Deep-auto-encoder based robot’s state mapping from task with task redundancy and fast global optimization involving task constraints

Shintaro Noda, Shunichi Nozawa, Yohei Kakiuchi, Kei Okada and Masayuki Inaba

Abstract—Although motion planning problems for robots are sometimes expressed as non-linear optimization problems with non-linear equality/inequality constraints due to the complexity of robot’s kinematics and dynamics, it is difficult to globally solve non-linear optimization in real time without good guess of starting point for searching. For solving these non-linear optimization problems quickly and globally, it would be beneficial to prepare a database of feasible motions in advance and to seek a desired motion according to the task objective. However, such database of robot’s motions with high degree of freedoms must be very large because of the curse of dimensionality. Besides, there is a problem of redundancy parametrization. When there are multiple robot’s states which satisfy the same task objective, and when the task redundancy has a non-linear relationship between robot’s state (e.g. the relationship between 7-dof robot’s joint angles and end-effector coordinate), it is unclear how to parametrize the task redundancy in database. In this paper, we express the map from task and task redundancy to robot’s state (Task-state map: TSM) by using deep auto encoder which has abilities of dimensional compression and auto parametrization of complex task redundant parameters, and we also propose an algorithm which quickly solves non-linear optimization in small TSM task redundancy space. Finally, we evaluate our algorithm by solving two non-linear optimizations: 7-dof arm kinematics learning and joint torques optimization involving redundant degree of freedom, and 4-dof arm reaching trajectories learning and jerk integration optimization.

I. INTRODUCTION

For solving motion planning problems of multi-link robots generally, non-linear optimization technique is a promising approach and a lot of previous works expressed these motion planning problems as non-linear optimization problems [1, 2, 3, 4]. However, these objective functions and equality/inequality constraints of motions are sometimes highly non-linear due to the complexity of robot’s kinematics and dynamics. Hence, non-linear optimization approaches are sometimes computationally expensive and are difficult to be solved in real time.

One simple solution for accelerating these motion planning problems are preparing a database of feasible motions in advance. For example, there is a previous work [5] which expressed the map from task to robot’s state by using the weighted sum of multi-dimensional B-spline, and which optimized the weights for satisfying constraints of feasible motions. This task-state map (TSM) was received user-defined task parameters such as reaching target coordinates, and directly calculated feasible robot’s states such as reaching trajectories.

1. Task-State map (TSM)

![Task-state map (TSM)](image)

Fig. 1. Task-state map (TSM) receives task parameters (e.g. end-effector coordinate) and task redundancy parameters (e.g. 7-dof arm’s redundant degree of freedom), and calculates robot’s state parameters (e.g. joint angles) which satisfy task constraints. By fixing task parameters and searching task redundancy parameters in TSM, fast optimization of robot’s state is achieved.

However, this database approach has a fatal problem of the curse of dimensionality. For expressing the task space by splitting the space equally per each dimension, the necessary parameters increase exponentially. For example, when expressing the map from reaching target coordinate with 3-dimensional position and 3-dimensional attitude (total 6-dimension) to 6-dof joint trajectory with 8 control points (total $6 \times 8 = 48$-dimension), and when equally splitting the 6-dimensional task space into 10 blocks per 1-dimension, $48 \times 10^6$ parameters are necessary. Such large-scale optimization problems would be extremely difficult to be solved. Besides, there is a problem of redundancy parametrization. When there are multiple robot’s states which satisfy the same task objective, and when the task redundancy has a non-linear relationship between robot’s state (e.g. the relationship between 7-dof robot’s joint angles and end-effector coordinate), it is unclear how to parametrize the task redundancy in database.

