On the Identification of Robot Parameters by the Classic Calibration Algorithms and Error Absorbing Trees

Takeshi Ohtsuki, Toshiko Iguchi, Yasuyuki Murata and Hiroshi Noborio
Department of Engineering Informatics
Osaka Electro-Communication University
Hatsu-Cho 18-8, Neyagawa, Osaka 572-8530, Japan

Abstract

In this paper, we propose a feasible method to construct a virtual manipulator in a 3-D graphics environment, which is equivalent to a real manipulator including its dynamics. For this purpose, we firstly calibrate parameters of robot dynamics by the classic two algorithms. One is to use a few special motions to identify viscous, dynamic and static frictions, masses of links, centers of the masses, mass matrices of the links, and so on. The other is to use the least square method to identify robot parameters from a huge number of general motions. Unfortunately, both classic methods are not practically stable because each motion pattern includes noise and error. To overcome this drawback, we indirectly absorb position differences between experimental true and calculated virtual manipulators by three types of learning trees. Moreover, we directly memorize many neighbor dynamic motions by the same learning trees. As a result, when real and virtual robots are independently supervised by the PD control, their angular errors of rotational joints amount to zero. In addition, even though both robots are independently supervised as slave arms from a master arm by the same sequence of forces in a bilateral control based on the PD control, motion sequences of real and virtual slave arms equal to each other. As a result, a virtual manipulator can be used in a 3-D graphics animation, which is truly replaced of a real manipulator.

1 Introduction

Recently, PC (e.g., Pentium 4.2GHz) and its graphics accelerator (e.g., GForce 3 for DirectX) become powerful, and we can easily program a high quality graphics animation with an image reality, especially, rendering such as lighting, shading, texture mapping and so on by using DirectX ver. 8.0 graphics software. Moreover, in order to get a motion reality, many kinds of models and calibration methods for friction and collision between rigid bodies are recently proposed [1],[2].

In such a background, we focus on how to generate a real sequence of dynamic motions of a mechanical system such as a robotic manipulator. For this purpose, we should calibrate two kinds of robot parameters. One is the geometric parameters, which define homogeneous transformations between successive links. The set of kinematic parameters is estimated by some method as [3]. Another is the dynamic parameters, consisting of the maximum of static friction, dynamic friction and viscous damping friction of each joint, and the mass, center of mass, inertia tensor of each link. In general, the robot dynamics should be calculated totally after the manipulator was constructed. Therefore, the set of dynamic parameters must be estimated by a lot of motion tests for the constructed manipulator.

The estimations have been proposed [3]∼[9]. They are mainly classified into two types: One is to calibrate all dynamic parameters by using a few special motions. The other is to investigate them by the least square method based on a lot of general motions. However, it is difficult for us to achieve special motions exactly. As an example, we will show a static test such that one joint is rotated and the others are stopped. This cannot be achieved by our direct-drive arm because there is few friction in each joint. Also, a few general motions always include noise and error, and consequently the least square method should select a bad set of calibrated parameters. As a result, we cannot acquire the set of exact dynamic parameters by the above two approaches.

In order to absorb such noise and error, two kinds of data structures are considered. One is a neural network, and another is a learning tree. The first application to learn a dynamic system such as a robot will be CMAC [10]. Then, the CMAC was refined by Miller [11], and it is applied for a robot manipulator with two degrees-of-freedom and a planar one-legged hopping robot by Atkeson [12]. In general, because a learning function works implicitly in the neural network, we cannot understand how to calibrate each or some parameters of robot dynamics. On the other
In this paper, we use Suzuki’s tree and simultaneously propose the other two types of learning trees. One is to memorize angular differences of real and virtual manipulators by the octree. Another is to memorize the same differences by the binary tree. In general, if we obtain a robot dynamics, its sequence of motions (its dynamic animation) is described by Runge-Kutta and/or Euler methods based on a calibrated dynamics. If the robot dynamics includes errors, a virtual angle calculated from the robot dynamics differs from a true angle obtained from a robot experiment. To solve this problem, we memorize a lot of angular differences in three kinds of trees. Moreover, we memorize many dynamic motion patterns in three trees, which correspond from the last joint angle, angular velocity, and torque to a present joint angle. This is a direct approach to describe directly a 3-D graphics animation of a practical manipulator in PC.

