follows.

$$\delta\dot{x} = A(t)\delta x(t) + B(t)\delta u_k \tag{3.16}$$

where

$$\begin{aligned} \delta x &= \begin{bmatrix} q - \bar{q} \\ \dot{q} - \dot{\bar{q}} \end{bmatrix} \\ A(t) &= \begin{bmatrix} 0 & I_r \\ -\bar{M}^{-1}(\partial R/\partial q) & -\bar{M}^{-1}(\partial R/\partial \dot{q}) \end{bmatrix} \\ B(t) &= \begin{bmatrix} 0 \\ \bar{M}^{-1} \end{bmatrix} \\ \delta u_k &= h(q(t_k),\dot{q}(t_k),\ddot{q}(t_k)) - h(\bar{q}(t_k),\dot{\bar{q}}(t_k),\ddot{\bar{q}}(t_k)) \end{aligned}$$

$\bar{M} = M(\bar{q}(t))$, and partial deviatives are calculated at $(\bar{q}(t), \dot{\bar{q}}(t), \ddot{\bar{q}}(t))$.

δu_k is obtained for a PD controller with a feedforward term, a computed torque controller, and a simple computed torque controller, respectively, as follows:

(i) A PD controller with a feedforward term

The control law is expressed as follows.

$$u_k = K_p(\bar{q}(t_k) - q(t_k)) + K_v(\dot{\bar{q}}(t_k) - \dot{q}(t_k)) + \bar{u}_k \tag{3.17}$$

where K_p and K_v are the proportional and differential gains respectively. Then we have the linearized equation for the input as follows.

$$\delta u_k = C_k \delta x(t_k) \tag{3.18}$$

where

$$C_k = -[K_p \; K_v].$$

(ii) A computed torque controller

The control law is expressed as follows.

$$u_k = R(q(t_k), \dot{q}(t_k), \ddot{q}^*(t_k)) \tag{3.19}$$

where

$$\ddot{q}^*(t_k) = \ddot{\bar{q}}(t_k) + K_p(\bar{q}(t_k) - q(t_k)) + K_v(\dot{\bar{q}}(t_k) - \dot{q}(t_k))$$

and K_p and K_v are the proportional and differential gains, respectively.

For this input, we have

$$\delta u_k = C_k \delta x(t_k) \tag{3.20}$$

where

$$C_k = [\partial R/\partial q - \bar{M}_k K_p \; \partial R/\partial \dot{q} - \bar{M}_k K_v]$$

$\bar{M}_k = M(\bar{q}(t_k))$, and partial deviatives are calculated at $(\bar{q}(t_k), \dot{\bar{q}}(t_k), \ddot{\bar{q}}(t_k))$.

(iii) A simple computed torque controller

If we use

$$\begin{aligned} u_k &= R(\bar{q}(t_k), \dot{\bar{q}}(t_k), \ddot{q}^*(t_k)) \\ &\text{or} \\ &= \bar{M}_k\{K_p(\bar{q}(t_k) - q(t_k)) + K_v(\dot{\bar{q}}(t_k) - \dot{q}(t_k))\} + \bar{u}_k \end{aligned}$$

then we have

$$\delta u_k = C_k \delta x(t_k) \tag{3.21}$$

where

$$C_k = -\bar{M}_k[K_p \; K_v].$$

Now, a method to determine matrices K_p, K_v and T is to use the algorithm shown in the above section for a multi-dimensional linear randomly sampled system. Namely:

step(i) *Select an appropriate time instance $\hat{t} \geq t_0$ and a sampling interval $\hat{\Delta}$,and calculate matrices $A(\hat{t})$, $B(\hat{t})$, $\Phi(\hat{t} + \hat{\Delta}, \hat{t})$ and $\Psi(\hat{t} + \hat{\Delta}, \hat{t})$.*

step(ii) *Choose appropriate eigenvalues λ_i's and vectors ξ_i's.*

step(iii) *Determine a feedback gain K and matrix T using algorithm given in Section 2.4.*

step(iv) *Determine matrices K_p and K_v as follows.*

For a PD controller with a feedforward term,

$$[K_p\ K_v] = -K$$

for a computed torque controller,

$$[K_p\ K_v] = -B_2(\hat{t})K - A_2(\hat{t})$$

and for a simple computed torque controller,

$$[K_p\ K_v] = -B_2(\hat{t})K$$

where $A_2(\hat{t})$ and $B_2(\hat{t})$ are the lower half sub-matrices of $A(\hat{t})$ and $B(\hat{t})$, respectively.

step(v) *Check the stability using Props. 6 or 7.*

3.3 Examples

In this section we apply the result given above to a simple three dof robot "ROBOTEC" shown in Fig. 3.1. The dynamic equation is given as follows.

$$\tau = M(q)\ddot{q} + C(q,\dot{q}) + B\dot{q} + G(q) \tag{3.22}$$

where

$$\begin{aligned}
M &= \begin{bmatrix} \alpha S_2^2 + 2\beta S_2 C_3 + \gamma C_3^2 + m_1 & 0 & 0 \\ 0 & \alpha + m_2 & \beta S_{23} \\ 0 & \beta S_{23} & \gamma + m_3 \end{bmatrix} \\
C &= \begin{bmatrix} 2\{(\alpha S_2 + \beta C_3)C_2\dot{q}_2 - (\beta S_2 + \gamma C_3)S_3\dot{q}_3\}\dot{q}_1 \\ -(\alpha S_2 + \beta S_3)C_2\dot{q}_1^2 + \beta C_{23}\dot{q}_3^2 \\ (\beta S_2 + \gamma S_3)S_3\dot{q}_1^2 + \beta C_{23}\dot{q}_2^2 \end{bmatrix} \\
B &= \begin{bmatrix} b_1 & 0 & 0 \\ 0 & b_2 & 0 \\ 0 & 0 & b_3 \end{bmatrix} \\
G &= \begin{bmatrix} 0 \\ -g_2 S_2 \\ -g_3 C_3 \end{bmatrix}
\end{aligned} \tag{3.23}$$

$S_i = sin(q_i)$, $S_{ij} = sin(q_i + q_j)$ and so on. The Coulomb friction is omitted here. Typical parameters are given in Table. 3.1. Note the gear ratio is 1 : 8 so that the dynamics of actuators does not dominate the robot dynamics. Here we set m_1, m_2, m_3, b_1, b_2, and b_3 to be zero to show the effect of the dynamic forces more clearly.

Now the desired trajectory was assumed to be a straight line segment in Cartesian space connecting $(10cm, 10cm, 0cm)$ at $t_0 = 0.0sec$ and $(10cm, -10cm, 10cm)$ at $t_f = 1.0sec$. The velocity and acceleration are determined using 5th order polynomials. We used $\hat{t} = 0.5[sec]$ and $\hat{\Delta} = 10msec$, respectively, and

selected $\{0.7, 0.7, 0.7, 0.4, 0.4, 0.4\}$ and $100[I_3 : I_3]$ for eigenvalues $\{\lambda_i\}$ and $[\xi_i]$, respectively. Then matrices K_p and K_v were calculated as follows. For the PD controller

$$[K_p : K_v] = \begin{bmatrix} 966.47 & -40.81 & 11.54 & 46.53 & 3.02 & -0.85 \\ 33.96 & 2300 & 310.6 & -3.04 & 112.3 & 16.72 \\ -16.84 & 305.6 & 780.9 & 0.83 & 16.70 & 38.41 \end{bmatrix} \times 6.272 \times 10^5 \tag{3.24}$$

for the computed torque controller

$$[K_p : K_v] = \begin{bmatrix} 1520 & -64.18 & 18.15 & 75.34 & -3.01 & 0.91 \\ 32.91 & 1491 & -9.73 & 1.50 & 73.92 & -0.48 \\ -36.18 & -28.32 & 1492 & -1.78 & -1.47 & 73.88 \end{bmatrix} \times 6.272 \times 10^5 \tag{3.25}$$

for the simple computed torque controller

$$[K_p : K_v] = \begin{bmatrix} 1520 & -64.18 & -18.15 & 73.18 & 4.76 & -1.34 \\ 28.65 & 1517 & -11.77 & -2.37 & 73.54 & 0.43 \\ -44.31 & -49.84 & 1503 & 2.58 & 1.19 & 73.50 \end{bmatrix} \times 6.272 \times 10^5. \tag{3.26}$$

Figs. 3.2, 3.3 and 3.4 show

$$g(t, \Delta) = \int_0^{\Delta} \gamma(t, \tau) d\tau$$

at $t = 0.05, 0.25, 0.5, 0.75, 0.95[sec]$ for the PD controller, the computed torque controller and the simple computed torque controller, respectively.

The following simulations were performed. At the initial time $t_0 = 0$, the robot hand was located at $(11cm, 11cm, 1cm)$. Fifteen streams of random variables were used to generate sampling interval Δ's for each distributions and the worst examples are shown in the figures.

- There is no α and β such that $g(t, \beta) < g(t, \alpha)$ except at $t = 0.5sec$ in Fig. 3.2. This means that the characteristics of the system varies so widely that the PD controller may not be able to stabilize the system except in the vicinity of $t = 0.5sec$ for any $[\alpha, \beta]$. In fact, the system is not stable for $\Delta \in U[5msec, 30msec]$ although the system is stable for a constant sampling rate $\Delta = 10msec$.

- Fig. 3.3 of $g(t, \Delta)$ is almost same for all t for the computed torque controller. This means that the characteristics are almost same for all t and the system is asymptotically stable along the trajectory if $\Delta \in \mathcal{U}[5msec, 30msec]$, for example. The simulation is shown in Fig. 3.5, where Ex, Ey and Ez are the errors from the desired trajectory in Cartesian space for the random sampling rate $\Delta \in \mathcal{U}[5msec, 30msec]$. Fig. 3.6 shows the simulation for $\Delta \in \mathcal{U}[5msec, 40msec]$. The system still remains stable, but there exist large vibrations around $t = 0.4sec$. The system is unstable for $\Delta \in \mathcal{U}[5msec, 45msec]$ for almost all random streams. From Fig. 3.3, we have: (i) if $\beta < 33msec$, there is an α such that the robot system is asymptotically stable, (ii) if $\alpha < 21msec$, there is a β for which the system is asymptotically stable.

- Fig. 3.4 for the simple computed torque shows the intermediate properties of the graphics for above two controllers. The characteristics change but not so widely that we can find stable distributions for all t. Figs. 3.7 and 3.8 show the error trajectories for the random rate $\Delta \in \mathcal{U}[5msec, 25msec]$ and $\Delta \in \mathcal{U}[5msec, 40msec]$, respectively. There are large vibrations around $t = 0.4sec$ in the latter figure which is almost same as one for the computed torque controller. The system is also unstable for $\Delta \in \mathcal{U}[5msec, 45msec]$ for almost all random streams. From Fig. 3.4, we have: (i) if $\beta < 28msec$, there is an α such that the robot system is asymptotically stable, (ii) if $\alpha < 19msec$, there is a β for which the system is asymptotically stable.

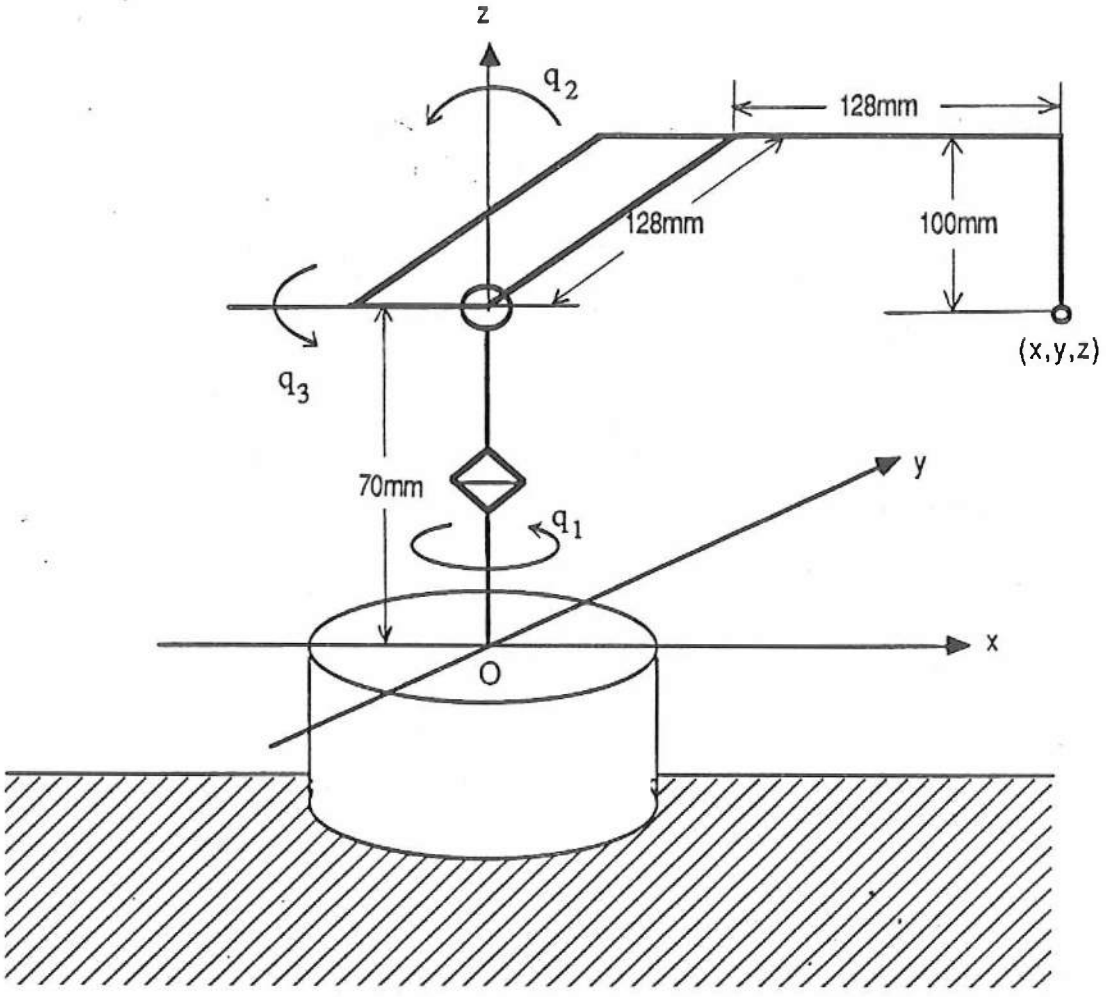


Figure 3.1: ROBOTEC

Table 3.2 shows expectations and standard variations of correlation coefficients btween $\gamma(t_k, \Delta_k)$ and $\gamma(t_{k+l}, \Delta_{k+l})$ for $k = 30$ and $l = 1, 5, 10, 20$, where $\Delta_k \in \mathcal{U}[5msec, 30msec]$. Five sets of 200 samples were calculated for each case. The correlation coefficients are all small even for a pair of $\gamma(t_{30}, \Delta_{30})$ and $\gamma(t_{31}, \Delta_{31})$. Therefore we can assume that these γ's are independent random variables one another.

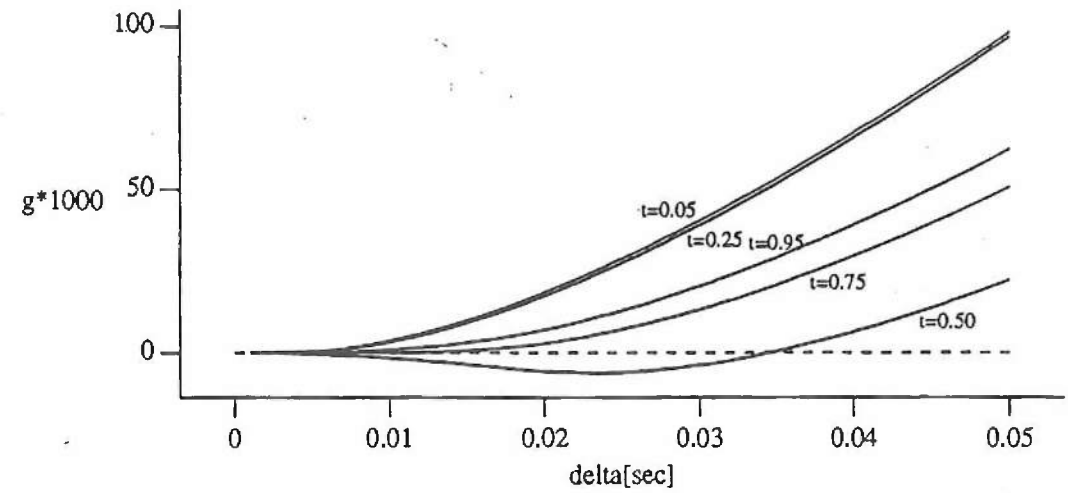


Figure 3.2: $g \times 1000$ of PD Controller

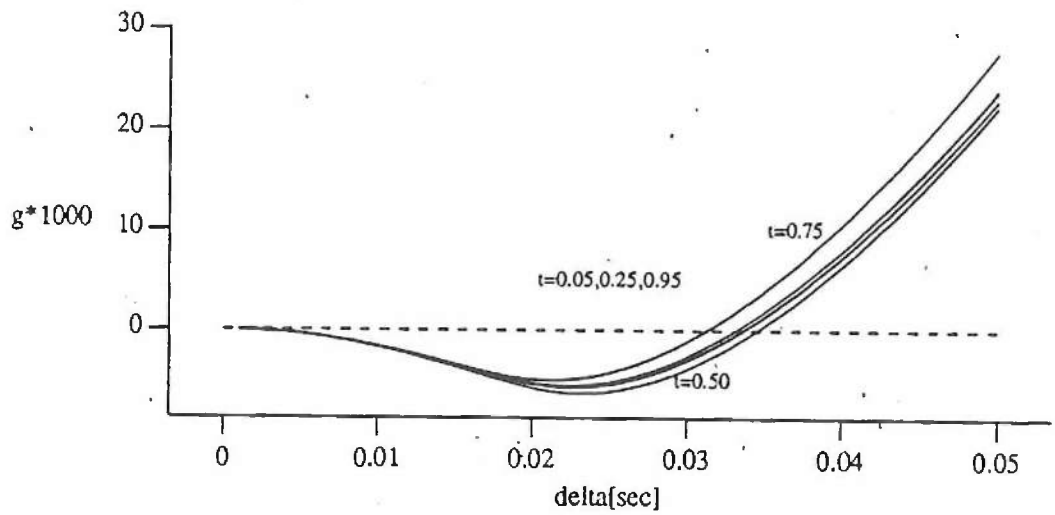


Table 3.1: Parameters of ROBOTEC

$\alpha[gcm^2]$	1.493×10^4		
$\beta[gcm^2]$	-4.421×10^3	$g1[gcm^2/sec^2]$	1.345×10^5
$\gamma[gcm^2]$	5.110×10^3	$g2[gcm^2/sec^2]$	8.389×10^3

