Efficient Implementation of Force Queries
For Compliant Robot Motion

STAVROS VOUGIOUKAS
Dept. of Applied Informatics
University of Macedonia
Egnatia 156, 54006 Thessaloniki
GREECE

Abstract: In high precision robotic applications, where there is contact with the environment, the direction of contact forces carries information about the contact geometry, which cannot be inferred by position sensors. A mapping of these unit contact forces to corrective motions can be built, before task execution, via simulation. A k-d tree data structure is proposed as an efficient implementation of such a mapping, and a novel scheme for executing fast conical region queries in this data structure is developed. This scheme makes it possible to lookup corrective motions from sensed forces in real-time, during the robot task execution.

Key Words: compliance mapping, data structures, force region queries.

1 Introduction

When a robot manipulator comes in contact with the environment, some form of compliance [1] is typically used to prevent excessive contact forces. The direction of each contact force (unit force) provides some information about the contact geometry and can be used to guide the robot operation. Such information is especially useful in tasks like high precision assembly, where the robot positional uncertainty exceeds the assembly parts clearances.

In [2][3] it has been shown that these unit contact forces can be computed quite accurately via an offline Monte-Carlo simulation of the robot task execution. During the simulation, each unit contact force can be mapped to some corrective motion, and stored in a lookup table. This table implements a compliance mapping. During the actual robot task execution, for each sensed contact force \( f \), the instantaneous commanded velocity of the robot is obtained by searching in the lookup table for all the pre-computed forces within a certain cone with \( f \) as its axis. The force entry picked, is the one, which is closest to \( f \), and its associated velocity is used for the robot.

The manner in which the lookup table is synthesized and used during execution, necessitates that the method used to represent this compliance mapping \( M(f) \) be optimized for the application. In the next section we will discuss why this is so, and what factors must be considered. Next we will present a suitable data structure for the mapping implementation, and a novel scheme for executing fast conical region queries in this data structure.

2 Data Structure Requirements

The amount of space required to store a compliance mapping is related to a number of factors such as the dimension of the configuration space of the task, the complexity of its surface, and the positional and control uncertainty of the robot. While it is not possible to express the size in closed form, the actual number of entries for real world tasks may be very large. Therefore, it is important to select an appropriate data structure for representing the compliance mapping based on the following application-driven criteria:

a) The data structure needs to allow for dynamic storage allocation, since it is not possible to predefine the number of entries.
b) The data structure may grow to be quite large, thus it needs to grow efficiently. In particular, it is desirable to select a data structure that grows linearly and that the linear growth factor be relatively small.
c) Since the data structure will need to be accessed in real-time it should be one that admits time-complexity efficient access.

To address the latter issue, there are three operations that, in general, a data structure should provide an efficient means to perform:

- insertion of a new entry into the table,
- region query, which asks for all the unit force entries \( f \), which lie in a cone with \( w \) at its axis,
- optimization of the data structure
Optimization of a data structure (e.g. tree balancing in a binary tree) is needed to guarantee that the performance of the first two operations meets the predicted theoretical optimum when it is available. Optimization does not pose a serious bottleneck because it does not need to be performed with every operation on the data structure. In our application, it would be sufficient to optimize the data structure only once, after the mapping has been completed, so that it may be used efficiently for runtime performance. It might be desirable, however to also optimize the data structure periodically during the compliance mapping synthesis so that the off-line synthesis too could be performed efficiently. The insertion operation is used repeatedly during the synthesis and verification procedure, but again these are performed off-line via simulation. Therefore, fast insertion is desirable, but not as essential as fast region query, since region queries must be performed in real-time during the execution of the compliant motion. Furthermore, in our formulation an insertion operation must be preceded by a region query to make certain that the new reaction force being added is not already represented in the table.

Hence, the most important factor to consider when selecting a representation method for our application is the efficiency with which a region query can be performed based on this representation. The region query operation is not only used repeatedly during simulation, but it also constitutes the most time consuming operation (excluding I/O) during real-time task execution.

