DETERMINING TASK OPTIMAL MODULAR ROBOT ASSEMBLY CONFIGURATIONS

I-Ming Chen  
School of Mechanical and Production Engineering  
Nanyang Technological University  
Singapore 2263  
Republic of Singapore

Joel W. Burdick  
Division of Engineering and Applied Science  
MS 104-44 Caltech  
Pasadena, CA 91125  
U. S. A.

Abstract

A “modular” robotic system consists of standardized joint and link units that can be assembled into a number of different kinematic configurations. Given a predetermined set of modules, this paper considers the problem of finding an “optimal” module assembly configuration for a specific task. We formulate the solution as a discrete optimization procedure. The formulation is based on an assembly incidence matrix representation of a modular robot and a general task-oriented objective function that can incorporate many realistic task criteria. Genetic algorithms (GA) are employed to solve this optimization problem, and a canonical method to represent a modular assembly in terms of genetic strings is introduced. An example involving a 3-DOF manipulator configuration is presented to demonstrate the feasibility of this approach.

1 Introduction

A modular reconfigurable robotic system is a collection of various sub-assemblies, at the level of links and joints, that can be easily separated and reassembled into different configurations through standardized connecting interfaces. By reconfiguring the modules, different robots can be created so as to suit a diversity of task requirements. Several prototype modular robotic systems have been built and demonstrated [3,5,12,14].

Let the arrangement of modules in a modular robot be called an assembly configuration. We consider the task-oriented optimal configuration problem in this paper—i.e., how to find an optimal assembly configuration from a set of modules for a specific task. We assume that the set of modules is fixed at planning time, and thus only freedom available for optimization is the recombination and rearrangement of modules. For simply designed module sets, the total number of possible assembly configurations can be manually enumerated, and this set of robot configurations can be easily tested against the task requirements. As the module design becomes more complicated and versatile, the possible number of the robot’s assembly configurations grows tremendously. An efficient and systematic method to find an optimal configuration is thus necessary.

Here we introduce a general framework for solving the optimal assembly configuration problem. We define a generally applicable task related objective function which evaluates a modular robot assembly configuration for a given task while avoiding sub-assemblies with undesirable kinematic properties. This function could be applied to all of the unique assembly configurations which result from the algorithm outlined in [1,2]. Alternatively, we can use this function as the basis for discrete or combinatorial optimization method. In this paper, we employ genetic algorithms because of the discrete nature of the assembly configuration set.

Paredis and Khosla [11] have considered task-based robot design for fixed configuration robots. In their formulation, all design parameters are assumed continuous values, and hence conventional optimization techniques can be employed. In the problem posed here, the search space, the set of assembly configurations, is a discrete set.

2 Conceptual Module Models

A conceptual module model set based on the features found in many real implementations is introduced [1,2]. The module set consists of link and joint modules only. The following 1- or 2-DOF joint modules are considered: revolute joint (R), prismatic joint (P), helical joint (H), and cylindrical joint (C). Joint modules are connected to the link modules through connecting ports and link modules possess multiple joint connections. We further assume that link modules have symmetrical geometry and symmetrically located connecting ports (though modules with only two connecting ports and no symmetry can be handles as well). The connecting ports are labeled accordingly. The symmetry design allows link modules to be re-oriented without altering the robot kinematics. For illustration purposes, we assume only two types of link modules are available: square prisms (10 ports) and cubic box units (6 ports), which are shown in Fig. 1. Other objects can be similarly treated.

3 Modular Robot Assembly Representation

To describe the location of a joint on a multi-port link module, we use a function termed assembly pattern. Let PORT be the set of port numbers on a link and ATT be the set of connecting status on a link. ATT contains eligible joint types and a zero indicating an empty port. For our modular robotic system, ATT = {0, R, H, C, P}. For a cubic box, PORT = {1,2,3,4,5,6}.

Definition 1 The assembly pattern on a link module is an injection mapping

\[ f : PORT \rightarrow ATT. \]
If the port is connected, $f$ will assign a joint type. If it is empty, a zero will be assigned. Because of the link symmetry, many assembly patterns can be reoriented in a way that they function identically in a large robot structure. An equivalence based on the symmetric rotations of link modules was defined on $f$ in [1] to classify distinct link-joint assembly patterns. Chen [2] introduced an enumeration algorithm, OrbitEnumerate, based on this equivalence, to list all distinct assembly patterns on a link module with a prescribed number and types of joints. Examples of assembly patterns are shown in Fig. 2.

