Proceedings of the 2005 IEEE
International Conference on Robotics and Automation
Barcelona, Spain, April 2005

Kinematic Redundancy in Robot Grasp Synthesis. An Efficient Tree-based Representation

César Fernández, Óscar Reinoso and Asunción Vicente
System Engineering and Automation Division
Miguel Hernandez University
Av. Universidad s/n, 03202 Elche (Alicante) Spain
{c.fernandez, o.reinoso, suni}@umh.es

Rafael Aracil
UPM-DISAM
Polytechnical University of Madrid
José Gutiérrez Abascal 2, 28006 Madrid, Spain
aracil@etsii.upm.es

***Abstract* - A redundancy resolution technique devoted to grasp synthesis is presented. Given a set of contact points and a certain robot arm and gripper, the goal is to select both the best assignment of gripper fingers to contact points and the best joint values that allow the fingers to reach such contact points. The system proposed is based on the generation of an inverse kinematics tree where fast searches can be performed in order to find the optimum configuration. Optimality is defined as similarity to previously stored examples over a hierarchical structure of configuration data, which includes finger assignments and robot joints.**

Index Terms* - *Grasp synthesis, inverse kinematics, redundancy resolution, machine learning..

I. Introduction

New robotic applications require robots to grasp previously unknown objects in non-structured environments; service robotics [1] or partially automated teleoperation (also known as collaborative control) [2] are among these new applications. In order to perform correct grasps autonomously, a certain degree of intelligence is required in robot controllers. Information about the objects to be grasped is obtained through different sensors, mainly video cameras or range sensors, and the robot has to process such information and decide which are the optimum contact points where to place the gripper fingers on the object. Besides, an inverse kinematics problem has to be solved in order to command the robot arm and gripper joints so that the fingers or end effectors reach the previously defined contact points.

The whole problem is referred to as *robot grasp synthesis* and has been addressed by many authors. Normally, the focus is centered on the selection of the optimum contact points, according to different criteria: fulfillment of form or force closure conditions [3], maximization of additional quality measures [4], similarity to previously stored examples [5], etc. However, the selection of the optimum contact points represents only a partial solution of the problem: once they are selected, a feasible configuration (i.e. joint values) has to be computed for both robot arm and hand so that the end effectors are able to reach the previously selected contact points. This is the problem addressed in the present paper.

II. Problem Statement

When attached to a robot arm, robot hands are highly redundant devices (except very simple grippers like two jaw parallel ones) so that a certain set of contact points can be reached by multiple different sets of joint values. A redundancy resolution technique has to be used in order to select the best configuration.

The problem can be stated as follows: given a certain set of n contact points on the surface of an object, which will be vectors in $\Re^3$ and can be denoted as P in (1); the goal is to select the optimum joint values for a certain robot arm and hand.

$$P = \{ p_1, ..., p_n \},\ p_i \in \Re^3 \forall i\ . \tag{1}$$

The robot arm is considered to have m_a DOF and the robot hand is supposed to be equipped with n fingers (one per contact point, pinch grips will be assumed) each of them with m_i DOF. So, in the general case, the joint space is m-dimensional, where m is defined in (2), and a certain configuration is defined by its m joint values, as Q in (3) where q_i represents the *i-th* joint of the robot arm and q_{ij} represents the *j-th* joint of finger *i.* Each joint can be rotational or translational.

$$m = m_a + \sum_{i=1}^{n} m_i\ . \tag{2}$$

$$Q = \{ q_1, ..., q_{m_a}, q_{11}, ..., q_{1m_1}, ..., q_{n1}, ..., q_{nm_n} \},\quad q_i, q_{ij} \in \Re \forall i, j. \tag{3}$$

It must be stated that, apart from the inherent redundancy of each finger of the robot gripper (which can reach a certain position with multiple different sets of joint values) there is another source of redundancy: the assignment of contact points to gripper fingers is not fixed. In this sense, all the possible combinations have to be considered. For a n finger robot hand reaching n contact points, the number of possible assignments equals $n!$, even though most of them will be unfeasible due to kinematical restrictions or collisions.