In this paper, we propose a deep-auto-encoder based TSM structure, and an algorithm to optimize motion by using...
TSM (Fig.1). Our TSM has two advantages in comparison with database approach: dimensional compression and auto parametrization of complex task redundancy parameters. Recently, deep neural network [6, 7] gathers attention because of its power of expression for large-scale data, and a few applications for robots start to appear [8]. Especially, we focus on the auto encoder [9], and its characteristic of dimensional compression and auto parametrization of learning data. Besides, we propose an algorithm for fast global optimization by utilizing TSM. Our algorithm is an global optimization process in TSM task space, and involves local planner for absorbing learning error. Finally, we evaluate our TSM and optimization algorithm in two aspects of computational speed and objective value through two experiments of kinematics learning and trajectory learning.

II. STRUCTURE OF DEEP-AUTO-ENCODER BASED TASK-STATE MAP AND OPTIMIZATION ALGORITHM UTILIZING TASK-STATE MAP

A. Structure of deep-auto-encoder based Task-State Map

![Deep-auto-encoder based task-state map](image)

Fig. 2. Deep-auto-encoder based task-state map with two input layers: Robot’s state input in bottom layer and task input in middle layer. Besides, middle layer also has task redundancy parameters which is automatically parameterized from teacher robot’s state values.

Deep auto encoder [9] is a structure of neural network which has the same values in input bottom layer and output top layer, and the middle layer has smaller neurons than input/output layer. Hence, when the input/output values are correctly learned, deep auto encoder expresses the input/output values with a different and small parameters.

Fig.2 shows the rough structure of deep auto encoder (surrounded by green frame) and TSM (surrounded by red frame). Fig.2 is slightly different from simple auto encoder in that it has two input layers: Robot’s state input in bottom layer and task input in middle layer. Besides, middle layer also has task redundancy parameters which is automatically parametrized from robot’s state values of teacher database in the same way as deep auto encoder. This task redundancy values are normalized by using sigmoid function for defining the domain of redundancy values from 0 to 1 and for effectively searching in task-redundancy space. Hence, when the input/output values are correctly learned in this network, TSM becomes a map which receives task parameters and task redundancy parameters within [0, 1], and directly calculates robot’s state.

B. Optimization algorithm utilizing Task-State Map

TSM is a map which receives task and task redundancy parameters and directly calculates robot’s state as described in Sec.II-A. Hence, TSM could be defined as follows:

\[
s = TSM(i, k)
\]  

(1)

In Eq.1, \(s\) is robot’s state parameters such as joint angles, \(i\) is a task redundancy parameters, and \(k\) is task parameters such as end-effector coordinate. By using this TSM function, the problem of searching robot’s state \(s\) which satisfies task constraints \(k\) could be transformed as the problem of searching task redundancy parameters \(i\) in TSM as follows:

\[
\begin{align*}
\min_i & \quad f(s) \\
\text{s.t.} & \quad s = TSM(i, k) \\
& \quad g(s) = 0 \\
& \quad h(s) < 0
\end{align*}
\]  

(2)

In Eq.2, \(f\) is an objective function, \(g\) is an equality constraints, and \(h\) is an inequality constraints. Because task redundancy \(i\) has smaller dimension than robot’s state \(s\), and because the TSM mapping expresses all or part of the equality task constraints, this optimization is expected to be solved faster than the original problem of searching in \(s\) space with task constraints.

Besides, it is important to combine a local planner which searches within a neighborhood of each \(s\) point because the learned redundancy parameters must have learning error between the desired teacher data. This error could be critical especially for the equality constraints. Hence, we defined the local planner \(TSM_{local}\) which optimizes robot’s state \(s + \Delta s\) around \(s = TSM(i, k)\) and which satisfies equality/inequality constraints as follows:

\[
TSM_{local} := \min_{\Delta s} \quad f(s + \Delta s) \\
\text{s.t.} \quad \Delta s \in S_e \\
& \quad g(s + \Delta s) = 0 \\
& \quad h(s + \Delta s) < 0
\]  

(3)

\(S_e\) is a set of \(\Delta s\) within a neighborhood of 0 vector (e.g. absolute value for each element of \(\Delta s\) is smaller than a small parameter \(c_e\)). When the learning error is critical, TSM function in Eq.2 should be replaced with \(TSM_{local}\) function described as Eq.3.
C. Problem settings of robot’s kinematics

In this subsection, we explain about the robot’s kinematics model which is used in the following experiments. We used the 7-dof manipulator model of hrp2jsk [10] which is modified based on hrp2 robot [11].