In the following we will present a special k-d tree data structure which allows for dynamic allocation, grows linearly, and admits efficient region queries, particularly in a multidimensional space such as the 6-DOF space in which robots are working. K-d trees were originally proposed as a means for storing multidimensional entities. We have developed a special procedure, based on bounding boxes, in order to make it efficient to perform conical-region query operations. We will now discuss our representation.

3 Review of k-d Trees

The multidimensional binary search tree, or k-d tree was first introduced by Bentley [4] as an efficient data structure for associative searching (often called multi-key searching). Each node of a k-d tree contains one record, and two pointers, which are either null, or point to another node in the tree. Each node is associated with a discriminator, which is an integer j between 0 and k-1. In our application the discriminator corresponds to the jth force coordinate. All nodes of the same level of the tree have the same discriminator. The root has discriminator zero, its two sons have discriminator one, and so on, until the k+1 level, which starts again with a discriminator equal to zero. The basic idea behind the k-d tree is the following: If a node P has discriminator j, and its jth coordinate is Pj then the jth coordinate of all nodes Q in its left subtree are smaller or equal to Pj. The right subtree contains nodes with jth coordinate greater than Pj. What this buys us is that, if random nodes are inserted into an initially empty tree, the resulting tree will have the same desirable properties as a randomly built binary search tree.

If n is the total number of forces we wish to store, the k-d tree requires O(n) storage. If these forces have not been encountered in a random order, it has been shown in [4] that the tree can be optimized (balanced) in O(nlog(n)) time. The average time required to perform a traditional insertion of a record into a traditional k-d tree is O(log(n)). We use the term "traditional insertion" to refer to the operation which checks whether the record already exists in the tree and inserts it, only if it does not exist. However, our mapping synthesis algorithm inserts a new force in the tree only when there exist no other entry in the tree, which lies in the uncertainty set (cone) of the new force. Therefore, the insertion operation for our mapping synthesis algorithm is equivalent to a region query.

4 Efficient Force Queries

If the query region is almost cubical and the number of records that satisfies the query is small then the required average time to perform the query is O(log(n)+F) (see [5]), where F is the number of records found in the region. The term F above depends in general on the factors which affect the size of the mapping. This size corresponds to an "average density" of unit-force entries on the surface of the hyper-sphere. The number F depends also on the unit vector f which is the axis of the cone, because the query region may fall in a sparsely, or densely "populated" part of the hyper-sphere.

For the reasons described above, the average F cannot be easily computed. Its upper bound however, depends only on the dimension k. The biggest obstacle for efficient region queries in our application is the mismatch between the way the k-d tree is constructed and the shape of the query region, which is the projection of a cone (a non
rectangular hyper-patch) on the unit hyper-sphere (an arc in 2D).

The reason for this mismatch is that the use of the force-vector coordinates as discriminators that split the k-d tree into sub-trees, during the construction of the tree, creates a partitioning of the k-dimensional space into rectilinearly oriented hyper-rectangular regions. These regions are represented at each node as boundary arrays.

Figure 1 illustrates a 2D example, where force vectors A through H have been inserted into the tree (in alphabetical order). The bound array at each node Q defines a bounding box for all the nodes in the left and right sub-tree of Q. Given two input forces \( F_1 \) and \( F_2 \), the region query should report no entries within the cone of \( F_1 \) and one entry (D) for the cone of \( F_2 \). An efficient region query algorithm that was described in [4], requires the computation of the intersection of these hyper-rectangles with the query region.

In our case, however, the query region is not cubical, so that performing region query in this manner would be very expensive. We have investigated two possible solutions to this problem. The first solution is to represent the unit forces in some k-dimensional spherical coordinate system. Then the partitioning would not take place in the Cartesian space \( \mathbb{R}^k \), but in the spherical angle space. This space it is very easy to compute a bounding box for the region. In two dimensions for example, if the unit vector \( \mathbf{f} \) is at angle \( \phi_0 \) with the x-axis, the query region (an arc) is enclosed by the cone \( \phi_0 - \alpha \leq \phi \leq \phi_0 + \alpha \). Unfortunately, the extension of this method to higher dimensions turned out to be very complicated and required the solution of higher order trigonometric equations. While we have solved these equations symbolically, we were confronted with a bigger obstacle, however, that is that in angle space there exist discontinuities which make it necessary to use multiple bounds to describe the region of interest. Furthermore, these bounds cannot be evaluated ahead of time, meaning that ultimately this region query algorithm would not be as fast as need be. Therefore this solution was not pursued further.

