# A Branch and Prune Algorithm for the Computation of Generalized Aspects of Parallel Robots

S. Caro[1], D. Chablat[1], A. Goldsztejn[2], D. Ishii[3], and C. Jermann[2]

[1] IRCCyN, Nantes, France
first.last@irccyn.ec-nantes.fr
[2] Université de Nantes/CNRS, LINA (UMR-6241), Nantes, France.
first.last@univ-nantes.fr
[3] National Institute of Informatics, JSPS, Tokyo, Japan
dsksh@acm.org

**Abstract.** Parallel robots enjoy enhanced mechanical characteristics that have to be contrasted with a more complicated design. In particular, they often have parallel singularities at some poses, and the robot may become uncontrollable, and could even be damaged, in such configurations. The computation of singularity free sets of reachable poses, called generalized aspects, is therefore a key issue in their design. A new methodology based on numerical constraint programming is proposed to compute a certified enclosure of such generalized aspects which fully automatically applies to arbitrary robot kinematic model.

**Keywords:** Numerical constraints; parallel robots; singularities; aspects.

## 1 Introduction

Mechanical manipulators, commonly called robots, are widely used in the industry to automatize various tasks. Robots are a mechanical assembly of rigid links connected by mobile joints. Some joints are actuated and they allow commanding the robot operating link, called its end-effector (or platform). One key characteristic of a robot is its reachable workspace, informally defined as the set of poses its end-effector can reach. Indeed, its size defines the scope of operational trajectories the robot will be able to perform. Robots comply with either a serial or a parallel (or possibly a hybrid) assembly, whether their links are connected in series or in parallel. Parallel robots [14, 16] present several advantages with respect to serial ones: They are naturally stiffer, leading to more accurate motions with larger loads, and allow high speed motions. These advantages are contrasted by a more complicated design as the computation and analysis of their workspace present several difficulties. First, one pose of the robot's end-effector may be reached by several different sets of actuated joint commands (which correspond to different working modes), and conversely one set of input commands may lead to several poses of its end-effector (which correspond to different assembly modes). Second, parallel robots generally have parallel singularities, i.e., specific configurations where they become uncontrollable and can even be damaged.

The kinematics of a parallel robot is modeled by a system of equations that relates the pose of its end-effector (which includes its position and possibly its orientation)

to its commands. Hence computing the pose knowing the commands, or conversely, requires solving a system of equations, called respectively the direct and inverse kinematic problems. Usually, the number of pose coordinates (also known as *degrees of freedom* (DOF)), the number of commands and the number of equations are the same. Therefore, the relation between the pose and the command is generically a local bijection. However, in some non-generic configurations, the pose and the command are not anymore related by a local bijection. This may affect the robot behavior, e.g., destroying it if some command is enforced with no corresponding pose. These non-generic cases are called robot singularities and they can be of two kinds [1]: Serial or parallel. One central issue in designing parallel robots is to compute connected sets of singularity free poses, so that the robot can safely operate inside those sets. Such a set is called a generalized aspect in [6] when it is maximal with respect to inclusion.

A key issue when computing the aspects is the certification of the results: Avoiding singularities is mandatory, and the connectivity between robot configurations must be certified. This ensures that the robot can actually move safely from one configuration to another. Very few frameworks provide such certifications, among which algebraic computations and interval analysis. The cylindrical algebraic decomposition was used in [5], with the usual restrictions of algebraic methods and with a connectivity analysis limited to robots with two DOF. Interval analysis was used in [17] for robots having a single solution to their inverse kinematic problem. Though limited, this method can still tackle important classes of robots like the Stewart platform. A quad-tree with certification of non-singularity was built in [4] for some planar robots with two DOF. This method extends to higher dimensional robots, but it requires the a priori separation of working modes by adhoc inequalities, and is not certified with respect to connectivity.

In this paper we propose a branch and prune algorithm incorporating the certification of the solutions and of their connectivity. This allows a fully automated computation of the generalized aspect from the model of arbitrary parallel robots, including robots with multiple solutions to their direct and inverse kinematic problems, without requiring any a priori study to separate their working modes. The algorithm is applicable to robots of any dimension, although the complexity of the computations currently restricts its application to robots with three degrees of freedom.