Joint configuration:

\[
\begin{align*}
q_0,1,2 & \quad q_3 & \quad q_{4,5,6} \\
\end{align*}
\]

EF coordinates:
\[
(p_x, p_y, p_z, r_x, r_y, r_z)
\]

Fig. 3. 7-dof manipulator configuration of hrp2jsk [10]

Fig.3 shows the configuration of hrp2jsk manipulator. This arm has 3-dof rotational joints in shoulder link \(q_{0,1,2}\), 1 dof in elbow link \(q_3\), and 3 dof in wrist link \(q_{4,5,6}\). End-effector position \(p_{x,y,z}\) and attitude \(r_{x,y,z}\) are calculated by using forward-kinematics function \(FK\) from joint angle values \(q_{0,1,2,3,4,5,6}\) as follows:

\[
\begin{align*}
q &= [q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6]^T \quad (4) \\
p &= [p_x \ p_y \ p_z]^T \quad (5) \\
r &= [r_x \ r_y \ r_z]^T \quad (6) \\
[p^T \ r^T]^T &= FK(q) \quad (7)
\end{align*}
\]

Besides, inverse-kinematics problem which calculates joint angles \(q\) from end-effector coordinate \([p^T \ r^T]^T\) is defined as follows:

\[
\begin{align*}
\min_q & \quad ||p_d - p||^2 + ||r_d - r||^2 \\
\text{s.t.} & \quad [p^T \ r^T]^T = FK(q)
\end{align*}
\]

Especially, when the joint parameter \(q\) has redundant degree of freedom, it is impossible to analytically solve this inverse-kinematics problem and non-linear optimization technique is often used [12].

III. LEARNING OF INVERSE KINEMATICS WITH TASK REDUNDANCY

Because of its nonlinearity, just learning the relationship between robot’s state and end-effector coordinate (or kinematics problem) is also effective to accelerate computation, and some previous works had this approach [13, 14]. On the other hand, our approach is powerful in that the redundant degrees of freedom of task are automatically parametrized. This feature makes it possible to quickly re-optimize in the redundant parameters space. In this section, we learned 7-dof manipulator’s kinematical relationship and evaluated the optimization algorithm which searches in task redundancy space.

A. Learning settings

All cells of each layer connects to the nearest layer’s all cells through sigmoid function \(\frac{1}{1 + e^{-x}}\). Besides, we used the mean square error (MSE) as the loss function between network output and teacher data. The number of cells for each layer was as follows (Table I):

<table>
<thead>
<tr>
<th>Layer</th>
<th>state</th>
<th>layer1</th>
<th>layer2</th>
<th>layer3</th>
<th>layer4</th>
<th>layer5</th>
<th>task_r</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>200</td>
<td>160</td>
<td>120</td>
<td>80</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>1+6=7</td>
<td>200</td>
<td>160</td>
<td>120</td>
<td>80</td>
<td>40</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

* task_r is the task redundant parameter, layerN is hidden layer.

Because all cells of each layer fully connects to the nearest layer’s cells, total number of connection weights of TSM network is calculated from Table I as 65680 (= \(7 \times 200 \times 200 \times 160 \times 160 \times 120 \times 120 \times 80 \times 80 \times 40 \times 40 \times 7\)). The data base with this scale could store 5052.3 (= 65680/13) pairs of 7-dof joint angles and 6-dof end-effector coordinates. When equally splitting the range of motions for each joint, only 5052.3/7 = 3.4 samples per each joint are possible. For storing 10 samples per each joint, the number of parameters is 1000000 which is 152.3 times larger than that of TSM.

