Enumeration, structural and dimensional synthesis of robotic hands: theory and implementation

Ali Tamimi, and Alba Perez-Gracia, Member, IEEE

Abstract—Designing robotic hands for specific tasks could help in the creation of optimized end-effectors for grasping and manipulation. However the systematic design of robotic hands for a simultaneous task of all fingertips presents many challenges. In this work the algorithms and implementation of an overall synthesis process is presented, which could be a first step towards a complete design tool for robotic end-effectors.

Type synthesis for a given task and number of fingers, solvability and dimensional synthesis for arbitrary topologies are developed and implemented. The resulting solver is a powerful tool that can aid in the creation of innovative robotic hands with arbitrary number of fingers and palms. Several examples of type synthesis, solvability calculations and dimensional synthesis are presented.

Index Terms—Robotic hands, Kinematic synthesis, Tree graphs.

I. INTRODUCTION

ROBOTIC hands are mechanical linkages where a common set of links spans a number of serial chains. Among the variety of robotic end-effectors, those generally defined as robotic hands are considered suited not only for grasping, but also for some dexterous manipulation.

When considering applications in robotic grasping and manipulation of grasped objects, many aspects of robotics have to converge, including sensing, identification, learning and planning, to name a few, and notable results are being obtained in these fields. One field that has not attracted so much attention is the systematic methodology for the physical embodiment of the robotic end effector; however, its effect on the successful completion of the task may be considerable.

The design of end-effector robotic tools has focused on three different strategies [12], which yield very different designs: anthropomorphism, designing for grasping tasks, and designing for dexterous manipulation. The anthropomorphic hands are constrained in their design, but they are considered a straightforward solution for human environment and human manipulation task mapping [10], [13]. Of the other two design strategies, hands oriented to grasping tasks are usually simpler or under-actuated; new efforts are being devoted to obtain under-actuated or simple hands with some degrees of dexterity [15], [19]. Hands for in-hand manipulation tend to be more complex, especially if a wide range of manipulation actions are targeted, however most of them are anthropomorphic in design. For a current review on design efforts of anthropomorphic hands, see [3].

More recently, the design process for robotic hands has started receiving some attention [17], [26], [2]. A task-based, systematic design process, needs to consider both the enumeration of topologies, or structural synthesis, and the dimensioning of the selected topologies, the dimensional synthesis, followed by a stage of detailed design and implementation.

The literature in type or structural synthesis is vast, especially for linkages with closed loops, which present bigger challenges in their classification. Type or structural synthesis is based on subgroups of motion, following [5] and on the use of screw theory, such as [17], among others [28], combined with graph theory for the enumeration and classification. Most of the current methods are based on defining subgroups of motion or subspaces of potential velocities for the system. A task-based approach for the structural synthesis needs to take into account the shape of the desired workspace, or kinematic task. Pucheta [18] applied a graph theory-based method and precision position method consecutively for planar linkages and for multiple kinematic tasks. Results on structural synthesis of hands based on mobility while grasping an object has been studied in [8] and [21], and more recently in [16].

For the dimensional synthesis stage, most of the research has focused on the design of individual, underactuated fingers; see [20] and [1]. The first tool for the systematic dimensional synthesis of complete multi-fingered robotic hands for given manipulation tasks, up to the authors’ knowledge, was developed by Simo-Serra et al. [24]. This tool allows to design multi-fingered robotic hands with a set of common wrist joints and a palm branching in different number of fingers, see also [25], and [23] for its theoretical development.

In this work we present a complete design methodology for arbitrary robotic hands. This includes topology enumeration and the corresponding arrays defining the topology, structural synthesis for an input task, and dimensional synthesis for hands that can present several splitting stages. Theoretical aspects, algorithmic implementation and computational aspects are included. The aim is to integrate these in a design tool to help in the creation of robotic hands tailored to specific applications.

II. TREE TOPOLOGIES

A Tree topology for a kinematic chain has a set of common joints spanning several chains, possibly in several stages, and ending in multiple end-effectors [22]. A branch of the hand is defined as a serial chain connecting the root node to one of the end-effectors, and a palm is a link that is ternary or above. The tree topology is represented as rooted a tree...
graph; the approach of Tsai [27] is followed, with the root vertex being fixed with respect to a reference system.

A multi-fingered hand is defined as a kinematic chain with several common joints - the wrist, which is a fundamental part of the hand manipulation - spanning several branches, possibly in several stages. At the end of each branch are the end-effectors, the fingertips. They are the main elements whose motion or contact with the environment is being defined by the task; this can be generalized to consider other intermediate vertices of the topology. Open hands, that is, hands not holding an object, are represented as kinematic chains with a tree or hybrid topology. For our synthesis formulation, the internal loops in the hand structure are removed using a reduction process [23], to obtain a tree topology with intermediate links that are ternary or above.

Tree topologies are denoted as $SC - (B_1, B_2, ..., B_b)$, where $SC$ is a serial kinematic chain representing the initial common joints and the dash indicates a branching or splitting, with the branches adjacent to $SC$ contained in the parenthesis, each branch $B_i$ characterized by its type and number of joints. Figure 1 shows the compacted and possibly reduced graph for a $2R - (2R, R - (3R, 3R, 3R), 2R)$, or $2 - (2, 1 - (3, 3, 3), 2)$ chain if we drop the $R$ in the case of all revolute joints. This hand has three branches, one of them branching again on three additional branches, for a total of five end-effectors or fingertips. The root vertex is indicated with a double circle. While most current robotic hands have a single splitting stage spanning several fingers, this can be generalized for greater adaptation to different applications by using hands with topologies such as the one presented in Figure 1.