III. Possible Approaches

The first option that has to be considered is to look for a closed form solution of the inverse kinematics problem; i.e. to find a certain function f so that $f(P)=Q$. However, this closed form solution can only be obtained for non-redundant robots with special geometries. When there is redundancy, a commonly used solution is to add constraints for the redundant DOF; i.e. to held fixed a certain joint or to establish some fixed relationships between different joints. However, these solutions decrease the dexterity of robots and grippers: some of the available DOF are not exploited.

A different approach is based on the use of iterative methods to approximate a good solution, normally based on

the Jacobian matrix *J*. In a general case, when there are multiple end effectors (e.g. grasping devices) the Jacobian matrix is defined as in (4), where p_i denotes the *i-th* end effector position (depending on the applications, p_i can represent both the end effector positions and orientations) and q_j denotes the *j-th* robot joint.

$$J(Q) = \left(\frac{\partial p_i}{\partial q_j} \right)_{i,j} \tag{4}$$

Inverse kinematics resolution based on the Jacobian matrix can be accomplished in many different ways: the Jacobian transpose method [6], the Jacobian pseudoinverse or null-space method [7], the damped least squares method [8], etc. Each approach offers different advantages; e.g. the null-space method allows solving redundancy by fixing secondary goals, as to avoid joint limits or to minimize the movement of a certain joint. However, none of these approaches is devoted to grasping processes and they do not take into account the grasp assignment problem; i.e. which finger should be assigned to each contact point.

Other redundancy resolution methods are based on parametric modeling rather than in the Jacobian matrix. In [9], an approach based on human motor control theories is presented. A taxonomy of robot motions is generated and, for each motion example, the joint values are computed in an off-line step called *skill acquisition*. When the robot is requested to perform a certain motion, function approximators are used to interpolate the joint values corresponding to the desired motion from the available skills. The main advantage of this method is its low on-line computational complexity. However, the same problem found with the Jacobian based methods persists: the skill based approaches are not devoted to grasping processes and the assignment problem is not addressed.

Some other approaches are specific to grasping processes, among them the one presented in [10], where grasps are computed starting from *generalized prototypes* or grasp primitives. These prototypes are adapted to the particular object being grasped. Somehow, these prototypes can be considered as *robot skills* following Dordevic's nomenclature [9]. Even though this approach solves the assignment problem, it can not work for any set of contact points, but only for a set of contact points that have been previously generated according to a certain grasp prototype. In this sense, it can not be considered general enough.

IV. Proposed Approach

The proposed approach for redundancy resolution is devoted to robot grasping processes, although it can be extended to other fields of application. Three main points constitute the basis of this approach:

- Imitation of human behavior, so that the system learns from grasp examples provided by the user.
- Establishment of a hierarchy of robot joints; the goal is to give priority to those joints more relevant to grasp quality.
- A novel tree-based representation of the robots inverse kinematics which allows fast search procedures.

Though general, the proposed approach requires a certain adaptation to the particular robot arm and hand used. The following paragraphs describe the off-line and on-line processes involved.

A. Off-line processing

As an off-line process, grasp examples of different objects are stored. These examples can be given by teleoperation of the robot with a master device or through a simulation environment. The data recorded for each grasp example includes local and global features of the user selected contact points (related to the geometry of the object to be grasped); the finger assignments; and the set of robot arm and hand joint values. The size of the data vector stored for each grasp example is shown as f_T in (5), where *m* represents the total number of joints, *n* the number of contact points (and the number of assignments), f_L the number of local features and f_G the number of global features. Local and global features for a certain set *S* are represented by the real valued vectors L^S and G^S in (6) and (7) respectively; where L^i corresponds to the local feature vector of the *i-th* point of the set; l_j^i corresponds to the *j-th* local feature of the *i-th* point; and g_j^S corresponds to the *j-th* global feature of the set *S*.