![Learning transition of mean-square-error of joint angels](image.png)

Fig. 5. Learning transition of mean-square-error of joint angels [\(\text{rad}^2\)]

For generating the training data set of inverse-kinematics learning, we equally split the range of motions of all joint angles into 10 blocks (total \(10^7\) combinations) and we calculated the end-effector coordinates by using forward-kinematics function. We used these joint angles as the output teacher data and auto-encoder input data. Besides, we used the corresponding end-effector coordinate as second input data in middle layer. Finally, we randomly shuffled the order of these data. Without shuffling, learning was not converged.

As the learning algorithm, we used Nesterov’s accelerated gradient [15] implemented in open-source deep learning framework: Caffe [16]. Although this learning was also succeeded by using Stochastic Gradient Descent (SGD) algorithm, sometimes SGD ended with a local solution that the redundancy parameter was always constant value (0 or 1). The maximum number of iterations was \(2 \times 10^7\). While the
1) Task1: Reaching postures for high place

2) Task2: Reaching postures for low place

Fig. 4. Reaching postures with parametrized redundancy. Each horizontal line has different target end-effector coordinate, and 5 images in the same line shows postures with the same target end-effector coordinates and with different redundancy parameters $\in [0, 1]$.

first half $1 \times 10^7$ iterations, the learning rate started from 0.05 and multiplied 0.95 at intervals of $1 \times 10^6$ calculations. After that, while the second half $1 \times 10^7$ iterations, the learning rate was fixed as 0.1.

B. Learning result

Total computation took 5.1 days by using Intel Core i7-4790 CPU, 3.60GHz. Fig.5 shows the transition of mean square error (MSE) while learning (the unit equals to radian$^2$), and the MSE seems to be successfully converged. Fig.4 shows the robot’s state learned in TSM. Each row has different target end-effector coordinate and each column has different redundancy parameter. Hence, multi robot’s state which satisfies the end-effector coordinate constraints are confirmed to be parametrized with the same task parameter and different redundancy parameters.

C. Test: Inverse kinematics

For evaluating the learning error of TSM, we solved the inverse-kinematics problem in TSM.

1) settings:

$$\begin{align*}
\min_i & \quad \|p - p_d\|^2 + \|r - r_d\|^2 \\
\text{s.t.} & \quad q = \text{TSM}_{ik}(i, p_d, r_d) \\
& \quad [p^T \ r^T]^T = FK(q)
\end{align*}$$

Eq.9 is an implementation of Eq.2 for solving the inverse-kinematics problem (Eq.8) by using TSM. The objective function $f$ is the distance between the desired end-effector coordinate and the coordinate which is calculated by using forward-kinematics function from joint angle vector $q = (\text{robot’s state } s)$. We solved this formulation by using open-source non-linear optimization library: NLOPT [17], and its implementation of DIRECT algorithm [18]. Now, we did not use local planner $\text{TSM}_{local}$ described as Eq.3.

2) result: We solved Eq.9 for $3^7 = 2187$ times and calculated the difference between the desired end-effector coordinates. As the target end-effector coordinates $[p_d^T \ r_d^T]^T$, we sampled random joint angles and calculated forward kinematics function from the sampled joint angles. Therefore, Eq.9 always has zero objective for this test. Fig.6 shows the histograms for all differences of position and attitude. These histograms are more similar to Laplace distribution $(\frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2\pi}})$ than Gaussian distribution $(\frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2\pi}})$.

Average computation time is 101.6 milliseconds. Each variance ($2\sigma$) is as follows (Table.II):

<table>
<thead>
<tr>
<th>$p_x$ [mm]</th>
<th>$p_y$ [mm]</th>
<th>$p_z$ [mm]</th>
<th>$r_x$ [deg]</th>
<th>$r_y$ [deg]</th>
<th>$r_z$ [deg]</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.3</td>
<td>18.1</td>
<td>15.9</td>
<td>12.8</td>
<td>9.5</td>
<td>10.1</td>
</tr>
</tbody>
</table>

Because Laplace distribution includes 95% data in $\pm 3\phi$ region, almost all of the differences are included in $\pm 30[\text{mm}]$ position and $\pm 15[\text{deg}]$ attitude. Although this error is not so small that it is possible to directly apply to accurate manipulation tasks, it is enough as the starting point of searching by TSM local planner (Eq.3).