A tree topology is represented by two arrays, which capture incidence and adjacency properties as well as information on the edges. Assume a numbering of the graph edges to define a parent-pointer array and a joint array. The length of both arrays is equal to number of edges of the tree graph after the reduction process is applied, that is, each edge, and each entry of the arrays, will correspond to a serial chain of the robotic hand.

The parent-pointer array implements the parent-pointer representation, where each element takes the value of the previous edge, the first edge being usually the one incident at the root vertex. The edges incident at the root vertex have no parent and they take the value zero. Each element of the joint array contains the number and type of joints for each edge. If we are limited to revolute joints, then the joint array element will be equal to the number of joints for that edge. As an example, for the tree topology shown in Figure 1 the parent-pointer array and joint array are defined as $p = \{0, 1, 1, 1, 3, 3, 3\}$ and $j = \{2, 2, 1, 2, 3, 3, 3\}$ for the given numbering of the edges.

III. KINEMATIC SYNTHESIS

KINEMATIC synthesis, the process of creating a mechanical system for a given motion task, can be used in order to select and size a topology as a candidate hand design. The synthesis process for robotic hands has four main steps that are detailed below: task definition, type or structural synthesis, solvability calculations, and dimensional synthesis. After this process we obtain a set of joints, their connectivity, and their relative position along a chain. Further steps of ranking, optimization and detailed design will be necessary to implement the candidates into functioning hands.

A. Task Definition

The task is the desired motion of the elements of the hand whose interaction with the environment is of interest. For a multi-fingered hand, a simultaneous motion of all fingertips or surface contacts, which could be any limb of the hand, is to be defined. For each fingertip or contact limb, a set of positions are defined as location plus orientation. Figure 2 shows a trajectory task for a hand with four fingertips.
B. Type Synthesis

Type synthesis, or structural synthesis, is the enumeration, selection and ranking of the kinematic chain topologies to be used as candidate designs. In the case of a robotic hand, it implies the selection or calculation of the number of fingers, the number of joints at the wrist, the number of splits or branchings, and the number of joints for the serial chain making each branch, as well as the type of joints to be used. This could be an a-priori selection by the designer or it could be calculated based on the task. Having an automatic type synthesis stage helps the designer exploring the sometimes vast field of possible solutions, and identify trends in the candidate topologies.

C. Solvability

In the case of simultaneous tasks of all fingertips, solvability is defined as the ability of all combination of fingers to perform their relative tasks, and it is a condition that needs to be checked in order to be able to do the dimensional synthesis. It consists of checking what is the maximum number of positions that do not overconstrain each root-to-end-effector(s) subgraph, and ensuring that each subgraph is less constrained than the overall graph. In this calculation, the subgraphs obtained by moving the root to each end-effector need to be included to account for relative motion between fingertips.

D. Dimensional Synthesis

In the dimensional synthesis stage, the position of the joint axes are to be calculated, for the selected solvable topology and for the desired kinematic task. There are many techniques to state and solve the dimensional synthesis equations; regardless of the formulation, the output is the position of the joint axes at a reference configuration. This output is equivalent to the set of parameters defining the relative location and orientation between adjacent joints. The kinematic solution can be then used for the detailed design of the hand.

IV. Type Synthesis and Enumeration

Given a simultaneous motion task for all fingertips, it is important to know how many, and what hand topologies are suited for the task. The number of candidate hand topologies of a certain type is usually very high and unbounded in some cases [9]. This number of suited topologies can be reduced if some additional constraints are added. At the end of the process, one or a few of these topologies will be selected for performing the additional design steps. The approach taken here is different from previous research such as [8], and it is based on free finger motion.

The conditions for considering a topology for the task are, at the least, to have the same number of end-effectors as the task and to be solvable, according to the criteria defined in [23] and [9]. The set of suited topologies can be ranked according to other criteria, such as number of edges, number of splits, and number of joints per edge, among others.

A. Candidate topology search

A search method and its algorithmic implementation is presented here to find solvable topologies for a defined task. The task is assumed to be a general subset in the SE(3) group of rigid motion and its derivatives, and the goal is to find all topologies that can be paired with the task for dimensional synthesis, given a set of user-defined restrictions.

User-defined inputs are the number of positions of the task \( m \), the number of end-effectors, or branches, \( b \), and the total number of edges of the graph \( e \). Remember that every edge corresponds to a serial chain. The output is the set of topologies that meet the solvability criteria subject to these conditions.

The task-sizing formula in [23] is applied in the first place to find all possible branch topologies for the given number of end-effectors. Start by calculating

\[
J = \sum_{i=1}^{e} j_i = \frac{(m - 1) * 6 * b}{(m + 3)} \tag{1}
\]

where \( J \) represent the total number of joints and \( j_i \) are the joints for the serial chain corresponding to the \( i \)-th edge of the topology. The number of joints per edge has to be between 1 and 5 for synthesis purposes, as a serial chain of length 6 or higher does not impose any restriction on the motion.

The presented method includes three steps. First, all possible tree structures (parent-pointer arrays \( p \)) which meet the input criteria are found, for the given number of branches and number of edges. In the algorithmic implementation, the parent-pointer array is filled up starting at the root and sequentially according to the following rules:

- \( p(1) = 0 \). The first edge is the root node and has not parent.
- If \( i \) is not an end-effector, \( p(i) \) can accept any value between \( p(i - 1) \) to \( i - 1 \). The values of parent pointer array are increasing \( (p(i) \geq p(i - 1)) \). This condition helps to avoid adjacent branch isomorphism.
- If \( i \) is an end effector, \( p(i) \) can accept any value between \( p(i - 1) \) to \( e - b \), since the last \( b \) edges are end-effectors and cannot be parents.
Second, for each structure found in first step, construct all possible joint arrays which meet the input criteria. This implies writing all joint arrays with length equal to \( e \) that satisfy Eq. 1 and with entries between 1 and 5. Constructing isomorphic trees is avoided by proper index and value assignment. Finally, after finding all possible joints arrays for each parent pointer array, check the solvability of each topology including parent pointer array and joint array. If it is solvable, add it to result as a candidate topology. This method yields all non-isomorphic trees \([6]\) for the input parameters.