A motivating example is presented in Section 2 followed by some preliminaries about numerical constraint programming and robotics in Section 3. The proposed algorithm for certified aspect computation is presented in Section 4. Finally, experiments on planar robots with two and three degrees of freedom are presented in Section 5.

**Notations** Boldface letters denote vectors. Thus $\mathbf{f}(\mathbf{x}) = 0$ denotes a system of equations $\mathbf{f}$ on a vector of variables $\mathbf{x}$: $f_1(x_1, \ldots, x_n) = 0, \ldots, f_k(x_1, \ldots, x_n) = 0$. The Jacobian matrix of $\mathbf{f}(\mathbf{x})$ with respect to variables $\mathbf{x}' \subseteq \mathbf{x}$ is denoted $\mathbf{F}_{\mathbf{x}'}(\mathbf{x})$, and $\det \mathbf{F}_{\mathbf{x}'}(\mathbf{x})$ denotes its determinant. Interval variables are denoted using bracketed symbols, e.g. $[x] = [\underline{x}, \overline{x}] := \{x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x}\}$. Hence, $[\mathbf{x}]$ is an interval vector (or box) and $[A] = ([a_{ij}])$ is an interval matrix. $\mathbb{IR}$ denotes the set of intervals and $\mathbb{IR}^n$ the set of $n$-dimensional boxes. For an interval $[x]$, we denote $\mathrm{wid}[x] := \overline{x} - \underline{x}$ its width, $\mathrm{int}[x] := \{x \in \mathbb{R} \mid \underline{x} < x < \overline{x}\}$ its interior, and $\mathrm{mid}[x] := (\underline{x} + \overline{x})/2$ its midpoint. These notations are extended to interval vectors.

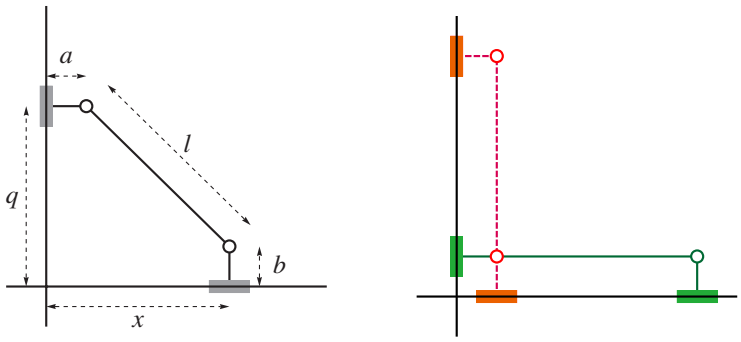



**Fig. 1.** The PRRP in a nonsingular pose (left) and in singular poses (right).

## 2 Motivating Example

*Description.* Consider the simple PRRP[4] planar robot depicted in Figure 1 which involves two prismatic joints (gray rectangles) sliding along two perpendicular axes. These prismatic joints are connected through three rigid bars (black lines) linked by two revolute joints (circles) which allow free rotations between the rigid bars. The positions of the prismatic joints are respectively denoted by $x$ and $q$, the end-effector position $x$ being on the horizontal axis and the command $q$ corresponding to the height on the vertical axis. The left-hand side diagram of Figure 1 shows one nonsingular configuration of the robot (note that there is another symmetric pose $x$ associated to the same command $q$ where $x$ is negative). From this configuration, when $q$ changes vertically $x$ changes horizontally, and both are related by a local bijection, hence this configuration is non-singular. The right-hand side diagram shows two singular configurations. In the green, plain line, pose (where the robot's main rigid bar is horizontal), increasing or decreasing the command $q$ entails a decrease of $x$, hence a locally non-bijective correspondence between these values. In the red, dashed line, pose (where the robot's main rigid bar is vertical), increasing or decreasing the command $q$ would entail a vertical motion of the end-effector which is impossible due to the robot architecture, hence a potential damage to the robot structure. The green configuration is a serial singularity, which restricts the robot mobility without damaging it; the red configuration is a parallel singularity, which is generally dangerous for the robot structure.