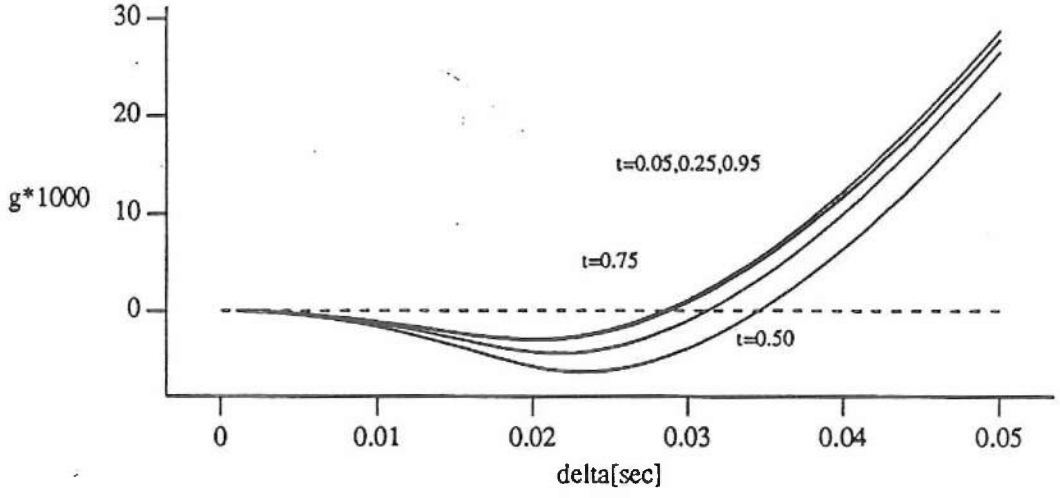


Figure 3.4: $g \times 1000$ of Simple Computed Torque Controller

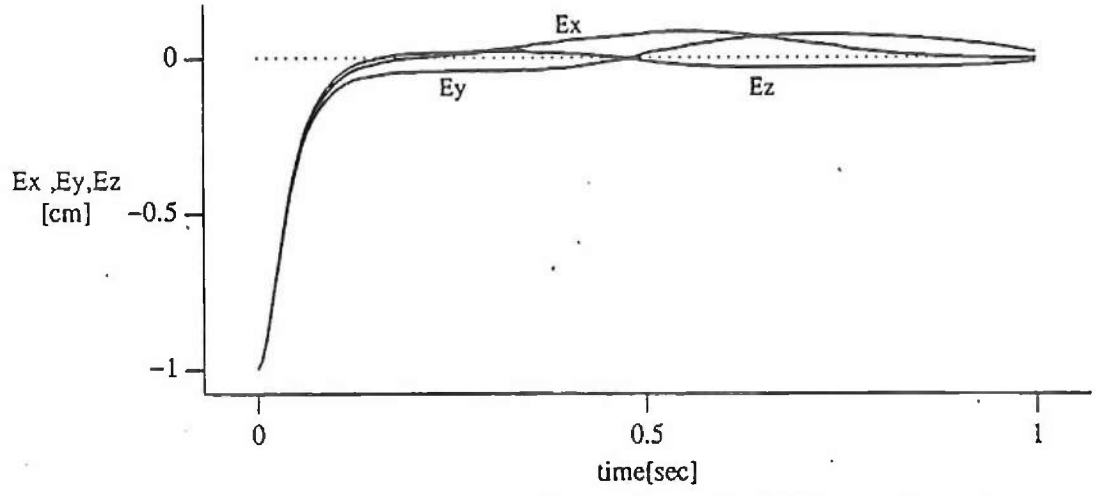


Figure 3.5: Computed Torque Controller - $\mathcal{U}[5msec, 30msec]$

Table 3.2: Expectations and Standard Deviations of Correlation Coefficients

$E[t_k] = 0.525[sec]$		PD Controller		CT Controller		SCT Controller	
l	$E[t_{k+l}][sec]$	MEAN	STD	MEAN	STD	MEAN	STD
1	0.543	0.0271	0.0829	-0.0110	0.0646	-0.0213	0.0676
5	0.613	0.0519	0.0755	0.0340	0.0780	0.0295	0.0772
10	0.700	0.0645	0.0432	0.0288	0.0517	0.0411	0.0527
20	0.875	-0.0484	0.0729	-0.0727	0.0650	-0.0632	0.0650

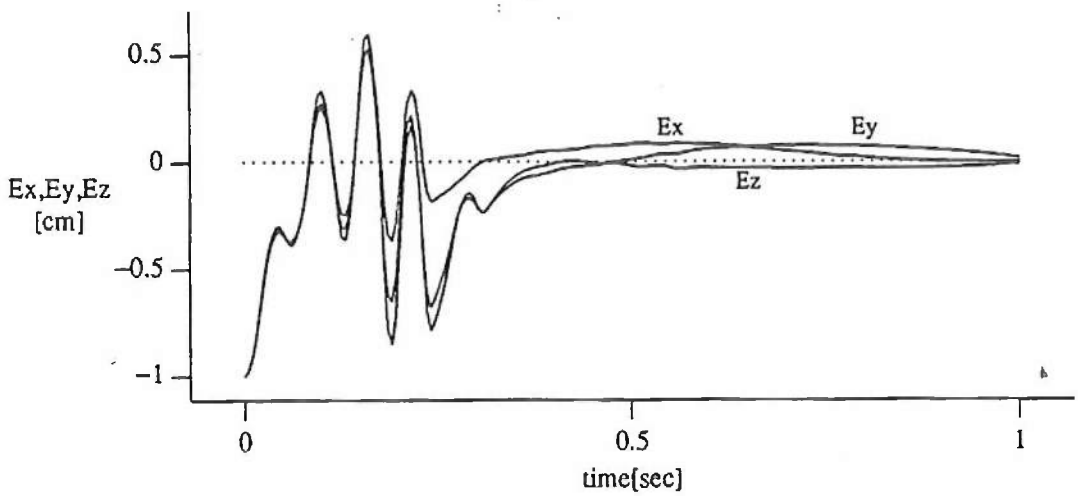


Figure 3.6: Computed Torque Controller - $\mathcal{U}[5msec, 40msec]$

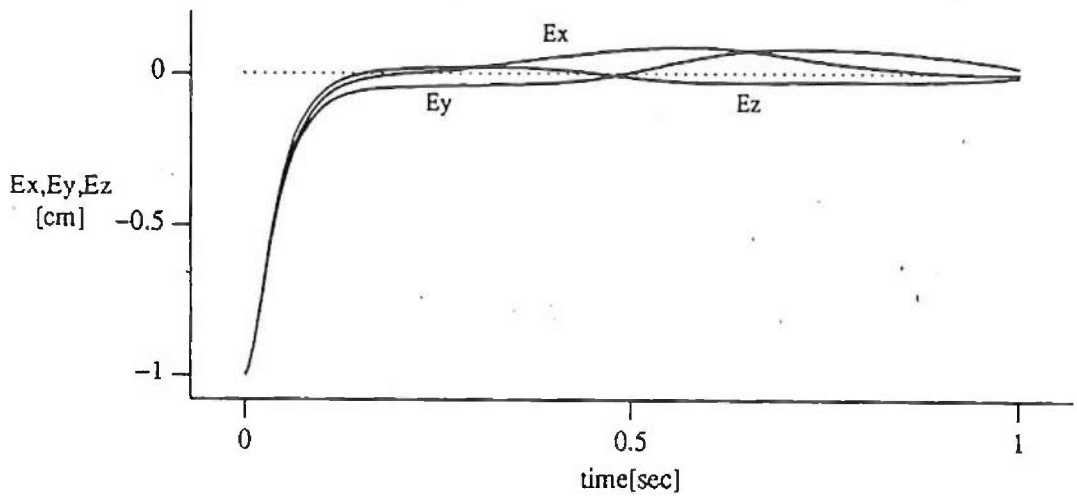


Figure 3.7: Simple Computed Torque Controller - $\mathcal{U}[5msec, 25msec]$

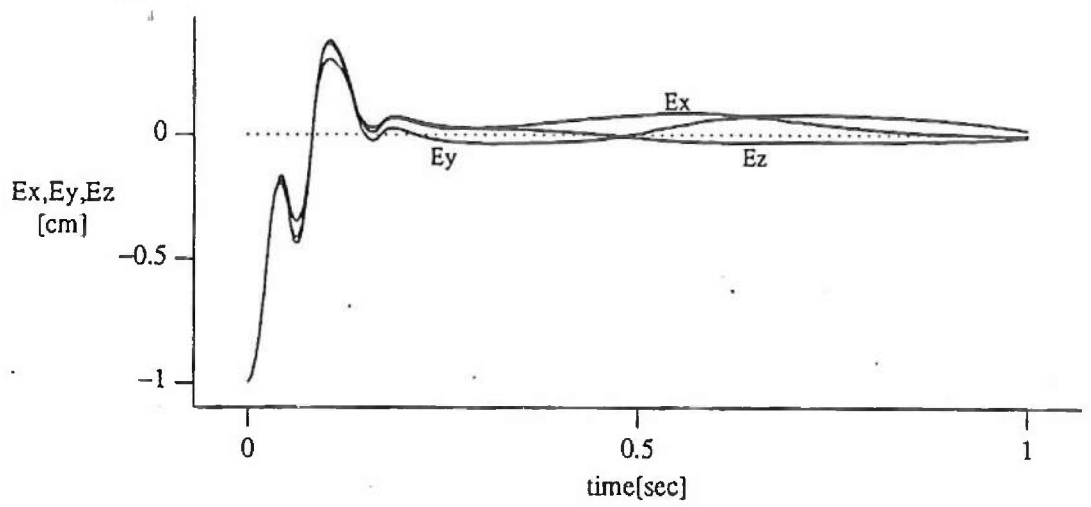


Figure 3.8: Simple Computed Torque Controller - $\mathcal{U}[5msec, 40msec]$

Chapter 4

Control of PUMA 260 under Random Sampling Intervals

In this chapter, we apply the results given so far to the control of PUMA 260 Robot Manipulator.

Corke [4] developed a new control system MMCS (Modular Motor Control System) which contains 2-axis control boards inserted into IBM PC bus and an adapter to connect to a high performance workstation computer(SUN 3). The controller supports up to 16 axes and can be used to control various motor units including robot manipulators, hand systems, camera mounts and tables. Control boards have a common clock and interrupt the host computer to calculate their input signals. This facility makes it possible to keep the sampling interval constant. The servo software in the kernel of the host computer supports position feedback and velocity feedback controls of each joint with a programmable digital filter and a Coulomb friction compensator. However if we wish to control the torques of motors directly through MMCS, then the control algorithm must be run in a user process and its scheduling cannot be guaranteed, with possibly consequences for unstable and non-smooth control, at the present stage of MMCS. This is unavoidable under a Unix machine. Therefore, in this chapter, we apply the results given above for random sampling intervals in order to control PUMA 260 with MMCS.

When we discussed the control problems of a two-dimensional linear control system in Section 2.5, we expected that we would be able to apply the results to robot manipulators which were controlled with a randomly sampled controller. In the following , first of all, the distribution of sampling intervals are measured and are approximated by a mixed uniform distribution. Next, feedback gains and a transformation matrix are calculated for the two-dimensional system using the algorithm given in Section 2.5. Finally, the stability is checked for the case where we use the same feedback gains to control PUMA 260 by SUN 3. To do so, linearized dynamic equation of PUMA 260 is generated automatically by a Lisp program. Some simulation results are also shown.

4.1 Control Scheme and Distribution of Sampling Intervals

If we use a simple computed torque controller, then inputs u_k are calculated by

$$u_k = M(\bar{q}(t_k))\ddot{q}^*(t_k) + C(\bar{q}(t_k), \dot{\bar{q}}(t_k)) + B(\dot{\bar{q}}(t_k)) + G(\bar{q}(t_k)) + CL(\dot{q}(t_k)), \tag{4.1}$$

where $CL(\dot{q})$ is a Coulomb friction force term, and

$$\ddot{q}^*(t_k) = \ddot{\bar{q}}(t_k) + K_p(\bar{q}(t_k) - q(t_k)) + K_v(\dot{\bar{q}}(t_k) - \dot{q}(t_k)).$$

Here the trajectory is assumed such that the hand is directed to downward and moves along a circle with the radius of 10 cm at the center of (25 cm, 0 cm, 0 cm) in the horizontal plane keeping the constant pose with respect to the shoulder coordinate system of PUMA 260. It rounds the circle in tr $sec.$ namely

$$T_6(t) = \begin{bmatrix} 0 & 0 & 1 & 25 + 10\sin(2\pi t/tr) \\ 0 & 1 & 0 & 10\cos(2\pi t/tr) \\ -1 & 0 & 0 & 0 \end{bmatrix} \tag{4.2}$$

Due to the randomness of the sampling intervals, we cannot identify the current time by counting the numbers on iterations and we must rely on function "clock()" to calculate the desired trajectory. The resolution of the function is 16.666 $msec$. Therefore we prepared a table of desired joint angles, angular velocities, and angular accelerations for each 16.666 $msec$ from the above trajectory. This also means that we cannot measure the histogram of sampling intervals of Eq. (4.1) by "clock()" precisely.

For PUMA 260, we use following control strategy:

- MMCS interrupts SUN 3 every 1 $msec$ and sends out current commands to motor drivers. Viscous and Coulomb friction are compensated in this level. This also means that the states of joints (which contain current angles and angular velocities of them) are updated each 1 $msec$.
- The controller reads the current state information from MMCS, calculates Eq. (4.1), and writes the command values $u(t)$ on MMCS repeatedly.

Now we assume that the output of "clock()" changes at $tc_0, tc_1, tc_2, \cdots$, where $(tc_{k+1} - tc_k = 16.666 msec)$ and that the i-th sampling interval begins at $t_i \in (tc_k, tc_{k+1}]$, and ends at $t_{i+1} \in [tc_{k+n}, tc_{k+n+1})$ (see Fig. 4.1). Then the "clock()" changes n-times in the sampling interval. We define this event as E_n, namely.

E_n : Event where "clock()" changes n-times in a sampling interval.

Now we assume that

- The starting times of sampling intervals after the latest clock change ($y_i = t_i - tc_k$) are subject to uniform distribution $\mathcal{U}[0, 16.666\ msec]$ and statistically independent not only of each other but also of sampling intervals. [1]

Then we have following proposition:

Proposition 8 (Probability of Event E_n)

$$P_0 = \int_0^h \frac{h - \Delta}{h} f(\Delta) d\Delta,$$

and for $n > 0$

$$P_n = \int_0^h \frac{\Delta}{h} f(\Delta + (n-1)h) d\Delta + \int_0^h \frac{h - \Delta}{h} f(\Delta + nh) d\Delta.$$

where $P_n = Prob[E_n], h = 16.666\ msec$, *and* $f(\Delta)$ *is the probability density function of sampling intervals. Furthermore we have*

$$E[\Delta] = \lim_{n=\infty} \sum_{i=1}^{n} ihP_i.$$

[1] Generally, this assumption is not true for control systems. Usually, y_i and y_{i+1} have a positive correlation if the deviation of sampling intervals are not so large. Nevertheless we assume this because of simplicity.

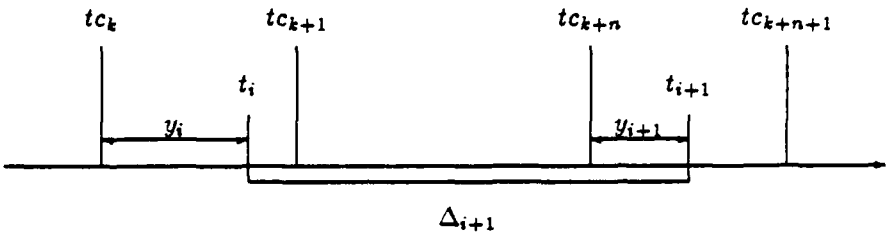


Figure 4.1: Event E_n

< proof >

Here we consider for the case of $n > 0$ only because the case of $n = 0$ will be trivial from it.

We assume that the i-the sampling interval starts y after the latest tc_k and continues Δ seconds. Then it is clear that if $\Delta < (n-1)h$ or $\Delta \leq (n+1)h$ then we have $Prob[E_n] = 0$. Next, if $(n-1)h \leq \Delta < nh$ then $y \geq nh - \Delta$ must be satisfied. Therefore, by the assumption on random variable y, the probability for the case where sampling intervals are contained in $(\Delta, \Delta + d\Delta)$ is

$$\frac{\Delta - (n-1)h}{h} f(\Delta) d\Delta.$$

On the other hand, if $nh \leq \Delta < (n+1)$ then y must be less than $(n+1)h - \Delta$. Therefore we have

$$\frac{(n+1)h - \Delta}{h} f(\Delta) d\Delta,$$

for the same small segment. Since these events are exclusive each other, we have

$$P_n = \int_{(n-1)h}^{nh} \frac{\Delta - (n-1)h}{h} f(\Delta) d\Delta + \int_{nh}^{(n+1)h} \frac{(n+1)h - \Delta}{h} f(\Delta) d\Delta.$$

Substructing the integral variables appropriately, we have first half part of the proposition. This equation can also be verified easily using characteristic functions.

Now we have

$$hP_1 = \int_0^h \Delta f(\Delta) d\Delta + \int_h^{2h} (2h - \Delta) f(\Delta) d\Delta.$$

Here we assume next equation:

$$\sum_{i=1}^{n} ihP_i = \int_0^{nh} \Delta f(\Delta) d\Delta + \int_{nh}^{(n+1)h} n\{(n+1)h - \Delta\} f(\Delta) d\Delta,$$

then we can easily show that

$$\sum_{i=1}^{n+1} ihP_i = \int_0^{(n+1)h} \Delta f(\Delta)d\Delta + \int_{(n+1)h}^{(n+2)h} (n+1)\{(n+2)h - \Delta\}f(\Delta)d\Delta.$$

Therefore for any $n > 0$,

$$\int_0^{nh} \Delta f(\Delta)d\Delta \leq \sum_{i=1}^{n} ihP_i \leq \int_0^{(n+1)h} \Delta f(\Delta)d\Delta.$$

This inequality means the proposition is true. □

The histogram of event E_n were measured five times for one minute using SUN 3 and the results are shown in Table 4.1.[2] We have five independent conditions ($P_n(n = 0, 1, 2, 3, 4)$). Therefore we can use five parameters to specify the distribution of sampling intervals. Note that the expectation of sampling times is determined by P_i only and does not depend on $f(\Delta)$ at all. This means that we cannot use the expectation to identify it.

Since we do not any information about the distribution of sampling intervals, it is reasonable to use a distribution as simple as possible. Hence we assume it as follows:

(1) $\Delta = a$ with probability p_1.

(2) Δ is subject to

$$\begin{cases} \mathcal{U}[h, 2h] & \text{with probability } p_2, \\ \mathcal{U}[2h, 3h] & \text{with probability } p_3, \\ \mathcal{U}[3h, 4h] & \text{with probability } p_4. \end{cases}$$

(1) corresponds to the normal situation and (2) to the case where some delay occurs due to, for example, interrupt latencies. By applying Prop. 8 to this distribution, we have

$$P_0 = \frac{a}{h}p_1 \tag{4.3}$$

$$P_1 = \frac{h-a}{h}p_1 + \frac{p_2}{2} \tag{4.4}$$

$$P_2 = \frac{p_2 + p_3}{2} \tag{4.5}$$

$$P_3 = \frac{p_3 + p_4}{2} \tag{4.6}$$

$$P_4 = \frac{p_4}{2} \tag{4.7}$$

These simultaneous equations can be solved easily and we have

$$\begin{cases} p_1 = 0.918845 \\ p_2 = 0.080298 \\ p_3 = 0.000784 \\ p_4 = 0.000070 \\ a = 2.147\ msec \end{cases} \tag{4.8}$$

[2] Of course, the distribution of sampling intervals varies depending on the number of login users and the quality of jobs. Therefore several data were gathered at different chances. Data were very similar to each other for this system since Sun 3 system in GRASP LAB. of Univ. of Penn. is devoted to a relatively restrictive purpose (to develop a control system of PUMA 260) and only a few people usually login it .

Trial	Time[sec]	No. of Iterations	E_0	E_1	E_2	E_3	E_4
1	60.014	11376	9120	1802	451	3	0
2	60.014	11190	8985	1736	458	11	0
3	60.014	11214	8974	1170	460	8	2
4	60.014	11285	8992	1826	467	0	0
5	60.014	11124	8907	1773	442	2	0
Means	60.014	11237.8	8995.6	1781.4	455.6	4.8	0.4
Rate	0.00534	1.0	0.8005	0.1585	0.0405	0.43×10^{-3}	0.35×10^{-4}

Table 4.1: Histogram of E_n

Stream	Time[sec]	No. of Iterations	E_0	E_1	E_2	E_3	E_4
1	60.016	10961	7846	2631	482	2	0
2	60.008	11390	8227	2727	435	1	0
3	60.016	11209	8081	2661	462	4	1
4	60.008	11271	8118	2710	439	4	0
5	60.029	11207	8081	2658	462	6	0
Means	60.016	11195.6	8070.2	2677.4	456.0	3.4	0.2
Rate	0.00536	1.0	0.7209	0.2391	0.0407	0.30×10^{-3}	0.18×10^{-4}

Table 4.2: Simulation of E_n

However there is a problem. If we use values given above, the expectation of sampling intervals becomes $4.017 msec$ and it is less than the measured value $5.34 msec$. To adjust this difference, we used $a = 3.587 msec$ instead of $2.147\ msec$.

Simulation were done using this distribution. The results are given in Table 4.2. The mean and probabilities of event E_2, E_3, and E_4 are very near the original data, while probabilities of E_0 and E_1 are different. This is because of the periodicity of the sampling times. The sampling intervals have same value ($3.587\ msec$) with probability 0.9 and y_i has a positive correlation each other. This distribution is still useful since it gives worse situation than the actual. Therefore we use this distribution to determine the feedback gains of Eq. (4.1) in the following section.

4.2 Feedback Gains and Stability

In order to apply the results given so far, we must derive a linearized equation of PUMA 260 along a reference trajectory, which requires a lot of times and the task is so cumbersome and complicated that we would not be able to do without lots of mistakes. To avoid this formidable business, a Lisp program is developed which automatically generates a linearized robot dynamic equation in a C program. The detail is given in Appendix A.