In this paper, section 2 describes two classic calibration methods to identify robot parameters. Section 3 explains how to generate a sequence of robot motions (dynamic animation). This is done by Runge-Kutta and Euler methods based on a calibrated dynamics. Then, in section 4, we describe three types of trees learning differences between joint angles of real and virtual arms, and also indicate three types of trees learning correspondences from a set of present angle, angular velocity and torque to a next angle of a real arm. Section 5 gives us several comparative results. Finally, in section 6, we will give a few conclusions.

2 A Calibration Method of Robot Parameters

The topics of this section is how to calibrate all parameters of robot dynamics. As an example of the robot, we use a direct-drive (DD) manipulator with two degrees of freedom (Fig.1).

The dynamics of the degrees-of-freedom manipulator is generally expressed by

$$J(\dot{\theta})\ddot{\theta} + C(\dot{\theta}, \theta) + D\dot{\theta} + E(\dot{\theta}, \theta) = \tau.$$ 

On the other hand, we can see the function in a learning tree explicitly. However, in the learning tree, there are many static examples, e.g., learning characters, but few dynamic examples. As an exception, Suzuki proposed a self-organization model for learning patterns and applied it into a navigation of a mobile robot with a CCD vision [14]. In this approach, a mobile robot always compares a present image with a lot of memorized images. Then, the mobile robot selects a wonderful behavior corresponding to a selected image. By using this, a mobile robot avoids an obstacle in a real environment practically. Many images are sorted in a binary tree and the comparison is quickly done by the binary search.

Finally, in section 6, we will give a few conclusions.

θ = (θ₁, θ₂)ᵀ is the vector of joints angles, ˙θ is the vector of angular velocities, ⃗θ is the vector of angular accelerations and τ = (τ₁, τ₂)ᵀ is the vector of joint torques. J(θ) ˙θ is regarded as the inertial force, C(θ, ˙θ) is considered as the centrifugal force and D ˙θ and E( ˙θ, θ) are the viscous and dynamic frictions. This is a horizontal manipulator, and consequently the gravity item is initially neglected.

Moreover, the following equation

$$C(\dot{\theta}, \theta) = d/dt\{J(\theta)\dot{\theta} - 1/2(\partial/\partial\theta)(\dot{\theta}^T J(\theta) \dot{\theta})\}$$

can be calculated by a set of parameters J(θ), D and E( ˙θ, θ).

$$J(\theta) = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix},
D = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix},
E(\dot{\theta}, \theta) = [E_1, E_2]^T$$

$$E_i = f_c |s_i| \dot{s}_i, \ (\dot{s}_i \neq 0) \ or \ E_i = F_i, \ (\dot{s}_i = 0)$$

$$J_{11} = I_1 + m_2 L_2^2 + I_2 + 2m_2 L_1(\dot{s}_2 x C_2 - \dot{s}_2 y S_2)
J_{12} = J_{21} = I_2 + m_2 L_1(\dot{s}_2 x C_2 - \dot{s}_2 y S_2), \ J_{22} = I_2.$$ 

Therefore, robot calibration is to identify dynamic parameters F₁, F₂, D₁, D₂, f₁, f₂, I₁ + m₂ L₁², I₂, m₂ ́s₂x and m₂ ́s₂y. Here, F₁, f₁ and D₁ are the maximum of static friction, dynamic friction and viscous damping friction of i-th joint, respectively. mᵢ, Iᵢ and Lᵢ are the mass, inertia tensor and length of i-th link. ́sᵢ is the center of mass mᵢ. After constructing a robotic manipulator, we cannot measure each parameter one by one.

To calibrate the parameter set, we have two typical methods. One is to use a set of special tests, and another is to use the least square method. The former directly investigates all dynamic parameters from a few special motions. The latter indirectly gathers them from a huge general motions.