*Kinematic model.* The kinematic model of this robot is easily derived: The coordinates of the revolute joints are respectively $(a, q)$ and $(x, b)$, where $a$ and $b$ are architecture parameters corresponding to the lengths of the two horizontal and vertical small rigid bars. Then the main oblique rigid bar enforces the distance between these two points to be equal to its length $l$, a third architecture parameter. Hence, the kinematic model is

$$(x - a)^2 + (q - b)^2 = l^2. \tag{1}$$

[4] In robotics, manipulators are typically named according to the sequence of joints they are made of, e.g., P for prismatic and R for revolute, actuated ones being underlined.

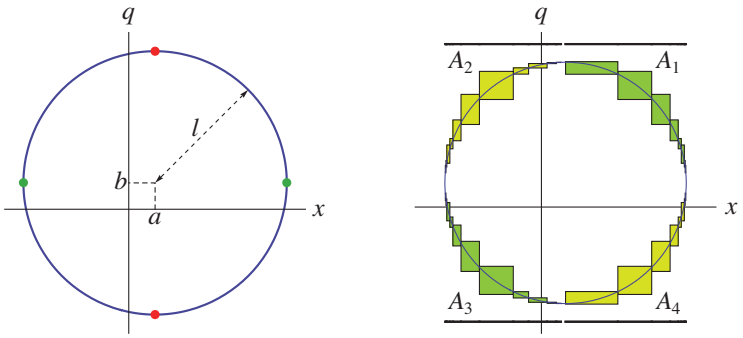



**Fig. 2.** Left: The PRRP kinematic model solutions set. Right: The computed paving.

The solution set of this model, the circle of center $(a, b)$ and radius $l$, is depicted on the left hand side diagram in Figure 2. The direct kinematic problem consists in computing $x$ knowing $q$, leading in the case of this robot to two solutions $a \pm \sqrt{l^2 - (q-b)^2}$ if $q \in [b-l, b+l]$, no solution otherwise. Similarly, the inverse kinematic problem consists in computing $q$ knowing $x$, leading to two solutions $b \pm \sqrt{l^2 - (x-a)^2}$ provided that $x \in [a-l, a+l]$, no solution otherwise. This simple robot is typical of parallel robots, which can have several solutions to their direct and inverse kinematic problems. It is also typical regarding its singularities: It has two serial singularities where the solution set has a vertical tangent (green, leftmost and rightmost, points on the left hand side diagram in Figure 2), and two parallel singularities where the solution set has a horizontal tangent (red, topmost and bottommost, points on the left hand side diagram in Figure 2). These four singularities split the solution set into four singularity free regions, called generalized aspects. Accordingly, we can determine the singularity free reachable workspace of the robot by projecting each aspect onto the $x$ component (see the thick lines above and under the paving on the right-hand side in Figure 2).

*Certified Enclosure of Generalized Aspects.* This paper aims at using numerical constraint programming in order to compute some certified enclosures of the different aspects. The standard branch and prune algorithm is adapted in such a way that solving the robot kinematic model together with non-singularity constraints leads to the enclosure depicted on the right-hand side of Figure 2. The solution boxes verify:

1. In each box, for each pose $x$ there exists a unique command $q$ such that $(x, q)$ satisfies the robot kinematic model;
2. Each box does not contain any singularity;
3. Each two neighbor boxes share a common solution.

The first two properties ensure that each box is crossed by a single aspect, and that this aspect covers the whole box projection on the $x$ subspace. The third certificate allows connecting neighbor boxes proving that they belong to the same aspect. Therefore, the connected components $A_1, A_2, A_3, A_4$ of the computed boxes shown in Figure 2 allow separating the four aspects, and provide, by projection, certified inner approximations of the singularity-free reachable workspace of the robot.