4.2.1 PD Controller

First of all, the expectations of function γ were calculated under distribution (4.8) for the two-dimensional system of Section 2.5 with a PD controller. The results are shown in Table 4.3. From this table, we

$\hat{\Delta}[msec]$	expectation
15.0	-0.0749
16.0	-0.0771
17.0	-0.0782
17.5	-0.0783
18.0	-0.0782
19.0	-0.0775
20.0	-0.0763

Table 4.3: $\hat{\Delta}$ and Expectation of γ function

use $\hat{\Delta} = 17.5\ msec$ because it gives the minimum expectation and we can expect the fastest convergence. Feedback gain matrices Kp and Kv of controllers and transformation matrix T to check the stability were given by Prop. 3 and Eq. (2.39) as follows:

$$Kp = \text{Diag}\{587.76\} \tag{4.9}$$

$$Kv = \text{Diag}\{46.29\} \tag{4.10}$$

$$T = \begin{bmatrix} \text{Diag}\{-0.0133\} & \text{Diag}\{-0.0165\} \\ \text{Diag}\{0.651\} & \text{Diag}\{0.333\} \end{bmatrix}, \tag{4.11}$$

where $\text{Diag}\{x\}$ is a six-dimensional diagonal matrix with the same diagonal element x. Figs. 4.2, 4.3, and 4.4 show γ functions of PUMA 260 calculated at the positions of $0, 72, 144, 216$, and $288\ deg$ for 0.2, 0.5 and $1\ rps$. The dot line shows γ function for the two-dimensional system. γ functions of PUMA 260 are very similar to each other in spite of the difference of the position and velocity. It is also shown that the expectation of γ functions for PUMA 260 will be larger a little than that of the two-dimensional system, but it may be still negative. In fact, Table 4.4 shows that the expectations of γ function for PUMA 260 are negative for all positions and velocities. Therefore PUMA 260 is asymptotically stable with probability one.

Figs. 4.5 $\sim$ 4.8 show simulation results for Stream 3 in Table 4.2. Fig. 4.5 shows the desired trajectory for the case of $0.2\ rps$. Figs. 4.6, 4.7, and 4.8 show the error angles in degree for each joint for the cases of 0.2, 0.5, and $1\ rps$, respectively. The mean of the absolute values of errors and the maximum for one minute are listed in Table 4.5. The errors become larger according to the increase of velocity. Figs. 4.9 and 4.10 show the trajectories in x-y plane and x-z plane for the case of $1\ rps$, respectively. The maximum errors in the radius and in the z-coordinates are listed in Table 4.6. This implies that the errors in joint angles generate not the deviation across the desired trajectory but the delay along it. The main cause of this is, of course, large delays in sampling times and another is the rough resolution of "clock()" function. If PUMA 260 travels along the circle at $1 rps$, then it moves about six degrees during $16.666\ msec$, which corresponds to about four degrees of Joint 1 at the largest value.

4.2.2 PID Controller

PID controllers make robot manipulator robust for slow changes of system parameters. Therefore we apply the results given in Subsection 2.6.1 to design a PID controller for PUMA 260. Namely, we use following controller:

$$u_k = M(\bar{q}(t_k))\ddot{q}^*(t_k) + C(\bar{q}(t_k), \dot{\bar{q}}(t_k)) + B(\dot{\bar{q}}(t_k)) + G(\bar{q}(t_k)) + CL(\dot{q}(t_k)), \tag{4.12}$$

angle[deg]	0.2[rps]	0.5[rps]	1.0[rps]
0	-0.07077	-0.07061	-0.06904
72	-0.06910	-0.06895	-0.06744
144	-0.06968	-0.06973	-0.06943
216	-0.07155	-0.07100	-0.06873
288	-0.07212	-0.07203	-0.07089

Table 4.4: Expectation of γ function for PUMA 260

	0.5[rps]		0.2[rps]		1.0[rps]	
Joint	Mean[deg/s]	Max[deg]	Mean[deg/s]	Max[deg]	Mean[deg/s]	Max[deg]
1	0.1340	0.7373	0.3345	1.7246	0.6408	3.8700
2	0.0919	0.3003	0.2219	0.6699	0.3834	1.3781
3	0.1929	0.6680	0.4680	1.4662	0.8256	3.5637
4	0.0019	0.0114	0.0074	0.0453	0.0469	0.3526
5	0.0963	0.3004	0.2277	0.6482	0.3755	1.1918
6	0.1321	0.6699	0.3127	1.5257	0.5453	3.3778

Table 4.5: Mean and Maximum of Joint Angle Errors

	Maximum Error	
RPS	Radius[cm]	Z-Coordinate[cm]
0.2	0.0295	0.0083
0.5	0.1490	0.0204
1.0	0.5302	0.0536

Table 4.6: Maximum Error in Cartesian Space for PD Controller

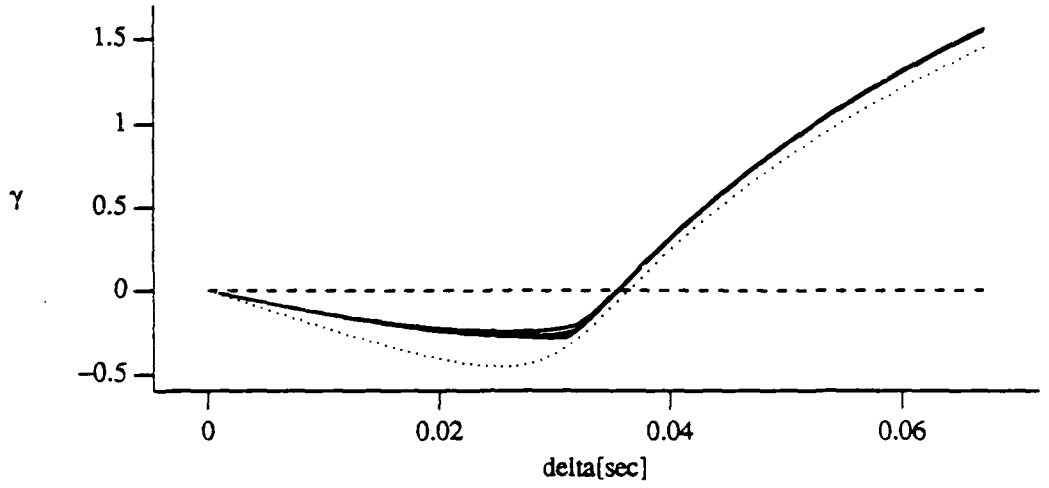


Figure 4.2: Simple Computed Torque Controller: 0.2 rps

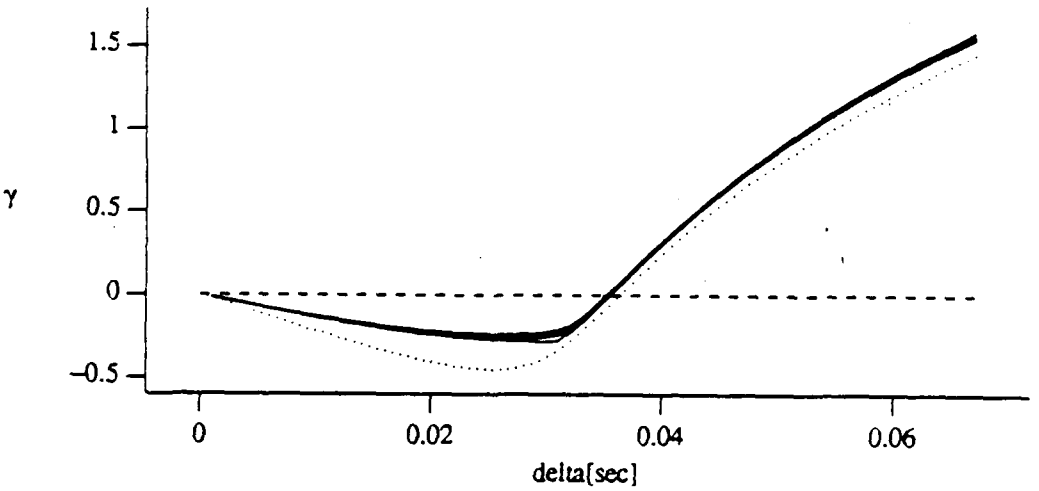


Figure 4.3: Simple Computed Torque Controller: 0.5 rps

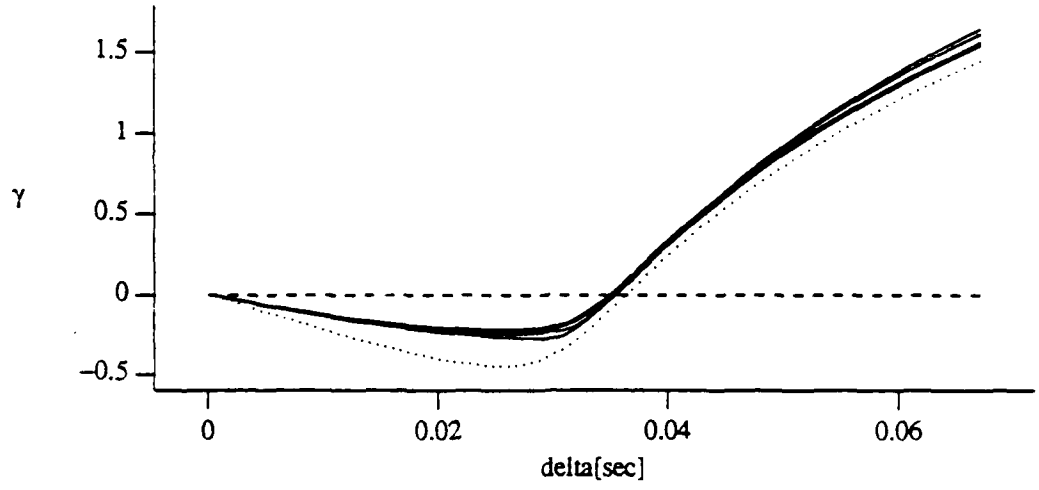


Figure 4.4: Simple Computed Torque Controller: 1.0 rps

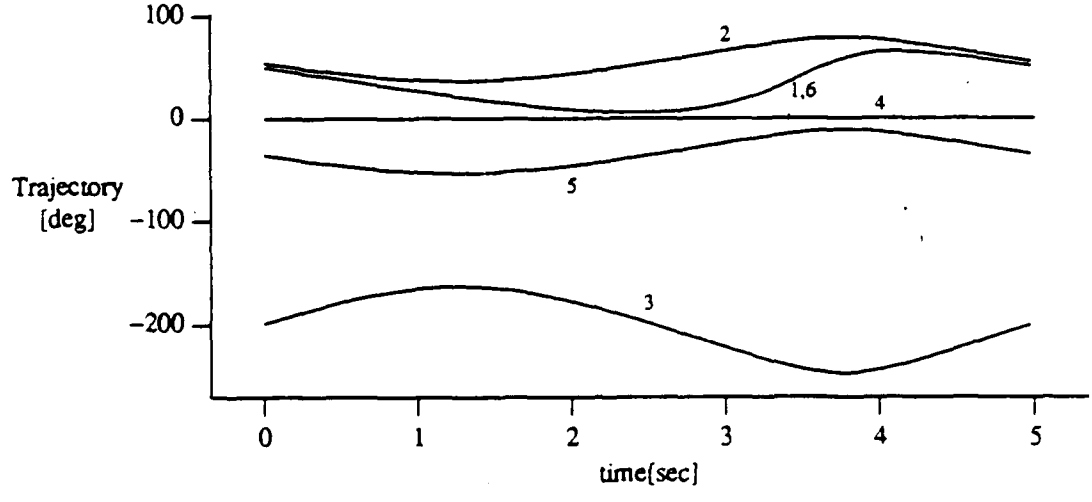


Figure 4.5: Trajectory for 0.2 rps

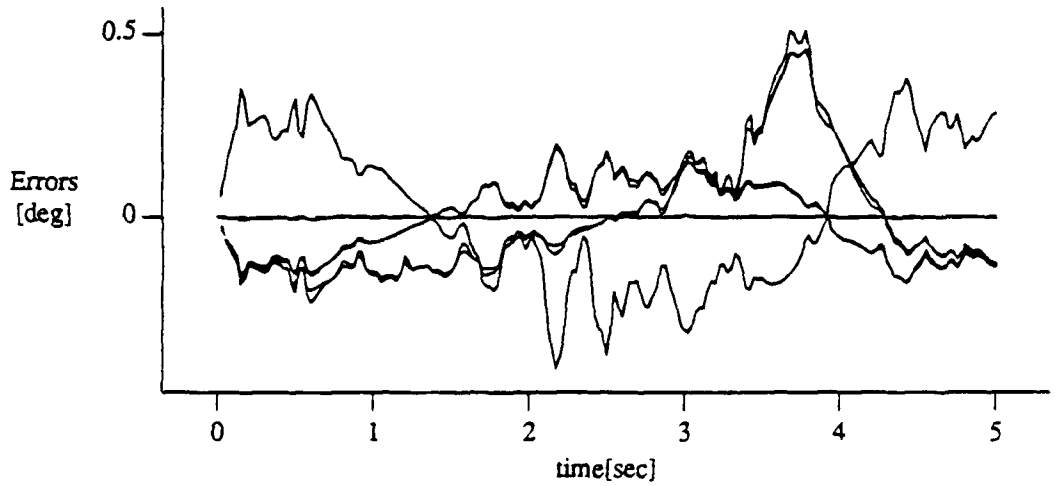


Figure 4.6: Joint Angle Errors for 0.2 rps

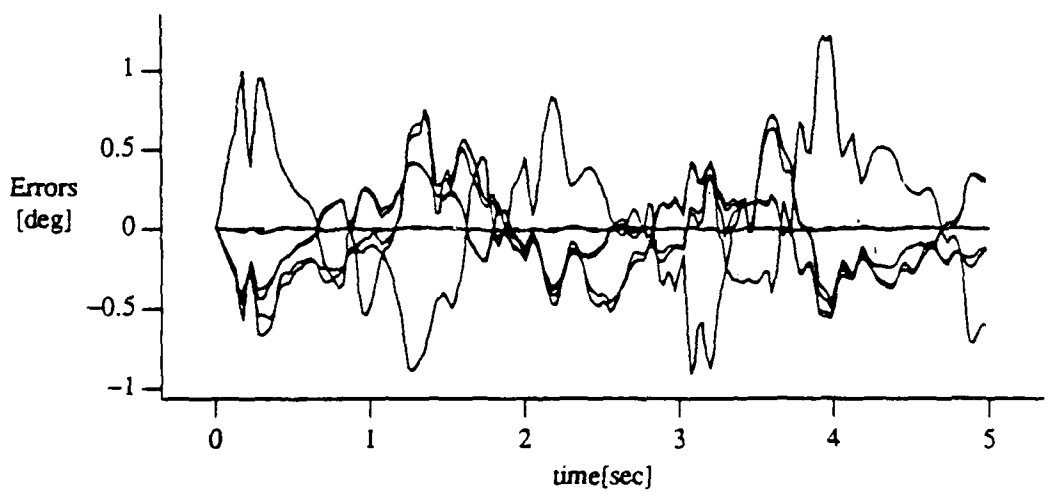


Figure 4.7: Joint Angle Errors for 0.5 rps

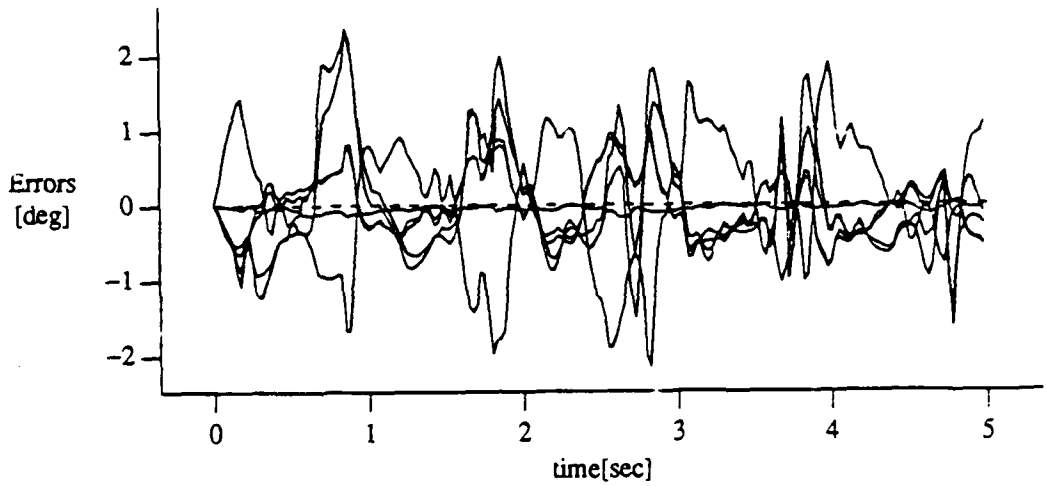


Figure 4.8: Joint Angle Errors for 1.0 rps

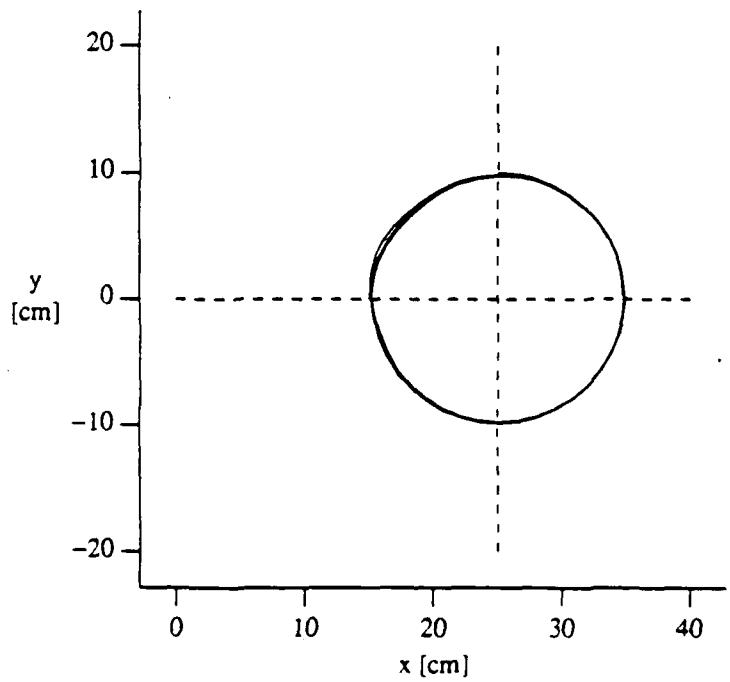


Figure 4.9: Trajectory in x-y Plane for 1.0 rps

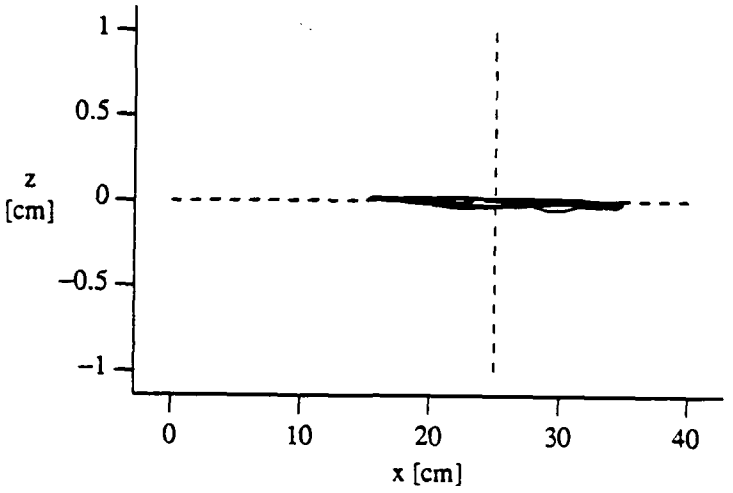


Figure 4.10: Trajectory in x-z Plane for 1.0 rps

	Maximum Error	
RPS	Radius[cm]	Z-Coordinate[cm]
0.2	0.0281	0.0249
0.5	0.1558	0.0653
1.0	0.6677	0.1851

Table 4.7: Maximum Error in Cartesian Space for PID Controller

where

$$\begin{aligned} z_{k+1} &= z_k + (q(t_k) - q(t_k)), \\ \ddot{q}^*(t_k) &= \ddot{\bar{q}}(t_k) + K_p(\bar{q}(t_k) - q(t_k)) + K_v(\dot{\bar{q}}(t_k) - \dot{q}(t_k)) + K_i z_k. \end{aligned} \tag{4.13}$$

Using the same procedure as we used for PD controllers, we have $\Delta = 13\,msec$, $K_p = Diag\{1307.69\}$, $K_v = Diag\{64.62\}$, and $K_i = Diag\{53.25\}$. Figs. 4.11 $\sim$ 4.14 show simulation results for five seconds, where the gravity compensation term $G(\bar{q}(t_k))$ was removed from Eq. 4.12. Figs. 4.11 and 4.12 are for the PD controller given above and the others for the PID controller designed here. The velocity is 1 *rps* for both controllers. For the PD controller, PUMA 260 falls about 0.7 *cm* at the most stretched position in Fig. 4.11, while the maximum error in the radius is about 0.6 *cm* (Fig. 4.12). If we use the PID controller, the vertical deviation becomes less than 0.2 *cm* at the worst (Fig. 4.13) and the maximum radius error becomes a little larger (Fig. 4.14). Table 4.7 shows the errors in the radius and the z-coordinate for 0.2, 0.5, and 1.0 *rps* from $t = 0.5$ *sec* to 5.0 *sec*.