In order to identify all the robot parameters, Mayeda et al. [4] initially proposed a set of special tests based on Lagrangian formulation. For identifying coefficients of dynamic equation derived from Lagrangian formulation, Mayeda found three sets of...
and a robot manipulator, we can measure the maximum
of dynamic parameters selected by IVM on LSM is
precisely detected by some sensors, an angular accel-
eration of the joint cannot be exactly detected. For
this reason, we should consider an error of the accel-
eration. To absorb the error in LSM, Kawasaki et al.
proposed a smart method by mixing the instru-
mental variable method (IVM) and LSM [9]. A set of
dynamic parameters selected by IVM on LSM is
better than that obtained by only LSM.

2.1 A Calibration Method by a Few Special
Motions

First of all, when one of two joints starts to move in
a robot manipulator, we can measure the maximum
static frictions $F_1$ and $F_2$ by considering balanced
torques. The detail is as follows:

The manipulator initially stops, i.e., $\theta_1 = \dot{\theta}_1 =
\ddot{\theta}_1 = \theta_2 = \dot{\theta}_2 = \ddot{\theta}_2 = 0$. In this case, since $J(\theta)\dot{\theta}$, $C(\theta, \dot{\theta})$
and $D\dot{\theta}$ vanish, the dynamic equation is simpli-
fied as $E(\theta, \dot{\theta}) = \tau$. After that, when the first joint starts
to rotate clockwise and counter-clockwise orders, we
measure positive and negative torques, respec-
tively. By these static tests, we obtain the maximum
static frictions as follows: $F_1 = \tau_1^+ = \tau_1^-$, and
$F_1 = (\tau_1^+ - \tau_1^-)/2$. Also, by applying the same
tests for the second joint, we obtain the maximum
static friction as follows: $F_2 = \tau_2^+ = \tau_2^-$, and
$F_2 = (\tau_2^+ - \tau_2^-)/2$.

Secondly, we describe a set of tests with constant
angular velocities to measure $D_1$, $D_2$, $f_{c1}$ and $f_{c2}$.
Since angular velocities are constant, their accelera-
tions $\ddot{\theta}_1$ and $\ddot{\theta}_2$ are zero. Therefore, $J(\theta)\ddot{\theta}$ and $C(\theta, \ddot{\theta})$
vanish. For this reason, the dynamic equation is sim-
plicated as $D\dot{\theta} = E(\theta, \dot{\theta}) = \tau$. In addition, the second
joint is completely fixed, i.e., $\theta_2 = \dot{\theta}_2 = \ddot{\theta}_2$. Then, two different velocities $\dot{\theta}_{1a}$ and $\dot{\theta}_{1b}$ occur by two types of
torques $\tau_{1a}$ and $\tau_{1b}$. By solving these equations, we
obtain frictions $D_1$ and $f_{c1}$ by solving two equations
$D_1\dot{\theta}_{1a} + f_{c1}\text{sgn}(\dot{\theta}_{1a}) = \tau_{1a}$ and $D_1\dot{\theta}_{1b} + f_{c1}\text{sgn}(\dot{\theta}_{1b}) = \tau_{1b}$.
Furthermore, by applying the same tests for the
second joint, we can measure frictions $D_2$ and $f_{c2}$ by solving the equations
$D_2\dot{\theta}_{2a} + f_{c2}\text{sgn}(\dot{\theta}_{2a}) = \tau_{2a}$ and
$D_2\dot{\theta}_{2b} + f_{c2}\text{sgn}(\dot{\theta}_{2b}) = \tau_{2b}$.

Finally, we explain a set of accelerated motion tests
in order to measure uncertain parameters $I_1$, $I_2$, $\dot{s}_{2x}$
and $\dot{s}_{2y}$. Because of the simplicity of $J_{22} = I_2$,
we firstly determine it. For this purpose, the first joint
is fixed as the normal figure, i.e., $\theta_1 = \dot{\theta}_1 = \ddot{\theta}_1 = 0$.