## 3 Preliminaries

### 3.1 Numerical Constraint Programming

Numerical constraint solving inherits principles and methods from discrete constraint solving [18] and interval analysis [19]. Indeed, as their variable domains are continuous subsets of $\mathbb{R}$, it is impossible to enumerate the possible assignments and numeric constraint solvers thus resorts to interval computations. As a result, we use an *interval extension* $[f] : \mathbb{IR}^n \to \mathbb{IR}$ of each function $f : \mathbb{R}^n \to \mathbb{R}$ involved in a constraint, such that $\forall [\mathbf{a}] \in \mathbb{IR}^n,\ \forall \mathbf{a} \in [\mathbf{a}],\ f(\mathbf{a}) \in [f]([\mathbf{a}])$.

**Numerical Constraint Satisfaction Problems** A *numerical constraint satisfaction problem* (NCSP) is defined as a triple $\langle \mathbf{v}, [\mathbf{v}], c \rangle$ that consists of

- a vector of *variables* $\mathbf{v} = (v_1, \ldots, v_n)$,
- an *initial domain*, in the form of a box, represented as $[\mathbf{v}] \in \mathbb{IR}^n$, and
- a *constraint* $c(\mathbf{v}) := (\mathbf{f}(\mathbf{v}) = 0 \ \wedge \ \mathbf{g}(\mathbf{v}) \geq 0)$, $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^e$ and $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^i$, i.e., a conjunction of equations and inequalities.

A *solution* of an NCSP is an assignment of its variables $\mathbf{v} \in [\mathbf{v}]$ that satisfies its constraints. The *solution set* $\Sigma$ of an NCSP is the region within its initial domain that satisfies its constraints, i.e., $\Sigma([\mathbf{v}]) := \{\mathbf{v} \in [\mathbf{v}] \mid c(\mathbf{v})\}$.

**The Branch and Prune Algorithm** The *branch and prune algorithm* [22] is the standard complete solving method for NCSPs. It takes a problem as an input and outputs two sets of boxes, called respectively the undecided ($\mathcal{U}$) and solution ($\mathcal{S}$) boxes. It interleaves a refutation phase, called *prune*, that eliminates inconsistent assignments within a box, and an exploration phase, called *branch*, that divides a box into several sub-boxes to be searched recursively, until a prescribed precision $\epsilon$ is reached. Algorithm 1 shows a generic description of this scheme. It involves four subroutines: $\mathrm{Extract}$ (extraction of the next box to be processed), $\mathrm{Prune}_c$ (reduction of the domains based on refutation of assignments that cannot satisfy a subset of constraint $c$), $\mathrm{Prove}_c$ (certification that a box contains a solution of the constraint $c$), and $\mathrm{Branch}$ (division of the processed box into sub-boxes to be further processed). Each of them has to be instantiated depending on the problem to be solved. The procedure $\mathrm{Prune}_c$ obviously depends on the type of constraint in the problem (e.g. inequalities only), as well as on other characteristics of the problem. The procedures $\mathrm{Extract}$ and $\mathrm{Branch}$ allow defining the search strategy (e.g. breadth-first, depth-first, etc.) which may be tuned differently with regards to the problem. The procedure $\mathrm{Prove}_c$ actually defines the aim of the branch and prune: Being a solution can take different meaning depending on the considered problem and the question asked.[5] For instance, if the question is to find the real solutions of a well-constrained system of equations, then it will generally implement a solution existence (and often uniqueness) theorem, e.g., Miranda, Brouwer or interval Newton [21], that guarantees that the considered box contains a (unique) real solution; on the other hand,

[5] For discrete CSPs, $\mathrm{Prove}_c$ usually checks the given assignment satisfies the constraint.