The connectivity of individual links and joints, or the topology, in a robot mechanism can be represented by a kinematic graph, in which vertices are links and edges are joints [4]. This graph, in turn, can be expressed as a vertex-edge incidence matrix [16] which contains only 0s and 1s. The number of columns and rows in this matrix are equal to the number of edges and vertices in the graph respectively. Entry $(i, j)$ is equal to 1 if edge $j$ is incident on vertex $i$, or it is equal to zero otherwise.

**Definition 2** Let $G$ be the graph of a modular robot and let $M(G)$ be its incidence matrix. The assembly configuration of this robot can be represented by an assembly incidence matrix, $A(G)$, obtained by replacing every entry of 1 in $M(G)$ with a non-zero integer, $k \in \text{PORT}$. The zero entries remain unchanged. $a_{ij} = k$ indicates that joint $e_j$ is attached to port $k$ of link $v_i$; $a_{ij} = 0$, otherwise.

For heterogeneous modular robots, which contain different kinds of link and joint modules, the assembly incidence matrix is augmented with an additional row and column that specifies the type of link and joint modules and is called the extended assembly incidence matrix (eAIM) [2]. Examples of assembly configurations of the following AIMS are shown in Fig. 3.

\[
A(G) = \begin{pmatrix}
10 & 5 & 1 \\
0 & 2 & 0 \\
0 & 0 & 0 \\
0 & 0 & 4
\end{pmatrix} \quad A(\bar{G}) = \begin{pmatrix}
10 & 5 & 1 & L \\
0 & 1 & 0 & B \\
0 & 0 & 0 & L \\
C & C & R & 0
\end{pmatrix}
\]

As pointed out in [1], different AIMS may have identical kinematic properties, such as the workspace and location of joint singularities. These AIMS are classified by an equivalence relation in order to reduce the number of unique robot assembly configurations. Two AIMS are said to be equivalent if the underlying graphs are isomorphic and the corresponding link assembly patterns are equivalent. The configuration enumeration algorithm, OrbitEnumerate, stated in [2] utilized this equivalence definition. The output is a set of kinematically unique robot assembly configurations.

4 General Formulation

To find an optimal solution, we wish to formulate an objective function that will evaluate how well an assembly configuration can accomplish an assigned task. Ideally, the form of the evaluation function should be general enough so that it is applicable to a wide variety of tasks, but flexible enough to incorporate different criteria which may be task specific. We term this function an assembly configuration evaluation function (ACEF).

In effect, the ACEF is used to evaluate every unique assembly configuration generated by RobotEnumerate for a certain task requirement. The assembly configuration with the greatest ACEF value is deemed optimal. Then, this function can be used as the basis for a discrete optimization scheme. The remainder of this paper focuses primarily on the task evaluation function and its use for a discrete optimization procedure.

It is also important to note that from a given set of modules it may be possible to construct robots with various topologies—e.g., robots with serial or parallel kinematic structures. Even with a fixed robot topology class, the number of degrees of freedom (DOF) can alter the kinematic functionality of the system. For simplicity, we discuss the case of fixed topology and fixed number of DOF. Fig. 4 depicts the scheme to solve the task-oriented optimal configuration problem for the case of a specific robot topology. The desired robot topology and the number of DOFs are fed into the configuration enumeration algorithm in advance. A quantifiable task definition and task evaluation criteria (TEC) are parameters to the ACEF. Furthermore, an
auxiliary function, termed the module assembly preference (MAP), is defined on individual assembly patterns to filter out those with inappropriate or undesirable kinematic features. The overall optimization problem becomes:

**GIVEN:** a robot task, a TEC, a MAP, and prescribed robot topology,

**FIND:** an AIM in the assembly configuration set whose ACEF value achieves a maximum.

If a set of robot configurations is given in advance, the enumeration algorithm can be bypassed, and the optimization procedure employed. The following sections explain the details of choosing task definitions, TEC, and MAPs. We use a serial type fixed base modular arm with revolute joints to demonstrate this problem solving strategy.