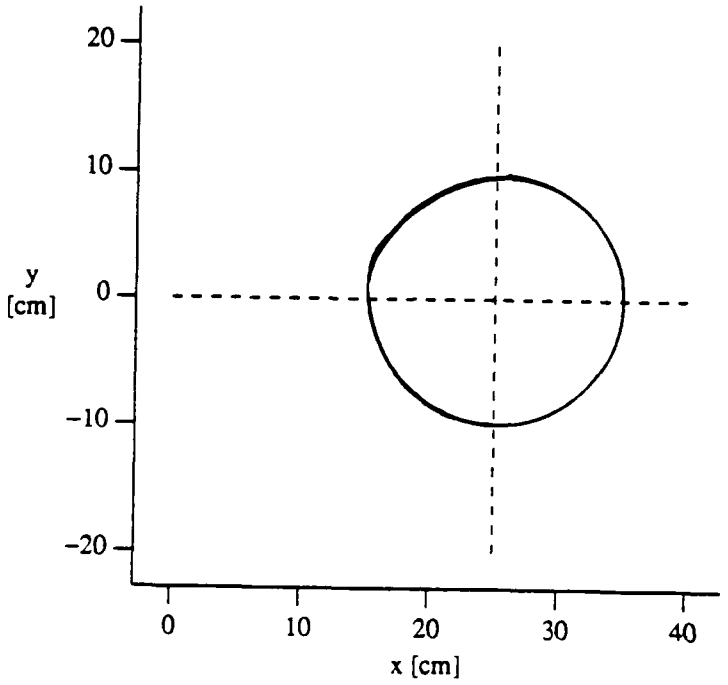


Figure 4.11: Trajectory in x-y Plane for PD Controller

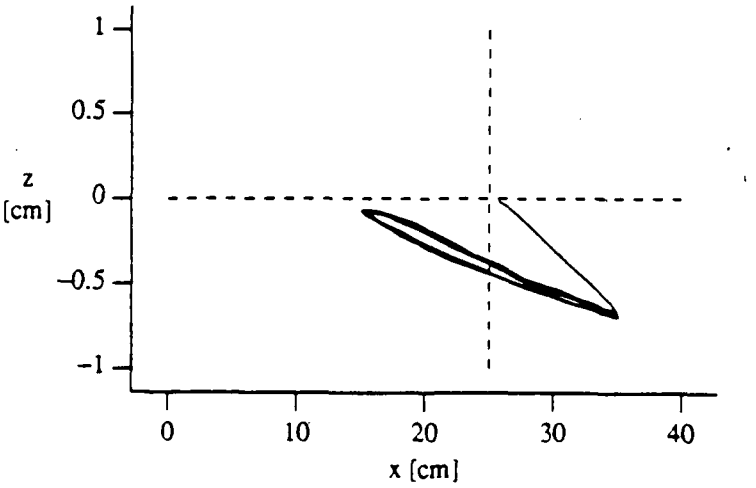


Figure 4.12: Trajectory in x-z Plane for PD Controller

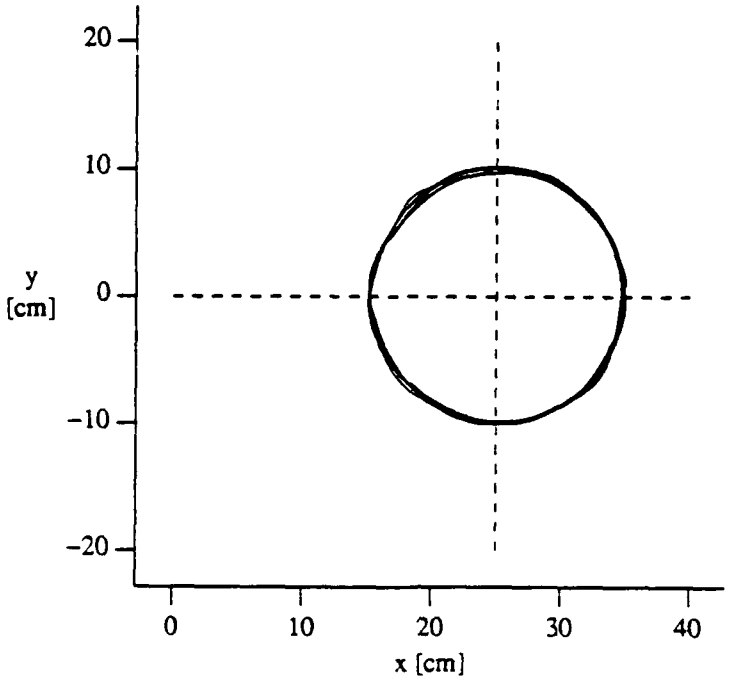


Figure 4.13: Trajectory in x-y Plane for PID Controller

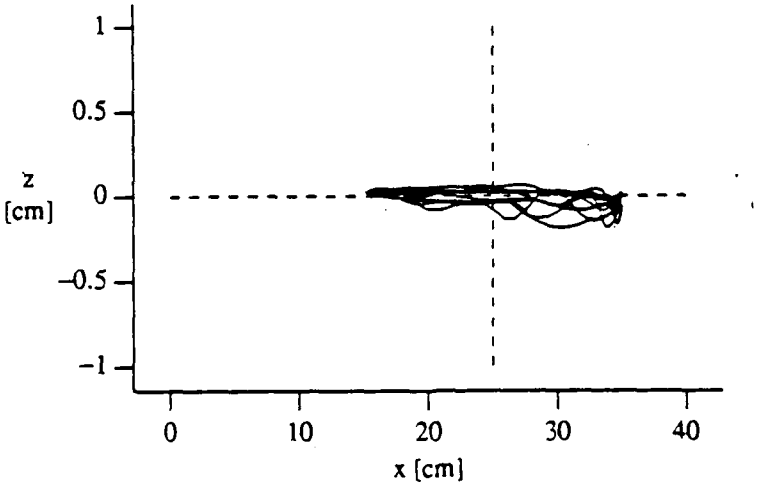


Figure 4.14: Trajectory in x-z Plane for PID Controller

Chapter 5

Conclusions

In this report, the stability of randomly sampled linear control systems was discussed and the following results were obtained.

- A necessary and sufficient condition for the asymptotical stability with probability one was obtained for one-dimensional systems.
- Applying the results to the systems whose sampling intervals are subject to Bernoulli distribution, uniform distribution and coupled uniform distribution, the stability conditions were derived explicitly.
- A sufficient condition for the asymptotical stability with probability one was obtained for multi-dimensional systems.
- For a typical two-dimensional system, a concrete design procedure shown for a PID controller and a PD controller with one step delay as well as a PD controller.
- The above results were applied to robot control systems and the effectiveness of the PD controller with a feedforward term, computed torque controller, and simple computed torque controller were compared with a random sampling rate. It was shown that PD controllers are very sensitive to the randomness of the sampling rate, while computed torque controllers and simple computed torque controllers are useful for random sampling rates which have fairly wide distributions.
- Finally, the stability problem of PUMA 260 controlled by work-station SUN 3 was discussed. First of all, the distribution of sampling intervals was measured and approximated by a mixed uniform distribution. Next, the feedback gains and the transformation matrix were obtained for the two dimensional system mentioned above. Then the stability of PUMA 260 with same gains were checked using same transformation matrix. To do so, a Lisp program which generates a linerized equations along a reference trajectory automatically was developed. Finally some simulation results were shown.

Appendix A

Linearized Dynamic Equation of Robot Manipulators

Here we develop a Lisp program to generate a C program which calculates linearized dynamic equations of manipulators along a given trajectory recursively.

In general, dynamic equations of robot manipulators are very complicated and to linearize the equations is a formidably burdensome work, while we often need linearized equations to assure the stability of the systems and so on. One way to deal with this work is to develop a lisp program to write the equations for us.

Since we have already had efficient recursive algorithms to calculate the actuator torques(forces) of robot manipulators for given configuration (for example [6]), we can easily deduce recursive equations to calculate small deviations of the torques(forces) given small deviations of joint variables. Since, of course, linearized equations are linear with respect to joint variables, we can calculate the coefficients from the i-th variable to the torques(forces) by setting the i-th variable to be equal to one and the others to be equal to zero. But if we use the general equations directly, the amount of the calculation becomes again formidable. Fortunately, the parameters of the equations contain a lot of zeros, ones, minus-ones, and common terms which appears many times during the calculation. It is also known that there is a linear dependence among parameters and the amount of the calculation can be reduced by using the dependence. We can use these facts if we develop a Lisp program to generate a C program for it.

Dynamic equations of manipulators are described as follows:

$$\tau = R(\mathrm{q}, \dot{\mathrm{q}}, \ddot{\mathrm{q}}), \tag{A.1}$$

$$= M(\mathrm{q})\ddot{\mathrm{q}} + C(\mathrm{q}, \dot{\mathrm{q}}) + G(\mathrm{q}), \tag{A.2}$$

where $\mathrm{q}, \dot{\mathrm{q}}, \ddot{\mathrm{q}}$ are angle(position), velocity, and acceleration vectors of joint variables, respectively. M is the inertia matrix, C the centrifugal and Coriolis force, and G the gravitational force. Here we neglect the Coulomb friction since it does not appear in linearized equations.

We assume that a desired trajectory $\{\bar{\mathrm{q}}, \dot{\bar{\mathrm{q}}}, \ddot{\bar{\mathrm{q}}}, \bar{\tau}\}$ is given where

$$\bar{\tau} = R(\bar{\mathrm{q}}, \dot{\bar{\mathrm{q}}}, \ddot{\bar{\mathrm{q}}}), \tag{A.3}$$

and define small deviation vectors $\delta\mathrm{q}, \delta\dot{\mathrm{q}}, \delta\ddot{\mathrm{q}}$, and $\delta\tau$ as follows:

$$\delta\mathrm{q} = \mathrm{q} - \bar{\mathrm{q}}, \tag{A.4}$$

$$\delta\dot{\mathbf{q}} = \dot{\mathbf{q}} - \dot{\bar{\mathbf{q}}}, \tag{A.5}$$
$$\delta\ddot{\mathbf{q}} = \ddot{\mathbf{q}} - \ddot{\bar{\mathbf{q}}}, \tag{A.6}$$
$$\delta\tau = \tau - \bar{\tau}. \tag{A.7}$$

Then we have

$$\delta\tau = P\delta\ddot{\mathbf{q}} + Q\delta\dot{\mathbf{q}} + R\delta\mathbf{q}, \tag{A.8}$$

where

$$P = M(\bar{\mathbf{q}}), \tag{A.9}$$
$$Q = [\partial R/\partial\dot{\mathbf{q}}](\bar{\mathbf{q}}, \dot{\bar{\mathbf{q}}}), \tag{A.10}$$
$$R = [\partial R/\partial\mathbf{q}](\bar{\mathbf{q}}, \dot{\bar{\mathbf{q}}}, \ddot{\bar{\mathbf{q}}}). \tag{A.11}$$

Linearized equation is given as follows:

$$\delta\dot{\mathbf{x}}(t) = A(t)\delta\mathbf{x}(t) + B(t)\delta\tau, \tag{A.12}$$

where

$$\delta x = \begin{bmatrix} q - \bar{q} \\ \dot{q} - \dot{\bar{q}} \end{bmatrix},$$
$$A(t) = \begin{bmatrix} 0 & I_r \\ -P^{-1}Q & -P^{-1}R \end{bmatrix},$$
$$B(t) = \begin{bmatrix} 0 \\ P^{-1} \end{bmatrix},$$

Recursive equations of dynamics of the manipulator are given as follows:
[Forward Equations]

$$\mathbf{w}_i = R_i^t(\mathbf{w}_{i-1} + \dot{\bar{q}}_i\mathbf{z}), \tag{A.13}$$
$$\dot{\mathbf{w}}_i = R_i^t(\dot{\mathbf{w}}_{i-1} + \ddot{\bar{q}}_i\mathbf{z} + \dot{\bar{q}}_i\mathbf{w}_{i-1} \times \mathbf{z}), \tag{A.14}$$
$$\dot{\mathbf{v}}_i = R_i^t[\dot{\mathbf{w}}_{i-1} \times \mathbf{p}_{i-1} + \mathbf{w}_{i-1} \times (\mathbf{w}_{i-1} \times \mathbf{p}_{i-1}) + \dot{\mathbf{v}}_{i-1}], \tag{A.15}$$

[Backward Equations]

$$F_i = m_i[\dot{\mathbf{v}}_i + \dot{\mathbf{w}}_i \times (\mathbf{p}_i + \mathbf{s}_i) + \mathbf{w}_i \times (\mathbf{w}_i \times (\mathbf{p}_i + \mathbf{s}_i))], \tag{A.16}$$
$$N_i = I_i\dot{\mathbf{w}}_i + \mathbf{w}_i \times (I_i\mathbf{w}_i), \tag{A.17}$$
$$\mathbf{f}_i = R_{i+1}\mathbf{f}_{i+1} + F_i, \tag{A.18}$$
$$\mathbf{n}_i = R_{i+1}[\mathbf{n}_{i+1} + (R_{i+1}^t\mathbf{p}_i) \times \mathbf{f}_{i+1}] + m_i(\mathbf{p}_i + \mathbf{s}_i) \times \dot{\mathbf{v}}_i + N_i, \tag{A.19}$$
$$\tau_i = \mathbf{n}_i^t(R_i^t\mathbf{z}) + IA_i\ddot{\bar{q}}_i + RD_i\dot{\bar{q}}_i \tag{A.20}$$

where $\dot{\bar{q}}_i$ and $\ddot{\bar{q}}_i$ are the velocity and the acceleration of the joint variable of link i. $\mathbf{w}_i$, $\dot{\mathbf{w}}_i$, and $\dot{\mathbf{v}}_i$ are the angular velocity, angular and linear acceleration of the frame i with respect to the frame i, respectively. F_i, N_i, $\mathbf{f}_i$, and $\mathbf{n}_i$ are the force due to the motion of link i, the torque due to the motion of link i, the total force in the link i, and the total torque in the link i expressed with respect to the frame i, respectively. $\mathbf{p}_i$ is the vector from the origin of frame $i-1$ to the origin of frame i and $\mathbf{s}_i$ the vector from the origin of frame i to the gravity center of link i with respect to frame i. m_i is the mass of link i and I_i is the inertia matrix of link i at the origin of frame i with respect to frame i. R_i is the rotation matrix from frame

$i-1$ to i. IA_i and RD_i are inertia and velocity friction parameters of the i-th actuator, respectively, and $z = (0,0,1)^t$.

Then the differential equations are given as follows:

[Forward Equations]

$$\delta w_i = R_i^t(\delta w_{i-1} + \delta\dot{q}_i z) + \delta q_i E_i w_i, \tag{A.21}$$

$$\delta\dot{w}_i = R_i^t(\delta\dot{w}_{i-1} + \delta\ddot{q}_i z + \delta\dot{q}_i w_{i-1} \times z + \dot{\bar{q}}_i \delta w_{i-1} \times z) + \delta q_i E_i \dot{w}_i, \tag{A.22}$$

$$\delta\dot{v}_i = R_i^t[\delta\dot{w}_{i-1} \times p_{i-1} + \delta w_{i-1} \times (w_{i-1} \times p_{i-1}) + w_{i-1} \times (\delta w_{i-1} \times p_{i-1}) + \delta\dot{v}_{i-1}] + \delta q_i E_i \dot{v}_i, \tag{A.23}$$

[Backward Equations]

$$\delta F_i = m_i[\delta\dot{v}_i + \delta\dot{w}_i \times (p_i + s_i) + \delta w_i \times (w_i \times (p_i + s_i)) + w_i \times (\delta w_i \times (p_i + s_i))], \tag{A.24}$$

$$\delta N_i = I_i \delta\dot{w}_i + \delta w_i \times (I_i w_i) + w_i \times (I_i \delta w_i), \tag{A.25}$$

$$\delta f_i = R_{i+1}\delta f_{i+1} + \delta F_i + \delta q_{i+1} DR_{i+1} f_{i+1}, \tag{A.26}$$

$$\delta n_i = R_{i+1}[\delta n_{i+1} + (R_{i+1}^t p_i) \times \delta f_{i+1} + \delta q_{i+1}(E_{i+1} R_{i+1}^t p_i) \times f_{i+1}] + \delta q_{i+1} DR_{i+1}[n_{i+1} + (R_{i+1}^t p_i) \times f_{i+1}] + m_i(p_i + s_i) \times \delta\dot{v}_i + \delta N_i, \tag{A.27}$$

$$\delta\tau_i = \delta n_i^t(R_i^t z) + \delta q_i n_i^t(E_i R_i^t z) + IA_i \delta\ddot{q}_i + RD_i \delta\dot{q}_i, \tag{A.28}$$

where $\partial R_i/\partial q_i = DR_i$ and $\partial R_i^t/\partial q_i = E_i R_i^t$.

If we use $\delta q = 0, \delta\dot{q} = 0$, and $\delta\ddot{q} = 0$ except $\delta q_j = 1$ for some j, then $\delta\tau_i$ gives c_{ij} element of C matrix. Therefore, if we use the equations given above directly to calculate matrices A, B, and C of the linearized equation at $\{\bar{q}(t), \dot{\bar{q}}(t), \ddot{\bar{q}}(t), \bar{\tau}(t)\}$ for some t, then we must calculate the dynamic equation once and differential equations 18 times.

The Lisp program developed here takes the symmetry property of A matrix and the linearly dependence of parameters of PUMA260 into the consideration. Alberto *et al.* [9] investigated the dependence among parameters of PUMA260 and showed only 23 parameters are significant which contain 6 actuator inertias, 6 velocity frictions, and 6 Coulomb frictions.

The Lisp program generates five procedures as follows:

D_equation(q, dq, ddq, tau) calculates input τ from (q, dq, ddq).

comm_terms(q, dq, ddq, tau) calculates 35 common terms used to calculate matrices P, Q, and R.

P_matrix(P) calculates P matrix.

Q_matrix(dq,Q) calculates Q matrix.

R_matrix(dq,R) calculates R matrix.

Table A.1 shows the number of operations contained in the C program generated by the Lisp program.

Linearized equation is calculated by these procedures as follows:

```
void Matrices(q,dq,ddq,tau,P,Q,R)
real q[mm],dq[mm],ddq[mm],tau[mm],P[mm],Q[mm],R[mm];
{
D_equation(q,dq,ddq,tau);
comm_terms(q,dq,ddq,tau);
P_matrix(P);
```

```
Q_matrix(dq,Q);
R_matrix(dq,R);
}

void L_Equation(q,dq,ddq,A,B)
real q[mm],dq[mm],ddq[mm],A[mm][mm],B[mm][mm];
{
real AA[mm][mm],tau[mm],P[mm][mm],Q[mm][mm],R[mm][mm],det,eps=1.0E-6;
int rank,i,j;

Matrices(q,dq,ddq,tau,P,Q,R);
Mat_diag(B,6,1.0);
Mat_sweep(P,B,6,6,6,&det,eps,&rank);

Mat_mul(B,R,AA,6,6,6);
for(i=0;i<6;i++)for(j=0;j<6;j++) A[i][j]= (-AA[i][j]);
Mat_mul(B,Q,AA,6,6,6);
for(i=0;i<6;i++)for(j=0;j<6;j++) A[i][j+6]= (-AA[i][j]);
}
```

where matrices A and B are 6×12 and 6×6 dimensional matrices and correspond to lower half submatrices of A and B in Eq. (A.12), respectively. The dynamic equation which calculates $\ddot{q}$ from $q, \dot{q}$, and τ is given by

```
void Dynamics(tau,q,dq,ddq)
real tau[mm],q[mm],dq[mm],ddq[mm];
{
real tau0[mm],P[mm][mm],AA[mm][mm], det,eps=1.0E-6,dq1[mm],ddq1[mm],tau1[mm];
int i,j,rank;

for(i=0;i<6;i++) ddq[i]=0.0;
D_equation(q,dq,ddq,tau0);
comm_terms(q,dq,ddq,tau0);
P_matrix(P);

for(i=0;i<6;i++) AA[i][0]=tau[i]-tau0[i];
Mat_sweep(P,AA,6,6,1,&det,eps,&rank);

for(i=0;i<6;i++) ddq[i]+=AA[i][0];
}
```

Operations	Dynamic Equation	Linear Equation	Total
Additions	76	473	549
Substructions	53	451	504
Multiplications	184	1311	1495

Table A.1: Number of Operations

Bibliography

[1] J. S. Albus, H. G. McCain, and R. Lumia. *NASA/NBS Standard Reference Model for Telerobot Control System Architecture.* Technical Report NBS Technical Note 1235, National Bureau of Standards, 1987.

[2] A. K. Bejczy, T. J. Tarn, X. Yun, and S. Han. Nonlinear feedback control of puma 560 arm by computer. In *Proceedings of 24th Conference of Decision and Control*, pages 1680–1688, 1985.

[3] C-T. Chen. *Linear System Theory and Design.* Holt, Rinehart and Winston, Inc., 1970.

[4] P. I. Corke. *A New Approach to Laboratory Motor Control MMCS - The Modular Motor Control Stsem.* Technical Report, CIS of Univ. of Penn, 1989.

[5] K. D. Dannenberg and J. L. Melsa. Stability analysis of randomly sampled digital control systems. *Automatica*, 11:99–103, 1975.

[6] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee. *Robotice : Control, Sensing, Vision, and Intelligence.* McGraw-Hill,Inc., 1987.

[7] S. Fujita and T. Fukao. Optimal stochastic control for discrete-time linear system with interrupted observation. *Automatica*, 8:425–432, 1972.

[8] Hikita, Koyama, and Miura. *Society of Instrument and Control Engineers*, 556–561, 1975. in Japanese.

[9] A. Izaguirre, M. Hashimoto, and R. P. Paul. *A New Computational Structure for Real Time Dynamics.* Technical Report, MS-CIS-87-107, CIS of Unv. of Penn., 1987.

[10] R. E. Kalman. *Analysis and Synthesis of Linear Dynamical Systems Operating on Ramdomly Sampled Data.* PhD thesis, Columbia University, New York, 1957.

[11] R. E. Kalman. Control of randomly varying linear dynamical systems. In *Proceedings of Symposia in Applied Mathematics*, pages 287–298, 1962. vol. XIII.

[12] W. L. De Koning. Infinite horizon optimal control of linear discrete time systems with stochastic parameters. *Automatica*, 18:443–453, 1982.

[13] W. L. De Koning. Stationary optimal control of stochastically sampled continuous-time systems. *Automatica*, 24:77–79, 1988.

[14] H. J. Kushner and L. Tobias. On the stability of randomly sampled systems. *IEEE Transactions on Automatic Control*, AC-14(4):319–324, August 1969.

[15] Oscar A. Z. Leneman. Random sampling of random processes: mean-square behavior of a first order closed- loop system. *IEEE Transactions on Automatic Control*, 429–432, August 1968.

[16] A. Li'egeois, A. Fournier, and M. Aldon. Model reference control of high-velocity industrial robots. In *Joint Automatic Control Conference*, 1980.

[17] B. R. Markievicz. *Analysis of the Computed Torque Drive Method and Comparision with Conventional Position Servo for a Computer Controlled Manipulator.* Technical Report Memo 33-601, JPL, 1973.

[18] A. Papoulis. Narrow-band systems and gaussinity. *IEEE Trans. on Information Theory*, IT-18(1):20–27, 1972.

[19] A. Papoulis. *Probability, Random Variables, and Stochastic Processes.* McGraw-Hill Book Company, 1984.