In this case, we obtain the dynamic equation
$J_{22}\ddot{\theta}_2 + D_2\dot{\theta}_2 + E_2 = \tau_2$. The parameters $D_2$ and $E_2$ were
already recognized by the above procedures. Thus, if
$\theta_2$, $\dot{\theta}_2$ and $\ddot{\theta}_2$ correspond to $\tau_2$, we calculate $J_{22} = I_2$.
After that, we calculate three complex inertia
tensors. For this purpose, the second joint is fixed as
special figures, i.e., $\dot{\theta}_2 = 0, \pi/2, -\pi/2$ and $\ddot{\theta}_2 = \ddot{\theta}_2 = 0$.
In the special figures, many $\sin$ and $\cos$ terms of
the tensors vanish, and consequently we obtain simpli-
fied equations as follows:
$\tau_1 + m_2L_2^2 \ddot{s}_{2x} = \alpha$, \quad $\tau_1 + m_2L_2^2 \ddot{s}_{2y} = \gamma$, \quad $\tau_1 + m_2L_2^2 = (\alpha + \gamma)/4$, \quad $\tau_2 = (\beta - \alpha - \gamma)/(2L_1)$
and $m_2\dot{s}_{2y} = (\beta - \alpha)/6L_1$. Note that $L_1$ should
be measured in advance.

Furthermore so as to get three constant value $\alpha$, $\beta$
and $\gamma$, $\dot{\theta}_1$, $\ddot{\theta}_1$ and $\ddot{\theta}_1$ of the first joint are obtained
by a torque $\tau_1$ in three kinds of dynamic equations
$J_{11}\ddot{\theta}_1 + D_1\dot{\theta}_1 + E_1 = \tau_1 \ (D_1$ and $E_1$ were already
recognized by the above procedures).

2.2 A Calibration Method of LSM and IVM
from a Huge Number of General
Motions

In this paragraph, we explain how to obtain a set $\sigma$
of dynamic parameters of a robotic manipulator by
LSM and IVM. The dynamic equation is expressed as
$\tau = W(\theta, \dot{\theta}, \ddot{\theta})\sigma$ and $\sigma = (I_1 + m_2L_2^2, m_2S_{2x}, m_2S_{2y}, I_2$,
$D_1, D_2, f_{c1}, f_{c2})^T$. In general, it is difficult for us to
measure $\sigma$ after constructing a robotic manipulator.

Since the equation is linear against $\sigma$, which is de-
signed under Newton-Euler equations, LSM can be
used for identifying $\sigma$ from many motion patterns
at sampling numbers $1 \sim N$. The estimation is de-
noted as $y = A\sigma + \nu$, $y = [\tau(1), \ldots, \tau(N)]^T$, $A =
[W(1), \ldots, W(N)]^T$, $\nu = [\phi(1), \ldots, \phi(N)]^T$. This
is evaluated by minimizing $PI = (y - A\sigma)^T \Omega (y -
A\sigma)$. This function $PI$ is differentiated by $\sigma$ as follows:
$\partial PI/\partial \sigma = 2A^T\Omega A\sigma - 2A^T\Omega y = 0$. By solving this
equation, we obtain an excellent value $\hat{\sigma}$.

If the inverse matrix of $A^T\Omega A$ exists, a better $\hat{\sigma}$
is directly calculated as $\hat{\sigma} = \sigma_{init} + (A^T\Omega A)^{-1}A^T\Omega
\nu$ ($\sigma_{init}$: an initial value of $\sigma$). The LSM, which means
the $\Omega$ is a unit matrix, does not yield the true
values of the unknown parameters because $\nu$ is correla-
tive to $A$. The IVM means $\Omega = A^T\Omega \ A$, $\nu$
in the instrumental variable matrix defined as $\hat{A} =
blockdiag(W(1), \hat{W}(2), \ldots, \hat{W}(N))$ with $\hat{W}(i) = W(\theta(i), \dot{\theta}(i), \ddot{\theta}(i))$.

The sampling time $t = ih$ and $h$ is the sampling
period. $\hat{\theta}$ is denoted as $\theta + \Delta \theta$. $\Delta \theta$ is the mea-
surement noize which is defined as a white noize with zero mean. A better \( \hat{\sigma} \) is calculated as
\[
\hat{\sigma} = \sigma_{\text{init}} + (A^T \Omega A)^{-1} A^T \Omega \nu.
\]
Even if no inverse matrix of \( A^T \Omega A \) exists, another algorithm successively calculates \( \sigma(i) \) from \( \hat{\sigma}(i - 1) \), \( \tau(i - 1) \), \( W(i - 1) \) \( (i = 1, \cdots, N) \). Then, this finally obtains a better \( \hat{\sigma} \) as \( \hat{\sigma}(N) \).