5 Task Specifications

**Definition of Robot Tasks**

A robot task is defined as a collection of working points, \( w_p \), in the operation space \( S^3 \) [8] or a collection of the end-effector positions/orientations \( w_e = (x, y, z, \theta, \phi, \psi) \), where \( \theta, \phi, \psi \) are Euler angles representing the orientation of the end-effector frame. If the robot is to follow a trajectory, this task can be approximated by a set of points along the path.

**Task Evaluation Criteria (TEC)**

Many monotonically increasing local kinematic performance measures can be employed for task evaluation [9,10,15]. These measures are typically evaluated at a particular working point with a specified mechanism posture. Many measures are based on the mechanism's Jacobian matrix, \( J_e \), the manipulability measure, \( \det(JJT)^{1/2} \) [15] and the condition number or minimum singular value of \( J \) [9,10]. Other configuration dependent criteria are possible.

Let \( \mu_i \) be the value of a modular robot performance measure at task point \( i \). For a collection of \( n \) task points, we choose to define the total task performance, \( \mu \), to be the smallest \( \mu_i \) among the task points, i.e.,

\[
\mu = \min_{i \in \{1, \ldots, n\}} \mu_i
\]

\( \mu \) represents the worst case among the collection of task points. Since \( \mu_i \geq 0 \), by definition, \( \mu \geq 0 \). This choice ensures that one robot assembly will be better than the other in the worst case.

6 Module Assembly Preference

In modular robots, the choice of the joint locations on a link module will affect the kinematic capability of the entire assembly. It is useful to be able to specify the preference of certain link/joint assembly patterns in the overall assembly configuration in order to avoid undesirable kinematic conditions, such as link interference or joint redundancy. By joint redundancy, we mean that the number of effective DOF of a robot is less than the number of its joints. For example, assembly patterns with collinear revolute joint axes (Fig. 5) exhibit joint redundancy. However, the same assembly patterns may be emphasized in other situations. For example, the redundant assembly patterns extend the dimension of the robot arm and enlarge the workspace. Besides, when one of the two joints fails, the other joint is able to drive that degree of freedom.

In order to retain or filter out a joint assembly pattern in a modular robot, a binary-valued function called the module assembly preference (MAP), \( \phi \), is defined on a set of distinct assembly patterns. Let \( F \) represent the set of distinct assembly patterns. \( [f] \in F \) represents a distinct joint pattern.

**Definition 3** A module assembly preference, \( \phi : F \rightarrow \{0, 1\} \), is a surjective function such that

\[
\phi([f]) := w, \quad w = 0, 1
\]

\( \phi([f]) = 0 \) represents an undesirable joint pattern. The MAP can be thought of as a quantifiable rule varying with different kinematic requirements. The preference of an entire robot assembly configuration is defined based on a MAP of assembly patterns. Suppose the robot has \( n \) links and the values of the MAP on Link \( i \) is \( w_i \). The structural preference of an entire robot, \( \Phi \), is defined as follows.

**Definition 4** The structural preference, \( \Phi \), of a modular robot assembly configuration \( A \) is the product of the MAPs of the robot's assembly patterns,

\[
\Phi(A) = \prod_{i=1}^{n} w_i
\]

where \( A \) is the AIM of a robot assembly configuration. Since \( w_i \) is equal to 0 or 1, \( \Phi(A) \) is equal to 0 or 1 as well. \( \Phi(A) = 0 \) indicates the assembly configuration contains undesirable assembly patterns. The choice of a MAP, \( \phi \), is illustrated in the following example.

**Example 1** Let a 3-DOF fixed base serial type with R-joints and prismatic links satisfy the kinematic conditions: minimum link interference and no joint redundancy. The corresponding MAP is described in Fig. 6. The intermediary prism link modules are connected by two joints. From 2 we obtain 9 distinct assembly patterns on link modules and 2 distinct patterns on cube modules. The choice of no joint redundancy implies that the MAP will be zero for patterns 4, 9, and 13. The requirement on minimum link interference will set the MAP of patterns 2 and 7 to zero, since two joints are attached to the same end of the prism. The remaining patterns are assigned a MAP value of one. For the end prism link, there exists only two distinct assembly patterns (patterns 10 and 11). The requirements on the robot structure have little influence on the end link patterns, so their MAPs are set to 1. If joint redundancy is allowed, patterns 4, 9, 13 will be set to 1. Using this MAP, the structural preferences of the 3-DOF robot
Figure 6: MAP for patterns on a prism and a cube configurations $A_1$ and $A_2$ can be obtained as follows.