[20] R. P. Paul. *Modeling, Trajectory Calculation and Servoing of a Computer Controlled Arm.* Technical Report AIM-177, Stanford university Artificial Inteligence Laboratory, 1972.

```
#include     <math.h>

#define real float
#define mm   6
#define G 9.81
#define M6 2.768114
#define Z6 0.01401
#define Y4 (-0.38219)
#define AD4 0.079781
#define CD4 0.077761
#define IA1 0.091631
#define IA2 0.136312
#define IA3 0.030843
#define IA4 0.001781
#define IA5 0.006759
#define IA6 0.001262
#define RD1 0.575662
#define RD2 0.94467
#define RD3 0.417502
#define RD4 0.066791
#define RD5 0.101721
#define RD6 0.030363
#define A2 0.2032
#define D3 0.12624
#define D4 0.2032

    static real q[mm],dq[mm],ddq[mm],tau[mm],P[mm][mm],Q[mm][mm],R[mm][mm];
    static real dl_Wx[6],dl_Wy[6],dl_Wz[6],dl_dWx[6],dl_dWy[6],dl_dWz[6];
    static real dl_dVx[6],dl_dVy[6],dl_dVz[6],dl_Fx[6],dl_Fy[6],dl_Fz[6];
    static real dl_Nx[6],dl_Ny[6],dl_Nz[6],dl_fx[6],dl_fy[6],dl_fz[6];
    static real dl_nx[6],dl_ny[6],dl_nz[6];

    static real S1,C1,S2,C2,S3,C3,S4,C4,S5,C5;
    static real S6,C6,Wx2,Wy2,dWx2,dWy2,dVx2,dVy2,Wx3,Wy3;
    static real Wz3,dWx3,dWy3,dWz3,dVx3,dVy3,dVz3,Wx4,Wy4,Wz4;
    static real dWx4,dWy4,dWz4,dVx4,dVy4,dVz4,Wx5,Wy5,Wz5,dWx5;
    static real dWy5,dWz5,dVx5,dVy5,dVz5,Wx6,Wy6,Wz6,dWx6,dWy6;
    static real dWz6,dVx6,dVy6,Fx6,Fy6,Fz6,nx6,ny6,fx5,fy5;
    static real nx5,ny5,Fx4,Fy4,Fz4,fx4,fy4,fz4,Nx4,Ny4;
    static real Nz4,nx4,ny4,nz4,fx3,fy3,nx3,ny3,nz3,fx2;
    static real fy2,fz2,nx2,ny2,nz2,CM0,CM1,CM2,CM3,CM4;
    static real CM5,CM6,CM7,CM8,CM9,CM10,CM11,CM12,CM13,CM14;
    static real CM15,CM16,CM17,CM18,CM19,CM20,CM21,CM22,CM23,CM24;
    static real CM25,CM26,CM27,CM28,CM29,CM30,CM31,CM32,CM33,CM34;
    static real CM35;



void D_equation(q,dq,ddq,tau)
    real q[mm],dq[mm],ddq[mm],tau[mm];
{

    real TM0,TM1,TM2,TM3,TM4,TM5,TM6,TM7,TM8,TM9;

    S1= sin(q[0]);
    C1= cos(q[0]);

    S2= sin(q[1]);
    C2= cos(q[1]);

    S3= sin(q[2]);
    C3= cos(q[2]);

    S4= sin(q[3]);
```

```
    C4= cos(q[3]);

    S5= sin(q[4]);
    C5= cos(q[4]);

    S6= sin(q[5]);
    C6= cos(q[5]);


/*****************************/


    Wx2= S2*dq[0];
    Wy2= C2*dq[0];

    TM0= dq[0]*dq[1];
    dWx2= S2*ddq[0]+C2*TM0;
    dWy2= C2*ddq[0]-S2*TM0;

    dVx2= S2*G;
    dVy2= C2*G;


    Wx3= S3*Wy2+C3*Wx2;
    Wy3= -(dq[1]+dq[2]);
    Wz3= C3*Wy2-S3*Wx2;

    TM0= Wx2*dq[2];
    TM1= Wy2*dq[2];
    TM2= dWy2-TM0;
    TM3= dWx2+TM1;
    dWx3= S3*TM2+C3*TM3;
    dWy3= -(ddq[2]+ddq[1]);
    dWz3= C3*TM2-S3*TM3;

    TM0= ddq[1]*A2;
    TM1= dq[1]*A2;
    TM2= Wy2*A2;
    TM3= Wx2*TM2;
    TM4= Wy2*TM2;
    TM5= dq[1]*TM1;
    TM6= TM3+TM0;
    TM7= TM6+dVy2;
    TM8= TM4+TM5;
    TM9= dVx2-TM8;
    dVx3= S3*TM7+C3*TM9;
    dVy3= dWy2*A2-Wx2*TM1;
    dVz3= C3*TM7-S3*TM9;


    Wx4= S4*Wy3+C4*Wx3;
    Wy4= Wz3+dq[3];
    Wz4= S4*Wx3-C4*Wy3;

    TM0= Wy3*dq[3];
    TM1= Wx3*dq[3];
    TM2= dWx3+TM0;
    TM3= dWy3-TM1;
    dWx4= S4*TM3+C4*TM2;
    dWy4= ddq[3]+dWz3;
    dWz4= S4*TM2-C4*TM3;

    TM0= dWz3*D3;
    TM1= Wz3*D3;
```

```
    TM2= Wx3*D3;
    TM3= Wy3*TM2;
    TM4= Wz3*TM1;
    TM5= Wx3*TM2;
    TM6= TM0-TM3;
    TM7= TM6+dVx3;
    TM8= TM4+TM5;
    TM9= TM8+dVy3;
    dVx4= S4*TM9+C4*TM7;
    dVy4= dVz3-(Wy3*TM1+dWx3*D3);
    dVz4= S4*TM7-C4*TM9;


    Wx5= S5*Wy4+C5*Wx4;
    Wy5= -(Wz4+dq[4]);
    Wz5= C5*Wy4-S5*Wx4;

    TM0= Wx4*dq[4];
    TM1= Wy4*dq[4];
    TM2= dWy4-TM0;
    TM3= dWx4+TM1;
    dWx5= S5*TM2+C5*TM3;
    dWy5= -(ddq[4]+dWz4);
    dWz5= C5*TM2-S5*TM3;

    TM0= dWz4*D4;
    TM1= Wz4*D4;
    TM2= Wx4*D4;
    TM3= Wz4*TM1;
    TM4= Wx4*TM2;
    TM5= Wy4*TM2;
    TM6= TM5-TM0;
    TM7= TM3+TM4;
    TM8= TM6+dVx4;
    TM9= dVy4-TM7;
    dVx5= S5*TM9+C5*TM8;
    dVy5= -(Wy4*TM1+dWx4*D4+dVz4);
    dVz5= C5*TM9-S5*TM8;


    Wx6= S6*Wy5+C6*Wx5;
    Wy6= C6*wy5-S6*Wx5;
    Wz6= Wz5+dq[5];

    TM0= Wx5*dq[5];
    TM1= Wy5*dq[5];
    TM2= dWy5-TM0;
    TM3= dWx5+TM1;
    dWx6= S6*TM2+C6*TM3;
    dWy6= C6*TM2-S6*TM3;
    dWz6= ddq[5]+dWz5;

    dVx6= S6*dVy5+C6*dVx5;
    dVy6= C6*dVy5-S6*dVx5;


/***** backwrad equation ****/

    TM0= Wx6*Z6;
    TM1= Wy6*Z6;
    Fx6= M6*dVx6+dWy6*Z6+Wz6*TM0;
    Fy6= M6*dVy6+Wz6*TM1-dWx6*Z6;
    Fz6= M6*dVz5-(Wx6*TM0+Wy6*TM1);
```

```
    nx6= -(Z6*dVy6);
    ny6= Z6*dVx6;

    tau[5]= RD6*dq[5]+IA6*ddq[5];

    fx5= C6*Fx6-S6*Fy6;
    fy5= C6*Fy6+S6*Fx6;

    nx5= C6*nx6-S6*ny6;
    ny5= C6*ny6+S6*nx6;

    tau[4]= RD5*dq[4]+IA5*ddq[4]-ny5;

    TM0= Wz4*Y4;
    TM1= Wx4*Y4;
    Fx4= Wy4*TM1-dWz4*Y4;
    Fy4= -(Wz4*TM0+Wx4*TM1);
    Fz4= dWx4*Y4+Wy4*TM0;

    fx4= C5*fx5-S5*Fz6+Fx4;
    fy4= C5*Fz6+S5*fx5+Fy4;
    fz4= Fz4-fy5;

    TM0= AD4*Wx4;
    TM1= CD4*Wz4;
    Nx4= Wy4*TM1+AD4*dWx4;
    Ny4= Wz4*TM0-Wx4*TM1;
    Nz4= CD4*dWz4-Wy4*TM0;

    TM0= S5*D4;
    TM1= C5*D4;
    TM2= TM0*fy5;
    TM3= TM1*fy5;
    TM4= nx5-TM3;
    nx4= Y4*dVz4+C5*TM4-S5*TM2+Nx4;
    ny4= C5*TM2+S5*TM4+Ny4;
    nz4= Nz4-(Y4*dVx4+TM1*fx5-TM0*Fz6+ny5);

    tau[3]= ny4+RD4*dq[3]+IA4*ddq[3];

    fx3= S4*fz4+C4*fx4;
    fy3= S4*fx4-C4*fz4;

    TM0= C4*D3;
    TM1= S4*D3;
    TM2= TM0*fy4;
    TM3= TM1*fy4;
    TM4= nx4-TM2;
    TM5= nz4-TM3;
    nx3= S4*TM5+C4*TM4;
    ny3= S4*TM4-C4*TM5;
    nz3= TM0*fx4+TM1*fz4+ny4;

    tau[2]= RD3*dq[2]+IA3*ddq[2]-ny3;

    fx2= C3*fx3-S3*fy4;
    fy2= C3*fy4+S3*fx3;
    fz2= -(fy3);

    TM0= C3*A2;
    TM1= S3*A2;
    TM2= TM0*fy3;
    TM3= TM1*fy3;
    TM4= TM2+nz3;
```

```
    TM5= TM3+nx3;
    nx2= C3*TM5-S3*TM4;
    ny2= C3*TM4+S3*TM5;
    nz2= TM1*fx3+TM0*fy4-ny3;

    tau[1]= nz2+RD2*dq[1]+IA2*ddq[1];

    tau[0]= C2*ny2+S2*nx2+RD1*dq[0]+IA1*ddq[0];

}


/*****************************************

   Additions       : 73
   Substractions   : 53
   Multiplications : 184

*****************************************/


void comm_terms(q,dq,ddq,tau)
    real q[mm],dq[mm],ddq[mm],tau[mm];
{

    CM0= C5*S4;
    CM1= AD4*S4;
    CM2= CD4*C4;
    CM3= C4*Y4;
    CM4= S4*Y4;
    CM5= C6*C4;
    CM6= S6*C4;
    CM7= C4*dq[5];
    CM8= C4*D4;
    CM9= S4*D4;
    CM10= S4*dq[4];
    CM11= S3*A2;
    CM12= C3*A2;
    CM13= C4*D3;
    CM14= S4*D3;
    CM15= C5*D4;
    CM16= S5*D4;
    CM17= AD4*Wx4;
    CM18= CD4*Wz4;
    CM19= Wz4*Y4;
    CM20= Wx4*Y4;
    CM21= Wy6*Z6;
    CM22= Wx6*Z6;
    CM23= Wz4*D4;
    CM24= Wx4*D4;
    CM25= Wz3*D3;
    CM26= Wx3*D3;
    CM27= Wy2*A2;
    CM28= Wx4*CM9;
    CM29= Wz4*CM8;
    CM30= S4*CM24;
    CM31= C4*CM23;
    CM32= Wy4*CM9;
    CM33= CM28-CM29;
    CM34= CM30-CM31;
    CM35= CM33+CM34;
```

```
void R_matrix1(dq,R)
    real dq[mm], R[mm][mm];
{


    R[5][0]= 0.0;

    R[4][0]= 0.0;

    R[3][0]= 0.0;

    R[2][0]= 0.0;

    R[1][0]= 0.0;

    R[0][0]= 0.0;


}

void R_matrix2(dq,R)
    real dq[mm], R[mm][mm];
{

    real TM0,TM1,TM2,TM3,TM4,TM5,TM6,TM7,TM8,TM9;
    real TM10,TM11,TM12,TM13,TM14,TM15;

    dl_Wx[2]= C3*Wy2-S3*Wx2;
    dl_Wz[2]= -(C3*Wx2+S3*Wy2);

    TM0= Wy2*dq[2];
    TM1= Wx2*dq[2];
    TM2= dWx2+TM0;
    TM3= dWy2-TM1;
    dl_dWx[2]= C3*TM3-S3*TM2;
    dl_dWz[2]= -(C3*TM2+S3*TM3);

    TM0= Wx2*A2;
    TM1= Wy2*CM27;
    TM2= Wx2*TM0;
    TM3= Wy2*TM0;
    TM4= Wx2*CM27;
    TM5= TM1-TM2;
    TM6= TM3+TM4;
    TM7= TM5-dVx2;
    TM8= TM6+dVy2;
    dl_dVx[2]= S3*TM7+C3*TM8;
    dl_dVy[2]= -(Wy2*dq[1]*A2+dWx2*A2);
    dl_dVz[2]= C3*TM7-S3*TM8;

    dl_Wx[3]= C4*dl_Wx[2];
    dl_Wz[3]= S4*dl_Wx[2];

    TM0= dl_Wx[2]*dq[3];
    dl_dWx[3]= C4*dl_dWx[2]-S4*TM0;
    dl_dWz[3]= C4*TM0+S4*dl_dWx[2];

    TM0= dl_dWz[2]*D3;
    TM1= dl_Wz[2]*D3;
    TM2= dl_Wx[2]*D3;
    TM3= Wy3*TM2;
    TM4= Wz3*TM1;
```

```
    TM5= Wx3*TM2;
    TM6= dl_Wz[2]*CM25;
    TM7= dl_Wx[2]*CM26;
    TM8= TM0-TM3;
    TM9= TM8+dl_dVx[2];
    TM10= TM4+TM5;
    TM11= TM6+TM7;
    TM12= TM10+TM11;
    TM13= TM12+dl_dVy[2];
    dl_dVx[3]= S4*TM13+C4*TM9;
    dl_dVy[3]= dl_dVz[2]-(Wy3*TM1+dl_dWx[2]*D3);
    dl_dVz[3]= S4*TM9-C4*TM13;

    dl_Wx[4]= S5*dl_Wz[2]+C5*dl_Wx[3];
    dl_Wz[4]= C5*dl_Wz[2]-S5*dl_Wx[3];

    TM0= dl_Wx[3]*dq[4];
    TM1= dl_Wz[2]*dq[4];
    TM2= dl_dWz[2]-TM0;
    TM3= dl_dWx[3]+TM1;
    dl_dWx[4]= S5*TM2+C5*TM3;

    TM0= dl_dWz[3]*D4;
    TM1= dl_Wz[3]*D4;
    TM2= dl_Wx[3]*D4;
    TM3= Wz4*TM1;
    TM4= Wx4*TM2;
    TM5= dl_Wz[3]*CM23;
    TM6= dl_Wx[3]*CM24;
    TM7= Wy4*TM2;
    TM8= dl_Wz[2]*CM24;
    TM9= TM8-TM0;
    TM10= TM3+TM4;
    TM11= TM5+TM6;
    TM12= TM7+TM9;
    TM13= TM12+dl_dVx[3];
    TM14= TM10+TM11;
    TM15= dl_dVy[3]-TM14;
    dl_dVx[4]= S5*TM15+C5*TM13;
    dl_dVy[4]= -(Wy4*TM1+dl_Wz[2]*CM23+dl_dWx[3]*D4+dl_dVz[3]);
    dl_dVz[4]= C5*TM15-S5*TM13;

    dl_dVx[5]= S6*dl_dVy[4]+C6*dl_dVx[4];
    dl_dVy[5]= C6*dl_dVy[4]-S6*dl_dVx[4];

    TM0= C6*dl_Wz[3];
    TM1= S6*dl_Wx[4];
    TM2= C6*dl_Wx[4];
    TM3= S6*dl_Wz[3];
    TM4= dl_Wz[3]*dq[5];
    TM5= dl_Wx[4]*dq[5];
    TM6= dl_dWx[4]-TM4;
    TM7= dl_dWz[3]+TM5;
    TM8= TM0+TM1;
    TM9= TM2-TM3;
    TM10= TM8*Z6;
    TM11= TM9*Z6;
    dl_Fx[5]= M6*dl_dVx[5]+Wz6*TM11+dl_Wz[4]*CM22-(C6*TM7+S6*TM6)*Z6;
    dl_Fy[5]= M6*dl_dVy[5]+dl_Wz[4]*CM21-(C6*TM6-S6*TM7)*Z6-Wz6*TM10;
    dl_Fz[5]= M6*dl_dVz[4]+Wy6*TM10-Wx6*TM11+TM8*CM21-TM9*CM22;

    dl_nx[5]= -(Z6*dl_dVy[5]);
    dl_ny[5]= Z6*dl_dVx[5];
```

```
    R[5][1]= 0.0;

    dl_fx[4]= C6*dl_Fx[5]-S6*dl_Fy[5];
    dl_fy[4]= S6*dl_Fx[5]+C6*dl_Fy[5];

    dl_nx[4]= C6*dl_nx[5]-S6*dl_ny[5];
    dl_ny[4]= S6*dl_nx[5]+C6*dl_ny[5];

    R[4][1]= -(dl_ny[4]);

    TM0= dl_Wz[3]*Y4;
    TM1= dl_Wx[3]*Y4;
    dl_Fx[3]= Wy4*TM1+dl_Wz[2]*CM20-dl_dWz[3]*Y4;
    dl_Fy[3]= -(Wz4*TM0+Wx4*TM1+dl_Wz[3]*CM19+dl_Wx[3]*CM20);
    dl_Fz[3]= Wy4*TM0+dl_Wz[2]*CM19+dl_dWx[3]*Y4;

    dl_fx[3]= dl_Fx[3]+C5*dl_fx[4]-S5*dl_Fz[5];
    dl_fy[3]= dl_Fy[3]+S5*dl_fx[4]+C5*dl_Fz[5];
    dl_fz[3]= dl_Fz[3]-dl_fy[4];

    TM0= AD4*dl_Wx[3];
    TM1= CD4*dl_Wz[3];
    dl_Nx[3]= Wy4*TM1+dl_Wz[2]*CM18+AD4*dl_dWx[3];
    dl_Ny[3]= Wz4*TM0-Wx4*TM1+dl_Wz[3]*CM17-dl_Wx[3]*CM18;
    dl_Nz[3]= CD4*dl_dWz[3]-dl_Wz[2]*CM17-Wy4*TM0;

    TM0= CM15*dl_fy[4];
    TM1= CM16*dl_fy[4];
    TM2= dl_nx[4]-TM0;
    dl_nx[3]= dl_Nx[3]+Y4*dl_dVz[3]+C5*TM2-S5*TM1;
    dl_ny[3]= dl_Ny[3]+S5*TM2+C5*TM1;
    dl_nz[3]= dl_Nz[3]-(Y4*dl_dVx[3]+dl_ny[4]+CM15*dl_fx[4]-CM16*dl_Fz[5]);

    R[3][1]= dl_ny[3];

    dl_fx[2]= C4*dl_fx[3]+S4*dl_fz[3];
    dl_fy[2]= S4*dl_fx[3]-C4*dl_fz[3];

    TM0= CM13*dl_fy[3];
    TM1= CM14*dl_fy[3];
    TM2= dl_nx[3]-TM0;
    TM3= TM1-dl_nz[3];
    dl_nx[2]= C4*TM2-S4*TM3;
    dl_ny[2]= S4*TM2+C4*TM3;
    dl_nz[2]= dl_ny[3]+CM13*dl_fx[3]+CM14*dl_fz[3];

    R[2][1]= -(dl_ny[2]);

    TM0= CM11*dl_fy[2];
    TM1= CM12*dl_fy[2];
    TM2= dl_nx[2]+TM0;
    TM3= dl_nz[2]+TM1;
    dl_nx[1]= C3*TM2-S3*TM3;
    dl_ny[1]= S3*TM2+C3*TM3;
    dl_nz[1]= CM11*dl_fx[2]+CM12*dl_fy[3]-dl_ny[2];

    R[1][1]= dl_nz[1];

    R[0][1]= S2*(dl_nx[1]-ny2)+C2*(dl_ny[1]+nx2);


}

void R_matrix3(dq,R)
```

```
    real dq[mm], R[mm][mm];
{

    real TM0,TM1,TM2,TM3,TM4,TM5,TM6,TM7,TM8,TM9;
    real TM10,TM11,TM12,TM13,TM14,TM15;

    dl_Wx[3]= C4*Wz3;
    dl_Wz[3]= S4*Wz3;

    TM0= Wz3*dq[3];
    dl_dWx[3]= C4*dWz3-S4*TM0;
    dl_dWz[3]= C4*TM0+S4*dWz3;

    TM0= dWx3*D3;
    TM1= Wy3*CM25;
    TM2= TM1+TM0;
    TM3= dVz3-TM2;
    dl_dVx[3]= C4*TM3;
    dl_dVy[3]= Wy3*CM26-dWz3*D3-dVx3;
    dl_dVz[3]= S4*TM3;

    dl_Wx[4]= C5*dl_Wx[3]-S5*Wx3;
    dl_Wz[4]= -(C5*Wx3+S5*dl_Wx[3]);

    TM0= dl_Wx[3]*dq[4];
    TM1= Wx3*dq[4];
    TM2= dWx3+TM0;
    TM3= dl_dWx[3]-TM1;
    dl_dWx[4]= C5*TM3-S5*TM2;

    TM0= dl_dWz[3]*D4;
    TM1= dl_Wz[3]*D4;
    TM2= dl_Wx[3]*D4;
    TM3= Wz4*TM1;
    TM4= Wx4*TM2;
    TM5= dl_Wz[3]*CM23;
    TM6= dl_Wx[3]*CM24;
    TM7= Wy4*TM2;
    TM8= Wx3*CM24;
    TM9= TM8+TM0;
    TM10= TM3+TM4;
    TM11= TM5+TM6;
    TM12= TM7-TM9;
    TM13= TM12+dl_dVx[3];
    TM14= TM10+TM11;
    TM15= dl_dVy[3]-TM14;
    dl_dVx[4]= S5*TM15+C5*TM13;
    dl_dVy[4]= -(Wy4*TM1+dl_dWx[3]*D4-Wx3*CM23+dl_dVz[3]);
    dl_dVz[4]= C5*TM15-S5*TM13;

    dl_dVx[5]= S6*dl_dVy[4]+C6*dl_dVx[4];
    dl_dVy[5]= C6*dl_dVy[4]-S6*dl_dVx[4];

    TM0= C6*dl_Wz[3];
    TM1= S6*dl_Wx[4];
    TM2= C6*dl_Wx[4];
    TM3= S6*dl_Wz[3];
    TM4= dl_Wz[3]*dq[5];
    TM5= dl_Wx[4]*dq[5];
    TM6= dl_dWx[4]-TM4;
    TM7= dl_dWz[3]+TM5;
    TM8= TM0+TM1;
    TM9= TM2-TM3;
    TM10= TM8*Z6;
```

```
    TM11= TM9*Z6;
    dl_Fx[5]= M6*dl_dVx[5]+Wz6*TM11+dl_Wz[4]*CM22-(C6*TM7+S6*TM6)*Z6;
    dl_Fy[5]= M6*dl_dVy[5]+dl_Wz[4]*CM21-(C6*TM6-S6*TM7)*Z6-Wz6*TM10;
    dl_Fz[5]= M6*dl_dVz[4]+Wy6*TM10-Wx6*TM11+TM8*CM21-TM9*CM22;

    dl_nx[5]= -(Z6*dl_dVy[5]);
    dl_ny[5]= Z6*dl_dVx[5];

    R[5][2]= 0.0;

    dl_fx[4]= C6*dl_Fx[5]-S6*dl_Fy[5];
    dl_fy[4]= S6*dl_Fx[5]+C6*dl_Fy[5];

    dl_nx[4]= C6*dl_nx[5]-S6*dl_ny[5];
    dl_ny[4]= S6*dl_nx[5]+C6*dl_ny[5];

    R[4][2]= -(dl_ny[4]);

    TM0= dl_Wz[3]*Y4;
    TM1= dl_Wx[3]*Y4;
    dl_Fx[3]= Wy4*TM1-(Wx3*CM20+dl_dWz[3]*Y4);
    dl_Fy[3]= -(Wz4*TM0+Wx4*TM1+dl_Wz[3]*CM19+dl_Wx[3]*CM20);
    dl_Fz[3]= Wy4*TM0+dl_dWx[3]*Y4-Wx3*CM19;

    dl_fx[3]= dl_Fx[3]+C5*dl_fx[4]-S5*dl_Fz[5];
    dl_fy[3]= dl_Fy[3]+S5*dl_fx[4]+C5*dl_Fz[5];
    dl_fz[3]= dl_Fz[3]-dl_fy[4];

    TM0= AD4*dl_Wx[3];
    TM1= CD4*dl_Wz[3];
    dl_Nx[3]= Wy4*TM1+AD4*dl_dWx[3]-Wx3*CM18;
    dl_Ny[3]= Wz4*TM0-Wx4*TM1+dl_Wz[3]*CM17-dl_Wx[3]*CM18;
    dl_Nz[3]= Wx3*CM17+CD4*dl_dWz[3]-Wy4*TM0;

    TM0= CM15*dl_fy[4];
    TM1= CM16*dl_fy[4];