D. Test: Torque optimal posture search

For evaluating the computational time and the objective value in comparison with a simple non-linear optimization approaches, we solved the problem of searching posture with minimal joint torques and with end-effector coordinate constraints.
Fig. 6. Histograms of differences between desired end-effector coordinates by using TSM inverse-kinematics optimization without local planner

1) settings:

\[
\begin{align*}
\min_{\tau} & \quad \|\tau\|^2 \\
\text{s.t.} & \quad q = TSM_{ik}(i, p_d, r_d) \\
& \quad [p^T r^T]^T = FK(q) \\
& \quad \tau = ID(q) \\
& \quad \forall i; \quad \|p_i - p_{d,i}\| < \epsilon_{p,i} \\
& \quad \forall i; \quad \|r_i - r_{d,i}\| < \epsilon_{r,i}
\end{align*}
\]

Eq.10 is an implementation of Eq.2 for searching the posture with the minimal joint torques and with the constraints of end-effector coordinate. The robot’s state parameter \( k \) is joint angle vector \( q \). The task parameter \( k \) is end-effector coordinate \( p, r \). The objective function \( f \) is the sum of square norm of joint torque vector. \( ID \) is the inverse-dynamics function which calculates the joint torque vector \( \tau \) from joint angle vector \( q \) without external contact force or moment. \( p_i, r_i \) and \( p_{d,i}, r_{d,i} \) are the \( i \)-th elements of \( p, r \) and \( p_d, r_d \). The difference between each element of \( p, r \) and its desired value \( p_d, r_d \) is limited less than its corresponding element of \( \epsilon_p, \epsilon_r \). Now, we used \( \epsilon_p = [50mm, 50mm, 50mm]^T \) and \( \epsilon_r = [30deg, 30deg, 30deg]^T \).

\[
TSM_{ik} := \min_{\Delta q} \|\tau\|^2 \\
\text{s.t.} \quad \forall i; \quad \|\Delta q_i\| < \epsilon_{q,i} \\
\quad [p^T r^T]^T = FK(q + \Delta q) \\
\quad \tau = ID(q + \Delta q) \\
\quad \forall i; \quad \|p_i - p_{d,i}\| < \epsilon_{p,i} \\
\quad \forall i; \quad \|r_i - r_{d,i}\| < \epsilon_{r,i}
\]

\( \Delta q_i \) is the \( i \)-th element of \( \Delta q \), and its value is limited less than \( \epsilon_{q,i} \). Now, we used \( \epsilon_{q,i} = 10deg \) for each \( i \).

2) results: we compared 4 types of algorithms. Each of them calculates a posture which satisfies the constraints of end-effector coordinates and outputs the sum of square of joint torque vector. Those descriptions are as follows:

i) IK: Solve inverse-kinematics problem (Eq.8) from random starting points by using gradient-based algorithm. After that, calculate joint torque vector by using the inverse-dynamics function.

ii) TSM: Solve Eq.10, optimization in TSM task-redundancy space without local planner Eq.11.

iii) TSM*: Solve Eq.10, optimization in TSM task-redundancy space with local planner Eq.11.

iv) Min: Search the posture which has the minimal joint torque vector norm and satisfies end-effector constraints in robot’s state space by using DIRECT algorithm. This output value is expected to be the global minimal.

<table>
<thead>
<tr>
<th>Table III</th>
<th>Optimal Torque in TSM vs Global Optima</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IK/Min</td>
</tr>
<tr>
<td>Average</td>
<td>2.31</td>
</tr>
<tr>
<td>Variance</td>
<td>2.25</td>
</tr>
</tbody>
</table>

We randomly sampled 1000 feasible end-effector coordinates as task constraints. Table.III shows the average and variance of the objective values for all 4 algorithms. The objective average of ‘IK’ is 2.31 times larger than that of ‘Min’ value, ‘TSM’ is 1.70 times, and ‘TSM*’ is 1.35 times larger. Comparing ‘IK’ and ‘TSM’, ‘TSM’ is 0.81 times smaller.