The second solution is to compute a rectilinearly oriented bounding box around the cone in Cartesian space, and perform a "classical" k-d tree region query, by intersecting this box with the k-d tree boundary arrays[4]. The left part of Figure 1 illustrates how easily the bounding boxes for two forces \( F_1, F_2 \) can be intersected with the partitioning of the space induced by the k-d tree.

The problem of the bounding box computation can be stated as a constrained optimization problem, which will be solved analytically in the next section. The reason for computing the bounding box analytically is that we cannot afford to perform numerical optimization during the real-time motion of the robot. The bounding box can be calculated with extremely low computational cost as it is shown next.

5 Bounding Box Computation

Consider a unit \( n \) dimensional hypersphere centered at the origin. A unit vector \( \mathbf{f} \) corresponds to a point on the surface of the sphere. Consider now all unit vectors \( \mathbf{v} \), such that

\[ \mathbf{v}^T \mathbf{f} \geq \cos \alpha \]

These vectors lie in a cone, which has \( \mathbf{f} \) as its axis, with \( \alpha \) being the half-angle of the cone. We wish to compute a bounding box

\[ B = \{(B_i^{\text{min}}, B_i^{\text{max}}), i = 1, \ldots, n\} \]

such that for all vectors \( \mathbf{v} \) within the cone,

\[ B_i^{\text{min}} \leq v_i \leq B_i^{\text{max}}, i = 1, \ldots, n \]

where \( i \) is the \( i \)th coordinate. Since \( B_i^{\text{max}} \) is an upper bound for the \( i \)th coordinate, we can compute it as the tightest upper bound, by computing the maximum \( i \)th coordinate of all vectors within the cone. If \( \mathbf{e}_i \) is the \( i \)th basis vector for \( \mathbb{R}^n \), then the \( i \)th coordinate of a vector \( \mathbf{v} \) is \( v_i = \mathbf{e}_i^T \mathbf{v} \). The maximum \( i \)th coordinate of all vectors in the cone can be computed as the \( v_i \) which is the solution to the minimization problem:
\[ \min(-v_i) \quad \text{subject to:} \]

\[
\begin{align*}
\cos \alpha - v^T f & \leq 0 \\
v^T v - 1 & = 0
\end{align*}
\]

For notation simplicity, we use \( v \) as a variable and as the optimal (extremal) solution. The above problem can be solved analytically. First we form the Lagrangian:

\[ L = -v_i + \frac{\lambda_1}{2}(\cos \alpha - v^T f) + \frac{\lambda_2}{2}(v^T v - 1) = 0 \]

The optimality, orthogonality and non-negativity KKT conditions (see [6]) at the optimal are:

\[
\begin{align*}
\frac{\partial L}{\partial v} &= -e_i - \lambda_1 f + 2\lambda_2 v = 0 \\
\lambda_1(\cos \alpha - v^T f) &= 0 \\
v^T v - 1 &= 0 \\
\lambda_1 &\geq 0 \\
\lambda_2 &> 0
\end{align*}
\]

From the optimality condition (Eq. 1), we can solve for the optimal vector

\[ v = \frac{1}{2\lambda_2} e_i + \frac{\lambda_1}{2\lambda_2} f \]

Let us also compute the following quantities

\[
\begin{align*}
v^T v &= 1 + 2\lambda_1 e_i + \frac{\lambda_1}{2\lambda_2} f \\
v^T f &= \frac{\lambda_1}{2\lambda_2} + f_i
\end{align*}
\]

There are two possibilities for the optimal vector \( v \), which has the maximum \( i \)th coordinate. It is either inside the cone, or at the boundary of the cone. We have to consider each case individually.

**Case 1.**

If the optimal vector \( v \) lies inside the cone, then constraint eq.1 is a strict inequality, and \( \lambda_1 = 0 \). From the unit vector constraint eq.2 and eq.9 we can find that \( \lambda_2 = 1/2 \). This value satisfies the non-negativity condition. From the cone strict inequality and eq.10 we can deduce the condition on \( f_i \) when the optimal vector \( v \) lies inside the cone, which is \( f_i > \cos \alpha \).