**Algorithm 1** Branch and prune

**Input:** NCSP $\langle \mathbf{v}, ([\mathbf{v}]), c \rangle$, precision $\epsilon > 0$
**Output:** pair of sets of boxes $(\mathcal{U}, \mathcal{S})$

$\mathcal{L} \leftarrow \{[\mathbf{v}]\}$, $\mathcal{S} \leftarrow \emptyset$ and $\mathcal{U} \leftarrow \emptyset$
**while** $\mathcal{L} \neq \emptyset$ **do**
  $[\mathbf{v}] \leftarrow \mathrm{Extract}(\mathcal{L})$
  $[\mathbf{v}] \leftarrow \mathrm{Prune}_c([\mathbf{v}])$
  **if** $[\mathbf{v}] \neq \emptyset$ **then**
    **if** $\mathrm{Prove}_c([\mathbf{v}])$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{[\mathbf{v}]\}$
    **else if** $\mathrm{wid}[\mathbf{v}] > \epsilon$ **then**
      $\mathcal{L} \leftarrow \mathcal{L} \cup \mathrm{Branch}([\mathbf{v}])$
    **else**
      $\mathcal{U} \leftarrow \mathcal{U} \cup \{[\mathbf{v}]\}$
    **end if**
  **end if**
**end while**
**return** $(\mathcal{U}, \mathcal{S})$

if the question is to compute the solution set of a conjunction of inequality constraints, then it will usually implement a solution universality test that guarantees that every real assignment in the considered box is a solution of the NCSP.

### 3.2 Parallel Robots, Singularities and Generalized Aspects

As illustrated in Section 2, the *kinematic model* of a parallel robot can be expressed as a system of equations relating its end-effector pose $\mathbf{x}$ and its commands $\mathbf{q}$:

$$\mathbf{f}(\mathbf{x}, \mathbf{q}) = 0. \tag{2}$$

The subspace restricted to the pose parameters $\mathbf{x}$ (resp. command parameters $\mathbf{q}$) is known as the *workspace* (resp. *joint-space*). The projection $\Sigma_{\mathbf{x}}$ (resp. $\Sigma_{\mathbf{q}}$) of the solution set of equation 2 is called the robot *reachable workspace* (resp. *reachable joint-space*). The solution set $\Sigma$ itself is called the *kinematic manifold* and lies in what is known as the (pose-command) *product space*. In this paper, we restrict to the most typical architectures which satisfy $\dim \mathbf{x} = \dim \mathbf{q} = \dim \mathbf{f} = n$, i.e., neither over- nor under-actuated manipulators. Then, by the implicit function theorem, this system of equations defines a local bijection between $\mathbf{x}$ and $\mathbf{q}$ provided the Jacobian matrices $\mathbf{F}_{\mathbf{x}}(\mathbf{x}, \mathbf{q})$ and $\mathbf{F}_{\mathbf{q}}(\mathbf{x}, \mathbf{q})$ are non-singular. The configurations $(\mathbf{x}, \mathbf{q})$ that do not satisfy these regularity conditions are called singularities, respectively serial or parallel whether $\mathbf{F}_{\mathbf{q}}(\mathbf{x}, \mathbf{q})$ or $\mathbf{F}_{\mathbf{x}}(\mathbf{x}, \mathbf{q})$ is singular. These singularity conditions correspond to the horizontal and vertical tangents of the kinematic manifold described in Section 2.

A key issue in robotics is to be able to control a robot avoiding singularities (in particular reaching a parallel singularity can dramatically damage a robot). This leads to the definition of *generalized aspects* [6] as maximal connected sets of nonsingular

configurations $(\mathbf{x}, \mathbf{q})$ that can all be reached without crossing any singularity. More formally, the set of reachable nonsingular configurations of the robot is

$$\{(\mathbf{x}, \mathbf{q}) \in \mathbb{R}^n \times \mathbb{R}^n \mid \mathbf{f}(\mathbf{x}, \mathbf{q}) = 0, \det \mathbf{F}_{\mathbf{x}}(\mathbf{x}, \mathbf{q}) \neq 0, \det \mathbf{F}_{\mathbf{q}}(\mathbf{x}, \mathbf{q}) \neq 0\}. \quad (3)$$

This corresponds e.g. to the left hand side diagram in Figure 2 where the four singularities (green and red points) are removed. As illustrated by this diagram, the set (3) is generally made of different maximal connected components. The generalized aspects of the robots are defined to be these maximal connected singularity free components.