Besides, we used Eq.11 as the local planning problem for absorbing learning error. For solving this local problem, we used COBYLA algorithm [19] implemented in NLOPT [17].
Table IV shows the computational times for all 4 algorithms. The average computational time of ‘TSM’ is about 1000 times faster than that of ‘Min’ algorithm, and ‘TSMi’ is about 100 times faster. Hence, we confirmed that ‘TSM’ and ‘TSMi’ could calculate the approximation of minimal value quickly. Besides, because we implemented the kinematics and dynamics computation by using a lispl language [20], it would be possible to furthermore accelerate this computation by re-implementing those functions with fast compiled language such C language.

<table>
<thead>
<tr>
<th>TABLE IV</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Computational Time for Each Algorithm</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Time [sec]</td>
</tr>
</tbody>
</table>

IV. APPLICATION TO REACHING TRAJECTORY LEARNING AND OPTIMIZATION

Because the computation of robot’s motion trajectory is an expansion of the computation of one robot’s posture in the time direction, it takes much more time to be solved. Hence, the acceleration of trajectory computation is important. Hence, the expression of motion trajectory, summation of B-spline function is sometimes used [1, 3, 21] because its convex hull properties is useful for expressing the infinite constraints of joint angle and its time derivative limitations.

In this section, we had an experiment on trajectory learning. We used the weights of B-spline summation as the robot’s state, and we used the initial posture and last end-effector position as the task parameters. By optimizing the jerk integration of trajectory in TSM space, we checked that our TSM was able to be applied to trajectory learning.

A. Definition of B-spline trajectory

The definition of robot’s state trajectory $s(t)$ as the summation of B-spline functions is defined as the weighted sum of N-order basis functions with M number of control points ($i \in [0, M]$; $b_i, N(t)$):

$$ s(t) = g^T b(t) \quad (12) $$

$$ b(t) = [b_{0,N}(t), b_{1,N}(t), \ldots, b_{M-1,N}(t)]^T \quad (13) $$

$$ g = [g_0, g_1, \ldots, g_{M-1}]^T \quad (14) $$

$t$ is time, $g$ is the weights of B-spline summation. Hence, the definition of L-dimensional robot’s state trajectories $s(t)$ are as follows:

$$ s(t) = G^T b(t) \quad (15) $$

$$ s(t) = [s_0(t), s_1(t), \ldots, s_{L-1}(t)]^T \quad (16) $$

$$ G = [g_0, g_1, \ldots, g_{L-1}] \quad (17) $$

Minimal jerk trajectory [22] is defined as follows:

$$ \text{minimize} \int_{t_e}^{t_0} \frac{\delta^3 s(t)^2}{\delta t^3} \quad (18) $$

For more detail about B-spline features, see [23].

B. Learning Settings

As the task parameters, we used 4-dof initial joint angles ($q_{0,1,2,3}$ was used and $q_{4,7,8}$ was fixed) and last end-effector position ($p_{e,x,y,z}$). As the robot’s state parameters, we used the weights of B-spline summation. We used 4-order basis functions with 8 control points ($M = 8, N = 4$) for 4-dof manipulator ($L = 4$). Hence, we used 8 control points $\times$ 4-dof = 32 parameters as robot’s state. The redundancy parameter was 1-dimension and the redundancy parameter was expected to express the last end-effector attitude. The number of cells for each layer was as follows (Table V):

<table>
<thead>
<tr>
<th>TABLE V</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The number of cells for each layer</strong></td>
</tr>
<tr>
<td>state</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>taskr</td>
</tr>
<tr>
<td>14+28</td>
</tr>
</tbody>
</table>

* taskr is the task redundancy parameter, and layerN is hidden layer.