$$f_T = n \cdot f_L + f_G + n + m\,. \tag{5}$$

$$L^S = \left\{ L^{p_1}, L^{p_2}, ..., L^{p_n} \right\} = \left\{ l_1^{p_1}, ..., l_{f_L}^{p_1}, ..., l_1^{p_n}, ..., l_{f_L}^{p_n} \right\}. \tag{6}$$

$$G^S = \left\{ g_1^S, g_2^S, ..., g_{f_G}^S \right\}. \tag{7}$$

Local features are computed for each contact point, and they include the distance to the center of mass of the object and a multiresolution measure of convexity. Global features are computed using as a reference point the centre of the convex hull defined by all the contact points, and include the distance from this reference point to the centre of mass of the object and a multiresolution measure of the angle between the normal at the contact point and the line directed to the reference point. Details about these local and global features can be found in our previous work [11]; in such work, these features are not only used for redundancy resolution but also for autonomously selecting the contact points on the surface of an object. Any other set of features would be valid for the proposed approach provided that they fulfill the following properties:

- Their computation is fast enough
- They provide insight about the kind of grasp being performed.

B. On-line processing

The on-line process starts when a robot configuration needs to be computed for a certain set of contact points. This process includes the generation of the tree and the search for the best inverse kinematics solution among those contained in the previously generated tree.

1) Tree generation: The goal is to generate an *n+m* depth tree, where *n* equals the number of fingers and *m* equals the total number of DOF. Each of the *n-1* upper levels of the tree below the root node corresponds to a certain gripper finger, and each of the *m* following tree

levels corresponds to a certain joint of the robot, where both the different fingers and the different joints are ordered according to a previously defined hierarchy. This hierarchy should reflect the relevance of each finger and joint to grasp quality. The relevance can be established following any application specific criteria. Normally, those joints closer to the contact point (closer to the end effector) should be considered more relevant, as Fig. 1 shows. However, any other criteria would also be applicable.

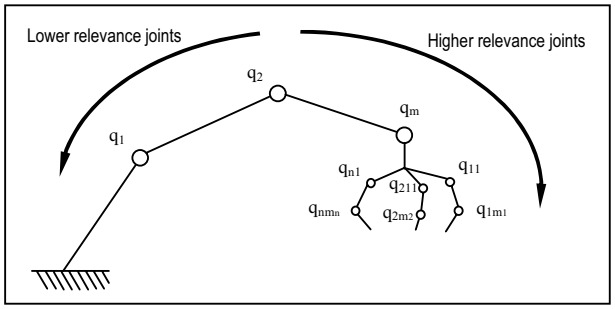

Fig. 1. Relevance of the different arm and hand joints.

For understandability reasons, the following paragraphs describe the generation of the whole inverse kinematics tree; afterwards it will be explained how only a small subset of the tree needs to be computed, thus highly reducing the computational load.

First, the generation of the upper subtree (the one corresponding to the first *n* levels, one per robot finger) will be explained. Once the hierarchy is established, this subtree is generated starting from a root node. A total of *n* child nodes have to be added to the root node, corresponding to the *n* possible assignments of finger number 1. For each of the *n* nodes of level 2, *n-1* descendants are created, corresponding to the *n-1* possible assignments of finger 2, and the process continues until finger *n-1*, which may have only two possible assignments. Fig. 2 represents an example of these first *n* levels of the tree for a three fingered robot gripper, where the second level corresponds to the first finger (which has 3 possible assignments), and the third level corresponds to the second finger (2 possible assignments). The last finger does not appear in the tree as it has only one possible assignment.

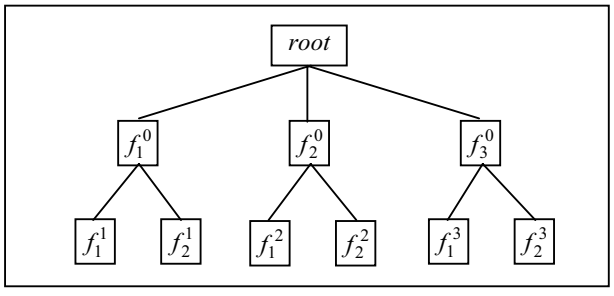

Fig. 2. Example of the n upper levels of an inverse kinematics tree.