From eq.8 we can easily find the optimal solution for this case, which is

\[ B_i^{max} = l \]

**Case 2.**

If the optimal vector \( v \) lies on the cone boundary, then \( \lambda_1 > 0 \), and constraint eq.1 combined with eq.10 can be written as

\[ \lambda_1 + f_i = 2\lambda_2 \cos \alpha \]

The unit magnitude constraint combined with eq.9 can be written as

\[ 1 + 2\lambda_1 f_i + \lambda_2^2 = 4\lambda_2^2 \]

We now have two equations, eq.12, eq.13, with the two Lagrange multipliers as unknowns. Solving them yields:

\[
\begin{align*}
\lambda_{i(1,2)} &= -f_i \pm \sqrt{\cos^2 \alpha(1-f_i^2)} \\
\lambda_2 &= \frac{\lambda_1 + f_i}{2\cos \alpha}
\end{align*}
\]

Since \( \lambda_1 \) may have two solutions, we have to choose the correct one. We should keep in mind that \( \lambda_1 \) must be positive at the optimal solution. Let the two candidate maximum coordinate values that correspond to the two solutions for \( \lambda_1 \) be \( B_i^{max} (\lambda_{11}) \) for the plus sign, and \( B_i^{max} (\lambda_{12}) \) for the minus sign. Their difference can be easily shown to be always greater than zero, and therefore the solution corresponding to \( \lambda_{11} \) is optimal. It is easy to show that \( \lambda_{11} \) is always greater than or equal to zero, when \( f_i \leq \cos \alpha \), which holds for Case 2. The maximum coordinate is then

\[ B_i^{max} = v_i = \frac{(1 + \lambda_{11} f_i) \cos \alpha}{\lambda_1 + f_i} \]

In the following, we summarize our results based on the given analysis.

**5.1 Bounding Box Maxima**

If \( f \) is the cone axis, and \( \alpha \) the half-angle of the cone and \( f_i \) is the \( i \)th coordinate of \( f \) and

\[
\Lambda = \sqrt{\frac{\cos^2 \alpha(1-f_i^2)}{1-\cos^2 \alpha}}
\]

then, the maximum \( i \)th coordinate for the bounding box \( B_i^{max} \) can be computed as follows:

**Case 1.**

If \( \cos \alpha \leq f_i \leq l \), then \( B_i^{max} = l \)
Case 2.
If \(-1 \leq f_i \leq \cos \alpha\), then
\[
 B_i^{\max} = \frac{(1 + \lambda_i f_i) \cos \alpha}{\lambda_i + f_i}, \lambda_i = -f_i + \Lambda
\]

5.2 Bounding Box Minima
Using similar analysis, we can compute the minimum \(i\)th coordinate of the bounding box \(B_i^{\min}\) as follows:

Case 1.
If \(-1 \leq f_i \leq -\cos \alpha\), then \(B_i^{\min} = -1\)

Case 2.
If \(-\cos \alpha \leq f_i \leq 1\), then
\[
 B_i^{\min} = \frac{(1 + \lambda_i f_i) \cos \alpha}{\lambda_i + f_i}, \lambda_i = f_i + \Lambda
\]

This concludes the analytic computation via minimization of the bounding box of the force conical neighborhood. The analytical solution of the minimization problem was obtained, so that the computation during real-time execution is kept minimal.

6 Implementation
The k-d tree data structure together with the bounding box computation were implemented in C, and were incorporated in the Monte-Carlo simulation procedure [2]. Simulations were run for three tasks of increasing complexity. These tasks were a 2D single peg insertion, a 3D single peg-in-slot insertion, and a 3D dual peg-in-slot insertion, in this order. The size of the mapping for the three simulations was found to be 26, 52, and 119 entries, respectively. As expected, as the geometry of the task got more complex, more contact forces were computed during the simulations. The convergence curves for the first 100 runs of these simulations are shown in Figure 2.

The k-d tree compliance mapping for the first task was also loaded and successfully used for real-time operation on a PUMA 560.

References:


