A Relational Positioning Methodology for Robot Task Specification and Execution

Adolfo Rodríguez, *Student Member, IEEE*, Luis Basañez, *Member, IEEE*, and Enric Celaya

***Abstract*—This paper presents a relational positioning methodology that allows to restrict totally or partially the movements of an object by specifying its allowed positions in terms of a set of intuitive geometric constraints. In order to derive these positions, a *geometric constraint solver* must be used. To this end, positioning mobile with respect to fixed (PMF), a geometric constraint solver for the relational positioning of rigid objects in free space is introduced. The solver exploits the fact that, in a set of geometric constraints, the rotational component can often be separated from the translational one and solved independently. PMF may be used as an interface for specifying offline-programmed robot tasks, as well as for assisting the execution of teleoperated tasks requiring constrained movements. Examples describing both the solver's operation and typical applications are discussed.**

***Index Terms*—Assembly planning, geometric constraint solving, relational positioning, robot programming.**

I. Introduction

When asked to perform a positioning operation, humans usually think of it in terms of satisfying geometric relations. For example, placing a glass on top of a table can be accomplished by making the bottom surface of the glass coincide with the tabletop. Positioning operations can restrict not only totally, but also partially the movement of an object. So, in the previous example, the glass can freely translate along directions parallel to the tabletop and can freely rotate about an axis perpendicular to it and still comply with the imposed relation. Furthermore, by using geometric relations, a positioning operation can be defined independently of the initial configurations of the involved objects, so if by some reason, the initial positions of the glass and the table change, the operation definition remains meaningful and need not be restated.

In the same way, many robot tasks require the positioning of objects with respect to their surroundings. Although this is a ubiquitous problem in robotics, most existing approaches fail to fulfill all the end user's needs, are not intuitive enough, and rarely support the notions of partial movement restriction and initial configuration independence. For example, in traditional offline programming, configurations are defined in terms of non-intuitive parameters such as homogeneous transformations and joint space coordinates. In gestural programming, the burden is placed on an operator that manually moves the robot end-effector along the desired trajectories, trading a simpler interface for possible workcell downtime and imprecision issues inherent to humans. Simulation-based programming is an attractive alternative since it imposes no workcell downtime, but its usefulness depends on a task representation interface that needs to be both intuitive and adapted to the end user's requirements. Relational positioning can be used to create such an interface.

Relational positioning is a powerful means for placing objects in space, in which the problem is formulated in terms of geometric constraints. A geometric constraint is a relation (distance, angle, tangency, etc.) between two or more geometric elements (points, curves, surfaces) that must be satisfied. These elements usually represent boundary or reference features of parent objects. For example, a point may represent the vertex of a cube, and a line may represent the axis of a cylinder. A geometric constraint solver is used to find the positions that each object should have to comply with these constraints. Relational positioning problems that can be solved by positioning one object at a time are called sequential.

Specifying a robot task can be done at multiple levels. Lower levels require defining all the details needed to complete the task (points, trajectories, etc.), while higher levels involve more abstract instructions leaving the details to automated processes. Relational positioning can be used at both levels: at low levels by using geometric constraints to define trajectory points and at high levels by using the constraints as an intermediate layer between an automatic task planner and the robot controller.

De Schutter *et al.* [1] present a constraint-based methodology for the specification of complex sensor-based robot tasks. They propose a means for dealing with geometric uncertainty and demonstrate the approach through various experiments, both

Manuscript received September 5, 2007; revised February 22, 2008. This paper was recommended for publication by Associate Editor P. Rocco and Editor H. Arai upon evaluation of the reviewers' comments. This work was supported in part by the Consejo Interinstitucional de Ciencia y Tecnología (CICYT) under Project DPI2005-00112 and Project DPI2007-63665 and in part by the Formación de Personal Investigador (FPI) Predoctoral Grant BES-2004-4070.

A. Rodríguez and L. Basañez are with the Institute of Industrial and Control Engineering (IOC), Technical University of Catalonia (UPC), Barcelona 08028, Spain (e-mail: adolfo.rodriguez@upc.edu; luis.basanez@upc.edu).

E. Celaya is with the Consejo Superior de Investigaciones Científicas, Institut de Robòtica i Informàtica Industrial (CSIC-UPC), Barcelona 08028, Spain (e-mail: celaya@iri.upc.edu).

This paper has supplementary downloadable multimedia material available at http://ieeexplore.ieee.org. provided by the author. This material includes videos, in which the implementation of PMF, the geometric constraint solver described in the paper is presented. In them, it is shown how the solver interface is used to interactively define relational positioning problems and display their solutions. More precisely, the following situations are reported: 1) A basic demonstrator of the solver's capabilities. It is shown how geometric constraints are used to restrict the movements of different objects, and how DOF parameters are used to translate and rotate such objects along their unconstrained directions. Two particular problems are solved. In the first one a point is constrained to the surface of a sphere and then to a circle, and in the second one two lines are set to coincide. 2) A simple assembly consisting on placing a cover on top of a base object is solved by progressively restricting the cover's movements and moving it along the unconstrained directions. The size is 11.6 MB. No particular requirements for the player, except for a standard MPEG1 decoder. Packing list includes pmf_demo.mpeg and simple_assembly.mpeg. Contact adolfo.rodriguez@upc.edu for further questions about this work.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2008.924263

simulated and real. Although very general and systematic, the approach requires the definition of appropriately chosen frames and geometric relationships, a process that can be convoluted in complex tasks, but can be greatly simplified with the addition of an intuitive relational positioning interface such as the one presented in this paper.

Thomas *et al.* [2] present a general approach for specifying and executing assembly tasks in the presence of uncertainty. Task descriptions can be generated with different modules, one of which is a geometric-restrictions-based assembly planner [3] that has been embedded in the Robcad Robot Simulation System. The planner has an intuitive interface but only supports a small set of geometric constraint types, limiting the complexity of the problems that can be formulated.

On the other hand, teleoperated task execution relies on operator skills. Some tasks involve movements that require the satisfaction of specific constraints, such as following a line or maintaining a fixed orientation. Turro *et al.* [4] present a system that can generate forces on a haptic device to restrict its movement to curves and surfaces, but these must be explicitly defined by the operator. DeJong *et al.* [5] use a combination of a structured light sensor system and an augmented-reality user interface to select curves and surfaces, which are then introduced to a constrained dynamic system simulation [6] for haptic rendering. In such situations, geometric constraints required for the correct execution of the task can be defined, and their effect can be fed back to the operator via visual displays and haptic devices.

The most important part of a relational positioning system is the geometric constraint solver, that translates the set of intuitive input constraints into an explicit set of allowed positions expressed in a way that is convenient for the automatic system that has to use them. There exist many methods for solving geometric constraint problems [7], most of which can be classified as graph-based, logic-based, algebraic, or a combination of these.

Graph-based methods construct a (hyper)graph in which the nodes represent geometric objects and the arcs, constraints. Topological features like cyclic dependencies and open chains can be easily detected. Graph analysis identifies simpler and solvable subproblems whose solutions are combined while maintaining compatibility with the initial problem. There exist algorithms with $O(n^2)$ [8], [9] and $O(nm)$ [10] time complexity, where n is the number of geometric elements and m is the number of constraints.

Logic-based methods represent the geometric elements and constraints using a set of axioms and assertions. The solution is obtained following general logic reasoning and constraint rewriting techniques [11], [12].

Algebraic methods translate the problem into a set of nonlinear equations, which can be solved using a variety of numeric and symbolic methods. Numeric methods range from the Newton–Raphson method [13] that is simple but does not guarantee convergence nor finding all possible solutions, to more sophisticated ones like homotopy [14] that guarantee both. They tend to have $O(n^2)$–$O(n^3)$ time complexity. Symbolic methods use elimination techniques such as Gröbner basis to find an exact generic solution to the problem, which can be evaluated with numerical values to obtain particular solutions [15]. These methods are extremely slow since they have $O(c^n)$ time complexity.

The application area for geometric constraint solvers is currently dominated by the computer-aided design (CAD) community, which has widely adopted them as an intuitive framework for parts and assembly design. Most CAD solvers deal with 2-D sketching problems [8], [9], but there exist methods that model parts directly in 3-D [12], [16]. Among other applications not so widespread figure mechanism design, kinematic modeling, molecular modeling, and robot task specification.

A method is said to be *general* if it admits the formulation of any geometric constraint problem and *complete* if it is able to solve—or detect the unsolvability—of all the problems whose formulation it admits. Kramer [10] proposes a geometric constraint solver for open spatial kinematic chains and certain families of closed ones that is neither complete nor general, specially in 3-D problems. Porta *et al.* [17] describe a complete and general numeric algebraic method based on Cayley–Menger determinants and branch-and-prune techniques. This method has the shortcoming that it is not well suited for relational positioning, because the problem cannot be directly formulated in terms of intuitive geometric constraints. Moreover, the form in which solutions are given needs to be processed before being used in a robot programming or teleoperation application.

This paper presents positioning mobile with respect to fixed (PMF) [18], [19] a sequential geometric constraint solver for the relational positioning of rigid objects in 3-D environments by means of distance and angular constraints between points, lines, and planes; PMF handles under-, well-, and overconstrained (redundant and incompatible) problems. The solver has been integrated in a framework for specifying offline-programmed robot tasks and assisting the execution of teleoperated ones.

The solver can be classified as *logic-based* since it contains a set of constraint rewriting rules that transform an input constraint set into an equivalent set whose solution is known. PMF is based on the linking matrix finder (LMF) solver [20], [21]. The most relevant improvements over the initial work are the ability to solve underconstrained problems, since it is often desirable to restrict only partially the motion of a robot and to guide it using the available degrees of freedom (DOFs); and the inclusion as inputs of certain types of second-order distance constraints (e.g., *point–point* and *point–line* distances), which considerably broaden the family of solvable problems.

An important requirement in the design of the solver has been that solution computation should be fast enough to be included in high-frequency loops and updated when the geometry of the problem changes (e.g., moving obstacles). Since there is a compromise between *completeness/generality* and *computational efficiency*, it has been opted for a solver that is neither complete nor general, but computationally very efficient, and at the same time, capable of handling most practical problems of the application domain. Such problems often turn out to be those whose solutions can be pictured qualitatively—but not quantitatively— by the user, so in most cases, the user is able to naturally formulate the problem in a way that can be solved by the system. On the other hand, CAD solvers [22] focus on

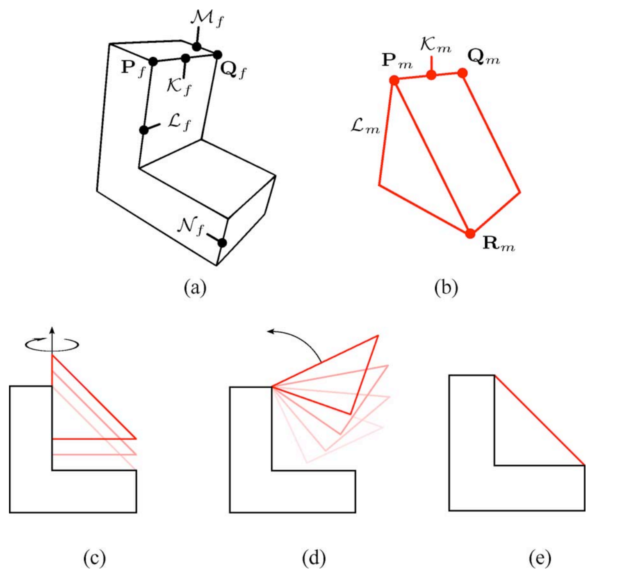


Fig. 1. Sample constraint sets between a fixed (a) and a mobile object (b). (c) Constraint $\mathcal{L}_m = \mathcal{L}_f$ can be decomposed into the pure rotational and translational constraints $\mathcal{L}_m \parallel \mathcal{L}_f$ and $\mathbf{P}_m \subset \mathcal{L}_f$, respectively. (d) Constraint set $\{\mathbf{P}_m = \mathbf{P}_f,\ \mathbf{Q}_m = \mathbf{Q}_f\}$ implies the rotational constraint $\mathcal{K}_m \parallel \mathcal{K}_f$. (e) Constraint set $\{\mathbf{P}_m = \mathcal{K}_f, \mathbf{Q}_m = \mathcal{M}_f$, and $\mathbf{R}_m = \mathcal{N}_f\}$ cannot be handled by the solver, but adding the additional redundant constraint $\mathbf{P}_m = \mathbf{P}_f$ renders it solvable.

completeness rather than on computational efficiency, because solutions are seldom required to be updated at high refresh rates.

The PMF solver is described in Sections II–V. Sample problems are listed in Section VI. Performance and implementation issues are covered in Section VII. Conclusions and future work are finally presented in Section VIII.

II. Solver Overview

The problem addressed by PMF is that of finding all possible configurations of a 3-D mobile object that satisfy a set of geometric constraints defined between the elements of the object and those of its (fixed) environment. The objects are assumed to be rigid and their positions known with respect to a fixed coordinate system.

PMF accepts as input constraints distance (d) and angle ($\angle$) relations between points, lines, and planes.[1] Also, the particular cases of coincidence/contained (=/$\subset$), parallelism ($\parallel$), and perpendicularity ($\perp$) relations are explicitly considered for commodity reasons, since they are used very often in practice.

The adopted notation for representing geometric entities throughout this discussion is the following: uppercase bold letters for points ($\mathbf{P}$, $\mathbf{Q}$), uppercase calligraphic letters for lines ($\mathcal{K}$, $\mathcal{L}$), uppercase Greek letters for planes (Π, Σ), and lowercase bold letters with a hat for unit vectors ($\hat{\mathbf{d}}$, $\hat{\mathbf{u}}$). Mobile (fixed) elements are identified by the subscript m (f).

A constraint is considered purely translational if it can be satisfied regardless of the orientation of the constrained object, and analogously, it is considered purely rotational if it can be satisfied regardless of the object's translation.

[1]Although the supported geometric elements are planar, the objects they belong to need not be polyhedral.

The solver takes advantage of two key facts, which will be described in the following, as well as illustrated with simple examples featuring the objects depicted in Fig. 1(a) and (b).

First, many geometric constraints restrict both rotational and translational DOFs, but often they can be expressed in terms of pure rotational and pure translational constraints without losing their original meaning. For example, consider the lines $\mathcal{L}_m$ and $\mathcal{L}_f$ in Fig. 1(a) and (b). The *line–line* coincidence constraint $\mathcal{L}_m = \mathcal{L}_f$ restricts all but one rotational and one translational DOF [see Fig. 1(c)], and is equivalent to the *line–line* parallelism constraint $\mathcal{L}_m \parallel \mathcal{L}_f$, which is purely rotational, and the *point–line* contained constraint $\mathbf{P}_m \subset \mathcal{L}_f$, which is purely translational (where $\mathbf{P}_m$ is a support point of $\mathcal{L}_m$).

Second, the simultaneous satisfaction of two or more pure translational constraints may give rise to an implicit rotational constraint. In fact, it is possible to fully restrict an object — rotations included—using exclusively translational constraints (e.g., a Stewart platfom). Conversely, the simultaneous satisfaction of any number of pure rotational constraints never gives rise to implicit translational constraints. Now consider the set of two *point–point* coincidence constraints $\{\mathbf{P}_m = \mathbf{P}_f,\ \mathbf{Q}_m = \mathbf{Q}_f\}$. While the individual satisfaction of either constraint restricts all of the mobile object's translational DOFs, the simultaneous satisfaction of both implies the *line–line* parallelism constraint $\mathcal{K}_m \parallel \mathcal{K}_f$, which additionally restricts two rotational DOFs, yielding a solution with only one rotational DOF [see Fig. 1(d)]. Notice that the two constraints are compatible only if the distance between $\mathbf{P}_m$ and $\mathbf{Q}_m$ is the same as the distance between $\mathbf{P}_f$ and $\mathbf{Q}_f$, otherwise the problem will have no solution.

An important observation is that the map between sets of geometric constraints and solutions is *not* injective, so there may exist multiple constraint sets associated to the same solution. For instance, the constraint set $\{\mathbf{P}_m \subset \mathcal{L}_f, \mathbf{Q}_m \subset \mathcal{M}_f\}$ yields the same solution as the aforementioned example [see Fig. 1(d)].

The core idea behind the solver consists in formulating a relational positioning problem in terms of a compact set of pure rotational and translational constraints—which will be referred to as fundamental constraints—and making all rotational constraints explicit. This permits separating the rotational component of the problem so that it can be solved first using only the rotational constraints. Then, the translational component corresponding to each allowed rotation can be easily found using the translational constraints. Each solution component is obtained by matching the corresponding subset of constraints with a list of handled scenarios.

This approach may fail for problems that cannot be reduced to a form with known solution. However, this is not the usual case in relational positioning, and when it appears, the user is notified that the problem cannot be handled by the solver. Sometimes unhandled problems can be restated in a solvable form, either by adding additional redundant constraints or by using a different constraint set that yields the same solution. For example, the constraint set $\{\mathbf{P}_m = \mathcal{K}_f, \mathbf{Q}_m = \mathcal{M}_f, \mathbf{R}_m = \mathcal{N}_f\}$ cannot be handled by the solver although a well-constrained solution exists [see Fig. 1(e)]. However, if the additional redundant constraint $\mathbf{P}_m = \mathbf{P}_f$ is added to the problem, it becomes solvable by PMF.

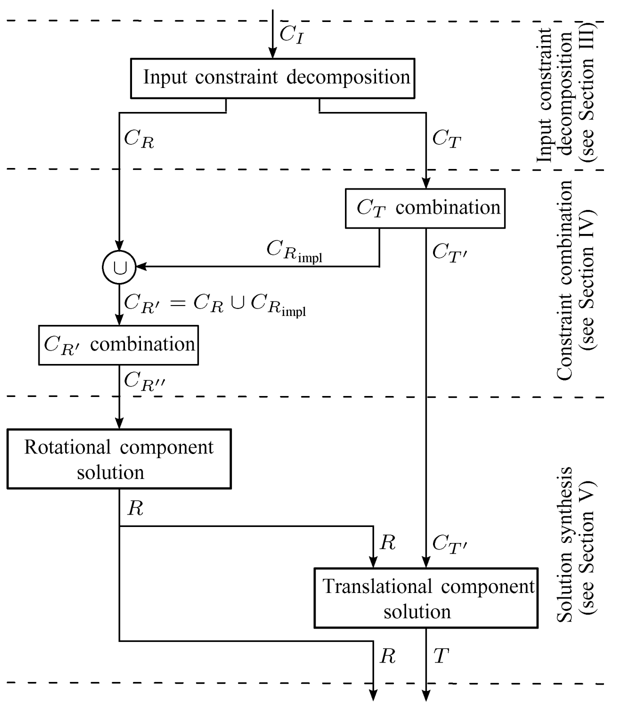

C_I
Input constraint decomposition
C_R
C_T
Input constraint decomposition (see Section III)
C_T combination
$C_{R_{\text{impl}}}$
$C_{T'}$
$\cup$
$C_{R'} = C_R \cup C_{R_{\text{impl}}}$
$C_{R'}$ combination
$C_{R''}$
Constraint combination (see Section IV)
Rotational component solution
R
R
$C_{T'}$
Translational component solution
Solution synthesis (see Section V)
R
T

Fig. 2. PMF solver solution process. C_I represents the input constraint set, C_R and C_T, respectively, represent the rotational and translational fundamental constraint sets ($C_{R_{\text{impl}}}$ contains the implicit rotational constraints), and R, T, respectively, represent the rotational and translational components of solution. Prime symbols indicate that the elements of a constraint set may have changed.

It is important to stress that the solver is *only* concerned on satisfying a set of geometric constraints. If the problem being solved models real objects, some of the solution configurations may be physically impossible to realize because of collisions between the involved objects, or because they lie outside the workspace of the actuation system (e.g., a robot arm). Also, it does not take into consideration issues such as geometric uncertainty or the dynamics of the constrained objects. However, PMF has been conceived with modularity in mind, so if required it can be interfaced with other modules such as collision detectors and robot simulators to filter out unfeasible configurations, and with sensory systems and estimators that update the geometric models as objects move or measurements are refined.

The solution process starts with the specification of the input constraints by the user, and consists on three main steps: input constraint decomposition, constraint combination, and solution synthesis. A scheme of the process is shown in Fig. 2.

III. Input Constraint Decomposition

Input constraints have been selected with the aim of providing an easy way to define the problem. Through input constraint decomposition, an input constraint set C_I is transformed into an equivalent set of pure rotational and translational fundamental constraints $C = C_R \cup C_T$ that contains fewer constraint types and is easier to work with.

There are three fundamental translational constraints, which express the distance between a point and another geometric element (point, line, or plane)

$$d(\mathbf{P}_a, \mathbf{P}_b) = p : \quad \textit{point–point}\text{ distance,}$$
$$d(\mathbf{P}_a, \mathcal{L}_b) = p : \quad \textit{point–line}\text{ distance,}$$
$$d(\mathbf{P}_a, \Pi_b) = 0 : \quad \textit{point–plane}\text{ coincidence,}$$

and one fundamental rotational constraint, which expresses the angle between two vectors:

$$\angle(\hat{\mathbf{u}}_a, \hat{\mathbf{u}}_b) = \alpha : \textit{vector-vector}\text{ angle.}$$

Subindices "a" and "b" represent the object to which a geometric element belongs. One object is always fixed, while the other is mobile [e.g., $d(\mathbf{P}_m, \mathbf{P}_f) = p, d(\mathbf{P}_f, \mathcal{L}_m) = p$].

The details of input constraint decomposition are listed in Table I. Input constraints that already are fundamental constraints do not need to be decomposed. The particular cases of coincidence/contained, parallelism, and perpendicularity are not explicitly shown in Table I, but can be obtained in a straightforward manner by setting the appropriate constraint parameter value. For *line–line* distance constraints, only the case that also enforces the lines to be parallel is considered, since the nonparallel case cannot be represented in terms of the adopted fundamental constraints.

TABLE I
Input Constraint Decomposition

Input constraint	Fundamental constraints	
	Translational	Rotational
$d(\mathbf{P}_a, \Pi_b) = p$	$d(\mathbf{P}_a, \Sigma_b) = 0$	-
$d(\mathcal{L}_a, \mathcal{L}_b) = p$ *	$d(\mathbf{R}_a, \mathcal{L}_b) = p$	$\angle(\hat{\mathbf{d}}_{\mathcal{L}_a}, \hat{\mathbf{d}}_{\mathcal{L}_b}) = 0$
$d(\mathcal{L}_a, \Pi_b) = p$	$d(\mathbf{R}_a, \Sigma_b) = 0$	$\angle(\hat{\mathbf{d}}_{\mathcal{L}_a}, \hat{\mathbf{n}}_{\Pi_b}) = \pi/2$
$d(\Pi_a, \Pi_b) = p$	$d(\mathbf{S}_a, \Sigma_b) = 0$	$\angle(\hat{\mathbf{n}}_{\Pi_a}, \hat{\mathbf{n}}_{\Pi_b}) = 0$
$\angle(\mathcal{L}_a, \mathcal{L}_b) = \alpha$	-	$\angle(\hat{\mathbf{d}}_{\mathcal{L}_a}, \hat{\mathbf{d}}_{\mathcal{L}_b}) = \alpha$
$\angle(\mathcal{L}_a, \Pi_b) = \alpha$	-	$\angle(\hat{\mathbf{d}}_{\mathcal{L}_a}, \hat{\mathbf{n}}_{\Pi_b}) = \pi/2 - \alpha$
$\angle(\Pi_a, \Pi_b) = \alpha$	-	$\angle(\hat{\mathbf{n}}_{\Pi_a}, \hat{\mathbf{n}}_{\Pi_b}) = \alpha$

with $d(\Pi_b, \Sigma_b) = p \qquad \mathbf{R}_a \subset \mathcal{L}_a \qquad \mathbf{S}_a \subset \Pi_a$
* Only the case that also enforces $\mathcal{L}_a \parallel \mathcal{L}_b$ is considered.

IV. Constraint Combination

In constraint combination, a set of rules defines a constraint rewriting engine that recursively tests constraints in pairs with the purpose of rewriting a set of fundamental constraints in a compact and explicit form with known solution. The tests verify constraint compatibility and redundancy, so ill-defined cases are labeled as unsolvable while compatible but redundant constraints are removed. Pairs of compatible constraints are tried to be expressed more compactly with a single equivalent constraint, or substituted by individually more restrictive, but globally equivalent constraints. Rotational constraints implicitly defined by pairs of translational ones are identified and explicitly introduced. Constraint combination is applied separately to C_T and $C_{R'}$, and in that order, so that implicit rotational constraints are incorporated to C_R before the combination tests are performed on it (Fig. 2).

The constraint rewriting rules are obtained by applying the following method to each pair of constraint types to be tested.

TABLE II
COMPATIBILITY CONDITIONS FOR CONSTRAINT COMBINATION

Constraint pair	Compatiblity condition
$d(\mathbf{P}_a, \mathbf{P}_b) = p,\ d(\mathbf{Q}_a, \mathbf{Q}_b) = q$	$(d_a + d_b \geq \lvert q - p \rvert) \vee (\lvert d_b - d_a \rvert \leq p + q)$
$d(\mathbf{P}_a, \mathbf{P}_b) = p,\ d(\mathbf{Q}_a, \mathcal{L}_b) = q$	$(d_a + d_b \geq q - p) \vee (d_b - d_a \leq p + q)$
$d(\mathbf{P}_a, \mathcal{L}_b) = p,\ d(\mathbf{Q}_a, \mathcal{K}_b) = q$	$[\,(\mathcal{L}_b \parallel \mathcal{K}_b) \wedge (d_a + d_b \geq \lvert q - p \rvert)\,] \vee (d_b - d_a \leq p + q)$
$d(\mathbf{P}_a, \mathcal{L}_b) = p,\ d(\mathbf{Q}_a, \Pi_b) = 0$	$d_b - d_a \leq p$
$d(\mathbf{P}_a, \mathbf{P}_b) = p,\ d(\mathbf{Q}_a, \Pi_b) = 0$	$d_b - d_a \leq p$
$d(\mathbf{P}_a, \Pi_b) = 0,\ d(\mathbf{Q}_a, \Sigma_b) = 0$	$d_b - d_a \leq 0$
$d(\mathbf{P}_a, \mathcal{L}_b) = p,\ d(\mathbf{Q}_b, \mathcal{L}_a) = q$	Always compatible
$d(\mathbf{P}_a, \mathcal{L}_b) = p,\ d(\mathbf{Q}_b, \Pi_a) = 0$	Always compatible
$\angle(\hat{\mathbf{u}}_a, \hat{\mathbf{u}}_b) = \alpha,\ \angle(\hat{\mathbf{v}}_a, \hat{\mathbf{v}}_b) = \beta$	$(\sigma_a + \sigma_b \geq \lvert \alpha - \beta \rvert) \vee (\lvert \sigma_a - \sigma_b \rvert \leq \alpha + \beta)$

Where d_a (d_b) represents the distance between the two elements belonging to object "a" ("b"). The same applies to σ_a (σ_b) but with angles instead of distances.

1) Find the compatibility conditions that enable the two constraints to be satisfied simultaneously. They will depend on up to four distance or angle parameters.
2) Create a rule that labels the problem as unsolvable if the compatibility condition is not satisfied (conflicting constraints).
3) Consider the compatibility conditions in its general form, as well as at the limit cases (e.g., parallel elements, equality of a greater-or-equal-than condition) for all possible combinations obtained by making the parameters equal to zero.
4) If any of the aforementioned configurations can be represented in terms of a single fundamental constraint, create a rule that substitutes the original pair with this single constraint.
5) If any of the aforementioned configurations can be represented in terms of a pair of individually more restrictive constraints, create a rule that substitutes the original pair with this new one.
6) If the constraints being tested are translational and the configuration reveals an implicit rotation, create a rule that explicitly adds this constraint to C_R.

This method fails to obtain new rules when the result of a combination test cannot be expressed in terms of the adopted fundamental constraints, or when more than two constraints need to be simultaneously considered to extract it.

When a new constraint is added to a constraint set, it must also be combined with the remaining constraints in the set, hence the recursivity of this step.

Sometimes the simultaneous satisfaction of two constraints yields a number of alternative possibilities. If each possible alternative can be expressed as a fundamental constraint, the current problem is branched into different problem instances, one for each possible alternative. These new problem instances are solved independently of each other. For example, the combination of $d(\mathbf{P}_m, \mathbf{P}_f) = 0$ and $d(\mathbf{Q}_m, \mathcal{L}_f) = 0$ [see Fig. 1(a) and (b)] gives rise to two possible alternative constraints of the form $d(\mathbf{Q}_m, \mathbf{R}_f) = 0$, where $\mathbf{R}_f$ is one of the two points of intersection of a sphere centered at $\mathbf{P}_f$ and the line $\mathcal{L}_f$.

Table II lists all combination scenarios handled by the solver indicating their corresponding compatibility conditions. For brevity, the complete set of rules has been omitted from this paper, but the interested reader can find them in [19]. Fig. 3 details the particular example of two *point–plane* coincidence constraints.

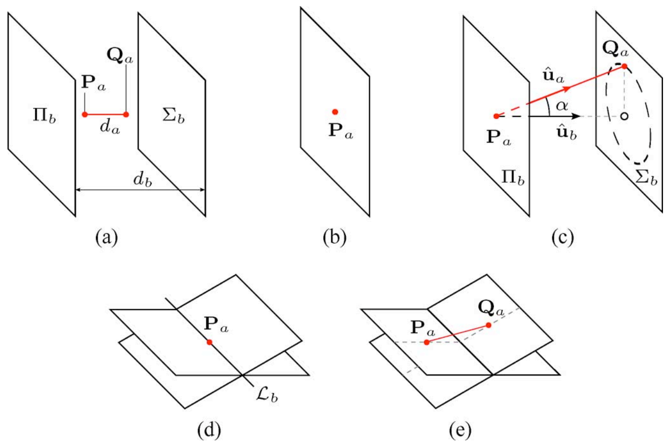


Fig. 3. Combination rules for two *point–plane* coincidence constraints $d(\mathbf{P}_a, \Pi_b) = 0, d(\mathbf{Q}_a, \Sigma_b) = 0$. The compatibility condition is $d_a \geq d_b$. (a) $(d_a < d_b)$: incompatible constraints. Label problem as unsolvable. (b) $(\Pi_b \parallel \Sigma_b)$ and $(d_a = d_b = 0)$: redundant constraints—remove one. (c) $(\Pi_b \parallel \Sigma_b)$ and $(d_a \neq 0)$ and $(d_a \geq d_b \geq 0)$: add to C_R the implicit rotation $\angle(\hat{\mathbf{u}}_a, \hat{\mathbf{u}}_b) = \alpha$, where $\alpha = \cos^{-1}(d_b/d_a)$. (d) $(\Pi_b \nparallel \Sigma_b)$ and $(d_a = 0)$: substitute both constraints with $d(\mathbf{P}_a, \mathcal{L}_b) = 0$. (e) $(\Pi_b \nparallel \Sigma_b)$ and $(d_a > 0)$: leave constraints unchanged.

The main reason why constraints involving more general curves and surfaces [e.g., nonuniform rational B-spline (NURBS), triangle meshes] have not been considered in the solver is because the compatibility conditions needed for constraint combination are not straightforward to obtain, and also because particular solutions of these constraints are not closed-form. More complex constraints could be integrated into the solver and enforced individually, but they would probably be impossible to combine according to the aforementioned methodology.

V. SOLUTION SYNTHESIS

Solution synthesis (bottom part of Fig. 2) computes transformations that position the mobile object in a configuration that simultaneously satisfies *all* the imposed geometric constraints. This is a two-step process that takes advantage of the separation of fundamental constraints into pure rotational and pure translational ones.

Algorithm 1 solution synthesis

```
Require: Fundamental constraint set $C = C_{R''} \cup C_{T'}$
  if problem is labelled as ill-defined then
    return problem has no solution
  end if
  if $C_{R''}$ can be handled (Table III) then
    solve rotational component $R$
  else
    return problem is unhandled
  end if
  if $C_{T'}$ can be handled (Table IV) then
    solve translational component $T$ for the computed rotation $R$
  else
    return problem is unhandled
  end if
  if $R$ or $T$ have DOFs then
    instantiate particular solution by initializing the free parameters
  end if
  return $R, T$
```

First, the rotational component of the solution R is solved using only the constraints in $C_{R''}$, where R maps the initial orientation of the mobile object to a submanifold of the 3-D space of rotations that satisfies all the rotational constraints. Then, from a configuration that already satisfies R, the translational component T is solved using the constraints in $C_{T'}$, where T maps the translation associated to an R-satisfying configuration of the mobile object to a submanifold of the 3-D space of translations that satisfies all the translational constraints. The dimension of the aforementioned submanifolds correspond to the number of available DOFs each solution component has. Notice that T depends on R, so changes in the rotational component of the solution in general also affect the translational component. Algorithm I summarizes the process.

Solutions are represented by a rigid transformation parameterized by as many parameters as available DOFs, so that a sweep across the allowed parameters values will span the range of reachable configurations. The solution to the particular case of a well-constrained problem, which has no DOFs will be a constant rigid transformation. A common representation for rigid transformations are 4×4 matrices, where R and T take the form of a 3×3 matrix and a 3×1 vector, respectively. A particular solution P can be then written as

$$P = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}. \quad (1)$$

Other rigid transformation representations may also be used. For example, the current implementation of the solver uses unit quaternions to represent rotations for numerical stability reasons, but for simplicity, the present discussion will adhere to the 4×4 matrix notation.

If branching occurred during the constraint combination step, Algorithm I must be applied to the fundamental constraint set C associated to each branch. The resulting solutions are all equally valid and satisfy the input constraints. However, a projection operation can be performed to obtain which solution yields the configuration that is closest (according to a suitably chosen metric) to the initial configuration of the mobile object.

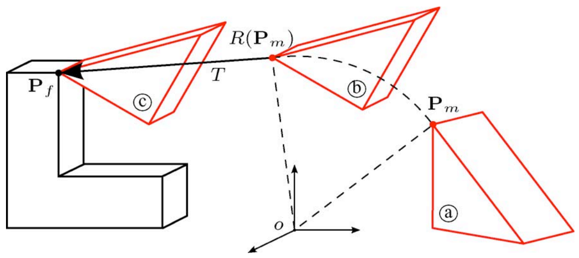


Fig. 4. Solution synthesis process: evaluation of R and T. ⓐ Initial configuration of the mobile object. ⓑ R-satisfying configuratin. ⓒ Final configuration. It satisfies both R and T.

To exemplify the solution synthesis process, consider the fixed and mobile objects depicted in Fig. 1(a) and (b), respectively; and let the mobile object be restricted with a constraint set that yields a solution such as the one depicted in Fig. 1(d). The process is illustrated in Fig. 4, where all geometric elements are represented in the fixed coordinate system o.

Solving R and applying it to the initial configuration of the mobile object ⓐ results in the R-satisfying configuration ⓑ. Then, T is solved for the current value of R (so that points $\mathbf{P}_m$ and $\mathbf{P}_f$ coincide), that when applied to ⓑ results in ⓒ, a configuration that satisfies both R and T.

In this case, $T = \mathbf{P}_f - R(\mathbf{P}_m)$, and a matrix representation of the solution would have the form

$$P = \begin{bmatrix} R & \mathbf{P}_f - R(\mathbf{P}_m) \\ 0 & 1 \end{bmatrix}. \quad (2)$$

In Section V-B, it will be shown that the general form of T is very similar to the one presented before.

The rotational component of the solution is solved before the translational one because rotations are *origin-preserving* transformations, that is, when the orientation of an object changes, all points except the coordinate system origin translate. On the other hand, translations are *orientation-preserving* transformations, so when the position of an object changes, its orientation remains unchanged. If the solution order is inverted so that the mobile object is first translated to satisfy the translational constraints and then rotated to satisfy the rotational ones, it can be observed that in the general case, the final configuration will no longer satisfy the translational constraints.

A. Rotational Component

Table III lists the possible sets of rotational constraints with their corresponding number of DOFs and solutions. After performing $C_{R'}$ reduction, *any* set of rotational constraints can be matched to one of these entries, so unmentioned cases have not been included in the list not because they are unhandled, but because they can be rewritten in a form that matches one of the table entries.

The rotation R that must be applied to the mobile object for four of the cases in Table III will be now discussed. The details of the remaining two cases have been omitted because of their length, but can be found in [19]. The trigonometric functions $\sin\alpha$ and $\cos\alpha$ will be abbreviated as s_α and c_α, respectively.

TABLE III
POSSIBLE SETS OF ROTATIONAL CONSTRAINTS

Constraints	DOF	# Solutions
$\angle(\hat{\mathbf{u}}_m, \hat{\mathbf{u}}_f) = 0$, $\angle(\hat{\mathbf{v}}_m, \hat{\mathbf{v}}_f) = 0$	0	1
$\angle(\hat{\mathbf{u}}_m, \hat{\mathbf{u}}_f) = \alpha$, $\angle(\hat{\mathbf{v}}_m, \hat{\mathbf{v}}_f) = \beta$, $\angle(\hat{\mathbf{w}}_m, \hat{\mathbf{w}}_f) = \gamma$ *	0	up to 8
$\angle(\hat{\mathbf{u}}_m, \hat{\mathbf{u}}_f) = 0$	1	1
$\angle(\hat{\mathbf{u}}_m, \hat{\mathbf{u}}_f) = \alpha$, $\angle(\hat{\mathbf{v}}_m, \hat{\mathbf{v}}_f) = \beta$	1	2
$\angle(\hat{\mathbf{u}}_m, \hat{\mathbf{u}}_f) = \alpha$	2	1
none	3	1

* Only certain cases are considered.

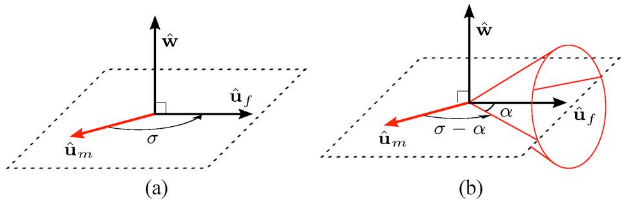


Fig. 5. Enforcement of two particular solutions of the rotational component. (a) One *vector–vector* parallelism constraint $\angle(\hat{\mathbf{u}}_m, \hat{\mathbf{u}}_f) = 0$. (b) One *vector–vector* angle constraint $\angle(\hat{\mathbf{u}}_m, \hat{\mathbf{u}}_f) = \alpha$.

1) Two Parallelism Constraints $\angle(\hat{\mathbf{u}}_m, \hat{\mathbf{u}}_f) = 0, \angle(\hat{\mathbf{v}}_m, \hat{\mathbf{v}}_f) = 0$*:* This is a particular case of the rotation associated to two pairs of independent unit vectors $R(\hat{\mathbf{u}}_m, \hat{\mathbf{v}}_m \rightarrow \hat{\mathbf{u}}_f, \hat{\mathbf{v}}_f)$, as explained in the Appendix.

2) One Parallelism Constraint $\angle(\hat{\mathbf{u}}_m, \hat{\mathbf{u}}_f) = 0$*:* Let σ be the angle between $\hat{\mathbf{u}}_m$ and $\hat{\mathbf{u}}_f$, and $\hat{\mathbf{w}} = \hat{\mathbf{u}}_m \times \hat{\mathbf{u}}_f$, then the rotation $R(\phi)$ is given by

$$R(\phi) = R(\hat{\mathbf{u}}_f, \phi)\, R(\hat{\mathbf{w}}, \sigma) \tag{3}$$

where $R(\hat{\mathbf{u}}_f, \phi)$ accounts for the free rotation about the direction of $\hat{\mathbf{u}}_f$ (expressed as an axis–angle pair) and $R(\hat{\mathbf{w}}, \sigma)$ is the fixed rotation that places $\hat{\mathbf{u}}_m$ parallel to $\hat{\mathbf{u}}_f$ [see Fig. 5(a)].

3) One Angle Constraint $\angle(\hat{\mathbf{u}}_m, \hat{\mathbf{u}}_f) = \alpha$*:* Let σ be the angle between $\hat{\mathbf{u}}_m$ and $\hat{\mathbf{u}}_f$, and $\hat{\mathbf{w}} = \hat{\mathbf{u}}_m \times \hat{\mathbf{u}}_f$, then the rotation $R(\phi, \theta)$ is given by

$$R(\phi, \theta) = R(\hat{\mathbf{u}}_f, \phi)\, R(\hat{\mathbf{w}}, \sigma - \alpha)\, R(\hat{\mathbf{u}}_m, \theta). \tag{4}$$

$R(\hat{\mathbf{u}}_m, \theta)$ and $R(\hat{\mathbf{u}}_f, \phi)$ account for the free rotations about the directions of $\hat{\mathbf{u}}_m$ and $\hat{\mathbf{u}}_f$, respectively, and $R(\hat{\mathbf{w}}, \sigma - \alpha)$ is the fixed rotation that places $\hat{\mathbf{u}}_m$ at an angle of α with $\hat{\mathbf{u}}_f$. Fig. 5(b) depicts the enforcement of this constraint.

4) No Rotational Constraints: The mobile object is able to freely rotate in any direction. However, the rotation has been parameterized according to the Yaw, Pitch, and Roll convention, in which the rotation axis corresponds to those of the fixed coordinate system

$$R(\phi, \theta, \psi) = R(\hat{\mathbf{k}}, \psi)\, R(\hat{\mathbf{j}}, \theta)\, R(\hat{\mathbf{i}}, \phi). \tag{5}$$

B. Translational Component

Table IV lists the handled sets of translational constraints with their associated number of DOFs. Basically, the handled cases correspond to the three fundamental translational constraints—considering separately the particular case when the distance parameter equals zero—and some configurations considered meaningful that cannot be expressed in terms of a single fundamental constraint, but rather two, such as the one shown in Fig. 3(e). Note that contrary to the rotational component solution, not all possible combinations of translational constraints can be reduced to a form that matches an entry of Table IV. All constraint sets that are not mentioned in the table but can be rewritten through C_T reduction in a form that matches one of its entries will have a valid solution, otherwise they will be considered unhandled.

TABLE IV
HANDLED SETS OF TRANSLATIONAL CONSTRAINTS

Constraints	DOF	Condition
$d(\mathbf{P}_a, \mathbf{P}_b) = 0$	0	
$d(\mathbf{P}_a, \mathcal{L}_b) = 0$, $d(\mathbf{Q}_a, \mathcal{K}_b) = 0$	0	$\mathcal{L}_b \cap \mathcal{K}_b \neq \emptyset$
$d(\mathbf{P}_a, \mathcal{L}_b) = 0$, $d(\mathbf{Q}_a, \Pi_b) = 0$	0	$\mathcal{L}_b \cap \Pi_b \neq \emptyset$
$d(\mathbf{P}_a, \mathcal{L}_b) = 0$	1	
$d(\mathbf{P}_a, \mathcal{L}_b) = p$, $d(\mathbf{P}_a, \Pi_b) = 0$	1	$\mathcal{L}_b \cap \Pi_b \neq \emptyset$
$d(\mathbf{P}_a, \Pi_b) = 0$, $d(\mathbf{Q}_a, \Sigma_b) = 0$	1	$\Pi_b \cap \Sigma_b \neq \emptyset$
$d(\mathbf{P}_a, \mathbf{P}_b) = p$	2	
$d(\mathbf{P}_a, \mathcal{L}_b) = p$	2	
$d(\mathbf{P}_a, \Pi_b) = 0$	2	
none	3	

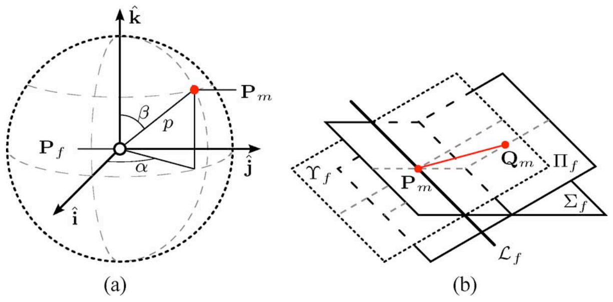


Fig. 6. Two particular solutions of the translational component. (a) One *point–point* distance constraint $[d(\mathbf{P}_m, \mathbf{P}_f) = p]$. (b) Two *point–plane* coincidence constraints $[d(\mathbf{P}_m, \Sigma_f) = 0, d(\mathbf{Q}_m, \Pi_f) = 0]$.

The translation T that must be applied to an R-satisfying mobile object has the general form

$$T = \mathbf{S}_f - R(\mathbf{S}_m) \tag{6}$$

where $\mathbf{S}_m$ and $\mathbf{S}_f$ belong to one of the submanifolds listed in Table V. The form taken by the expression of T given in (6) will now be explained in detail for five particular cases of Table IV: the first three solutions are associated to single constraints and the last two correspond to solutions defined by pairs of constraints. In these examples, subindices "a" and "b" have been substituted by "m" or "f" to explicitly identify the mobile and fixed elements. In order to obtain the expressions associated to the remaining cases, the same line of thought—with minor differences—can be applied.

1) One Point–Point *Coincidence Constraint* $d(\mathbf{P}_m, \mathbf{P}_f) = 0$*:* Both $\mathbf{S}_m$ and $\mathbf{S}_f$ belong to point submanifolds, so (6) becomes

$$T = \mathbf{P}_f - R(\mathbf{P}_m). \tag{7}$$

2) One Point–Point *Distance Constraint* $d(\mathbf{P}_m, \mathbf{P}_f) = p$*:* The mobile point $\mathbf{P}_m$ is constrained to the surface of a sphere with center $\mathbf{P}_f$ and radius p, as shown in Fig. 6(a), so $\mathbf{S}_m$ belongs to a point submanifold and $\mathbf{S}_f$ belongs to a spherical

TABLE V
PARAMETRIC REPRESENTATION OF TRANSLATIONAL SUBMANIFOLDS

Submanifold	Parametric representation ($\hat{\mathbf{d}}_1$, $\hat{\mathbf{d}}_2$, and $\hat{\mathbf{d}}_3$ are mutually perpendicular)
Point	$\mathbf{S} = \mathbf{P}$
Line	$\mathbf{S}(\lambda) = \mathbf{P} + \lambda\hat{\mathbf{d}}$
Plane	$\mathbf{S}(\lambda_1, \lambda_2) = \mathbf{P} + \lambda_1\hat{\mathbf{d}}_1 + \lambda_2\hat{\mathbf{d}}_2$
Ellipse/circle	$\mathbf{S}(\alpha) = \mathbf{P} + ac_\alpha\hat{\mathbf{d}}_1 + bs_\alpha\hat{\mathbf{d}}_2$
Sphere	$\mathbf{S}(\alpha,\beta) = \mathbf{P} + a(c_\alpha s_\beta\hat{\mathbf{d}}_1 + s_\alpha s_\beta\hat{\mathbf{d}}_2 + c_\beta\hat{\mathbf{d}}_3)$
Cylinder	$\mathbf{S}(\lambda,\alpha) = \mathbf{P} + \lambda\hat{\mathbf{d}}_1 + a(c_\alpha\hat{\mathbf{d}}_2 + s_\alpha\hat{\mathbf{d}}_3)$
$\mathbb{R}^3$	$\mathbf{S}(\lambda_1,\lambda_2,\lambda_3) = \mathbf{P} + \lambda_1\hat{\mathbf{d}}_1 + \lambda_2\hat{\mathbf{d}}_2 + \lambda_3\hat{\mathbf{d}}_3$

submanifold:

$$T(\alpha,\beta) = \underbrace{\mathbf{P}_f + p(c_\alpha s_\beta\hat{\mathbf{i}} + s_\alpha s_\beta\hat{\mathbf{j}} + c_\beta\hat{\mathbf{k}})}_{\mathbf{S}_f} - R(\underbrace{\mathbf{P}_m}_{\mathbf{S}_m}). \quad (8)$$

3) One Point–Line *Coincidence Constraint* $d(\mathbf{P}_f, \mathcal{L}_m) = 0$*:* The mobile line $\mathcal{L}_m$ is constrained to contain the fixed point $\mathbf{P}_f$ at all times, so $\mathbf{S}_m$ belongs to a line submanifold and $\mathbf{S}_f$ belongs to a point submanifold:

$$T(\lambda) = \underbrace{\mathbf{P}_f}_{\mathbf{S}_f} - R(\underbrace{\mathbf{P}_m + \lambda\hat{\mathbf{d}}_{\mathcal{L}_m}}_{\mathbf{S}_m}) \quad (9)$$

where $\hat{\mathbf{d}}_{\mathcal{L}_m}$ and $\mathbf{P}_m$ are the direction vector and a support point of $\mathcal{L}_m$, respectively.

4) One Point–Line *Distance and One* Point–Plane *Coincidence Constraints* $d(\mathbf{P}_m, \mathcal{K}_f) = p, d(\mathbf{P}_m, \Pi_f) = 0$*:* The mobile point $\mathbf{P}_m$ is constrained to an ellipse (or circle), so $\mathbf{S}_m$ belongs to a point submanifold and $\mathbf{S}_f$ belongs to an elliptical submanifold:

$$T(\alpha) = \underbrace{\mathbf{R}_f + p\left[c_\alpha\hat{\mathbf{d}}_1 + (s_\alpha/s_{\sigma_f})\hat{\mathbf{d}}_2\right]}_{\mathbf{S}_f} - R(\underbrace{\mathbf{P}_m}_{\mathbf{S}_m}) \quad (10)$$

where

$$\mathbf{R}_f = \mathcal{K}_f \cap \Pi_f$$
$$\hat{\mathbf{d}}_1 = \hat{\mathbf{n}}_{\Pi_f} \times \hat{\mathbf{d}}_{\mathcal{K}_f}$$
$$\hat{\mathbf{d}}_2 = \hat{\mathbf{d}}_1 \times \hat{\mathbf{n}}_{\Pi_f}.$$

5) Two Point–Plane *Coincidence Constraints* $d(\mathbf{P}_m, \Sigma_f) = 0, \mathrm{d}(\mathbf{Q}_\mathrm{m}, \Pi_\mathrm{f}) = 0$*:* To solve this case, the two constraints are transformed into an equivalent pair of *point–plane* constraints that share the same mobile point.

Stating (6) for the second constraint yields

$$T = \mathbf{Q}_f - R(\mathbf{Q}_m) \quad (11)$$

where $\mathbf{Q}_f$ is a point contained in the planar submanifold associated to Π_f. Adding and subtracting $\mathbf{P}_m$ and rearranging, we get

$$T = \mathbf{Q}_f - R(\mathbf{Q}_m + \mathbf{P}_m - \mathbf{P}_m) \quad (12)$$

$$T = [\,\mathbf{Q}_f - R(\mathbf{Q}_m - \mathbf{P}_m)\,] - R(\mathbf{P}_m). \quad (13)$$

By defining $\mathbf{S}'_f = \mathbf{Q}_f - R(\mathbf{Q}_m - \mathbf{P}_m)$, (13) becomes

$$T = \mathbf{S}'_f - R(\mathbf{P}_m). \quad (14)$$

Equation (14) is equivalent to the *point–plane* coincidence constraint $d(\mathbf{P}_m, \Upsilon_f) = 0$, where Υ_f is the plane parallel to Π_f that passes through $\mathbf{S}'_f$.

The original constraint pair can now be expressed through the equivalent $d(\mathbf{P}_m, \Sigma_f) = 0, d(\mathbf{P}_m, \Upsilon_f) = 0$, which shares the same mobile point, which in turn can be further simplified into the case of one *point–line* coincidence constraint $d(\mathbf{P}_m, \mathcal{L}_f) = 0$, where $\mathcal{L}_f = \Sigma_f \cap \Upsilon_f$. Notice that $\mathcal{L}_f$ depends on R and must be recomputed every time its value changes. The solution is depicted in Fig. 6(b).

VI. Sample Problems

This section presents two examples that illustrate how PMF works and how it can be applied to its target applications. The first example details the solution process for a simple problem and emphasizes the contribution of each step. The second example consists of a simple assembly, and it is shown how the same constraint set can be used to execute the task in both offline-programmed and teleoperation modes.

A. Simple Example

Consider one more time the fixed and mobile objects depicted in Fig. 1(a) and (b), respectively. The mobile object is to be positioned according to the input constraint set

$$C_I = \{\mathbf{P}_m \subset \mathcal{K}_f, \mathbf{P}_m \subset \mathcal{L}_f, \mathbf{Q}_m = \mathbf{Q}_f\}$$

where $\mathbf{P}_m = [0\ 5\ 3]^T$, $\mathbf{Q}_m = [0\ 7\ 3]^T$, $\mathbf{Q}_f = [-2\ 0\ 3]^T$, and $\mathcal{K}_f$, $\mathcal{L}_f$ are the lines with direction vectors $\hat{\mathbf{d}}_{\mathcal{K}_f} = [0\ 1\ 0]^T$, $\hat{\mathbf{d}}_{\mathcal{L}_f} = [0\ 0\ 1]^T$, and share the support point $\mathbf{P}_f = [0\ 0\ 3]^T$.

Fig. 7 outlines the solution process for this problem, and shows how constraint sets are affected by each step. The solution, depicted in Fig. 1(d), has one rotational DOF.

1) Input Constraint Decomposition: According to Table I, the constraints in C_I become

$$C_T = \{d(\mathbf{P}_m, \mathcal{K}_f) = 0, \quad d(\mathbf{P}_m, \mathcal{L}_f) = 0, \quad d(\mathbf{Q}_m, \mathbf{Q}_f) = 0\}$$
$$C_R = \emptyset.$$

2) Constraint Combination:

a) The first two constraints share the same mobile point, and since $\mathcal{K}_f \cap \mathcal{L}_f = \mathbf{P}_f$, they can be rewritten as $d(\mathbf{P}_m, \mathbf{P}_f) = 0$, yielding

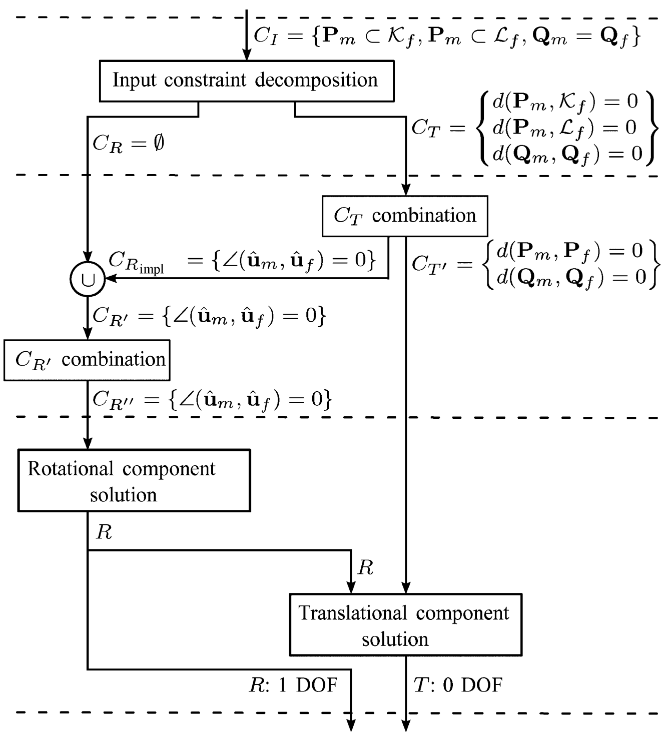
$$C_{T'} = \{d(\mathbf{P}_m, \mathbf{P}_f) = 0, \qquad d(\mathbf{Q}_m, \mathbf{Q}_f) = 0\}.$$

b) Since $d(\mathbf{P}_m, \mathbf{Q}_m) = d(\mathbf{P}_f, \mathbf{Q}_f) \neq 0$, the two constraints in $C_{T'}$ imply the rotation $\angle(\hat{\mathbf{u}}_m, \hat{\mathbf{u}}_f) = 0$, where $\hat{\mathbf{u}}_m = [0\ 1\ 0]^T$ points in the direction of $\overrightarrow{\mathbf{P}_m\mathbf{Q}_m}$ and $\hat{\mathbf{u}}_f = [-1\ 0\ 0]^T$ points in the direction of $\overrightarrow{\mathbf{P}_f\mathbf{Q}_f}$

$$C_{R'} = \{\angle(\hat{\mathbf{u}}_m, \hat{\mathbf{u}}_f) = 0\}.$$

3) Solution Synthesis:

a) *Rotational component:* R is computed for one parallelism constraint (Section V-A2) and has one DOF.



$C_I = \{\mathbf{P}_m \subset \mathcal{K}_f, \mathbf{P}_m \subset \mathcal{L}_f, \mathbf{Q}_m = \mathbf{Q}_f\}$
Input constraint decomposition
$C_R = \emptyset$
$C_T = \begin{Bmatrix} d(\mathbf{P}_m, \mathcal{K}_f) = 0 \\ d(\mathbf{P}_m, \mathcal{L}_f) = 0 \\ d(\mathbf{Q}_m, \mathbf{Q}_f) = 0 \end{Bmatrix}$
C_T combination
$C_{R_{\text{impl}}} = \{\angle(\hat{\mathbf{u}}_m, \hat{\mathbf{u}}_f) = 0\}$
$\cup$
$C_{T'} = \begin{Bmatrix} d(\mathbf{P}_m, \mathbf{P}_f) = 0 \\ d(\mathbf{Q}_m, \mathbf{Q}_f) = 0 \end{Bmatrix}$
$C_{R'} = \{\angle(\hat{\mathbf{u}}_m, \hat{\mathbf{u}}_f) = 0\}$
$C_{R'}$ combination
$C_{R''} = \{\angle(\hat{\mathbf{u}}_m, \hat{\mathbf{u}}_f) = 0\}$
Rotational component solution
R
R
Translational component solution
R: 1 DOF
T: 0 DOF

Fig. 7. Solution process for a problem involving the objects from Fig. 1(a) and (b).

Let $\hat{\mathbf{w}} = \hat{\mathbf{u}}_m \times \hat{\mathbf{u}}_f = [0\ 0\ 1]^T$ and $\sigma = \pi/2$ be the angle between $\hat{\mathbf{u}}_m$ and $\hat{\mathbf{u}}_f$, then

$$R(\phi) = R(\hat{\mathbf{u}}_f, \phi)\, R(\hat{\mathbf{w}}, \sigma)$$

$$R(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & s_\phi \\ 0 & -s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R(\phi) = \begin{bmatrix} 0 & -1 & 0 \\ c_\phi & 0 & s_\phi \\ -s_\phi & 0 & c_\phi \end{bmatrix} \tag{15}$$

b) *Translational component:* $C_{T'}$ is matched to the most restrictive entry of Table IV; hence, T is computed for one *point–point* coincidence constraint[2] (Section V-B1), and has no DOFs.

$$T = \mathbf{P}_f - R(\mathbf{P}_m)$$
$$T = [0\ 0\ 3]^T - R[0\ 5\ 3]^T. \tag{16}$$

Particular instances of the solution can be found by setting the DOF parameter. For example, for $\phi = 0$, the rigid transformation P is obtained

$$P = \begin{bmatrix} 0 & -1 & 0 & 5 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{17}$$

[2]Since $C_{T'}$ consists of two *point–point* coincidence constraints, either choice is valid and will yield a correct result. In this particular case, $d(\mathbf{P}_m, \mathbf{P}_f) = 0$ has been chosen.

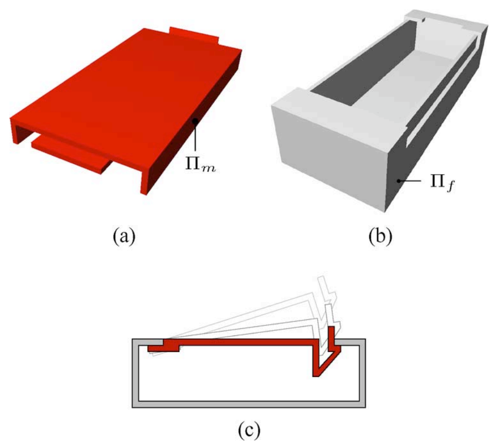


Fig. 8. Assembly task consisting on positioning *cover* (a) with respect to *base* (b) as shown in (c). (a) Cover (mobile object). (b) Base (fixed object). (c) Final configuration.