The algorithmic implementation of the structural synthesis for tree topologies is detailed in Algorithm 1.

### B. Type synthesis enumeration

Even though this search may be unbounded, reduced atlas can be created for a certain range on the number of end-effectors and precision positions.

Table I shows different values for the inputs and the number of candidate topologies that can be found. In this table, \( m \) is number of task positions for each fingertip, \( b \) is number of end-effectors, \( e \) is the number of edges of the graph. The overall number of joints is calculated under Joints and the number of different joint arrays \( (j) \) and parent-pointer arrays \( (p) \) are calculated. The candidate topologies are the solvable combinations of joint arrays and parent-pointer arrays.

#### TABLE I

| INPUTS | OUTPUTS | Candidate Topologies |
|--------|---------|----------------------|
| m b e  | Joints j p |                   |
| 3 2 2  | 4 2 1 1 |                     |
| 3 2 3  | 4 2 1 2 |                     |
| 3 3 3  | 6 3 1 1 |                     |
| 5 2 3  | 6 6 1 4 |                     |
| 5 3 3  | 9 3 1 5 |                     |
| 5 3 4  | 9 45 2 9 |                  |
| 5 4 4  | 9 46 1 19 |                  |
| 5 4 5  | 12 8 1 1 |                     |
| 5 4 6  | 12 187 3 14 |                  |
| 5 4 7  | 12 478 3 72 |                  |
| 5 5 7  | 12 206 1 47 |                  |
| 6 3 3  | 10 58 2 4 |                     |
| 6 3 5  | 10 76 1 13 |                    |
| 9 4 4  | 16 5 1 1 |                     |
| 9 4 5  | 16 250 3 26 |                  |
| 9 4 6  | 16 1442 3 237 |                  |
| 9 4 7  | 16 1313 1 292 |                  |
| 13 2 3 | 9 11 1 6  |                     |
| 13 4 5 | 18 187 3 4  |                     |
| 13 4 6 | 18 1645 3 161 |                    |
| 13 4 7 | 18 2137 1 233 |                    |
| 21 2 3 | 10 10 1 10 |                     |
| 21 3 3 | 15 1 1 |                     |
| 21 3 4 | 15 45 2 |                     |
| 21 5 5 | 25 1 1 |                     |
| 21 5 6 | 25 168 4 |                     |

To illustrate the results of Table I, Table II shows some of the candidate topologies that can be found using this method, where \( p \) denotes the parent-pointer array and \( j \) the joint array of the topology. Due to the high number of solvable
candidate topologies, it is not possible to present them all in the table, however the final number is presented in Table I for each example. Figure 3 presents the three non-isomorphic topologies for two fingertips and three precision positions.

TABLE II
EXAMPLES OF TYPE SYNTHESIS

| Example 1 | Example 2 |
|-----------|-----------|
| m=3 b=2 e=2&3 |
| p = (0, 0, 1, 1, 2, 2, 2) |
| j = (2, 2, 2, 2, 2) |
| p = (0, 1, 1, 1, 1, 1, 1) |
| j = (4, 3, 4, 4, 4) |
| p = (0, 1, 1, 1, 1, 1) |
| j = (4, 4, 4, 4) |
| p = (0, 0, 0, 0, 0, 0, 0) |
| j = (2, 2, 2, 2, 2, 2, 2) |

| Example 3 | Example 4 |
|-----------|-----------|
| m=13 b=5 e=5 |
| p = (0, 1, 1, 1, 1, 1) |
| j = (2, 4, 4, 4, 4) |
| p = (0, 0, 0, 0, 0, 0) |
| j = (2, 5, 5, 5, 5, 5) |
| p = (0, 1, 1, 1, 1, 1) |
| j = (4, 3, 4, 4, 4) |
| p = (0, 1, 1, 1, 1, 1) |
| j = (5, 5, 3, 4, 3, 3) |

Example 3:

- Some selected topologies:
  - p = (0, 1, 1, 1, 1)
  - j = (2, 4, 4, 4, 4)
  - p = (0, 0, 0, 0, 0, 0)
  - j = (2, 5, 5, 5, 5, 5)
  - p = (0, 1, 1, 1, 1, 1)
  - j = (4, 3, 4, 4, 4)
  - p = (0, 1, 1, 1, 1, 1)
  - j = (5, 5, 3, 4, 3, 3)

V. SOLVABILITY

We define a hand as solvable when we can design it for a meaningful simultaneous task of all the fingertips or end-effectors, that is, a positive rational task with at least two positions. Because some fingers may be overconstrained while others are underconstrained for a given topology, solvability needs to be checked systematically for all root-to-end-effector subgraphs of the hand, including those obtained when changing the root vertex to one of the end-effectors.

Equation 2 calculates number of positions for the exact kinematic synthesis of a tree topology. If the number of positions so obtained for the kinematic task of all subtrees is greater or equal than the number of positions for the overall tree, the tree is solvable for kinematic synthesis.

\[
m = \frac{D_s}{D_s - 2} + 1.
\]

In this equation, \(D_s\) is the vector containing the number of structural variables for each edge, \(E\) is the vector of ones for the edges belonging to the subgraph, \(D^e\) is the vector of possible extra constraints for each branch, \(B\) is the vector of ones for the branches belonging to the subgraph, \(D_{ee}\) is the vector of degrees of freedom for the motion of each end effector, and \(D_j\) is the vector containing the number of joint variables for each edge. These vectors are calculated with the help of the root-to-end-effector path matrix of the graph.

The algorithmic implementation of the solvability condition has two steps, a first one to create all possible subgraphs and their corresponding arrays, and a second one to calculate the solvability for those subgraphs. Algorithm 2 is implemented in order to calculate the solvability.

Algorithm 2 Solvability

\[
M \leftarrow \text{Number Of Position(Tree)} \quad \triangleright \text{equation 2}
\]

for all SubTree do

\[
m \leftarrow \text{Number Of Position(SubTree)} \quad \triangleright \text{equation 2}
\]

if \(M > m\) then

return NOTSolvable

end if

end for

for all end-effectors do

\[
\text{RemoveCommonEdge(Tree)} \quad \triangleright \text{ Explain in Algorithm 3}
\]

\[
\text{newRoot} \leftarrow \text{endEffector(i)}
\]

\[
\text{Reconstruct(Tree, newRoot)} \quad \triangleright \text{ Explain in Algorithm 3}
\]

if only one end-effectors remain then

return Solvable

end if

end for

The process of assigning a new parent-pointer array to the subtrees is shown in Algorithm 3.

As an example, for the tree topology shown in Figure 4 solvability needs to be checked for the original tree, the following root-changing trees (Figure 4) and all of their subtrees. Figure 5 shows the new parent-pointer representation assignment.

The process can be itemized as follows:
Algorithm 3 Change Parent Pointer Array

procedure RemoveCommonEdge(Tree)
    for all edge in edges do
        if edge is in all branches then Remove(edge) ▷
    end if
end for

procedure Reconstruct(Tree, newRoot) ▷ change parent pointer for edges which are connected to the path between last root and current root
    joint ← newRoot
    while joint !=" zero do
        for all edge in edges do
            if (parent(edge)== parent(joint)) then
                parent(edge) ← joint
            end if
        end for
        joint ← parent(joint)
    end while
    ▷ change parent pointer for edges which are in the path between last root and current root (it means change the direction of path)

    p ← 0
    q ← newRoot
    for all p in path do
        parent(q) ← p
        p ← q
    end for
    parent(q) ← p
end procedure

The value -1 means that the edge has been removed.

b) There is a path between previous root and new root.
In this step, the parent-pointer value of the edges that are connected in this path is updated. In the example of Figure 5, the path includes edges 3 and 6. In this step, the value of parent pointer for edges 2, 4, 5, 7 is changed. The parent-pointer array after this step changes from \( ppt = \{-1, 0, 0, 0, 3, 3, 3\} \) to \( ppt = \{-1, 3, 0, 0, 6, 3, 6\} \).

c) Finally, the parent-pointer value for the edges which are in the path is updated, by changing the value of parent pointer for edges 3 and 6. The parent-pointer array after this step changes from \( ppt = \{-1, 3, 0, 0, 3, 6, 3, 6\} \) to \( ppt = \{-1, 3, 6, 3, 6, 0, 6\} \).

d) After the previous step, the new tree is ready and \( m \) can be calculated for all the combinations of the branches, as the algorithm shows. Knowing the branch connectivity is needed for making the \( [B] \) matrix, and the tree with \( b \) branches has \( 2^b - 1 \) combinations of branches for the original root node; when switching the root node to each end-effector, that yields \( 2(2^b - 1) - b \) different subtree combinations. These are defined by changing \( 1 \rightarrow 2^j \) to binary numbers using \( j \) digits, \( j = 1, \ldots, b \). Finally, all the needed matrices are available for calculating \( m \) and comparing them to \( M \).

A. Solvability examples

Table III shows the results of the solvability checking algorithm for some hand topologies. For the cases in which the topology is solvable, the number of positions to be used for exact kinematic synthesis is returned. If the tree is not solvable, the overconstrained subtrees are identified.

VI. Dimensional Synthesis

KIEMATIC dimensional synthesis is used to shape the new designs for robotic hands, able to grasp and/or manipulate in a given application. Dimensional synthesis has the candidate topology and the kinematic task as inputs. The kinematic task consists of a set of simultaneous displacements for each fingertip, as well as velocities and accelerations defined at some or all of those positions.

A. Automatic forward kinematics equations

For the design of robotic hands with arbitrary topologies, including multiple splitting stages, forward kinematics equations need to be automatically created from the tree topology.
and its associated arrays, identifying the common joints that will appear in the equations of several branches. The strategy to accomplish this is to divide the forward kinematics in serial chains ‐corresponding to graph edges‐, branching points and end-effector points. Three types of objects are defined as outlined below:

1) **Chain**: a set of joint axes connected in series. There are two different types of chains, those ending on an end-effector and those ending at a branching point. This second type is common to several branches, however from the point of view of the object, they are generated equally.

2) **Tip Contact Point (TCP)**: TCPs are created for each end-effector and then attached to the corresponding end-effector chain. In the most general case, we can attach a TCP to any link, such as a palm link or intermediate finger link.

3) **Splitter**: a vertex that spans more than one edge. Splitters are identified and created, and the chains spanning from each of them are attached to the splitter. If the splitter has a predecessor, then the splitter is attached to the common serial chain.

This sequential process takes place until design equations are created for each chain from root node to end-effector node. First step is generating the end-effector chains. For each end-effector chain, generate and attach a TCP. For a single-branch topology \((b=1)\), the process is done. For multi-finger topologies, Splitters are generated for each common joint and chains are attached to them, from end-effector to root. Finally, the tree is attached to a first Splitter \((sp0)\) for topologies with no wrist, or to the serial chain of the first common edge in case of a wristed hand. Algorithm 4 shows this process.