$$A_1 = \begin{pmatrix} 1 & 0 & 0 & FB \\ 1 & 2 & 0 & L \\ 0 & 5 & 2 & L \\ R & R & R & 0 \end{pmatrix}, A_2 = \begin{pmatrix} 1 & 0 & 0 & FB \\ 1 & 2 & 0 & L \\ 0 & 1 & 9 & L \\ R & R & R & 0 \end{pmatrix}$$

The structural preference of $A_1$ is calculated starting from the base module. Since there is only one assembly state on the fixed base (pattern 1), its MAP is set to 1. There are two R-joints connected to port 1 and 2 of link 2 (pattern 1), so $w_2 = 1$. R-joints are connected to port 2 and 5 of link 3 (pattern 5), so $w_3 = 1$. For the end link (pattern 10), $w_4 = 1$. Therefore, $\Phi(A_1) = w_2 \cdot w_3 \cdot w_4 = 1$. Similarly, in $A_2$, $w_2 = 1$ (pattern 1), $w_3 = 0$ (pattern 8), and $w_4 = 1$ (pattern 10), so $\Phi(A_2) = 0$.

7 Assembly Configuration Evaluation Function

The structure of the ACEF for a serial modular robot is shown in Fig. 7. This function evaluates the "goodness" of a robot assembly configuration for a required task and structure specification. The "goodness" is represented by a non-negative real number. An AIM with large real value represents a good assembly configuration.

The input of the ACEF is an AIM with a predefined number of DOFs and predefined topology. The function is divided into task and structure parts. The MAP, $\Phi$, is given to determine the structural preference $\Phi$. Task points and the TEC are given for task evaluation. In the first part of task evaluation, the workspace check procedure determines if an assembly, represented by an AIM, can carry out the specified task. If a task point is outside of the robot's workspace, there is no need to proceed with the task evaluation, and $\mu$ will be set to zero. If all task points are reachable, the TEC is applied to calculate $\mu$.

Figure 7: ACEF for serial modular robots

Definition 5 The ACEF, $\Psi$, for the performance of a robot assembly, $A$, is

$$\Psi(A) = \Phi(A) \cdot \mu(A). \quad (4)$$

Since $\Phi(A)$ is equal to zero or one, and $\mu(A) \geq 0$, by definition, $\Psi(A) \geq 0$ as well. Note that the structure preference $\Phi$ functions as a filter for the ACEF. For assembly configurations satisfying the kinematic constraints, $\Phi$ is always equal to 1. Therefore, $\Psi(A) = \mu(A)$. Comparing the total performance of acceptable AIMS is equivalent to the comparison of the TECs of the task points only.

Workspace Check Procedure

The workspace check is accomplished by solving the inverse kinematics of the manipulator for a given task point. If a real valued solution can be found, the task point is within the workspace, otherwise, the task point is out of reach. A numerical inverse kinematics technique proposed by [7] is adopted here for its robustness and efficiency. The detail of the workspace checking can be found in [2].

8 Genetic Algorithms for Modular Robots

If the number of assembly configurations that can be generated from a given set of modules is small, an exhaustive evaluation of each assembly configuration can be performed in a reasonable amount of time. However, as the number of robot DOFs increases, the set of assembly configurations may become factorially large and the exhaustive search becomes undesirable. Instead, a discrete random search technique can be used for efficiency. Here, we explore the use of genetic algorithms (GA) to solve this problem. Other researchers
have applied GAs to distributed robotic systems and
task-based robot design [8,13].