For a given generalized aspect $\mathbb{A}$, its projection $\mathbb{A}_{\mathbf{x}}$ is a maximal singularity-free region in the robot reachable workspace. Knowing these regions allows roboticians to safely plan robot motions: Any two poses in $\mathbb{A}_{\mathbf{x}}$ are connected by at least one singularity-free path. In addition, the study of aspects provides important information about robot characteristics, e.g., if $(\mathbf{x}, \mathbf{q})$ and $(\mathbf{x}, \mathbf{q}')$ exist in an aspect $\mathbb{A}$ and $\mathbf{q} \neq \mathbf{q}'$, i.e., two different commands yield the same pose, then the robot is said to be *cuspidal* [20]. Cuspidal robots can change working mode without crossing singularities, yielding an extra flexibility in their usage. Finally, the computation of aspects allows roboticians to make informed choices when designing a robot for a given task.

## 4 Description of the Method

The proposed method for the generalized aspect computation relies on solving the following NCSP whose solutions are the nonsingular configurations of the robot:

$$\Big\langle (\mathbf{x}, \mathbf{q}) \ , \ ([\mathbf{x}], [\mathbf{q}]) \ , \ \mathbf{f}(\mathbf{x}, \mathbf{q}) = 0 \wedge \det \mathbf{F}_{\mathbf{x}}(\mathbf{x}, \mathbf{q}) \neq 0 \wedge \det \mathbf{F}_{\mathbf{q}}(\mathbf{x}, \mathbf{q}) \neq 0 \Big\rangle. \quad (4)$$

Let $\Sigma([\mathbf{x}], [\mathbf{q}])$ be the solution set of this NCSP. Our method computes a set of boxes partly covering this solution set, grouped into connected subsets that represent approximations of the aspects of the considered robot. The computed boxes have to satisfy the specific properties stated in Subsection 4.1. The corresponding branch and prune instantiation is described in Subsection 4.2. The connections between the output boxes have to be certified as described in Subsection 4.3.

### 4.1 From the NCSP Model to the Generalized Aspects Computation

We aim at computing a (finite) set of boxes $\mathcal{S} \subseteq \mathbb{IR}^n \times \mathbb{IR}^n$ together with (undirected) links $\mathcal{N} \subseteq \mathcal{S}^2$, satisfying the following three properties:

**(P1)** $\forall([\mathbf{x}], [\mathbf{q}]) \in \mathcal{S}, \ \forall \mathbf{x} \in [\mathbf{x}], \ \exists \text{ a unique } \mathbf{q} \in [\mathbf{q}], \ \mathbf{f}(\mathbf{x}, \mathbf{q}) = 0;$
**(P2)** $\forall([\mathbf{x}], [\mathbf{q}]) \in \mathcal{S}, \ \forall \mathbf{x} \in [\mathbf{x}], \ \forall \mathbf{q} \in [\mathbf{q}], \det \mathbf{F}_{\mathbf{x}}(\mathbf{x}, \mathbf{q}) \neq 0 \wedge \det \mathbf{F}_{\mathbf{q}}(\mathbf{x}, \mathbf{q}) \neq 0;$
**(P3)** $\forall\big(([\mathbf{x}], [\mathbf{q}]), ([\mathbf{x}'], [\mathbf{q}'])\big) \in \mathcal{N}, \ \exists (\mathbf{x}, \mathbf{q}) \in ([\mathbf{x}], [\mathbf{q}]) \cap ([\mathbf{x}'], [\mathbf{q}']), \mathbf{f}(\mathbf{x}, \mathbf{q}) = 0.$

Property (P1) allows defining in each $([\mathbf{x}], [\mathbf{q}]) \in \mathcal{S}$ a function $\kappa_{([\mathbf{x}],[\mathbf{q}])} : [\mathbf{x}] \to [\mathbf{q}]$ that associates the unique command $\mathbf{q} = \kappa_{([\mathbf{x}],[\mathbf{q}])}(\mathbf{x})$ with a given position $\mathbf{x} \in [\mathbf{x}]$ (i.e., the solution of the inverse kinematic problem locally defined inside $([\mathbf{x}], [\mathbf{q}])$). Property (P2) allows applying the Implicit Function Theorem to prove that $\kappa_{([\mathbf{x}],[\mathbf{q}])}$ is