### B. Exact synthesis

Exact dimensional synthesis has been explored in [23]. The approach followed to create dimensional synthesis equations consists on equating the forward kinematics of each root-to-fingertip branch in the hand to the set of positions defined for the fingertip. Given a set of \(m_p\) task positions \(P_i^k\), \(k = 1 \ldots m_p\) for each end-effector (denoted by superscript \(i\)), \(m_v\) task velocities \(V_i^r\), for each end-effector \(i\), \(r = 1 \ldots m_v\), and \(m_a\) task accelerations \(A_i^s\) for each end-effector \(i\), \(s = 1 \ldots m_a\), where \(m = m_p + m_v + m_a\), design equations are created. Compute the relative displacements from a selected reference position, usually position 1, and equate the relative forward kinematics to those relative positions \(P_i^1\). The twist of each end-effector \(V_i^r\) is equated to the linear combination of twists for each joint axes, and similarly for the acceleration of the end effectors. The Plucker coordinates of the joint axes appear explicitly in the forward kinematics when these are computed as the product of exponentials for relative displacements, and linearly in the velocity and acceleration equations. For a hand with \(b\) fingertips, this yields \(b\) sets of equations that are to be solved simultaneously.

### Table III

**Examples of solvability calculation**

| Topology | Solvability |
|----------|-------------|
| \(3R-(2R,R-(3R,3R,3R,3R))\) | Solvable \(m=7\) |
| \(p = (0, 1, 1, 2, 2, 2)\) | |
| \(j = (3, 1, 2, 1, 1, 1)\) | |
| \(R-(2R-(3R-(R,R,R,R,R,R)),\) | Not Solvable |
| \(2R-(3R-(R,R,R,R,R,R))\) | R-(R) overconstrained |
| \(p = (0, 1, 1, 2, 3, 3, 4, 4, 5, 5, 9, 9)\) | |
| \(j = (1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1)\) | |
| \(R-(R-(2R-(R,R,R,R,R,R)),\) | Solvable \(m=3\) |
| \(R-(R,R,R,R))\) | |
| \(p = (0, 1, 1, 2, 3, 3, 4, 4, 5, 5, 9, 9)\) | |
| \(j = (1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1)\) | |
| \(2R-(3R-(2R,2R,2R,3R))\) | Solvable \(m=5\) |
| \(p = (0, 1, 1, 1, 3, 3, 3)\) | |
| \(j = (2, 3, 1, 3, 2, 2, 2)\) | |

**TABLE III**

**Examples of solvability calculation**

| Topology | Solvability |
|----------|-------------|
| \(3R-(2R,R-(3R,3R,3R,3R))\) | Solvable \(m=7\) |
| \(p = (0, 1, 1, 2, 2, 2)\) | |
| \(j = (3, 1, 2, 1, 1, 1)\) | |
| \(R-(2R-(3R-(R,R,R,R,R,R)),\) | Not Solvable |
| \(2R-(3R-(R,R,R,R,R,R))\) | R-(R) overconstrained |
| \(p = (0, 1, 1, 2, 3, 3, 4, 4, 5, 5, 9, 9)\) | |
| \(j = (1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1)\) | |
| \(R-(R-(2R-(R,R,R,R,R,R)),\) | Solvable \(m=3\) |
| \(R-(R,R,R,R))\) | |
| \(p = (0, 1, 1, 2, 3, 3, 4, 4, 5, 5, 9, 9)\) | |
| \(j = (1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1)\) | |
| \(2R-(3R-(2R,2R,2R,3R))\) | Solvable \(m=5\) |
| \(p = (0, 1, 1, 1, 3, 3, 3)\) | |
| \(j = (2, 3, 1, 3, 2, 2, 2)\) | |
Algorithm 4 Automatic FK object attachment

if (no_general_wrist) then
    sp0=Create Splitter.
end if

endEffectors ← FINDENDEFFECTORS(parentpointer_array)
▷ endEffectors are those joints which are not pointed in
parentpointer_array
for all e IN endEffectors do
    Create chain.
    attach TCP to e.
    if (no_general_wrist) and (parentpointer_array[e]=0)
        then
            attach e to sp0.
        end if
end for

common_joints ← FIND_PALMS(parentpointer_array)
▷ palms or common joints are those joints which are attached
by more than 1 joint.
for all common_joints do
    Create chain.
end for

groups ← MAKEGROUP(parentpointer_array)
▷ group[i] contains those joints which are attached to common_joint[i]
for i ← #groups, 1, step(-1) do
    for all joints IN group[i] do
        attach joints to Splitter[i]
    end for
    attach Splitter[i] to common_joint[i]
end for

for all cj IN common_joints do
    if (no_general_wrist) and (parentpointer_array[cj]=0)
        then
            attach cj to sp0.
        end if
end for

if b=1 then
    ▷ topology is single branch
    SEND(endEffector[1])
end if
if no_general_wrist then
    ▷ topology is without wrist
    SEND(sp0)
end if
if NOT(no_general_wrist) then
    ▷ topology is with wrist
    SEND(common_joint[1])
end if

$\hat{P}_{ik} = \prod_{j \in \{B_i\}} e^{\frac{\Delta \theta^+_{j} + \Delta \theta^-_{j}}{2} S_j}$,

$V_i = \sum_{j \in \{B_i\}} S_j^i \dot{\theta}_{j}^i$,

$A_i^r = \sum_{j \in \{B_i\}} S_j^r \dot{\theta}_{j}^r + \sum_{j,h \in \{B_i\}} \dot{\theta}_{j}^r \dot{\theta}_{h}^r [S_j^r, S_h^r]$,

where the number of end-effectors, or branches as root-to-fingertip chains, is indicated by b, $m_p$ is the number of exact positions, and $\{B_i\}$ is the set of ordered indices of the joints belonging to branch i, which can be obtained from the graph matrices. The set of ordered indices $\{T_i\}$ and $\{R_i\}$ correspond to positions where twists and accelerations have been defined, for each branch i. Notice that some of the joints will be common to several branches. The joint axes at the reference configuration are denoted as $S_j$, and the joint axes at the configuration given by position k are denoted as $S_j^k$.