    TM2= dl_nx[4]-TM0;
    dl_nx[3]= dl_Nx[3]+Y4*dl_dVz[3]+C5*TM2-S5*TM1;
    dl_ny[3]= dl_Ny[3]+S5*TM2+C5*TM1;
    dl_nz[3]= dl_Nz[3]-(Y4*dl_dVx[3]+dl_ny[4]+CM15*dl_fx[4]-CM16*dl_Fz[5]);

    R[3][2]= dl_ny[3];

    dl_fx[2]= C4*dl_fx[3]+S4*dl_fz[3];
    dl_fy[2]= S4*dl_fx[3]-C4*dl_fz[3];

    TM0= CM13*dl_fy[3];
    TM1= CM14*dl_fy[3];
    TM2= dl_nx[3]-TM0;
    TM3= TM1-dl_nz[3];
    dl_nx[2]= C4*TM2-S4*TM3;
    dl_ny[2]= S4*TM2+C4*TM3;
    dl_nz[2]= dl_ny[3]+CM13*dl_fx[3]+CM14*dl_fz[3];

    R[2][2]= -(dl_ny[2]);

    TM0= CM11*dl_fy[2];
    TM1= CM12*dl_fy[2];
    TM2= CM12*fy3;
    TM3= CM11*fy3;
    TM4= dl_nx[2]+TM0;
    TM5= nz3+TM2;
    TM6= dl_nz[2]+TM1;
    TM7= nx3+TM3;
```

```
    TM8= TM2+TM4;
    TM9= TM6-TM3;
    TM10= TM8-TM5;
    TM11= TM9+TM7;
    dl_nx[1]= C3*TM10-S3*TM11;
    dl_ny[1]= S3*TM10+C3*TM11;
    dl_nz[1]= -(CM11*fy4-CM12*fx3+dl_ny[2]-(CM11*dl_fx[2]+CM12*dl_fy[3]));

    R[1][2]= dl_nz[1];

    R[0][2]= S2*dl_nx[1]+C2*dl_ny[1];


}

void R_matrix4(dq,R)
    real dq[mm], R[mm][mm];
{

    real TM0,TM1,TM2,TM3,TM4,TM5,TM6,TM7,TM8,TM9;
    real TM10,TM11;

    dl_Wx[4]= -(C5*Wz4);
    dl_Wz[4]= S5*Wz4;

    TM0= Wz4*dq[4];
    dl_dWx[4]= S5*TM0-C5*dWz4;

    TM0= dWx4*D4;
    TM1= Wy4*CM23;
    TM2= TM1+TM0;
    TM3= TM2+dVz4;
    dl_dVx[4]= -(C5*TM3);
    dl_dVy[4]= -(Wy4*CM24-dWz4*D4+dVx4);
    dl_dVz[4]= S5*TM3;

    dl_dVx[5]= S6*dl_dVy[4]+C6*dl_dVx[4];
    dl_dVy[5]= C6*dl_dVy[4]-S6*dl_dVx[4];

    TM0= C6*Wx4;
    TM1= S6*dl_Wx[4];
    TM2= C6*dl_Wx[4];
    TM3= S6*Wx4;
    TM4= Wx4*dq[5];
    TM5= dl_Wx[4]*dq[5];
    TM6= dl_dWx[4]-TM4;
    TM7= dWx4+TM5;
    TM8= TM0+TM1;
    TM9= TM2-TM3;
    TM10= TM8*Z6;
    TM11= TM9*Z6;
    dl_Fx[5]= M6*dl_dVx[5]+Wz6*TM11+dl_Wz[4]*CM22-(C6*TM7+S6*TM6)*Z6;
    dl_Fy[5]= M6*dl_dVy[5]+dl_Wz[4]*CM21-(C6*TM6-S6*TM7)*Z6-Wz6*TM10;
    dl_Fz[5]= M6*dl_dVz[4]+Wy6*TM10-Wx6*TM11+TM8*CM21-TM9*CM22;

    dl_nx[5]= -(Z6*dl_dVy[5]);
    dl_ny[5]= Z6*dl_dVx[5];

    R[5][3]= 0.0;

    dl_fx[4]= C6*dl_Fx[5]-S6*dl_Fy[5];
    dl_fy[4]= S6*dl_Fx[5]+C6*dl_Fy[5];

    dl_nx[4]= C6*dl_nx[5]-S6*dl_ny[5];
```

```
    dl_ny[4]= S6*dl_nx[5]+C6*dl_ny[5];

    R[4][3]= -(dl_ny[4]);

    dl_Fx[3]= -(Wy4*CM19+dWx4*Y4);
    dl_Fz[3]= Wy4*CM20-dWz4*Y4;

    dl_fx[3]= dl_Fx[3]+C5*dl_fx[4]-S5*dl_Fz[5];
    dl_fy[3]= S5*dl_fx[4]+C5*dl_Fz[5];
    dl_fz[3]= dl_Fz[3]-dl_fy[4];

    TM0= AD4*Wz4;
    TM1= CD4*Wx4;
    dl_Nx[3]= Wy4*TM1-AD4*dWz4;
    dl_Ny[3]= Wx4*CM17+Wz4*CM18-(Wz4*TM0+Wx4*TM1);
    dl_Nz[3]= Wy4*TM0+CD4*dWx4;

    TM0= CM15*dl_fy[4];
    TM1= CM16*dl_fy[4];
    TM2= dl_nx[4]-TM0;
    dl_nx[3]= dl_Nx[3]+Y4*dVx4+C5*TM2-S5*TM1;
    dl_ny[3]= dl_Ny[3]+S5*TM2+C5*TM1;
    dl_nz[3]= dl_Nz[3]+Y4*dVz4-(dl_ny[4]+CM15*dl_fx[4]-CM16*dl_Fz[5]);

    R[3][3]= dl_ny[3];

    TM0= dl_fx[3]+fz4;
    TM1= fx4-dl_fz[3];
    dl_fx[2]= C4*TM0-S4*TM1;
    dl_fy[2]= S4*TM0+C4*TM1;

    TM0= CM13*dl_fy[3];
    TM1= CM14*dl_fy[3];
    TM2= CM14*fy4;
    TM3= CM13*fy4;
    TM4= dl_nx[3]-TM0;
    TM5= nz4-TM2;
    TM6= dl_nz[3]-TM1;
    TM7= nx4-TM3;
    TM8= TM2+TM4;
    TM9= TM3-TM6;
    TM10= TM8+TM5;
    TM11= TM9+TM7;
    dl_nx[2]= C4*TM10-S4*TM11;
    dl_ny[2]= S4*TM10+C4*TM11;
    dl_nz[2]= CM13*fz4-CM14*fx4+dl_ny[3]+CM13*dl_fx[3]+CM14*dl_fz[3];

    R[2][3]= -(dl_ny[2]);

    TM0= CM11*dl_fy[2];
    TM1= CM12*dl_fy[2];
    TM2= dl_nx[2]+TM0;
    TM3= dl_nz[2]+TM1;
    dl_nx[1]= C3*TM2-S3*TM3;
    dl_ny[1]= S3*TM2+C3*TM3;
    dl_nz[1]= CM11*dl_fx[2]+CM12*dl_fy[3]-dl_ny[2];

    R[1][3]= dl_nz[1];

    R[0][3]= S2*dl_nx[1]+C2*dl_ny[1];


}
```

```
void R_matrix5(dq,R)
    real dq[mm], R[mm][mm];
{

    real TM0,TM1,TM2,TM3,TM4,TM5,TM6,TM7,TM8,TM9;

    dl_dVx[5]= C6*dVz5;
    dl_dVy[5]= -(S6*dVz5);

    TM0= S6*Wz5;
    TM1= C6*Wz5;
    TM2= Wz5*dq[5];
    TM3= TM0*Z6;
    TM4= TM1*Z6;
    dl_Fx[5]= M6*dl_dVx[5]+Wz6*TM4-(Wx5*CM22+(C6*TM2+S6*dWz5)*Z6);
    dl_Fy[5]= M6*dl_dVy[5]-(Wz6*TM3+Wx5*CM21+(C6*dWz5-S6*TM2)*Z6);
    dl_Fz[5]= Wy6*TM3-Wx6*TM4+TM0*CM21-TM1*CM22-M6*dVx5;

    dl_nx[5]= -(Z6*dl_dVy[5]);
    dl_ny[5]= Z6*dl_dVx[5];

    R[5][4]= 0.0;

    dl_fx[4]= C6*dl_Fx[5]-S6*dl_Fy[5];
    dl_fy[4]= S6*dl_Fx[5]+C6*dl_Fy[5];

    dl_nx[4]= C6*dl_nx[5]-S6*dl_ny[5];
    dl_ny[4]= S6*dl_nx[5]+C6*dl_ny[5];

    R[4][4]= -(dl_ny[4]);

    TM0= dl_fx[4]-Fz6;
    TM1= dl_Fz[5]+fx5;
    dl_fx[3]= C5*TM0-S5*TM1;
    dl_fy[3]= S5*TM0+C5*TM1;

    TM0= CM15*dl_fy[4];
    TM1= CM16*dl_fy[4];
    TM2= CM16*fy5;
    TM3= CM15*fy5;
    TM4= dl_nx[4]-TM0;
    TM5= nx5-TM3;
    TM6= TM3+TM1;
    TM7= TM2+TM4;
    TM8= TM7-TM2;
    TM9= TM6+TM5;
    dl_nx[3]= C5*TM8-S5*TM9;
    dl_ny[3]= S5*TM8+C5*TM9;
    dl_nz[3]= CM16*fx5+CM15*Fz6-(dl_ny[4]+CM15*dl_fx[4]-CM16*dl_Fz[5]);

    R[3][4]= dl_ny[3];

    dl_fx[2]= C4*dl_fx[3]-S4*dl_fy[4];
    dl_fy[2]= S4*dl_fx[3]+C4*dl_fy[4];

    TM0= CM13*dl_fy[3];
    TM1= CM14*dl_fy[3];
    TM2= dl_nx[3]-TM0;
    TM3= TM1-dl_nz[3];
    dl_nx[2]= C4*TM2-S4*TM3;
    dl_ny[2]= S4*TM2+C4*TM3;
    dl_nz[2]= dl_ny[3]+CM13*dl_fx[3]-CM14*dl_fy[4];

    R[2][4]= -(dl_ny[2]);
```

```
    TM0= CM11*dl_fy[2];
    TM1= CM12*dl_fy[2];
    TM2= dl_nx[2]+TM0;
    TM3= dl_nz[2]+TM1;
    dl_nx[1]= C3*TM2-S3*TM3;
    dl_ny[1]= S3*TM2+C3*TM3;
    dl_nz[1]= CM11*dl_fx[2]+CM12*dl_fy[3]-dl_ny[2];

    R[1][4]= dl_nz[1];

    R[0][4]= S2*dl_nx[1]+C2*dl_ny[1];


}

void R_matrix6(dq,R)
    real dq[mm], R[mm][mm];
{

    real TM0,TM1,TM2,TM3;

    dl_Fx[5]= M6*dVy6+Wz6*CM21-dWx6*Z6;
    dl_Fy[5]= -(M6*dVx6+Wz6*CM22+dWy6*Z6);

    dl_nx[5]= Z6*dVx6;
    dl_ny[5]= Z6*dVy6;

    R[5][5]= 0.0;

    TM0= dl_Fx[5]-Fy6;
    TM1= dl_Fy[5]+Fx6;
    dl_fx[4]= C6*TM0-S6*TM1;
    dl_fy[4]= S6*TM0+C6*TM1;

    TM0= dl_nx[5]-ny6;
    TM1= dl_ny[5]+nx6;
    dl_nx[4]= C6*TM0-S6*TM1;
    dl_ny[4]= S6*TM0+C6*TM1;

    R[4][5]= -(dl_ny[4]);

    dl_fx[3]= C5*dl_fx[4];
    dl_fy[3]= S5*dl_fx[4];

    TM0= CM15*dl_fy[4];
    TM1= CM16*dl_fy[4];
    TM2= dl_nx[4]-TM0;
    dl_nx[3]= C5*TM2-S5*TM1;
    dl_ny[3]= S5*TM2+C5*TM1;
    dl_nz[3]= -(dl_ny[4]+CM15*dl_fx[4]);

    R[3][5]= dl_ny[3];

    dl_fx[2]= C4*dl_fx[3]-S4*dl_fy[4];
    dl_fy[2]= S4*dl_fx[3]+C4*dl_fy[4];

    TM0= CM13*dl_fy[3];
    TM1= CM14*dl_fy[3];
    TM2= dl_nx[3]-TM0;
    TM3= TM1-dl_nz[3];
    dl_nx[2]= C4*TM2-S4*TM3;
    dl_ny[2]= S4*TM2+C4*TM3;
    dl_nz[2]= dl_ny[3]+CM13*dl_fx[3]-CM14*dl_fy[4];
```

```
    R[2][5]= -(dl_ny[2]);

    TM0= CM11*dl_fy[2];
    TM1= CM12*dl_fy[2];
    TM2= dl_nx[2]+TM0;
    TM3= dl_nz[2]+TM1;
    dl_nx[1]= C3*TM2-S3*TM3;
    dl_ny[1]= S3*TM2+C3*TM3;
    dl_nz[1]= CM11*dl_fx[2]+CM12*dl_fy[3]-dl_ny[2];

    R[1][5]= dl_nz[1];

    R[0][5]= S2*dl_nx[1]+C2*dl_ny[1];


}

void Q_matrix1(dq,Q)
    real dq[mm], Q[mm][mm];
{

    real TM0,TM1,TM2,TM3,TM4,TM5,TM6,TM7,TM8,TM9;
    real TM10,TM11,TM12,TM13,TM14,TM15;

    dl_dWx[1]= C2*dq[1];
    dl_dWy[1]= -(S2*dq[1]);

    dl_Wx[2]= S3*C2+C3*S2;
    dl_Wz[2]= C3*C2-S3*S2;

    TM0= S2*dq[2];
    TM1= C2*dq[2];
    TM2= dl_dWy[1]-TM0;
    TM3= dl_dWx[1]+TM1;
    dl_dWx[2]= S3*TM2+C3*TM3;
    dl_dWz[2]= C3*TM2-S3*TM3;

    TM0= C2*A2;
    TM1= Wx2*TM0;
    TM2= S2*CM27;
    TM3= Wy2*TM0;
    TM4= C2*CM27;
    TM5= TM1+TM2;
    TM6= TM3+TM4;
    dl_dVx[2]= S3*TM5-C3*TM6;
    dl_dVy[2]= dl_dWy[1]*A2-S2*dq[1]*A2;
    dl_dVz[2]= C3*TM5+S3*TM6;

    dl_Wx[3]= C4*dl_Wx[2];
    dl_Wz[3]= S4*dl_Wx[2];

    TM0= dl_Wx[2]*dq[3];
    dl_dWx[3]= C4*dl_dWx[2]-S4*TM0;
    dl_dWz[3]= C4*TM0+S4*dl_dWx[2];

    TM0= dl_dWz[2]*D3;
    TM1= dl_Wz[2]*D3;
    TM2= dl_Wx[2]*D3;
    TM3= Wy3*TM2;
    TM4= Wz3*TM1;
    TM5= Wx3*TM2;
    TM6= dl_Wz[2]*CM25;
    TM7= dl_Wx[2]*CM26;
```

```
    TM8= TM0-TM3;
    TM9= TM8+dl_dVx[2];
    TM10= TM4+TM5;
    TM11= TM6+TM7;
    TM12= TM10+TM11;
    TM13= TM12+dl_dVy[2];
    dl_dVx[3]= S4*TM13+C4*TM9;
    dl_dVy[3]= dl_dVz[2]-(Wy3*TM1+dl_dWx[2]*D3);
    dl_dVz[3]= S4*TM9-C4*TM13;

    dl_Wx[4]= S5*dl_Wz[2]+C5*dl_Wx[3];
    dl_Wz[4]= C5*dl_Wz[2]-S5*dl_Wx[3];

    TM0= dl_Wx[3]*dq[4];
    TM1= dl_Wz[2]*dq[4];
    TM2= dl_dWz[2]-TM0;
    TM3= dl_dWx[3]+TM1;
    dl_dWx[4]= S5*TM2+C5*TM3;

    TM0= dl_dWz[3]*D4;
    TM1= dl_Wz[3]*D4;
    TM2= dl_Wx[3]*D4;
    TM3= Wz4*TM1;
    TM4= Wx4*TM2;
    TM5= dl_Wz[3]*CM23;
    TM6= dl_Wx[3]*CM24;
    TM7= Wy4*TM2;
    TM8= dl_Wz[2]*CM24;
    TM9= TM8-TM0;
    TM10= TM3+TM4;
    TM11= TM5+TM6;
    TM12= TM7+TM9;
    TM13= TM12+dl_dVx[3];
    TM14= TM10+TM11;
    TM15= dl_dVy[3]-TM14;
    dl_dVx[4]= S5*TM15+C5*TM13;
    dl_dVy[4]= -(Wy4*TM1+dl_Wz[2]*CM23+dl_dWx[3]*D4+dl_dVz[3]);
    dl_dVz[4]= C5*TM15-S5*TM13;

    dl_dVx[5]= S6*dl_dVy[4]+C6*dl_dVx[4];
    dl_dVy[5]= C6*dl_dVy[4]-S6*dl_dVx[4];

    TM0= C6*dl_Wz[3];
    TM1= S6*dl_Wx[4];
    TM2= C6*dl_Wx[4];
    TM3= S6*dl_Wz[3];
    TM4= dl_Wz[3]*dq[5];
    TM5= dl_Wx[4]*dq[5];
    TM6= dl_dWx[4]-TM4;
    TM7= dl_dWz[3]+TM5;
    TM8= TM0+TM1;
    TM9= TM2-TM3;
    TM10= TM8*Z6;
    TM11= TM9*Z6;
    dl_Fx[5]= M6*dl_dVx[5]+Wz6*TM11+dl_Wz[4]*CM22-(C6*TM7+S6*TM6)*Z6;
    dl_Fy[5]= M6*dl_dVy[5]+dl_Wz[4]*CM21-(C6*TM6-S6*TM7)*Z6-Wz6*TM10;
    dl_Fz[5]= M6*dl_dVz[4]+Wy6*TM10-Wx6*TM11+TM8*CM21-TM9*CM22;

    dl_nx[5]= -(Z6*dl_dVy[5]);
    dl_ny[5]= Z6*dl_dVx[5];

    Q[5][0]= 0.0;

    dl_fx[4]= C6*dl_Fx[5]-S6*dl_Fy[5];
```

```
    dl_fy[4]= S6*dl_Fx[5]+C6*dl_Fy[5];

    dl_nx[4]= C6*dl_nx[5]-S6*dl_ny[5];
    dl_ny[4]= S6*dl_nx[5]+C6*dl_ny[5];

    Q[4][0]= -(dl_ny[4]);

    TM0= dl_Wz[3]*Y4;
    TM1= dl_Wx[3]*Y4;
    dl_Fx[3]= Wy4*TM1+dl_Wz[2]*CM20-dl_dWz[3]*Y4;
    dl_Fy[3]= -(Wz4*TM0+Wx4*TM1+dl_Wz[3]*CM19+dl_Wx[3]*CM20);
    dl_Fz[3]= Wy4*TM0+dl_Wz[2]*CM19+dl_dWx[3]*Y4;

    dl_fx[3]= dl_Fx[3]+C5*dl_fx[4]-S5*dl_Fz[5];
    dl_fy[3]= dl_Fy[3]+S5*dl_fx[4]+C5*dl_Fz[5];
    dl_fz[3]= dl_Fz[3]-dl_fy[4];

    TM0= AD4*dl_Wx[3];
    TM1= CD4*dl_Wz[3];
    dl_Nx[3]= Wy4*TM1+dl_Wz[2]*CM18+AD4*dl_dWx[3];
    dl_Ny[3]= Wz4*TM0-Wx4*TM1+dl_Wz[3]*CM17-dl_Wx[3]*CM18;
    dl_Nz[3]= CD4*dl_dWz[3]-dl_Wz[2]*CM17-Wy4*TM0;

    TM0= CM15*dl_fy[4];
    TM1= CM16*dl_fy[4];
    TM2= dl_nx[4]-TM0;
    dl_nx[3]= dl_Nx[3]+Y4*dl_dVz[3]+C5*TM2-S5*TM1;
    dl_ny[3]= dl_Ny[3]+S5*TM2+C5*TM1;
    dl_nz[3]= dl_Nz[3]-(Y4*dl_dVx[3]+dl_nx[4]+CM15*dl_fx[4]-CM16*dl_Fz[5]);

    Q[3][0]= dl_ny[3];

    dl_fx[2]= C4*dl_fx[3]+S4*dl_fz[3];
    dl_fy[2]= S4*dl_fx[3]-C4*dl_fz[3];

    TM0= CM13*dl_fy[3];
    TM1= CM14*dl_fy[3];
    TM2= dl_nx[3]-TM0;
    TM3= TM1-dl_nz[3];
    dl_nx[2]= C4*TM2-S4*TM3;
    dl_ny[2]= S4*TM2+C4*TM3;
    dl_nz[2]= dl_ny[3]+CM13*dl_fx[3]+CM14*dl_fz[3];

    Q[2][0]= -(dl_ny[2]);

    TM0= CM11*dl_fy[2];
    TM1= CM12*dl_fy[2];
    TM2= dl_nx[2]+TM0;
    TM3= dl_nz[2]+TM1;
    dl_nx[1]= C3*TM2-S3*TM3;
    dl_ny[1]= S3*TM2+C3*TM3;
    dl_nz[1]= CM11*dl_fx[2]+CM12*dl_fy[3]-dl_ny[2];

    Q[1][0]= dl_nz[1];

    Q[0][0]= RD1+S2*dl_nx[1]+C2*dl_ny[1];


}

void Q_matrix2(dq,Q)
    real dq[mm], Q[mm][mm];
{
```

```
    real TM0,TM1,TM2,TM3,TM4,TM5,TM6,TM7,TM8;

    dl_dWx[1]= C2*dq[0];
    dl_dWy[1]= -(S2*dq[0]);

    dl_dWx[2]= S3*dl_dWy[1]+C3*dl_dWx[1];
    dl_dWz[2]= C3*dl_dWy[1]-S3*dl_dWx[1];

    TM0= dq[1]*A2;
    TM1= 2.0*TM0;
    dl_dVx[2]= -(C3*TM1);
    dl_dVy[2]= A2*(dl_dWy[1]-Wx2);
    dl_dVz[2]= S3*TM1;

    TM0= dl_dWx[2]-dq[3];
    dl_dWx[3]= C4*TM0;
    dl_dWz[3]= S4*TM0;

    TM0= dl_dWz[2]*D3;
    TM1= CM26+TM0;
    TM2= TM1+dl_dVx[2];
    dl_dVx[3]= S4*dl_dVy[2]+C4*TM2;
    dl_dVy[3]= CM25-dl_dWx[2]*D3+dl_dVz[2];
    dl_dVz[3]= S4*TM2-C4*dl_dVy[2];

    dl_Wx[4]= -(CM0);
    dl_Wz[4]= S5*S4;

    TM0= dl_dWz[2]+CM1(;
    dl_dWx[4]= S5*TM0+C5*dl_dWx[3];

    TM0= dl_dWz[3]*D4;
    TM1= CM32+TM0;
    TM2= dl_dVx[3]-TM1;
    TM3= CM35+dl_dVy[3];
    dl_dVx[4]= S5*TM3+C5*TM2;
    dl_dVy[4]= -(Wy4*CM8+dl_dWx[3]*D4+dl_dVz[3]);
    dl_dVz[4]= C5*TM3-S5*TM2;

    dl_dVx[5]= S6*dl_dVy[4]+C6*dl_dVx[4];
    dl_dVy[5]= C6*dl_dVy[4]-S6*dl_dVx[4];

    TM0= C6*CM0;
    TM1= S6*CM0;
    TM2= CM0*dq[5];
    TM3= TM2-dl_dWz[3];
    TM4= dl_dWx[4]-CM7;
    TM5= CM6+TM0;
    TM6= TM1-CM5;
    TM7= TM5*Z6;
    TM8= TM6*Z6;
    dl_Fx[5]= M6*dl_dVx[5]+dl_Wz[4]*CM22+(C6*TM3-S6*TM4)*Z6-Wz6*TM7;
    dl_Fy[5]= M6*dl_dVy[5]+Wz6*TM8+dl_Wz[4]*CM21-(S6*TM3+C6*TM4)*Z6;
    dl_Fz[5]= M6*dl_dVz[4]+Wx6*TM7-Wy6*TM8+TM5*CM22-TM6*CM21;

    dl_nx[5]= -(Z6*dl_dVy[5]);
    dl_ny[5]= Z6*dl_dVx[5];

    Q[5][1]= 0.0;

    dl_fx[4]= C6*dl_Fx[5]-S6*dl_Fy[5];
    dl_fy[4]= S6*dl_Fx[5]+C6*dl_Fy[5];

    dl_nx[4]= C6*dl_nx[5]-S6*dl_ny[5];
```

```
    dl_ny[4]= S6*dl_nx[5]+C6*dl_ny[5];

    Q[4][1]= -(dl_ny[4]);

    dl_Fx[3]= -(Wy4*CM4+dl_dWz[3]*Y4);
    dl_Fy[3]= Wx4*CM4-Wz4*CM3+S4*CM20-C4*CM19;
    dl_Fz[3]= Wy4*CM3+dl_dWx[3]*Y4;

    dl_fx[3]= dl_Fx[3]+C5*dl_fx[4]-S5*dl_Fz[5];
    dl_fy[3]= dl_Fy[3]+S5*dl_fx[4]+C5*dl_Fz[5];
    dl_fz[3]= dl_Fz[3]-dl_fy[4];

    dl_Nx[3]= Wy4*CM2+AD4*dl_dWx[3];
    dl_Ny[3]= C4*CM17+S4*CM18-(Wz4*CM1+Wx4*CM2);
    dl_Nz[3]= Wy4*CM1+CD4*dl_dWz[3];

    TM0= CM15*dl_fy[4];
    TM1= CM16*dl_fy[4];
    TM2= dl_nx[4]-TM0;
    dl_nx[3]= dl_Nx[3]+Y4*dl_dVz[3]+C5*TM2-S5*TM1;
    dl_ny[3]= dl_Ny[3]+S5*TM2+C5*TM1;
    dl_nz[3]= dl_Nz[3]-(Y4*dl_dVx[3]+dl_ny[4]+CM15*dl_fx[4]-CM16*dl_Fz[5]);

    Q[3][1]= dl_ny[3];

    dl_fx[2]= C4*dl_fx[3]+S4*dl_fz[3];
    dl_fy[2]= S4*dl_fx[3]-C4*dl_fz[3];

    TM0= CM13*dl_fy[3];
    TM1= CM14*dl_fy[3];
    TM2= dl_nx[3]-TM0;
    TM3= TM1-dl_nz[3];
    dl_nx[2]= C4*TM2-S4*TM3;
    dl_ny[2]= S4*TM2+C4*TM3;
    dl_nz[2]= dl_ny[3]+CM13*dl_fx[3]+CM14*dl_fz[3];