As the learning algorithm, we used Nesterov’s accelerated gradient [15] implemented in Caffe [16]. The maximum number of iterations was $1 \times 10^7$. While the first $1 \times 10^6$ iterations, the learning rate started from 0.05 and multiplied 0.95 at each $10^6$ calculations. For generating the training data set, the ranges of motions of all initial joint angles and last joint angles were equally split into 6 blocks (total $6^8 = 1679616$ combinations), and calculated the minimal jerk trajectories which satisfy the initial and last joint angles constraints. While the second $9 \times 10^6$ iterations, the learning rate was fixed as 0.1, and the training data set was randomly expanded about 5 times. Total computation took 7.5 hours by using Intel Core i7-4790 CPU, 3.60GHz.

Fig.7 shows the reaching trajectory learned in TSM. Each row has different target end-effector coordinate and each column has different redundancy parameter. Multi robot’s state trajectory which satisfies the task constraints are confirmed to be parametrized with the same task parameter and different redundancy parameters.

C. Test: Minimize the distance between desired initial posture and last end-effector coordinates

For evaluating the learning error of TSM, we solved the optimization problem for minimizing the distance between desired task state as follow:

$$ \text{min}_i \| p_{t_e} - p_{d} \|^2 + \| q_{t_e} - q_{d} \|^2 \quad (19) $$

s.t.  \quad G = TSM_{\text{task}}(i, p_d, q_d)

$$ q_{t_e} = G^T b(t_e) $$

$$ p_{t_e} = G^T b(t_e) $$

$$ [p_{t_e}^T, r_{t_e}^T]^T = FK(q_{t_e}) $$

Eq.19 is an implementation of Eq.2. $q_{t_e}$ is the initial posture of trajectory. $q_{t_e}, p_{t_e}, r_{t_e}$ is the last posture, end-effector position and attitude. For solving Eq.19, we used DIRECT algorithm in NLOPT, and we did not use local
1) Task1: Reaching trajectory for high place
2) Task2: Reaching postures for higher place

Fig. 7. Reaching trajectory with parametrized redundancy. Each horizontal line has different target end-effector position, and 5 images in the same line shows trajectories with the same target end-effector positions and with different redundancy parameters.

We solved Eq.19 for $3^7 = 2187$ times and calculated the objective function. The average computational time was 0.19 seconds.

Table VI shows the variance of distance between the desired task value. The 95% range of the end-effector position distance is about ±20nm and this order is similar to that of inverse-kinematics learning (Sec.III-C). In comparison with the end-effector range of motion which is about ±1m, this error is about ±2%. In contrast, The 95% range of the initial posture distance is less than 1% of each joint’s range of motion ($\theta_0 = 200\text{deg}; \theta_1 = 105\text{deg}; \theta_2 = 184\text{deg}; \theta_3 = 139\text{deg}$), and this is smaller than that of end-effector position. The reason of this difference is that the joint angle parameters have linear relationships between robot’s state parameters ($= G$, the summation weights of B-spline), and it is easy to be learned by sigmoid function.

Table.VI-VII shows the variance of distance between the desired task value. The 95% range of the end-effector position distance is about ±20nm and this order is similar to that of inverse-kinematics learning (Sec.III-C). In comparison with the end-effector range of motion which is about ±1m, this error is about ±2%. In contrast, The 95% range of the initial posture distance is less than 1% of each joint’s range of motion ($\theta_0 = 200\text{deg}; \theta_1 = 105\text{deg}; \theta_2 = 184\text{deg}; \theta_3 = 139\text{deg}$), and this is smaller than that of end-effector position. The reason of this difference is that the joint angle parameters have linear relationships between robot’s state parameters ($= G$, the summation weights of B-spline), and it is easy to be learned by sigmoid function.