In conclusion, we describe a defective point of these methods which the data contain errors. The former method directly investigates all the dynamic parameters. However if special motions include errors, the parameters also include errors. The latter method indirectly gathers the same parameters from a huge number of general motions. However, if few general motions include errors, LSM converges to a local minimum and consequently does not generate any good parameters. To overcome this problem, we propose many kinds of error absorbing trees.

### 3 Motion Sequence Based on Dynamic Equation and Runge-Kutta Method

In this section, we explain a classic approach to make a sequence of joint angles of a virtual manipulator in a 3-D graphics world. The approach consists of Runge-Kutta and Euler methods based on a calibrated dynamic equation.

1. We initialize \( t = 0 \) and \( \dot{\theta}_i^1 = \ddot{\theta}_i^1 = 0 \). Then, in order to move a virtual manipulator, we give an arbitrary torque \( \tau^1 \).

2. We calculate \( \dot{\theta}_i^1 \) by a calibrated dynamic equation
\[
\dot{\theta}_i^1 = J(\dot{\theta}_i^1)^{-1}(\dot{r}_i - C(\dot{\theta}_i^1, \ddot{\theta}_i^1) - D\ddot{\theta}_i^1 - E((\dot{\theta}_i^1, \ddot{\theta}_i^1))).
\]

3. By Runge-Kutta method, we calculate a next angular velocity \( \dot{\theta}_i^{t+1} \) from a present angular acceleration \( \dot{\theta}_i^t \) as follows: \( f(t, \dot{\theta}_i^t) = \ddot{\theta}_i^t, k_1 = h f(t, \dot{\theta}_i^t), k_2 = h f(t + h/2, \dot{\theta}_i^t + k_1/2), k_3 = h f(t + h/2, \dot{\theta}_i^t + k_2/2), k_4 = h f(t + h, \dot{\theta}_i^t + k_3), k = (k_1 + 2k_2 + 2k_3 + k_4)/6, \dot{\theta}_i^{t+1} = \dot{\theta}_i^t + k. \) In succession, by Euler method, we calculate \( \theta_i^{t+1} \) from \( \dot{\theta}_i^t \) as follows: \( \theta_i^{t+1} = \theta_i^t + \dot{\theta}_i^t h. \)

4. If \( t \geq t_{\text{end}} \) is kept, the algorithm ends. Otherwise, we set \( t = t + h \) and return to step 2.

### 4 Error Absorbing and Motion Memorizing by Trees

In this section, we firstly give a method to absorb an error of real and virtual angles by three kinds of learning trees. A real angle is obtained from an experiment, and its virtual angle is calculated from Runge-Kutta and Euler methods based on a calibrated dynamic equation. A difference between true and false angles is memorized into an external node of each tree. Therefore, by adding a calculated angle and a memorized error, we always obtain an exact sequence of angles of a robotic manipulator. Secondly, we propose a method to memorize a next angle from a set of present angle, angular velocity and torque by three kinds of learning trees. The method does not require any dynamic equation of a robotic manipulator. In both types of trees, after a set of present angle, angular velocity and torque is given, we obtain a next angle exactly. As a result, an operator always watches a precise dynamic animation of a manipulator in a 3-D graphics PC.

#### 4.1 Using Octree

In this paragraph, we explain how to calculate an exact sequence of joint angles from a given sequence of joint torques. For this purpose, we propose two methods. One is to memorize a difference between \( \theta_i^t \) and \( \dot{\theta}_i^t \) by three kinds of trees. \( \theta_i^t \) is virtually calculated by Runge-Kutta and Euler methods based on a calibrated dynamic equation. On the other hand, \( \dot{\theta}_i^t \) is directly detected as an experimental result. In this method, a next virtual angle is calculated by the above methods based on an approximated dynamics, and then its updated angle is calculated by adding the virtual angle and an error memorized in the trees. Another is to memorize a correspondence from present angle, angular velocity and torque to a next angle by the same trees. In both methods, we give a present set of \( \theta_i^{t-1}, \dot{\theta}_i^{t-1} \) and \( \tau_i^{t-1} \) as input data, and consequently obtain a next angle \( \theta_i^t \) as output data. As mentioned above, we use the octree representation to memorize two kinds of correspondences between input and output data [16].