This yields a total of $6(m_p - 1 + m_v + m_a)b$ independent equations to be simultaneously solved. The method has been applied to simultaneous rigid-body motion tasks for all fingertips (24), defined by a finite set of positions, and to simultaneous fingertip tasks defined by a finite set of displacements and associated twists. For most topologies, this method yields many potential designs.

C. Multiple velocity synthesis or constrained-motion synthesis

For tasks aiming to define a free trajectory for each fingertip, the definition of a finite set of positions, with a single twist vector defining the velocity for each position, and possibly a single acceleration 6D vector, gives a full characterization of the task. However for tasks which are constrained by the contact between the fingertip and an object, the definition of the allowed subspace of velocities at each point can be used to ensure the desired behavior for some grasping actions such as finger sliding or finger rolling, for a suited hand topology. Notice that the velocities must always be defined at a given position of the end-effector.

Consider the desired angular velocity of the end-effector and linear velocity of the origin of the end-effector frame at a given position, and calculate the fixed-frame six-dimensional twist. In this twist, the point velocity is calculated at the origin, so that it would yield the desired linear velocity at the origin of the end-effector frame.

A constrained motion given by a contact is defined, at a given position, as a subspace of wrenches, $W$, whose magnitudes can be as high as needed. The subspace of reciprocal twists, $V$, define the potential directions of allowed motion at that position.

A fingertip in contact with a surface can be kinematically modeled using one of the standard fingertip joints, see for instance (11), such as pointy fingers or soft fingers, which are defined by their degrees of freedom and friction cone if applicable. For a general case, the dimension of the subspace of reciprocal twists can be made to coincide with the mobility of the parallel mechanism formed when the hand is in contact with an object (defined by $n$ links and $j$ joints of $f_i$ degrees of freedom each),

$$\dim(V) = 6(n - 1) - \sum_{i=1}^{j} (6 - f_i).$$

Using this method, a hand can be synthesized for a desired $m$-dimensional subspace of twists at each precision position, just by defining a set of $m$ independent twists at that position.
As an example, consider a hand with a $2 - (2, 2)$ topology, with two fingers and soft finger joints at the fingertip. Figure 6 shows the graph and kinematic sketch of the $1 - (4, 4)$ hand. This hand has two revolute joints at the wrist and two revolute joints at each of the two fingers.

![Image](image_url)

Fig. 6. The 2-(2,2) hand: Up, topology; down, kinematic sketch.

This topology is solvable for a total of $m = 5$ precision positions. Define a task with $m_p = 3$, with one position having two specified twists each ($m_v = 2$), and the rest of positions having no specified velocities. If this is a task in which both fingers touch an object, and assuming a general grasp, the $2 - (2, 2)$ hand has 4 degrees of freedom according to the general mobility formula. Two of them corresponds to the wrist rotations, while the other two are in-hand degrees of freedom. This allows us to include in the task the ability of the fingers to be compatible with a contact constraint at a given position.

It is possible to state the synthesis problem with as many velocities as desired, as long as they are compatible with the conditions in Eqs. (1) and (2).

VII. KINEMATIC SOLVER

The algorithms presented in this work have been implemented in a kinematic design software. A first version of the solver for dimensional synthesis, ArtTreeKS (Articulated Tree Kinematic Synthesis), was developed [24] for tree topologies with a single branching, corresponding to anthropomorphic or simple hands. A single root is sought for the system of equations using a hybrid solver, based on a Genetic Algorithm (GA) built on top of a Levenberg-Marquadt local optimizer, which minimize the average error of the dual quaternions representing the task. This numerical solver yields a single solution but allows dealing with tree topologies with a very high number of joints and fingers.

As all meta-heuristic algorithms, this genetic algorithm must be adjusted experimentally according to the problem being solved. Each entity in the genetic algorithm is represented as a vector of real numbers that allows simple integration with other numerical solver libraries like MINPACK [14] which is used in the Hybrid solver.

This numerical solver has been integrated in the kinematic design package. The package includes a type synthesis stage, solvability checking, the ability to synthesize new designs with arbitrary branching stages (corresponding to hands with several palms), and the ability to define a task with positions and several velocities or accelerations at a given position. This second feature is important in order to fully determine manipulation actions such as finger rolling or finger sliding, or simple dexterity without changing the grasping point.

VIII. OVERALL SOFTWARE ARCHITECTURE

The software implementation follows a three-layer architecture, which is shown in Figure 7 and uses the elements described below. The user interface and writing of input files is done using Lua, while the solver is programmed using C++.

- Input Files: Input information from the designer.
  - Type synthesis input: contains the number of branches, task positions and edges.
  - Dimensional synthesis input: Contains the tree topology and the values for the task positions, velocities and accelerations.
- Output files: Results of calculations that the designer can access.