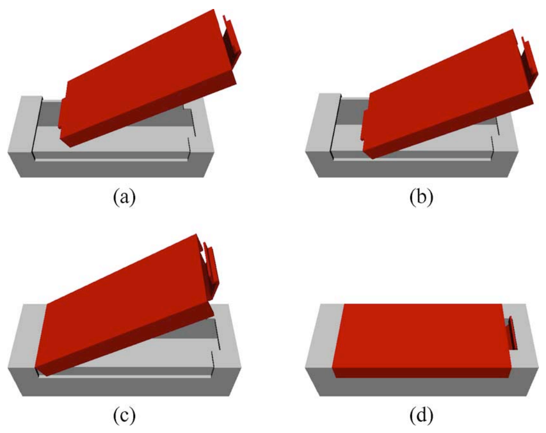


Fig. 9. Sequence followed to accomplish the assembly. Robot gripper is handling *cover*, but is not shown for clarity.

B. Assembly Task

Consider the objects *cover* and *base*, depicted in Fig. 8(a) and (b), respectively. The task consists on securing *cover* (which is being grasped by a robot arm) to *base*, as shown in Fig. 8(c). Notice that since *cover* features a spring-activated locking mechanism, it must be positioned following the right sequence of configurations rather than directly heading to its final state. Consider now the following sequence of configurations for successfully performing the task.

a) Position *cover* at an angle with respect to *base* ensuring the objects do not collide [see Fig. 9(a)].
b) Lower *cover* until it makes contact with the rails on *base*'s sides [see Fig. 9(b)].
c) Translate *cover* along the rails until the back of *cover* makes contact with *base* [see Fig. 9(c)].
d) Rotate *cover* until the spring on its front locks [see Figs. 9(d) and 8(c)].

The constraint set $C_I = \{\Pi_m = \Pi_f\}$ restricts Π_m to coincide with Π_f, reducing the DOFs of *cover* from six to three: one rotation about $\hat{\mathbf{n}}_{\Pi_f}$—the normal of Π_f—and two translations perpendicular to $\hat{\mathbf{n}}_{\Pi_f}$ [whose directions coincide with the horizontal and vertical directions of Fig. 8(c)]. Now it will be shown

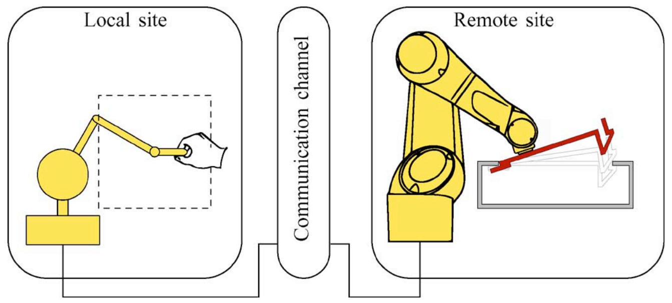


Fig. 10. Teleoperated task where virtual forces are exerted on the haptic device to maintain its end-effector contained in a plane (shown in dotted lines).

how the task can be executed in both offline-programmed and teleoperated modes.

1) Offline-Programmed Task Specification: Configuration a) is a contact-free situation and can be achieved by setting appropriate initial values to the DOF parameters (position control in the DOF parameter space). The remaining configurations can be reached by sequentially moving *cover* along each of its DOFs. However, such configurations involve physical contact between the objects, so contact states should be taken into account.

If the robot that is executing the task has force-sensing capabilities, motion along each DOF direction can be not only position-controlled, but also force-controlled. In such cases, instead of attaining a position setpoint, motion along a DOF direction is carried out until a predefined force threshold is reached. Configurations b) and c) can, respectively, be reached by translating *cover* along the vertical and horizontal translational DOFs until contact is detected, and configuration d) can be reached by actuating the rotational DOF until the final position is attained (identified by a reaction torque). Since the object is moving along its unrestricted directions, the trajectories that it describes will *always* satisfy the imposed geometric constraint.

One important remark is that if the constraints are defined symbolically (i.e., using references to the geometric entities instead of their particular values at some instant in time), the task need not be redefined if the initial configurations of *cover* or *base* change, or if the value of a particular geometric measure is updated by an (external) online sensory module. The solver will automatically recompute the new solutions when necessary. This enables changing the layout of the robot workcell with minimal downtime and no task reprogramming.

2) Teleoperated Execution: Consider a teleoperation scheme such as the one depicted in Fig 10. If the scheme is bilateral, force/torque and probably visual feedback are sent to the operator, which provide him/her with information concerning the contact state of the objects. Also, haptic guidance can be used to enforce the coincidence constraint between Π_m and Π_f by generating virtual forces on the operator, so that he/she only needs to concentrate on actuating the unrestricted directions.

By combining contact state information coming from the real-time execution of the task with haptic guidance, the performance of a teleoperated task can be improved greatly. Of course, the precision with which geometric constraints are haptically enforced is highly dependent on the specific characteristics of the haptic display and the guidance algorithm.

The generation of virtual forces requires translating the kinematic information provided by the solver into a dynamic model. An early implementation of PMF was interfaced in [23] with a bilateral teleoperation scheme where haptic guidance is implemented by attaching a virtual spring–damper system between the actual and desired positions of the haptic end-effector. This work uses a PHANToM impedance haptic display, which has a fairly large workspace but a limited range of exertable forces/torques. Typical translational constraint following errors are in the order of 10^{-3} m.

VII. Performance and Implementation Issues

A. Time Complexity

In the following, the time complexity of the methodology is proven to be quadratic $O(n^2)$—where n represents the number of input constraints—by showing that the individual steps that comprise the solution process are at most $O(n^2)$.

Input constraint decomposition has $O(n)$ complexity because the decomposition of one input constraint into fundamental ones is a constant time operation that must be performed once for each input constraint. Since the decomposition of an input constraint produces at most one translational and one rotational fundamental constraint (Table I), the number of fundamental constraints is upper bounded to n translational and n rotational ones.

Constraint combination has $O(n^2)$ complexity. Given the combinatorial nature of this step, the maximum number of combination tests—which are constant time operations—that are performed on a problem instance is upper bounded by $\binom{n}{2} = n(n-1)/2$. This bound is rarely reached because the number of fundamental constraints usually decreases during the constraint reduction steps. Additionally, since each geometric constraint restricts at least one DOF, the number of input constraints for most practical problems—including those with multiple redundant constraints—is usually small.

Solution synthesis is a constant time operation $O(1)$.

Since multiple solutions are treated by the solver as multiple problem instances, each solution branch has $O(n^2)$ complexity. However, the way in which constraint combination has been implemented is such that combination tests performed prior to the branching operation need not be repeated.

B. Implementation

The current implementation of the PMF solver is written in C++. Its performance has been measured on a desktop PC with an Intel Pentium 4 processor running at 3.4 GHz. The solution times range from 0.05 ms for a simple problem with one constraint and one solution to 1.5 ms for a fully constrained problem with five constraints and 32 distinct solutions. In contrast, methods containing iterative processes are more sensitive to singularities and degeneracies, and tend to have much greater variations in their solution times. Comparatively, the examples of [17] featuring one mobile object have solution time variations of up to five orders of magnitude, as opposed to two for PMF.

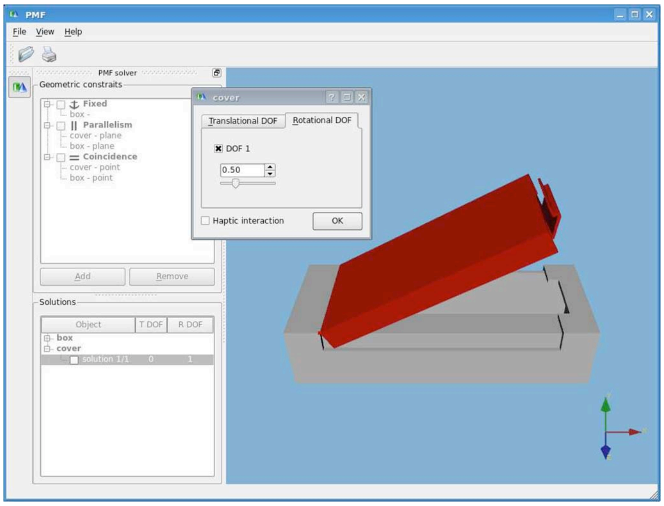


Fig. 11. Screenshot of the solver user interface.

Fig. 11 shows a screenshot of the solver user interface that includes a 3-D model of the task (right) and a control panel (left) that permits the graphical interactive definition of input constraints displays information regarding the solutions of the current problem, and permits moving the mobile object along its current solution submanifold with a mouse/keyboard or a haptic device. Two supplementary moving picture experts group (MPEG) videos that show how the solver interface is used to define relational positioning problems and display its solutions have been included. They will be available at http://ieeexplore.ieee.org.

VIII. Conclusion And Future Work

A relational positioning methodology that serves as an interface for flexibly and intuitively specifying offline-programmed robot tasks, as well as for assisting the execution of teleoperated tasks demanding constrained movements has been presented.

It was shown how an object positioning operation can be formulated in terms of geometric constraints for restricting totally or partially the movements of an object independently of its initial configuration. As a means to find the solutions to such a problem, PMF, a 3-D sequential geometric constraint solver, has been proposed. The solver exploits the fact that in sets of geometric constraints, the rotational component can often be separated from the translational one and solved independently. PMF can handle under-, well-, and overconstrained problems with multiple solutions, and although it is not complete, the solvable subset handles most of the problems a user would be interested in. Solvable problems are those whose rotational part can be expressed in terms of fundamental constraints, and that after constraint combination (Section IV), the resulting constraint sets can be matched to one of the handled scenarios (Section V). Incompatible problems are labeled as unsolvable (ill-defined) when detected, and no best-fit solutions are attempted. Issues such as collisions between objects, geometric uncertainty, workspace of the actuation system, and dynamics are *not* considered by the solver, but if necessary can be dealt with by interfacing the solver with third-party modules.