The octree represents a 3-D coordinate system defined by three parameters \( \theta_i, \dot{\theta}_i \) and \( \tau_i \). A root node means a whole space of the coordinate system and its eight descendents means their eight subspaces. Each data allocated into a node are distributed into eight descendents according to \( \theta_i, \dot{\theta}_i \) and \( \tau_i \). Then, if the maximum of error dispersion of data allocated into a descendant is smaller than a threshold \( T \), the descendant is represented as an external node and the average of errors of allocated data is memorized into the external node. Otherwise, the descendant has eight descendents and all data are distributed again.

Finally, if no data is allocated into an external node \( n \), an approximated error should be given in the node \( n \) as follows: In the octree, we can easily design a neighbor finding algorithm to find the closest node with an average error from the node \( n \) (Its quadtree version is addressed in [17]). Therefore, the average error in the closest node is allocated into the node \( n \). By this modification, some of external nodes have average errors whose dispersions are larger than \( T \).

1. (initialization): A whole space is expressed as the root \( n \), which is defined by parameters \( \theta_i, \dot{\theta}_i \) and \( \tau_i \). Then, all data \( (i = 1, \cdots, N) \) are stored into \( n \).
2. If the maximum of error dispersion of data allocated into a node \( n \) is smaller than a threshold \( T \), the node \( n \) is expressed as an external node with the average of differences or next angles.

3. Otherwise, the node \( n \) is expanded to eight subnodes which are newly generated as \( n_i \) \((i = 0, \cdots, 7)\). Eight subnodes correspond to uniformly divided subregions. Then, all data allocated in \( n \) are assigned into eight subnodes \( n_i \) \((i = 0, \cdots, 7)\). Finally, \( n \) is successively replaced by each \( n_i \) and return to step 2.

**Figure 2**: A typical look-up table designed by the octree.

### 4.2 Using Binary Tree

In this paragraph, we use the binary tree to memorize differences between real and virtual manipulators or next joint angles of a real manipulator. The difference against the octree is as follows: A level of the octree corresponds to three levels of the binary tree. In other words, we divide three parameters \( \theta^t, \dot{\theta}^t, \tau^t \) simultaneously at an internal node of the octree, but we divide each of them successively at an internal nodes of the binary tree.

1. (initialization): A whole space is expressed as the root \( n \), which is defined by parameters \( \theta_1, \dot{\theta}_1 \) and \( \tau_1 \). Then, all data \((i = 1, \cdots, N)\) are stored into \( n \).

2. If the maximum of error dispersion of data allocated into a node \( n \) is smaller than a threshold \( T \), the node \( n \) is represented as an external node with the average of differences or next angles.

3. Otherwise, the node \( n \) is expanded to two subnodes which are newly generated as \( n_i \) \((i = 0, 1)\). Two subnodes correspond to uniformly divided two subregions. Then, all data allocated in \( n \) are assigned into two subnodes \( n_i \) \((i = 0, 1)\). Finally, \( n \) is successively replaced by each \( n_i \) and return to step 2.

### 4.3 Using Suzuki Tree

Each of two trees addressed in the previous two paragraphs grows by considering whether the maximum of errors of all data located in a space is smaller than a threshold \( T \) or not. However, if no data or few data is allocated into a space, the judgment is impossible. In addition, we cannot easily get a set of unknown dynamic parameters because a sequence of robot motions is continuous. For this reason, an error of a space is substituted by another error of its closest space. If the substitution frequently occurs, the error threshold \( T \) is meaningless, and an error larger than \( T \) frequently appears.

The following method escapes from such problems. As long as all data are inserted continuously, the binary tree grows successively.

1. (initialization): The root node is denoted as a node \( n \), and a number \( i \) is denoted by 1, and then a data whose number \( i \) is stored into the node \( n \). Then, it is evaluated by the equation \( H(data_i) = K_1 \cdot \theta_i + K_2 \cdot \dot{\theta}_i + K_3 \cdot \tau_i \) \((K_1 = K_2 = K_3 = 1)\).

2. If \( i \) is over \( N \), the algorithm ends. Otherwise, increment \( i \). Then, a data whose number is \( i \) is stored into \( n \) with \( H(data_j) \).

3. \( H(data_j) \) of \( n \) is retrieved. Then, if \(| H(data_i) - H(data_j) | \leq T \), the average of differences or next angles located in \( n \) is calculated and then memorized into the node \( n \). Then, return to step 2. Otherwise, if \( H(data_i) \leq H(data_j) \), \( H(data_i) \) is stored into the left subnode \( n_{left} \) of \( n \) and simultaneously \( H(data_j) \) is stored into the right subnode \( n_{right} \) of \( n \), otherwise, \( H(data_i) \) is stored into the right subnode \( n_{right} \) of \( n \) and simultaneously \( H(data_j) \) is stored into the left subnode \( n_{left} \) of \( n \).

4. If a stored node \( n_{left} \) or \( n_{right} \) is newly generated as an external node, return to step 2. Otherwise, replace \( n \) of the stored node \( n_{left} \) or \( n_{right} \), and return to step 3.

### 5 Comparative Results

In this section, we prepare two kinds of experimental results to compare several types of identifications.
One is to evaluate them in the PD control, and the other is to evaluate them in a bilateral control based on the PD control. We firstly obtain a sequence of angles $\theta_i$ ($i = 1, \cdots, N$) and that of angles $\theta'_i$ via joint encoders of the DD manipulator (17bit: the resolution is denoted as $360/2^{17} = 0.002747$ degree). Each angle has a zero mean white noise. To eliminate the noise, we secondly use the Kalman Filter.

5.1 On the Angular Errors in Eight Methods

In this section, we compare eight methods such as Maeda’s method (type a.), Maeda’s and Kawasaki’s methods (type b.), two methods plus error absorbing octree (type c.), two methods plus error absorbing binary tree (type d.), two methods plus error absorbing Suzuki tree (type e.), motion memorizing octree (type f.), motion memorizing binary tree (type g.), motion memorizing Suzuki tree (type h.).

Kawasaki’s LSM starts from initial parameters calculated from Maeda’s method. After real and virtual manipulators are synchronously controlled by a sequence of torques, both sequences of angular positions are compared. Our purpose is to compare differences of angular positions of real and virtual manipulators affected by the same torque sequence from the same initial state. The differences are compared per 33[ms] because a human operator watches dynamic motions at the video-frame rate. Needless to say, compared sequences are obviously different from learned sequences for calibrating the dynamics of a robotic manipulator. The number of learned sequences is 500000 and the number of compared sequences is 10000. Table 1 shows us the average angular differences (degrees) in eight methods a., b., c., d., e.,
Table 1: The average error (degree) between angular positions generated by real and virtual arms in a practical and a graphics world.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>a.</th>
<th>b.</th>
<th>c.</th>
<th>d.</th>
<th>e.</th>
<th>f.</th>
<th>g.</th>
<th>h.</th>
</tr>
</thead>
<tbody>
<tr>
<td>First joint</td>
<td>1.202</td>
<td>1.183</td>
<td>0.674</td>
<td>0.686</td>
<td>0.850</td>
<td>0.029</td>
<td>0.029</td>
<td>1.306</td>
</tr>
<tr>
<td>Second joint</td>
<td>1.742</td>
<td>1.631</td>
<td>1.634</td>
<td>1.009</td>
<td>2.459</td>
<td>0.049</td>
<td>0.052</td>
<td>1.008</td>
</tr>
</tbody>
</table>

f., g. and h. Also in Tables 2 and 3, we describe memory storage, tree depth, average, maximum and minimum numbers of data in each external node as a threshold $T$ is changed in three methods c., d. and e. including a calibrated dynamics, respectively. In succession, in Tables 4 and 5, we describe memory storage, tree depth, average, maximum and minimum numbers of data in each external node as the threshold $T$ is changed in three methods f., g. and h. without any calibrated dynamics. Furthermore in Figures 5, 6, 7 and 8, we describe how memory storage and angle precision are changed as $T$ is changed.

As shown in these tables and figures, concerning to angle precision, the memorizing octree without dynamics is the best, the memorizing binary tree without dynamics is the better, and Suzuki trees are not so good. The reason is that our evaluation function for expanding a node in Suzuki trees is not well defined. As contrasted with this, concerning to memory storage, Suzuki trees are the best, the binary tree memorizing dynamics and the octree memorizing dynamics are better. As a result, we recommend the octree memorizing dynamics.

Finally, neglecting a calibrated dynamics is always better than using this. This reason will be as follows: scale and timing of joint angle, angular velocity, acceleration and torque between dynamics calculating and error memorizing should be carefully adjusted. It is difficult for us to mix two different procedures. The memorizing a robot dynamics does not require the adjustment. For this reason, a simple algorithm without considering robot dynamics is better than another complex one with regarding the dynamics.

Table 2: If a threshold $T$ is fixed as 0.001 in three methods c., d., e., many values evaluating three trees are described.

<table>
<thead>
<tr>
<th>Method</th>
<th>Octree</th>
<th>Binary Tree</th>
<th>Suzuki Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory storage (MB)</td>
<td>240.838</td>
<td>242.741</td>
<td>200.356</td>
</tr>
<tr>
<td>Tree depth</td>
<td>28</td>
<td>23</td>
<td>84</td>
</tr>
<tr>
<td>Average number of data at one external node</td>
<td>0.264</td>
<td>0.262</td>
<td>0.605</td>
</tr>
<tr>
<td>Maximum number</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Minimum number</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Average time for data retrieving (sec)</td>
<td>0.000872</td>
<td>0.000877</td>
<td>0.000897</td>
</tr>
</tbody>
</table>

5.2 A Usefulness of the Best Model in Bilateral Control

In this paragraph, we ascertain a usefulness of the wonderful virtual manipulator in a bilateral control based on the PD one. A six degrees-of-freedom manipulator Joyarm (Mitsui Zosen Co.) is used as the master arm, and the direct-drive manipulator (Shinmeiwa Co.) calibrated by our proposed method is used as the slave arm. For the comparison, a real DD arm and its virtual DD arm are simultaneously supervised by the Joyarm on a bilateral control based on the PD control with the same sequence of forces (Fig.9(a),(b)). The force sequence is reserved for the master manipulator Joyarm in advance.

Figure 9: (a) A photo of our bilateral control between the Joyarm and the real DD arm. (b) A photo of our bilateral control between the Joyarm and the virtual DD arm.

The Fig.10(a),(b) shows us the consistency between motions of real and virtual arms. As a result, we can practically use a virtual manipulator in replace of its real manipulator in a graphics world for several kinds of controls.

Figure 10: (a) A sequence of motions supervised by the Joyarm and its sequence of motions achieved by the real DD arm. (b) A sequence of motions supervised by the Joyarm and its sequence of motions achieved by the virtual DD arm.

6 Conclusions

In this research, we focus on a reality of dynamic animation of a robotic manipulator which runs in a 3-D graphics environment. For this purpose, we firstly calibrate a set of dynamic parameters of a robotic manipulator. However, it is not enough. A sequence
of angular positions of a virtual arm extremely differs from that of a real arm. To overcome this, we absorbed errors of positions of real and virtual arms.

In general, in order to make a sequence of angular positions, we calculate a next angle from present angle, velocity and torque by Runge-Kutta and Euler methods based on a calibrated dynamics. Therefore, we have three data for input and have one data for output. To control relationships from three input data to one output data, we use the octree representation. Furthermore, to save memory storage of the learning tree, we replace the octree by the binary tree. Then, we compare our two methods with a classic smart method to learn errors by another binary tree. Finally, we learned an enormous number of neighbor motion patterns by three types of trees without considering the dynamic equation.

As a result, concerning to position precision and memory storage, we can see that the octree or binary tree memorizing the dynamic patterns without calculating an approximated dynamics is the best way.

Acknowledgments

This research is supported in part by 2002 Grants-aid for Scientific Research from the Ministry of Education, Science and Culture, Japan (No.14550247).

References