    Q[2][1]= -(dl_ny[2]);

    TM0= CM11*dl_fy[2];
    TM1= CM12*dl_fy[2];
    TM2= dl_nx[2]+TM0;
    TM3= dl_nz[2]+TM1;
    dl_nx[1]= C3*TM2-S3*TM3;
    dl_ny[1]= S3*TM2+C3*TM3;
    dl_nz[1]= CM11*dl_fx[2]+CM12*dl_fy[3]-dl_ny[2];

    Q[1][1]= RD2+dl_nz[1];

    Q[0][1]= S2*dl_nx[1]+C2*dl_ny[1];


}

void Q_matrix3(dq,Q)
    real dq[mm], Q[mm][mm];
{

    real TM0,TM1,TM2,TM3,TM4,TM5,TM6,TM7,TM8;

    dl_dWx[2]= C3*Wy2-S3*Wx2;
    dl_dWz[2]= -(C3*Wx2+S3*Wy2);

    TM0= dl_dWx[2]-dq[3];
    dl_dWx[3]= C4*TM0;
```

```
    dl_dWz[3]= S4*TM0;

    TM0= dl_dWz[2]*D3;
    TM1= CM26+TM0;
    dl_dVx[3]= C4*TM1;
    dl_dVy[3]= CM25-dl_dWx[2]*D3;
    dl_dVz[3]= S4*TM1;

    dl_Wx[4]= -(CM0);
    dl_Wz[4]= S5*S4;

    TM0= dl_dWz[2]+CM10;
    dl_dWx[4]= S5*TM0+C5*dl_dWx[3];

    TM0= dl_dWz[3]*D4;
    TM1= CM32+TM0;
    TM2= dl_dVx[3]-TM1;
    TM3= CM35+dl_dVy[3];
    dl_dVx[4]= S5*TM3+C5*TM2;
    dl_dVy[4]= -(Wy4*CM8+dl_dWx[3]*D4+dl_dVz[3]);
    dl_dVz[4]= C5*TM3-S5*TM2;

    dl_dVx[5]= S6*dl_dVy[4]+C6*dl_dVx[4];
    dl_dVy[5]= C6*dl_dVy[4]-S6*dl_dVx[4];

    TM0= C6*CM0;
    TM1= S6*CM0;
    TM2= CM0*dq[5];
    TM3= TM2-dl_dWz[3];
    TM4= dl_dWx[4]-CM7;
    TM5= CM6+TM0;
    TM6= TM1-CM5;
    TM7= TM5*Z6;
    TM8= TM6*Z6;
    dl_Fx[5]= M6*dl_dVx[5]+dl_Wz[4]*CM22+(C6*TM3-S6*TM4)*Z6-Wz6*TM7;
    dl_Fy[5]= M6*dl_dVy[5]+Wz6*TM8+dl_Wz[4]*CM21-(S6*TM3+C6*TM4)*Z6;
    dl_Fz[5]= M6*dl_dVz[4]+Wx6*TM7-Wy6*TM8+TM5*CM22-TM6*CM21;

    dl_nx[5]= -(Z6*dl_dVy[5]);
    dl_ny[5]= Z6*dl_dVx[5];

    Q[5][2]= 0.0;

    dl_fx[4]= C6*dl_Fx[5]-S6*dl_Fy[5];
    dl_fy[4]= S6*dl_Fx[5]+C6*dl_Fy[5];

    dl_nx[4]= C6*dl_nx[5]-S6*dl_ny[5];
    dl_ny[4]= S6*dl_nx[5]+C6*dl_ny[5];

    Q[4][2]= -(dl_ny[4]);

    dl_Fx[3]= -(Wy4*CM4+dl_dWz[3]*Y4);
    dl_Fy[3]= Wx4*CM4-Wz4*CM3+S4*CM20-C4*CM19;
    dl_Fz[3]= Wy4*CM3+dl_dWx[3]*Y4;

    dl_fx[3]= dl_Fx[3]+C5*dl_fx[4]-S5*dl_Fz[5];
    dl_fy[3]= dl_Fy[3]+S5*dl_fx[4]+C5*dl_Fz[5];
    dl_fz[3]= dl_Fz[3]-dl_fy[4];

    dl_Nx[3]= Wy4*CM2+AD4*dl_dWx[3];
    dl_Ny[3]= C4*CM17+S4*CM18-(Wz4*CM1+Wx4*CM2);
    dl_Nz[3]= Wy4*CM1+CD4*dl_dWz[3];

    TM0= CM15*dl_fy[4];
```

```
    TM1= CM16*dl_fy[4];
    TM2= dl_nx[4]-TM0;
    dl_nx[3]= dl_Nx[3]+Y4*dl_dVz[3]+C5*TM2-S5*TM1;
    dl_ny[3]= dl_Ny[3]+S5*TM2+C5*TM1;
    dl_nz[3]= dl_Nz[3]-(Y4*dl_dVx[3]+dl_ny[4]+CM15*dl_fx[4]-CM16*dl_Fz[5]);

    Q[3][2]= dl_ny[3];

    dl_fx[2]= C4*dl_fx[3]+S4*dl_fz[3];
    dl_fy[2]= S4*dl_fx[3]-C4*dl_fz[3];

    TM0= CM13*dl_fy[3];
    TM1= CM14*dl_fy[3];
    TM2= dl_nx[3]-TM0;
    TM3= TM1-dl_nz[3];
    dl_nx[2]= C4*TM2-S4*TM3;
    dl_ny[2]= S4*TM2+C4*TM3;
    dl_nz[2]= dl_ny[3]+CM13*dl_fx[3]+CM14*dl_fz[3];

    Q[2][2]= RD3-dl_ny[2];

    TM0= CM11*dl_fy[2];
    TM1= CM12*dl_fy[2];
    TM2= dl_nx[2]+TM0;
    TM3= dl_nz[2]+TM1;
    dl_nx[1]= C3*TM2-S3*TM3;
    dl_ny[1]= S3*TM2+C3*TM3;
    dl_nz[1]= CM11*dl_fx[2]+CM12*dl_fy[3]-dl_ny[2];

    Q[1][2]= dl_nz[1];

    Q[0][2]= S2*dl_nx[1]+C2*dl_ny[1];


}

void Q_matrix4(dq,Q)
    real dq[mm], Q[mm][mm];
{

    real TM0,TM1,TM2,TM3,TM4,TM5;

    dl_dWx[3]= C4*Wy3-S4*Wx3;
    dl_dWz[3]= C4*Wx3+S4*Wy3;

    TM0= dl_dWx[3]+dq[4];
    dl_dWx[4]= C5*TM0;

    TM0= dl_dWz[3]*D4;
    TM1= CM24-TM0;
    dl_dVx[4]= C5*TM1;
    dl_dVy[4]= -(CM23+dl_dWx[3]*D4);
    dl_dVz[4]= -(S5*TM1);

    dl_dVx[5]= S6*dl_dVy[4]+C6*dl_dVx[4];
    dl_dVy[5]= C6*dl_dVy[4]-S6*dl_dVx[4];

    TM0= S6*S5;
    TM1= C6*S5;
    TM2= S5*dq[5];
    TM3= TM0*Z6;
    TM4= TM1*Z6;
    TM5= dl_dWz[3]+TM2;
    dl_Fx[5]= M6*dl_dVx[5]+Wz6*TM4+C5*CM22-(C6*TM5+S6*dl_dWx[4])*Z6;
```

```
    dl_Fy[5]= M6*dl_dVy[5]+C5*CM21-(C6*dl_dWx[4]-S6*TM5)*Z6-Wz6*TM3;
    dl_Fz[5]= M6*dl_dVz[4]+Wy6*TM3-Wx6*TM4+TM0*CM21-TM1*CM22;

    dl_nx[5]= -(Z6*dl_dVy[5]);
    dl_ny[5]= Z6*dl_dVx[5];

    Q[5][3]= 0.0;

    dl_fx[4]= C6*dl_Fx[5]-S6*dl_Fy[5];
    dl_fy[4]= S6*dl_Fx[5]+C6*dl_Fy[5];

    dl_nx[4]= C6*dl_nx[5]-S6*dl_ny[5];
    dl_ny[4]= S6*dl_nx[5]+C6*dl_ny[5];

    Q[4][3]= -(dl_ny[4]);

    dl_Fx[3]= CM20-dl_dWz[3]*Y4;
    dl_Fz[3]= CM19+dl_dWx[3]*Y4;

    dl_fx[3]= dl_Fx[3]+C5*dl_fx[4]-S5*dl_Fz[5];
    dl_fy[3]= S5*dl_fx[4]+C5*dl_Fz[5];
    dl_fz[3]= dl_Fz[3]-dl_fy[4];

    dl_Nx[3]= CM18+AD4*dl_dWx[3];
    dl_Nz[3]= CD4*dl_dWz[3]-CM17;

    TM0= CM15*dl_fy[4];
    TM1= CM16*dl_fy[4];
    TM2= dl_nx[4]-TM0;
    dl_nx[3]= dl_Nx[3]+C5*TM2-S5*TM1;
    dl_ny[3]= S5*TM2+C5*TM1;
    dl_nz[3]= dl_Nz[3]-(dl_ny[4]+CM15*dl_fx[4]-CM16*dl_Fz[5]);

    Q[3][3]= RD4+dl_ny[3];

    dl_fx[2]= C4*dl_fx[3]+S4*dl_fz[3];
    dl_fy[2]= S4*dl_fx[3]-C4*dl_fz[3];

    TM0= CM13*dl_fy[3];
    TM1= CM14*dl_fy[3];
    TM2= dl_nx[3]-TM0;
    TM3= TM1-dl_nz[3];
    dl_nx[2]= C4*TM2-S4*TM3;
    dl_ny[2]= S4*TM2+C4*TM3;
    dl_nz[2]= dl_ny[3]+CM13*dl_fx[3]+CM14*dl_fz[3];

    Q[2][3]= -(dl_ny[2]);

    TM0= CM11*dl_fy[2];
    TM1= CM12*dl_fy[2];
    TM2= dl_nx[2]+TM0;
    TM3= dl_nz[2]+TM1;
    dl_nx[1]= C3*TM2-S3*TM3;
    dl_ny[1]= S3*TM2+C3*TM3;
    dl_nz[1]= CM11*dl_fx[2]+CM12*dl_fy[3]-dl_ny[2];

    Q[1][3]= dl_nz[1];

    Q[0][3]= S2*dl_nx[1]+C2*dl_ny[1];


}

void Q_matrix5(dq,C
```

```
    real dq[mm], Q[mm][mm];
{

    real TM0,TM1,TM2,TM3;

    dl_dWx[4]= C5*Wy4-S5*Wx4;

    TM0= S6*Z6;
    TM1= C6*Z6;
    TM2= dl_dWx[4]-dq[5];
    dl_Fx[5]= -(Wz6*TM0+S6*TM2*Z6);
    dl_Fy[5]= -(Wz6*TM1+C6*TM2*Z6);
    dl_Fz[5]= Wx6*TM0+Wy6*TM1+S6*CM22+C6*CM21;

    Q[5][4]= 0.0;

    dl_fx[4]= C6*dl_Fx[5]-S6*dl_Fy[5];
    dl_fy[4]= S6*dl_Fx[5]+C6*dl_Fy[5];

    Q[4][4]= RD5;

    dl_fx[3]= C5*dl_fx[4]-S5*dl_Fz[5];
    dl_fy[3]= S5*dl_fx[4]+C5*dl_Fz[5];

    TM0= CM16*dl_fy[4];
    TM1= CM15*dl_fy[4];
    dl_nx[3]= -(C5*TM1+S5*TM0);
    dl_ny[3]= C5*TM0-S5*TM1;
    dl_nz[3]= CM16*dl_Fz[5]-CM15*dl_fx[4];

    Q[3][4]= dl_ny[3];

    dl_fx[2]= C4*dl_fx[3]-S4*dl_fy[4];
    dl_fy[2]= S4*dl_fx[3]+C4*dl_fy[4];

    TM0= CM13*dl_fy[3];
    TM1= CM14*dl_fy[3];
    TM2= dl_nx[3]-TM0;
    TM3= TM1-dl_nz[3];
    dl_nx[2]= C4*TM2-S4*TM3;
    dl_ny[2]= S4*TM2+C4*TM3;
    dl_nz[2]= dl_ny[3]+CM13*dl_fx[3]-CM14*dl_fy[4];

    Q[2][4]= -(dl_ny[2]);

    TM0= CM11*dl_fy[2];
    TM1= CM12*dl_fy[2];
    TM2= dl_nx[2]+TM0;
    TM3= dl_nz[2]+TM1;
    dl_nx[1]= C3*TM2-S3*TM3;
    dl_ny[1]= S3*TM2+C3*TM3;
    dl_nz[1]= CM11*dl_fx[2]+CM12*dl_fy[3]-dl_ny[2];

    Q[1][4]= dl_nz[1];

    Q[0][4]= S2*dl_nx[1]+C2*dl_ny[1];


}

void Q_matrix6(dq,Q)
    real dq[mm], Q[mm][mm];
{
```

```
    real TM0,TM1,TM2,TM3;

    dl_Fx[5]= CM22-(C6*Wx5+S6*Wy5)*Z6;
    dl_Fy[5]= CM21-(C6*Wy5-S6*Wx5)*Z6;

    Q[5][5]= RD6;

    dl_fx[4]= C6*dl_Fx[5]-S6*dl_Fy[5];
    dl_fy[4]= S6*dl_Fx[5]+C6*dl_Fy[5];

    Q[4][5]= 0.0;

    dl_fx[3]= C5*dl_fx[4];
    dl_fy[3]= S5*dl_fx[4];

    TM0= CM16*dl_fy[4];
    TM1= CM15*dl_fy[4];
    dl_nx[3]= -(C5*TM1+S5*TM0);
    dl_ny[3]= C5*TM0-S5*TM1;
    dl_nz[3]= -(CM15*dl_fx[4]);

    Q[3][5]= dl_ny[3];

    dl_fx[2]= C4*dl_fx[3]-S4*dl_fy[4];
    dl_fy[2]= S4*dl_fx[3]+C4*dl_fy[4];

    TM0= CM13*dl_fy[3];
    TM1= CM14*dl_fy[3];
    TM2= dl_nx[3]-TM0;
    TM3= TM1-dl_nz[3];
    dl_nx[2]= C4*TM2-S4*TM3;
    dl_ny[2]= S4*TM2+C4*TM3;
    dl_nz[2]= dl_ny[3]+CM13*dl_fx[3]-CM14*dl_fy[4];

    Q[2][5]= -(dl_ny[2]);

    TM0= CM11*dl_fy[2];
    TM1= CM12*dl_fy[2];
    TM2= dl_nx[2]+TM0;
    TM3= dl_nz[2]+TM1;
    dl_nx[1]= C3*TM2-S3*TM3;
    dl_ny[1]= S3*TM2+C3*TM3;
    dl_nz[1]= CM11*dl_fx[2]+CM12*dl_fy[3]-dl_ny[2];

    Q[1][5]= dl_nz[1];

    Q[0][5]= S2*dl_nx[1]+C2*dl_ny[1];


}

void P_matrix1(P)
    real P[mm][mm];
{

    real TM0,TM1,TM2,TM3;

    dl_dWx[2]= S3*C2+C3*S2;
    dl_dWz[2]= C3*C2-S3*S2;

    dl_dVy[2]= C2*A2;

    dl_dWx[3]= C4*dl_dWx[2];
    dl_dWz[3]= S4*dl_dWx[2];
```

```
    TM0= dl_dWz[2]*D3;
    dl_dVx[3]= S4*dl_dVy[2]+C4*TM0;
    dl_dVy[3]= -(dl_dWx[2]*D3);
    dl_dVz[3]= S4*TM0-C4*dl_dVy[2];

    dl_dWx[4]= S5*dl_dWz[2]+C5*dl_dWx[3];

    TM0= dl_dWz[3]*D4;
    TM1= dl_dVx[3]-TM0;
    dl_dVx[4]= S5*dl_dVy[3]+C5*TM1;
    dl_dVy[4]= -(dl_dWx[3]*D4+dl_dVz[3]);
    dl_dVz[4]= C5*dl_dVy[3]-S5*TM1;

    dl_dVx[5]= S6*dl_dVy[4]+C6*dl_dVx[4];
    dl_dVy[5]= C6*dl_dVy[4]-S6*dl_dVx[4];

    dl_Fx[5]= M6*dl_dVx[5]-(C6*dl_dWz[3]+S6*dl_dWx[4])*Z6;
    dl_Fy[5]= M6*dl_dVy[5]-(C6*dl_dWx[4]-S6*dl_dWz[3])*Z6;
    dl_Fz[5]= M6*dl_dVz[4];

    dl_nx[5]= -(Z6*dl_dVy[5]);
    dl_ny[5]= Z6*dl_dVx[5];

    P[5][0]= 0.0;

    dl_fx[4]= C6*dl_Fx[5]-S6*dl_Fy[5];
    dl_fy[4]= S6*dl_Fx[5]+C6*dl_Fy[5];

    dl_nx[4]= C6*dl_nx[5]-S6*dl_ny[5];
    dl_ny[4]= S6*dl_nx[5]+C6*dl_ny[5];

    P[4][0]= -(dl_ny[4]);

    dl_Fx[3]= -(dl_dWz[3]*Y4);
    dl_Fz[3]= dl_dWx[3]*Y4;

    dl_fx[3]= dl_Fx[3]+C5*dl_fx[4]-S5*dl_Fz[5];
    dl_fy[3]= S5*dl_fx[4]+C5*dl_Fz[5];
    dl_fz[3]= dl_Fz[3]-dl_fy[4];

    dl_Nx[3]= AD4*dl_dWx[3];
    dl_Nz[3]= CD4*dl_dWz[3];

    TM0= CM15*dl_fy[4];
    TM1= CM16*dl_fy[4];
    TM2= dl_nx[4]-TM0;
    dl_nx[3]= dl_Nx[3]+Y4*dl_dVz[3]+C5*TM2-S5*TM1;
    dl_ny[3]= S5*TM2+C5*TM1;
    dl_nz[3]= dl_Nz[3]-(Y4*dl_dVx[3]+dl_ny[4]+CM15*dl_fx[4]-CM16*dl_Fz[5]);

    P[3][0]= dl_ny[3];

    dl_fx[2]= C4*dl_fx[3]+S4*dl_fz[3];
    dl_fy[2]= S4*dl_fx[3]-C4*dl_fz[3];

    TM0= CM13*dl_fy[3];
    TM1= CM14*dl_fy[3];
    TM2= dl_nx[3]-TM0;
    TM3= TM1-dl_nz[3];
    dl_nx[2]= C4*TM2-S4*TM3;
    dl_ny[2]= S4*TM2+C4*TM3;
    dl_nz[2]= dl_ny[3]+CM13*dl_fx[3]+CM14*dl_fz[3];
```

```
    P[2][0]= -(dl_ny[2]);

    TM0= CM11*dl_fy[2];
    TM1= CM12*dl_fy[2];
    TM2= dl_nx[2]+TM0;
    TM3= dl_nz[2]+TM1;
    dl_nx[1]= C3*TM2-S3*TM3;
    dl_ny[1]= S3*TM2+C3*TM3;
    dl_nz[1]= CM11*dl_fx[2]+CM12*dl_fy[3]-dl_ny[2];

    P[1][0]= dl_nz[1];

    P[0][0]= IA1+S2*dl_nx[1]+C2*dl_ny[1];


}

void P_matrix2(P)
    real P[mm][mm];
{

    real TM0,TM1,TM2,TM3;

    dl_dVx[3]= C4*CM11;
    dl_dVz[3]= S4*CM11;

    dl_dWx[4]= -(CM0);

    TM0= dl_dVx[3]-CM8;
    dl_dVx[4]= S5*CM12+C5*TM0;
    dl_dVy[4]= CM9-dl_dVz[3];
    dl_dVz[4]= C5*CM12-S5*TM0;

    dl_dVx[5]= S6*dl_dVy[4]+C6*dl_dVx[4];
    dl_dVy[5]= C6*dl_dVy[4]-S6*dl_dVx[4];

    dl_Fx[5]= M6*dl_dVx[5]+(S6*CM0-CM5)*Z6;
    dl_Fy[5]= M6*dl_dVy[5]+(CM6+C6*CM0)*Z6;
    dl_Fz[5]= M6*dl_dVz[4];

    dl_nx[5]= -(Z6*dl_dVy[5]);
    dl_ny[5]= Z6*dl_dVx[5];

    P[5][1]= 0.0;

    dl_fx[4]= C6*dl_Fx[5]-S6*dl_Fy[5];
    dl_fy[4]= S6*dl_Fx[5]+C6*dl_Fy[5];

    dl_nx[4]= C6*dl_nx[5]-S6*dl_ny[5];
    dl_ny[4]= S6*dl_nx[5]+C6*dl_ny[5];

    P[4][1]= -(dl_ny[4]);

    dl_Fx[3]= -(CM3);
    dl_Fz[3]= -(CM4);

    dl_fx[3]= C5*dl_fx[4]-S5*dl_Fz[5]-CM3;
    dl_fy[3]= S5*dl_fx[4]+C5*dl_Fz[5];
    dl_fz[3]= -(CM4+dl_fy[4]);

    dl_Nx[3]= -(CM1);

    TM0= CM15*dl_fy[4];
    TM1= CM16*dl_fy[4];
```

```
    TM2= dl_nx[4]-TM0;
    dl_nx[3]= Y4*dl_dVz[3]+C5*TM2-S5*TM1-CM1;
    dl_ny[3]= S5*TM2+C5*TM1;
    dl_nz[3]= CM2-(Y4*dl_dVx[3]+dl_ny[4]+CM15*dl_fx[4]-CM16*dl_Fz[5]);

    P[3][1]= dl_ny[3];

    dl_fx[2]= C4*dl_fx[3]+S4*dl_fz[3];

    TM0= CM13*dl_fy[3];
    TM1= CM14*dl_fy[3];
    TM2= dl_nx[3]-TM0;
    TM3= TM1-dl_nz[3];
    dl_ny[2]= S4*TM2+C4*TM3;

    P[2][1]= -(dl_ny[2]);

    P[1][1]= IA2+CM11*dl_fx[2]+CM12*dl_fy[3]-dl_ny[2];

    P[0][1]=P[1][0];


}

void P_matrix3(P)
    real P[mm][mm];
{

    real TM0,TM1,TM2;

    dl_dWx[4]= -(CM0);

    dl_dVx[4]= -(C5*CM8);
    dl_dVz[4]= S5*CM8;

    dl_dVx[5]= S6*CM9+C6*dl_dVx[4];
    dl_dVy[5]= C6*CM9-S6*dl_dVx[4];

    dl_Fx[5]= M6*dl_dVx[5]+(S6*CM0-CM5)*Z6;
    dl_Fy[5]= M6*dl_dVy[5]+(CM6+C6*CM0)*Z6;
    dl_Fz[5]= M6*dl_dVz[4];

    dl_nx[5]= -(Z6*dl_dVy[5]);
    dl_ny[5]= Z6*dl_dVx[5];

    P[5][2]= 0.0;

    dl_fx[4]= C6*dl_Fx[5]-S6*dl_Fy[5];
    dl_fy[4]= S6*dl_Fx[5]+C6*dl_Fy[5];

    dl_nx[4]= C6*dl_nx[5]-S6*dl_ny[5];
    dl_ny[4]= S6*dl_nx[5]+C6*dl_ny[5];

    P[4][2]= -(dl_ny[4]);

    dl_fy[3]= S5*dl_fx[4]+C5*dl_Fz[5];

    dl_Nx[3]= -(CM1);

    TM0= CM15*dl_fy[4];
    TM1= CM16*dl_fy[4];
    TM2= dl_nx[4]-TM0;
    dl_nx[3]= C5*TM2-S5*TM1-CM1;
    dl_ny[3]= S5*TM2+C5*TM1;
```

```
    dl_nz[3]= CM2-(dl_ny[4]+CM15*dl_fx[4]-CM16*dl_Fz[5]);

    P[3][2]= dl_ny[3];

    P[2][2]= IA3-(S4*(dl_nx[3]-CM13*dl_fy[3])+C4*(CM14*dl_fy[3]-dl_nz[3]));

    P[1][2]=P[2][1];

    P[0][2]=P[2][0];


}

void P_matrix4(P)
    real P[mm][mm];
{


    dl_Fx[5]= -(S6*S5*Z6);
    dl_Fy[5]= -(C6*S5*Z6);

    P[5][3]= 0.0;

    dl_fy[4]= S6*dl_Fx[5]+C6*dl_Fy[5];

    P[4][3]= 0.0;

    P[3][3]= IA4+C5*CM16*dl_fy[4]-S5*CM15*dl_fy[4];

    P[2][3]=P[3][2];

    P[1][3]=P[3][1];

    P[0][3]=P[3][0];


}

void P_matrix5(P)
    real P[mm][mm];
{


    P[5][4]= 0.0;

    P[4][4]= IA5;

    P[3][4]=P[4][3];

    P[2][4]=P[4][2];

    P[1][4]=P[4][1];

    P[0][4]=P[4][0];


}

void P_matrix6(P)
    real P[mm][mm];
{


    P[5][5]= IA6;
```

```
    P[4][5]=P[5][4];

    P[3][5]=P[5][3];

    P[2][5]=P[5][2];

    P[1][5]=P[5][1];

    P[0][5]=P[5][0];


}

void P_matrix(P)
    real P[mm][mm];
{

    P_matrix1(P);
    P_matrix2(P);
    P_matrix3(P);
    P_matrix4(P);
    P_matrix5(P);
    P_matrix6(P);

}

void Q_matrix(dq,Q)
    real dq[mm], Q[mm][mm];
{

    Q_matrix1(dq,Q);
    Q_matrix2(dq,Q);
    Q_matrix3(dq,Q);
    Q_matrix4(dq,Q);
    Q_matrix5(dq,Q);
    Q_matrix6(dq,Q);

}

void R_matrix(dq,R)
    real dq[mm], R[mm][mm];
{

    R_matrix1(dq,R);
    R_matrix2(dq,R);
    R_matrix3(dq,R);
    R_matrix4(dq,R);
    R_matrix5(dq,R);
    R_matrix6(dq,R);

}


/*******************************************

   Additions       : 473
   Substractions   : 451
   Multiplications : 1311

*******************************************/
```