It has been demonstrated that the solution process has quadratic time complexity $O(n^2)$ for the number of input constraints, and experimental data show that the solution times of the current implementation allow real-time interaction with a human operator and the inclusion of the solver in high-frequency loops that require response times within the milliseconds order of magnitude.

Future lines of work comprise extending the solver's capabilities for handling multiple mobile objects and interfacing it with kinematics and path-planning modules to enable the specification and execution of more complex (multi)robot tasks.

Appendix

Rotation Defined by Two Pairs of Vectors

Given two pairs of independent unit vectors $\{\hat{\mathbf{u}}, \hat{\mathbf{v}}\}$ and $\{\hat{\mathbf{r}}, \hat{\mathbf{s}}\}$, the rotation $R(\hat{\mathbf{u}}, \hat{\mathbf{v}} \rightarrow \hat{\mathbf{r}}, \hat{\mathbf{s}})$ places $\hat{\mathbf{u}}$ parallel to $\hat{\mathbf{r}}$ and $\hat{\mathbf{v}}$ in the plane defined by $\hat{\mathbf{r}}$ and $\hat{\mathbf{s}}$. If $\hat{\mathbf{u}} \cdot \hat{\mathbf{v}} = \hat{\mathbf{r}} \cdot \hat{\mathbf{s}}$, then $\hat{\mathbf{v}}$ will additionally be parallel to $\hat{\mathbf{s}}$.

$$R(\hat{\mathbf{u}}, \hat{\mathbf{v}} \rightarrow \hat{\mathbf{r}}, \hat{\mathbf{s}}) = [\,\hat{\mathbf{r}} \quad \hat{\mathbf{s}}_{\mathbf{r}} \quad \hat{\mathbf{t}}\,]\,[\,\hat{\mathbf{u}} \quad \hat{\mathbf{v}}_{\mathbf{u}} \quad \hat{\mathbf{w}}\,]^T \tag{A1}$$

where $\{\hat{\mathbf{u}}, \hat{\mathbf{v}}_{\mathbf{u}}, \hat{\mathbf{w}}\}$ and $\{\hat{\mathbf{r}}, \hat{\mathbf{s}}_{\mathbf{r}}, \hat{\mathbf{t}}\}$ are the orthonormal basis associated to $\{\hat{\mathbf{u}}, \hat{\mathbf{v}}\}$ and $\{\hat{\mathbf{r}}, \hat{\mathbf{s}}\}$, respectively.

Given two independent unit vectors $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$, the orthonormal basis associated to them is $\{\hat{\mathbf{u}}, \hat{\mathbf{v}}_{\mathbf{u}}, \hat{\mathbf{w}}\}$, where

$$\hat{\mathbf{v}}_{\mathbf{u}} = \frac{\hat{\mathbf{v}} - (\hat{\mathbf{u}} \cdot \hat{\mathbf{v}})\hat{\mathbf{u}}}{|\hat{\mathbf{v}} - (\hat{\mathbf{u}} \cdot \hat{\mathbf{v}})\hat{\mathbf{u}}|} \tag{A2}$$

$$\hat{\mathbf{w}} = \hat{\mathbf{u}} \times \hat{\mathbf{v}}_{\mathbf{u}}. \tag{A3}$$

References

[1] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx, "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *Int. J. Robot. Res.*, vol. 26, no. 5, pp. 433–455, 2007.

[2] U. Thomas, B. Finkemeyer, T. Kröger, and F. Wahl, "Error-tolerant execution of complex robot tasks based on skill primitives," in *Proc. IEEE Int. Conf. Robot. Autom.*, Taipei, Taiwan, R.O.C., Sep. 14–19, 2003, pp. 3069–3075.

[3] U. Thomas and F. Wahl, "A system for automatic planning, evaluation and execution of assembly sequences for industrial robots," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, Maui, HI, Oct. 29–Nov. 3, 2001, pp. 1458–1464.

[4] N. Turro, O. Khatib, and E. Coste-Maniere, "Haptically augmented teleoperation," in *Proc. IEEE Int. Conf. Robot. Autom.*, Seoul, Korea, May. 21–26, 2001, pp. 386–392.

[5] B. P. DeJong, E. L. Faulring, J. E. Colgate, and M. A. Peshkin, "Lessons learned from a novel teleoperation testbed," *Ind. Robot: Int. J.*, vol. 33, no. 3, pp. 187–193, 2006.

[6] E. L. Faulring, K. M. Lynch, J. Colgate, and M. A. Peshkin, "Haptic display of constrained dynamic systems via admittance displays," *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 101–111, Feb. 2007.

[7] C. Hoffmann and R. Joan-Arinyo, "A brief on constraint solving," *Comput.-Aided Des. Appl.*, vol. 2, no. 5, pp. 655–664, 2005.

[8] I. Fudos and C. Hoffmann, "A graph-constructive approach to solving systems of geometric constraints," *ACM Trans. Graph.*, vol. 16, no. 2, pp. 179–216, 1997.

[9] J. Owen, "Algebraic solution for geometry from dimensional constraints," in *Proc. Symp. Solid Modeling Foundations CAD/CAM Appl.*, R. Rossignac and J. Turner, Eds. Austin, TX: ACM, Jun. 5–7, 1991, pp. 397–407.

[10] G. Kramer, *Solving Geometric Constraint Systems*. Cambridge, MA: MIT Press, 1992.

[11] B. Bruderlin, "Symbolic computer geometry for computer aided geometric design," in *Advances in Design and Manufacturing Systems*. Tempe, AZ: NSF, Jan. 8–12, 1990.

[12] B. Bruderlin, "Using geometric rewrite rules for solving geometric problems symbolically," in *Theoretical Computer Science*. vol. 116, Amsterdam, The Netherlands: Elsevier, 1993, pp. 291–303.
[13] G. Nelson, "Juno, a constraint-based graphics system," in *Proc. SIGGRAPH*, San Francisco, CA, Jul. 22–26, 1985, pp. 235–243.
[14] H. Lamure and D. Michelucci, "Solving geometric constraints by homotopy," in *Proc. 3rd Symp. Solid Modeling Appl.*, C. Hoffmann and J. Rossignac, Eds. Salt Lake City, UT: ACM, May 17–19, 1995, pp. 263–269.
[15] K. Kondo, "Algebraic method for manipulation of dimensional relationships in geometric models," *Comput.-Aided Des.*, vol. 24, no. 3, pp. 141–147, Mar. 1992.
[16] A. Kumar and L. Yu, "Sequential constraint imposition for dimension-driven solid models," in *Comput.-Aided Des.*, vol. 33, no. 6, pp. 475–486, 2001.
[17] J. M. Porta, L. Ros, F. Thomas, and C. Torras, "A branch-and-prune solver for distance constraints," *IEEE Trans. Robot.*, vol. 21, no. 2, pp. 176–187, Apr. 2005.
[18] A. Rodríguez, L. Basañez, and E. Celaya, "Robot task specification and execution through relational positioning," presented at the IFAC Workshop Intell. Manuf. Syst., Alicante, Spain, May 23–25, 2007.
[19] A. Rodríguez, L. Basañez, and E. Celaya. (2007, Sep.). Description of a robotics-oriented relational positioning methodology. Tech. Univ. Catalonia. Barcelona, Spain, Tech. Rep. [Online]. Available: https://upcommons.upc.edu/e-prints/handle/2117/1531?locale=en
[20] E. Celaya, "Geometric reasoning for the determination of the position of objects linked by spatial relationships," Ph.D. dissertation, Univ. Politècnica Catalunya. Barcelona, Spain, 1992.
[21] E. Celaya, "LMF: A program for positioning objects using geometric relationships," in *Proc. VII Int. Conf. Appl. Artif. Intell. Eng.*, D. G. *et al.*, Ed. Amsterdam, The Netherlands: Computational Mechanics/Elsevier, 1992, pp. 873–881.
[22] C. Hoffmann, "Constraint-based CAD," *J. Comput. Inf. Sci. Eng.*, vol. 5, no. 3, pp. 182–197, 2005.
[23] E. Nuño, A. Rodríguez, and L. Basañez, "Force reflecting teleoperation via ipv6 protocol with geometric constraints haptic guidance," in *Springer Transactions in Advanced Robotics—Advances in Telerobotics*, vol. 31, S. Verlag, Ed. New York: Springer-Verlag, 2007, ch. 26, pp. 445–458.

Adolfo Rodríguez (S'08) received the B.Sc. (Hons.) degree in mechanical engineering from Simón Bolívar University, Caracas, Venezuela, in 2003. He is currently working toward the Ph.D. degree in robotics at the Institute of Industrial and Control Engineering (IOC), Barcelona, Spain.

His current research interests include geometric constraint solving, haptic interfaces, and semi-autonomous robots.

Luis Basañez (M'72) received the Ph.D. degree in electrical engineering from the Technical University of Catalonia (UPC), Barcelona, Spain, in 1975.

From 1976 to 1987, he was the Vice Director of the Institute of Cybernetics (UPC), where he was the Director from 1987 to 1990. Since 1986, he has been a Full Professor of system engineering and automatic control at the UPC, where since 1990, he has been the Head of the Robotics Division, Institute of Industrial and Control Engineering. His current research interests include task planning, multirobot systems coordination, teleoperation, sensor integration, and active perception.

Dr. Basañez was an Executive Committee Member of the International Federation of Robotics (IFR) from 1987 to 1992. He is currently the Spanish delegate at the IFR. In 2005, he was a Fellow of the International Federation of Automatic Control (IFAC). Since then, he has been a Member of the IFAC Council.

Enric Celaya received the B.Sc. degree in physics from the Universitat de Barcelona, Catalonia, Spain, in 1979, and the Ph.D. degree in artificial intelligence from the Universitat Politècnica de Catalunya, Barcelona, Spain, in 1992.

He is currently a Full Researcher at the Consejo Superior de Investigaciones Científicas, Institut de Robòtica i Informàtica Industrial (CSIC-UPC), Barcelona. He has been engaged in research projects in the field of vision-based robot navigation in natural environments. His current research interests include geometric reasoning, legged robot locomotion, reinforcement learning in robotics, and visual saliency detection.