Each node in the tree is represented as f_i^j where the superscript refers to the parent node and the subscript allows distinguishing between all the descendants of a certain node (node f_i^j is the *i-th* descendant of the *j-th* node of the previous level). More than one superscript is needed if more levels are added to the tree; the nodes at the first level are considered child nodes of the root node or node 0. Each path from the root node to a leaf of this subtree represents a possible set of finger assignments.

The remaining *m* levels of the whole tree correspond to each of the *m* DOF of the robot arm and hand. To explain how these remaining tree nodes are generated, let us start at one of the leaves of the high level subtree, which will be considered as a root node for one of the lower level subtrees. The subtree is generated starting from the most relevant robot joint. Focusing in this first joint, a partial inverse kinematics analysis is performed trying to find the joint values for which a solution exists. Different results can be obtained:

- The inverse kinematics can only be solved for a certain joint value. This means that there is no redundancy in this particular joint, and only one tree node is added at this level, which will contain the previously computed joint value.
- The inverse kinematics can be solved for a finite number of different joint values. There is redundancy, and a node should be added for each of the valid values.
- The inverse kinematics can be solved for infinite different joint values (ranges of joint values). There is a high redundancy, and the valid ranges need to be discretized in order to add as many nodes as valid joint values. The resolution used should make a justifiable compromise between accuracy and computational load, and can be chosen on an application specific basis.

Once the first subtree level has been generated, the process continues with the next robot joint in the hierarchy. For each of the nodes created in the previous level, a new partial kinematics analysis is performed. Once again, the goal is to detect the values of the second joint for which a solution exists. Let us assume that the first subtree level consists of k_1 nodes; then, k_1 inverse kinematics problems have to be solved in this step of the algorithm, with *m-1* DOF (one of the joints has now a fixed value). Depending on the results of this analysis, one or multiple child nodes are added. The process continues with the following levels, until the lowest joint in the hierarchy is analyzed. The results of this last analysis correspond to the tree leaves. The number of tree leaves shows the number of possible solutions to the inverse kinematics (to the resolution level used in the discretizations). Even though the whole tree does not need to be computed (as will be explained in the next section) it must be stated that generating a full tree does not represent a very high computationally load as the number of DOF decreases at each tree level and the redundancy is also expected to be reduced when going deeper in the tree.

Pseudocode for the lower level subtree generation algorithm is shown in Fig. 3 (the generation of the upper level subtree is straightforward) where the *kinematics* function is dependant on the particular robot arm and hand used; the *sort_joints* function depends on the relevance criteria used, and the remaining functions are generic.

An example of low level subtree is shown in Fig. 4, where the joint value corresponding to each node is

represented as $q_i^{i_1,i_2,\ldots,i_k}$: the superscripts give information about the parent nodes up to the root (a node at level k will have *k-1* parent nodes) and the subscript allows to distinguish between child nodes of the same parent. Different situations are shown in the example: the initial inverse kinematics (first joint, second tree level) gives infinite solutions, which have been discretized in k_1 values. Looking at the third tree level, different results are obtained for the second joint, which depend on the value of the parent node (parent joint): for the first value of joint 1, there are infinite solutions for joint 2, which are discretized to k_2 solutions; for the second value, only one solution is found; and for the k_1-*th* value, a finite number of solutions (2 solutions) are valid. Going deeper in the tree, the remaining joints can take only one value; this is a common situation as the number of DOF is highly reduced at these tree levels.

```
sort_joints(relevance)
create_root_node()
TreeLevel := 1
LevelNodes := 1
repeat
        ParentNode := 1
        UpperLevelNodes := LevelNodes
        LevelNodes := 0
        repeat
                ValidJointValues := kinematics(TreeLevel, ParentNode)
                add_child_nodes(ParentNode, ValidJointValues)
                LevelNodes := LevelNodes + count(ValidJointValues)
                ParentNode := ParentNode + 1
        while ParentNode <= UpperLevelNodes
        TreeLevel := TreeLevel + 1
while TreeLevel <= Joints
```

Fig. 3. Lower subtree generation algorithm.

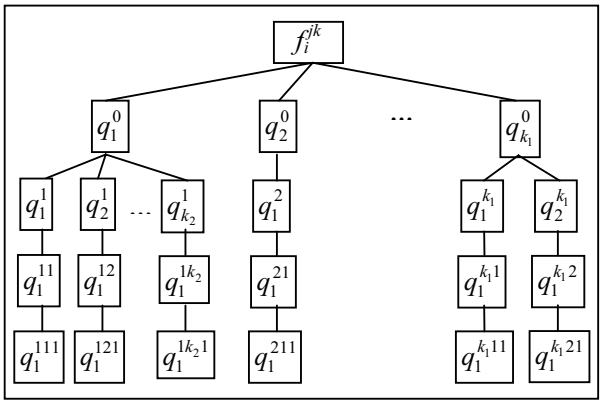

Fig. 4. Example of the m lower levels of an inverse kinematics tree.

2) Tree search: Once the tree has been generated, different search strategies can be used to look for the configuration closer to the examples. The first question to be answered is how to measure this similarity.

Let us consider that the user has performed n_e grasp examples of different objects. The goal is, first, to select the grasp example whose contact points are more similar to those for which the inverse kinematics needs to be computed; i.e. to select the most similar *grasp type* among the examples. For this purpose, the previously mentioned local and global features are used; and a simple L_2 norm is computed, with a previous normalization of the feature values in order to avoid preponderances of features with higher absolute values. The distance between two different sets of n points (S_1, S_2) in the feature space is computed using a L_2 metric, and it is represented as d_F in (8), where $\hat{l}$ and $\hat{g}$ represent normalized local and global feature values. The normalization used, which is described in (9) only for local features in order to avoid redundancy, fits every feature to the range [0, 1] by subtracting the minimum value and dividing by the difference between the maximum and the minimum value. It is a common normalization for multivariable distance measurement, particularly used in nearest neighbor classifiers [12]. A different normalization could have been used, in order to set every feature to have zero mean and unit variance, with similar results.

$$d_F(S_1,S_2)=\sqrt{\sum_{i=1}^{f_L}\left(\hat{l}_i^{S_1}-\hat{l}_i^{S_2}\right)^2+\sum_{i=1}^{f_G}\left(\hat{g}_i^{S_1}-\hat{g}_i^{S_2}\right)^2}\ . \tag{8}$$

$$\hat{l}_i^{S_j}=\frac{l_i^{S_j}-\left[min\left(l_i^{S_k}\right)\forall k\right]}{\left[max\left(l_i^{S_k}\right)\forall k\right]-\left[min\left(l_i^{S_k}\right)\forall k\right]}\ . \tag{9}$$

Computing the closest example is straightforward once the distance has been defined; however, and in order to help the system find solutions for the inverse kinematics, not only the closest example but the T closest examples are used for redundancy resolution (T can be adjusted depending on each particular application).

Once the T closest examples are selected, the tree can be searched. The goal is to find in the tree a robot configuration as close as possible to that of one of the examples. There are several possibilities, among them the brute force approach which will perform a full search, either depth-first or breadth-first. The result will obviously be the best match, but computational cost will be extremely high. A heuristic search is proposed instead; it is not guaranteed to find the best match, but computational load is highly reduced as the tree is traversed only once from the root node to the selected leaf.

The heuristic search proposed works as follows: starting from the root node of the tree, it has to be decided which child node to select at the first tree level. Child nodes at this level represent all the possible assignments of finger 1. In order to select the most similar assignment to those of the recorded examples, local features are used: features related to a single contact point. In this case, the space is f_L-dimensional (f_L local features), so the distance between two contact points in the configuration space can be expressed as d_C in (10), using again a L_2 metric.

$$d_C(p_1,p_2)=\sqrt{\sum_{i=1}^{f_L}\left(\hat{l}_i^{p_1}-\hat{l}_i^{p_2}\right)^2}\ . \tag{10}$$

However, distances in the configuration space and distances in the feature space have to be considered simultaneously; when selecting between candidate nodes, two criteria have to be fulfilled:

- The configuration value of the node (joint value or assignment) should be as similar as possible to that of one of the selected examples.

- The local and global features of such example should be as similar as possible to those of the contact points under consideration.

Storing not only the most similar example but the *T* most similar ones gives us more chances to find a good solution for the inverse kinematics, but distance computation becomes more complex. A parameterized distance measure is proposed, where both effects can be weighted according to a parameter (α); such distance is represented as d_T in (11). The factors $n \cdot f_L + f_G$ and f_L have been introduced in order to obtain equal weights for both effects when the parameter α is set to 0.5.

$$d_T\left(S_1, S_2, q_i^1, q_i^2\right) = \\ = \frac{\alpha}{n \cdot f_L + f_G} \cdot d_F\left(S_1, S_2\right) + \frac{1-\alpha}{f_L} \cdot d_C\left(p_1, p_2\right). \quad (11)$$

Once the distance has been defined, its application in order to select a certain node in the first level of the tree is straightforward: it suffices to compute the distance between the *T* examples and each of the nodes at the first level: the minimum distance node should be selected.

For subsequent levels, a simpler measure distance has to be used: once an example has been selected (among the set of *T* examples) it should be kept for the remaining levels of the tree; i.e. the configuration most similar to that of the selected example will be searched for. So, subsequent comparisons up to the first *n* levels of the tree will use the configuration distance shown in (10). For comparisons at deeper levels, the distance measure to be used is slightly different. At these levels the information stored in each node does not represent finger assignments but robot joint values. The joint space is *m*-dimensional (*m* joints) but only one joint is considered at a time, so that a 1-dimensional distance measurement has to be computed; this can be as simple as a L_1 norm computation. Such simple distance measure between two different values for a certain joint *i* is defined as d_J in (12).

$$d_J\left(q_i^1, q_i^2\right) = \left|q_i^1 - q_i^2\right|. \quad (12)$$

The proposed search is extremely fast compared to a full search and, even though this heuristic search may not find the optimal match, the hierarchy of joints considered when building the tree ensures that the possible deviations from the optimal match will happen at lower levels of the tree, where the joint values have little relevance.

At this point, it has to be considered that it is not necessary to build the complete tree: building and searching can be performed simultaneously (as the proposed search strategy never performs backtracking) and child nodes need only to be computed for the selected nodes of the previous level. In this way, the whole process can be performed with a low computational load.

Besides, some intermediate or hybrid solutions can also be accomplished: up to a certain level, a full search and full generation of the tree may be performed, and then the heuristic approach can be used for the lower levels of the tree. Such solution is selected for the application example presented in the next section: a full search is performed in the upper tree levels (those corresponding to the finger assignments) thus resulting in a set of leaf nodes; and then heuristic searches are performed in the lower tree levels, starting at each previous leaf node.

V. Application Example

As an example, the proposed grasp redundancy resolution method will be used to select the optimum configuration in a 2D planar problem. A three-fingered articulated gripper like the one shown in Fig. 5 is selected. The gripper has 4 DOF: 3 of them translational (opening and closing of the fingers) and one rotational (abduction of the two articulated fingers, which are coupled). As it can be seen in the figure, multiple different grasps can be performed with such gripper, given three contact points on the object surface. First, three different finger assignments can be chosen (rows of the figure) and then, for each assignment it is possible to reach the contact points with different abduction angles (columns of the figure).

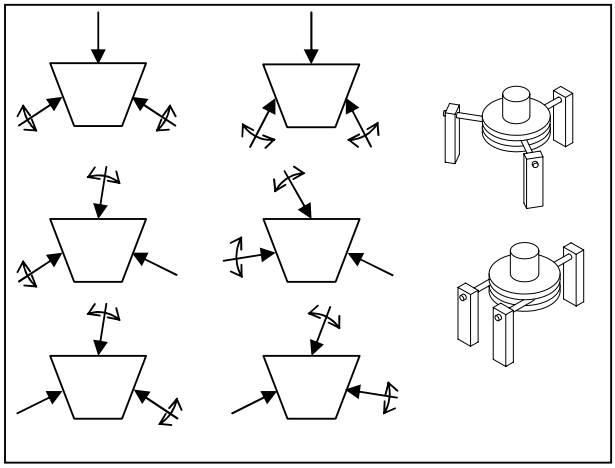

Fig. 5. Redundancy of the three-fingered gripper used as an example.

The gripper is supposed to be attached to a SCARA robot arm, which in a 2D representation adds 3 DOF. Fig. 6 shows the planar structure of arm and gripper. The joints are numbered from 1 to 7, following the hierarchy order chosen for the application of the algorithm.

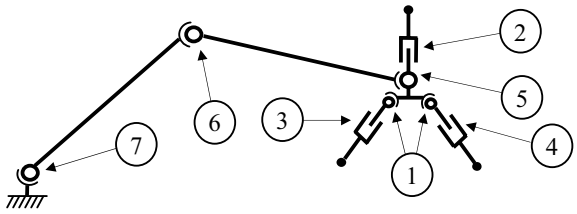

Fig. 6. Joints of the example robot arm and hand.

The more relevant joint has been considered the one that controls finger abduction (only one joint as abduction is coupled in both articulated fingers). Even though there are other joints closer to the contact points, this is the one that better represents the kind of grasp being performed. Next, the translational gripper joints: no difference in relevance is found in them, so they have been randomly ordered. After

the gripper joints, the SCARA arm joints follow in order of their distance to the contact points.

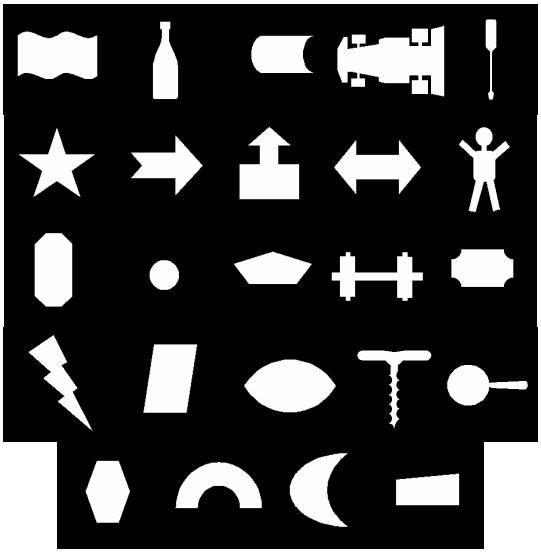

Fig. 8. Object database used for example generation.

In order to generate grasp examples, a simulation environment has been developed. Different objects are presented to the user, who should operate the robot arm and gripper in order to grasp each object. Once an object has been grasped, both the contact points and the robot joint values are stored. The database used for the experiments is shown in Fig. 8: these are the objects presented to the user and also the objects the system will have to grasp autonomously once trained. In each experiment, a total of 50 randomly selected objects are presented to the user, who performs grasps for all of them.

Once the examples recorded, the next step is the generation of the tree. A hybrid approach has been used, with full tree generation for the assignment levels of the tree and heuristic tree generation for the joint levels.

The assignment tree is quite simple due to the gripper being considered: once the first finger is assigned to a certain contact point, the other two fingers have fixed assignments. The result is an upper subtree with only three leaves.

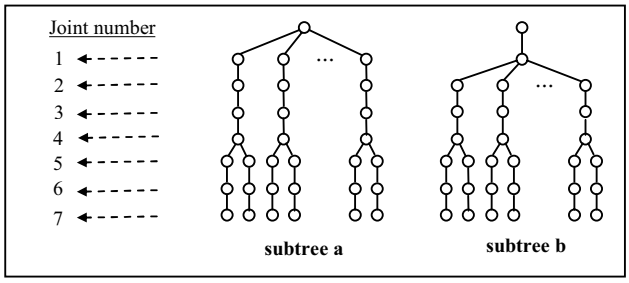

Fig. 9. Joint subtrees for the example application.

For each upper subtree leaf, a heuristic generation of the joint subtree is performed, starting with joint 1. Let us call j_1 the angular value of joint 1. Without getting into geometry details, depending on the set of contact points a single solution $j_1=0$ or a range of solutions $j_1>0$ can be found. Depending on that result, joint 2 will have infinite valid values (when $j_1=0$) or a single value (when $j_1>0$). The remaining joints will be restricted to a single value except those of the SCARA robot arm which can reach the desired endpoint with elbow up or elbow down configurations. The two possible subtrees are schematically shown in Fig. 9.

The computation of the trees and the search process performed simultaneously take about 0.3 sec. on a standard PC (256 MB RAM, 1.7 GHz) when the range of motion of the different joints is discretized in 20 values.

VI. Conclusions

Robot grasp synthesis requires efficient redundancy resolution techniques. Previous approaches to redundancy resolution are not devoted to grasp synthesis, so they do not consider specific problems, like finger assignment.

A tree based representation of the inverse kinematics like the one proposed can handle both the assignment redundancy and the joint redundancy, and allows fast searches to be performed. The experimental results obtained show the validity of the approach.

The use of similarity to the examples given by the user as a way to solve the redundancy is a novelty and has many potential fields of application, among them humanoid robots. The method proposed for measuring similarity (joint hierarchy and inverse kinematics tree) is specific to grasping processes, but can be easily adapted to other robot operations.

Future work is focused in the extension of the presented technique to redundancy resolution in path planning.

References

[1] C. Schaeffer and T. May, "Care-o-bot: a system for assisting elderly or disabled persons in home environments," in Assistive Technology on the Threshold of the New Millennium. C. Bhler and H. Knops, Eds. IOS Press, 1999, pp. 340-345.

[2] R.J. Anderson, "Autonomous, teleoperated, and shared control for robot systems," in Proceedings of the 1996 IEEE Int. Conf. on Robotics and Automation, 1996, pp. 2025-2032.

[3] V.D. Nguyen, "The synthesis of stable force-closure grasps," Technical report AI-TR-905, MIT Artificial Intelligence Laboratory, 1986.

[4] J. Cornellá and R. Suárez, "On 2D 4-finger frictionless optimal grasps," in Proc. of 16th IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Las Vegas, 2003, vol.3, pp.3680-3685.

[5] D. Schwammkrug, J. Walter and H. Ritter, "Rapid learning of robot grasping positions," in Proc. 7th Int. Symp. Intelligent Robotics, 1999, pp. 149-155.

[6] W.A. Wolovich, H. Elliot, "A computational technique for inverse kinematics," in Proc. 23rd IEEE Conference on Decision and Control, 1984, pp. 1359-1363.

[7] J. Baillieul, "Kinematic programming alternatives for redundant manipulators," in Proc. IEEE Int. Conf. on Robotics and Automation, 1985, pp. 722-728.

[8] Y. Nakamura and H. Hanafusa, "Inverse kinematics solutions with singularity robustness for robot manipulator control," in Journal of Dynamic Systems, Measurement and Control, 108, 1996, pp. 163-171.

[9] G.S. Dordevic, M. Rasic and R. Shadmehr, "Parametric models for motion planning and control in biomimetic robotics", in IEEE Transactions on Robotics, in press (*paper no. W03-017R*).

[10] N.S. Pollard, "Synthesizing grasps from generalized prototypes," Proc. of the IEEE Int. Conf. on Robotics and Automation, 1996, pp. 2124-2130.

[11] C. Fernandez, A. Vicente, O. Reinoso and R. Aracil, "A decision tree based approach to grasp synthesis," in Proc. Int. Conf. on Intelligent Manipulation and Grasping, 2004, pp. 486-491.

[12] I.H. Witten and E. Frank, "Data mining," Morgan Kaufmann, 2000.