A "genetic algorithm" is an optimization method based
on a model of an ecological system in which the mechan-
ics of natural selection and natural genetics are the pri-
mary factors for improving the performance of a pop-
ulation [6]. In this algorithm, candidate solutions are
coded into string structures that are analogous to a ge-
netic code. A fitness function will assign a fitness value
to every string. The candidate strings are combined
among themselves with a structured, yet randomized,
information change in order to form a new generation of
candidate solutions. The change is based on operations
that mimic the adaptive process of natural systems:
reproduction, crossover, and mutation. The strings of the
new generation are created by using bits and pieces of
the fittest strings in the previous generation. These bits
and pieces of the string contribute to the overall
performance of the string, and create offsprings with
higher fitness values. It is interesting to note the anal-
ogy between the effect of module sub-assembly patterns
on overall performance of a modular robot, and the in-
fluence of sub-strings on overall string fitness.

CODING SCHEMES FOR AIMS

To employ GAs effectively, a coding scheme to trans-
form an AIM into a string structure is necessary. A
binary string of Os or Is is chosen to represent an AIM
because it can offer the maximum number of schemata
per bit of any coding [6]. This string contains sub-
strings, where each substring represents the types of
joints and the types of link and joint patterns in an
AIM. We call this binary string an "assembly string." The
lengths of the substrings are determined by the num-
bers of joint types and the number of distance joints
and link patterns. Since joints and links are arranged in
alternating sequence in a serial type robot, the sub-
strings representing the types of joints and links are
arranged in the same alternating sequence as in the
AIM shown in Fig. 8.

Suppose the substring for the joint type is of length
$n_1$. Each joint type is assigned a unique bit string pat-
tern of length $l_1$. Similarly, a substring of length $l_2$
is used to represent the link type. The possible types
of joint/link assembly patterns is assigned a substring
of length $l_3$. In order to reduce the size of the search
space, bit string patterns are only assigned to kinemati-
cally unique patterns. Note that in a serial robot, the
number of joints connected to the intermediary links,
The initial set of AIMs is shown in Fig. 10. The figure, is reached. After the destination generation is reached, we choose the assembly string in this generation that has the largest fitness value as the optimal assembly configuration satisfying the required task and structure specifications.

**Example 2** We wish to find a 3-DOF fixed base serial robot with R-joints that passes through a set of task points listed in the following table. We also demand that there are no redundant joints and minimum link interference. The TEC is chosen to be the manipulator's manipulability measure and the MAP is described in Example 1.

<table>
<thead>
<tr>
<th>Task Point</th>
<th>Task Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (3.0, 0.5, 0.5)</td>
<td>4 (2.0, 0.0, 0.5)</td>
</tr>
<tr>
<td>2 (3.0, 0.5, 1.5)</td>
<td>5 (0.5, 3.0, 0.5)</td>
</tr>
<tr>
<td>3 (2.0, 2.0, 1.5)</td>
<td>6 (0.5, 3.0, 1.5)</td>
</tr>
</tbody>
</table>

The initial set of AIMs is shown in Fig. 10. The empty boxes represent assembly strings that do not correspond to any AIM. Their fitness values are set to zero. The fitness values of these AIMs obtained from the ACEF are (5.7963, 0.0, 5.7963, 0.0). The parameters used in the GA are $P_{cross} = 0.6$, the probability of crossover operation, and $P_{mutate} = 0.1$, the probability of mutation. The destination generation is chosen to be the 10th generation. After evolving ten generations, the AIMs in the final generation are shown in Fig. 11. Their fitness values are (5.7963, 0.0, 5.7963, 0.0). Fig. 12 shows the average and maximum fitness value in every generation. Assembly configurations 1, 3, and 4 are identical and have the highest fitness value of 5.7963 so they are chosen as the optimal one.

**Figure 10: Initial configurations**

**Figure 11: Final configurations**

**Figure 12: Avg. & max. fitness in each generation**

9 Conclusion

In order to evaluate a modular robot assembly configuration, we introduced a flexible and general assembly configuration evaluation function. This function can be used in two ways to find the optimal modular robot assembly configuration that solves a given task. First, the function could be used to evaluate the suitability of each of the kinematically unique assembly configuration that is enumerated by the algorithm in [1]. The configuration with the highest evaluation is obviously the most suitable. However, in some instances, such a procedure will be computationally expensive. Instead, a discrete combinatorial optimization algorithm, which employs the modular robot evaluation function, is a desirable alternative. We have found the Genetic Algorithm method to be especially well suited for the modular robot problem, and we presented a method to effectively encode modular robot assembly configurations in the bit strings required for the GA procedure.

**References**