Table VI-VII shows the variance of distance between the desired task value. The 95% range of the end-effector position distance is about ±20nm and this order is similar to that of inverse-kinematics learning (Sec.III-C). In comparison with the end-effector range of motion which is about ±1m, this error is about ±2%. In contrast, The 95% range of the initial posture distance is less than 1% of each joint’s range of motion ($\theta_0 = 200\text{deg}; \theta_1 = 105\text{deg}; \theta_2 = 184\text{deg}; \theta_3 = 139\text{deg}$), and this is smaller than that of end-effector position. The reason of this difference is that the joint angle parameters have linear relationships between robot’s state parameters ($= G$, the summation weights of B-spline), and it is easy to be learned by sigmoid function.

D. Test: Minimize jerk integration of trajectory involving end-effector position constraint

The optimization problem which minimizes the jerk integration of trajectory and satisfies the end-effector position constraint is a non-linear optimization problem due to the nonlinearity of the relationship between joint angles and end-effector coordinate. Hence, this problem is a little computationally expensive. However, by using TSM\(_{\text{traj}}\) and by searching only redundant parameter of task, it is possible to accelerate computation.

$$\min_i \int_{t_s}^{t_e} dt \| G^T \frac{\delta^3 b(t)}{\delta t^3} \|^2$$

s.t. $G = TSM_{\text{traj}}(i, p_d, q_d)$

$$q_{ts} = G^T b(t_s)$$

$$q_{te} = G^T b(t_e)$$

$$[p_{ts}^T, r_{ts}^T]^T = FK(q_{ts})$$

$$\forall i: \| q_{ts,i} - q_{d,i} \| < \epsilon_{q,i}$$

$$\forall i: \| p_{ts,i} - p_{d,i} \| < \epsilon_{p,i}$$

$\epsilon_{q,i}$, $\epsilon_{p,i}$ is the error tolerance of initial posture and last end-effector position. Now, we used $\epsilon_{q,i} = 10\text{deg}$, $\epsilon_{p,i} = 10\text{cm}$. We solved Eq.20 for $3^7 = 2187$ times with randomly sampled feasible task values. The average computation time is 0.51 seconds.

Table VIII shows the average and variance of those objective functions.
functions. In comparison with minimal average and maximum average, the difference is 2197.6 and 4283.5 rad/sec. Hence, we confirmed that TSM_{traj} has multiple trajectory with the same task parameters and with the difference redundancy parameter.

V. CONCLUSION

For accelerating the computation time of non-linear optimization for robot’s state, we propose a deep-auto-encoder based task-state map (TSM) structure, and an algorithm to optimize robot’s state by utilizing TSM. Our approach accelerates computation by preparing a database of feasible robot’s state (TSM) in advance and by seeking a desired state according to the task objective. TSM receives task parameters (e.g. end-effector coordinate) and task redundancy parameters (e.g. 7-dof arm’s redundant degree of freedom), and directly calculates robot’s state parameters (e.g. joint angles) which satisfy task constraints. Our TSM has two advantages. First, the task redundancies are automatically parameterized as the result of deep-auto-encoder learning. Second, the number of parameters for determining the database structure is relatively smaller than that needed by the simple database which stores all pairs of tasks and robot’s states. Besides, we propose an algorithm for fast global optimization by using TSM. This algorithm is an global optimization process in TSM task space, and involves local planner for absorbing learning error. Finally, we evaluate our TSM and optimization algorithm in two aspects of computational speed and objective value through two experiments of kinematics learning and trajectory learning. And we confirmed that our approach was effective for quickly calculating the approximation of minimal value.

The remaining challenge is TSM application to robot’s whole-body trajectory such as walking trajectory mapped from footstep task, and real-time optimization of whole-body trajectory in the TSM task space. Because even our experiments took some days for learning the single arm reaching trajectory, whole-body trajectory learning took much more time, possibly a few months or more. Besides, the size of necessary training data would be also a problem. For example, for equally sampling in 30-dof joint parameters space such as 10 samples per 1-dof, 10^{30} sampling combinations are needed. For this problem, it would be efficient to learn with relatively small training data, to generate another training data at the same time with learning, to swap those training data after learning, and to restart learning again and again.

REFERENCES