![Software Architecture Diagram](image_url)

Fig. 7. Software architecture.
· Solvability output: output file with the results of solvability calculations for a given topology.
· Type synthesis output: output file with the atlas of solvable topologies for a given number of fingers and positions.
· Design Synthesis File: output file containing the design equations.
· Dimensional synthesis output: output file containing the result of the dimensional synthesis: Plucker coordinates of joint axes and joint variables.

- Process files: internal calculations
  · Solvability Library: Lua functions to calculate tree solvability.
  · Type synthesis Library: Lua file to construct all possible topologies for a set of input conditions.
  · Generator File: Lua functions to check solvability, assemble forward kinematics equations and assign initial values.
  · Synthesis Library: Library of functions to communicate solver and Synthesis file.
  · Solvers: Genetic algorithm and Minpack C++ code to generate candidate solutions and perform minimization.

**IX. DESIGN EXAMPLE**

As an illustration for the overall design process, let us consider a hand task that can be defined with five positions of each fingertip. For this task we can use a minimum of three fingers and a maximum of five fingers, for grasping and manipulation purposes. Three fingers may be sufficient for stable grasping but adding the extra two fingers may help in some manipulation strategies.

We start the design process defining \( m_p = 5 \) number of task positions, three to five branches \( b = 3 \) to \( b = 5 \), and we limit the number of edges to the interval from \( e = 1 \) to \( e = 9 \) in order to have a bounded search and to limit the complexity of the design.

Applying Algorithms 1 and 2 we find, for \( b = 3 \), a total of 29 solvable non-isomorphic topologies. For \( b = 4 \) there are 134 solvable topologies, and for \( b = 5 \) we find a total of 728 solvable topologies. For \( b = 4 \) and \( b = 5 \) fingertips, we notice that the minimum number of edges for the solvable topologies is \( e = 4 \) and \( e = 5 \) respectively. In order not to complicate the design too much, we limit the search to a maximum of \( e = 5 \) number of edges. Table IV shows the solutions of the type synthesis stage for 3 fingertips, and Table V contains the solvable topologies for 4 and 5 fingertips and up to 5 edges.

The simplest solvable topology able to perform this task has parent-pointer array \( p = \{0, 0, 0\} \) and joint array \( j = \{3, 3, 3\} \), that is, a 0 - (3, 3, 3) topology with three R-R-R fingers and no wrist. Some of the most complex topologies are the 3 - (2, 1 - (1, 2)) topology, with \( p = \{0, 1, 1, 2\} \) and \( j = \{3, 2, 1, 1, 2\} \), or the 0 - (3, 2 - (2, 2, 3)), with \( p = \{0, 0, 1, 1, 1\} \) and \( j = \{2, 3, 2, 2, 3\} \).

Out of the 45 candidate topologies, we select for the design the topology with \( p = \{0, 1, 1, 2, 2\} \) and \( j = \{2, 1, 2, 2, 2\} \), corresponding to the 2 - (1 - (2, 2), 2) hand, with two R joints at the wrist spanning two fingers, the first one spanning two more fingers for a total of three end-effectors. Figure 8 shows the graph of this topology.

Dimensional synthesis is used to shape this topology with five random finite displacements. The resulting set of equations from Eq. (5) consists of 96 highly nonlinear equations in 90 unknowns, 72 of which are independent. The results of five
runs with different initial conditions are presented in Table VI. All five obtained solutions were feasible and different, which makes us infer that there will be a big number of solutions for this topology.

**TABLE VI**

**DIMENSIONAL SYNTHESIS SOLVER RESULTS**

| Run | Final error | Iterations | Running time |
|-----|-------------|------------|--------------|
| 1   | $1.15 \times 10^{-13}$ | 1          | 12 sec.      |
| 2   | $6.0 \times 10^{-12}$   | 2          | 10 sec.      |
| 3   | $1.4 \times 10^{-13}$   | 1          | 5 sec.       |
| 4   | $2.0 \times 10^{-13}$   | 9          | 29 sec.      |
| 5   | $1.0 \times 10^{-13}$   | 1          | 7 sec.       |

The solutions obtained with the dimensional synthesis solver have been modeled using the automatic drawing procedure developed in [4] and are presented in Figure 10 and Figure 11. The positions used for this design are presented in Figure 9.

The output of the kinematic synthesis stage is to be used as the input for a detailed design, using computer-aided tools. This work presents the development and implementation of the different stages of kinematic design within a tool for the creation of innovative multi-fingered robotic hands. The resulting design package is able to perform type and dimensional kinematic synthesis for arbitrary tree topologies, enumerating candidate topologies and creating wristed hands with arbitrary number and type of fingers and arbitrary number and type of branchings. The solver accepts several inputs, basically a kinematic task and some limits on the desired topologies such as number of fingers or some bounds on the number of edges. The kinematic task may include finite displacements of each fingertip, and multiple velocities and accelerations for the fingertips at some of those finite positions.

Type synthesis and solvability are implemented using an enumeration technique which constructs non-isomorphic trees. The implementation of the dimensional synthesis combines the automatic construction of the tree forward kinematics with a solver consisting of a genetic algorithm and a Levenberg-Marquardt stage in order to explore the space of solutions, and shows fast convergence to a solution for each run. The current version of the solver is freely available at the project webpage. Future work will focus on generalizing some other features of the solver and on the automatic connection to subsequent stages in the design process.

The output of the design process is a kinematic design: a set of joint axes, defined by their Plucker coordinates at a reference configuration, and a set of joint variables and
Fig. 11. 2-(1-(2,2),2) hand design for the specified positions.

Joint rates. Each kinematic design can be implemented in a final design in an unlimited number of ways, selected by the designer and constrained by additional specifications. The rationale is that a hand design tailored to an application may simplify many other aspects of the process, increasing the success of the grasping and manipulation actions.

REFERENCES
[1] M. Ceccarelli and M. Zottola. Design and simulation of an underactuated finger mecanism for larm hand. *Robotica*, 2015.
[2] A. Ciocarlie and P. Allen. Data-driven optimization for underactuated robotic hands. In *Proc. of the 2010 International Conference on Robotics and Automation*, pages 1292–1299, Anchorage, Alaska, USA, 2010.
[3] M. Controzzi, C. Cipriani, and M.C. Carrozza. The Human Hand as an Inspiration for Robot Hand. In R. Balasubramian and V.J. Santos, editors, *Design of Artificial Hands: A Review*, volume 95 of *Springer Tracts in Advanced Robotics*, pages 219–244. Springer Switzerland, 2014. ISBN 978-3-319-03016-6.
[4] N. Hasanzadeh, X. He, and A. Perez-Gracia. A design implementation process for robotic hand synthesis. In *ASME Int. Design Engineering Technical Conferences*, 2015.
[5] J.M. Hervé. Analyse structurelle des mécanismes par groupe des déplacements. *Mechanism and Machine Theory*, 13(4):437 – 450, 1978. ISSN 0094-114X. doi: DOI:10.1016/0094-114X(78)90017-4. URL. [http://www.sciencedirect.com/science/article/pii/0094114X78900174](http://www.sciencedirect.com/science/article/pii/0094114X78900174)
[6] A.D. Jaggard and J.J. Marincel. Generating tree isomorphisms for pattern-avoiding involutions. *Ann. Comb.*, 15: 437–448, 2011.
[7] X. Kong and C. Gosselin. Type synthesis of 3t1r 4-dof parallel manipulators based on screw theory. *IEEE Transactions on Robotics and Automation*, 20(2):181–190, 2004.
[8] J.J. Lee and L.W. Tsai. Structural synthesis of multi-fingered hands. *Journal of Mechanical Design*, 124:272–276, 2002.
[9] A. Makhal and A. Perez-Gracia. Solvable multi-fingered hands for exact kinematic synthesis. In *Advances in Robot Kinematics*, Ljubljana, Slovenia, June 2014.
[10] J. Martin and M. Grossard. Design of a fully modular and back drivable dexterous hand. *The International Journal of Robotics Research*, 33(5):783–798, 2014.
[11] M.T. Mason. Mechanics of Robot Manipulation. The MIT Press, 2001.
[12] M.T. Mason, S.S. Srinivasa, A.S. Vazquez, and A. Rodriguez. Generality and simple hands. *International Journal of Robotics Research*, 2010.
[13] E. Mattar. A survey of bio-inspired robotics hands implementation: New directions in dexterous manipulation. *Robotics and Autonomous Systems*, 61(5), 2013.
[14] J.J. More, B.S. Garbow, and K.E. Hillstrom. User guide for minpack-1. *Argonne National Laboratory Report*, ANL-80-74, 1980.
[15] L.U. Odhner and et al. A compliant, underactuated hand for robust manipulation. *The International Journal of Robotics Research*, 33(5):736–752, 2014.
[16] E. Ozgur, G. Gogu, and Y. Mezouar. Structural synthesis of dexterous hands. In *Intelligent Robots and Systems (IROS) Conference*, 2014.
[17] J.E. Parada Puig, N.E. Nava Rodriguez, and M. Ceccarelli. A methodology for the design of robotic hands with multiple fingers. *Advanced Robotic Systems*, 5(2): 177–184, 2008.
[18] MA. Pucheta and A. Cardona. Topological and dimensional synthesis of planar linkages for multiple kinematic tasks. *Multibody System Dynamics*, 29(2):189–211, February 2013.
[19] M. Quigley, C. Salisbury, A.Y. Ng, and J.K. Salisbury. Mechatronic design of an integrated robotic hand. *The International Journal of Robotics Research*, 33(5):706–720, 2014.
[20] N. Robson, J. Allington, and G.S. Soh. Development of Underactuated Mechanical Fingers Based on Anthropometric Data and Anthropomorphic Tasks. Buffalo, USA, 2014. ASME.
[21] J.K. Salisbury and B. Roth. Kinematic and force analysis of articulated mechanical hands. *Journal of Mechanisms, Transmissions and Automation in Design*, 105(1):35–41, 1983.
[22] J. M. Selig. *Geometric Fundamentals of Robotics (Monographs in Computer Science)*. SpringerVerlag, 2004. ISBN 0387208747.
[23] E. Simo-Serra and A. Perez-Gracia. Kinematic synthesis using tree topologies. *Mechanism and Machine Theory*, 72 C:94–113, 2014.
[24] E. Simo-Serra, F. Moreno-Noguer, and A. Perez-Gracia. Design of non-anthropomorphic robotic hands for anthrop-
pomorphic tasks. August 28-31, 2011.

[25] E. Simo-Serra, A. Perez-Gracia, H. Moon, and N. Robinson. Design of multi fingered robotic hands for finite and infinitesimal tasks using kinematic synthesis. In *Advances in Robot Kinematics*, Innsbruck, Austria, June 2012.

[26] D.B. Sterusand and C.J. Turner. A design methodology based process for robotic gripper design. In *Proc. of the 2011 ASME International Design Engineering Technical Conferences, IDETC-CIE*, Washington D.C., USA, August 28-31, 2011.

[27] Lung W. Tsai. *Mechanism Design: Enumeration of Kinematic Structures According to Function*. CRC Press, Boca Raton, 2001.

[28] Q. Zeng and Y. Fang. Structural synthesis and analysis of serial–parallel hybrid mechanisms with spatial multi-loop kinematic chains. *Mechanism and Machine Theory*, 49:198–215, March 2012.