differentiable (and hence continuous) inside $[\mathbf{x}]$. Therefore, for a given box $([\mathbf{x}], [\mathbf{q}]) \in \mathcal{S}$, the solution set restricted to this box

$$\Sigma([\mathbf{x}], [\mathbf{q}]) = \left\{ \left( \mathbf{x}, \kappa_{([\mathbf{x}],[\mathbf{q}])}(\mathbf{x}) \right) : \mathbf{x} \in [\mathbf{x}] \right\} \tag{5}$$

is proved to be connected and singularity free, and is thus a subset of one generalized aspect. These properties are satisfied by the motivating example output shown on the right-hand side in Figure 2. Remark that given a box $([\mathbf{x}], [\mathbf{q}]) \in \mathcal{S}$ and a position $\mathbf{x} \in [\mathbf{x}]$, the corresponding command $\kappa_{([\mathbf{x}],[\mathbf{q}])}(\mathbf{x})$ is easily computed using Newton iterations applied to the system $\mathbf{f}(\mathbf{x}, \cdot) = 0$ with initial iterate $\tilde{\mathbf{q}} \in [\mathbf{q}]$ (e.g. $\tilde{\mathbf{q}} = \mathrm{mid}[\mathbf{q}]$).

Property (P3) basically entails that $\Sigma([\mathbf{x}], [\mathbf{q}])$ and $\Sigma([\mathbf{x}'], [\mathbf{q}'])$ are connected, and are thus subsets of the same aspect. Finally, assuming $\mathcal{S}_k \subseteq \mathcal{S}$ to be a connected component of the undirected graph $(\mathcal{S}, \mathcal{N})$, the solution set

$$\bigcup_{([\mathbf{x}],[\mathbf{q}]) \in \mathcal{S}_k} \Sigma([\mathbf{x}], [\mathbf{q}]) \tag{6}$$

is proved to belong to one generalized aspect. The next two subsections explain how to instantiate the branch and prune algorithm in order to achieve these three properties.

### 4.2 Instantiaton of the Branch and Prune Algorithm

**Pruning** In our context, implementing the $\mathrm{Prune}_c$ function as a standard AC3-like fixed-point propagation of contracting operators that enforce local consistencies, like the Hull [2, 15] or the Box consistencies [2, 10], is sufficient. Indeed, this allows an inexpensive refutation of non-solution boxes. Moreover, stronger consistency can be achieved at no additional cost thanks to the solution test described below: the interval-Newton-based operator applied for certifying that a box covers an aspect can also refute non-solution boxes and allows pruning with respect to the whole constraint.

**Search Strategy** The standard search strategy for NCSPs applies appropriately in our context. Because boxes are output as soon as they are certified or they have reached a prescribed precision, using a DFS approach to the $\mathrm{Extract}$ function is adequate and avoids the risk of filling up the memory, unlike a BFS or LFS[6] approach. The $\mathrm{Branch}$ function typically selects a variable in a round-robin manner (i.e., all domains are selected cyclically) and splits the corresponding interval at its midpoint (i.e., a domain is split into two halves).

**Solution Test** The $\mathrm{Prove}_c$ procedure of Algorithm 1 has to return true only when Property (P1) and Property (P2) are verified. The former is related to proving the existence of solution which is performed using a parametric Newton operator as described in the following paragraph. The latter requires checking the regularity of some interval matrices as described in the next paragraph.

[6] Largest-first search

*Existence proof.* The standard way to prove that a box $([\mathbf{x}], [\mathbf{q}])$ satisfies Property (P1) is to use a parametric interval Newton existence test [8, 9, 11]. Using the Hansen-Sengupta [21] version of the interval Newton, the following sequence is computed

$$[\mathbf{q}^0] := [\mathbf{q}], \ldots, [\mathbf{q}^{k+1}] := [H]([\mathbf{q}^k]) \cap [\mathbf{q}^k] \tag{7}$$

where $[H]$ is the Hansen-Sengupta operator applied to the system $\mathbf{f}([\mathbf{x}], \cdot) = 0$. As soon as $\emptyset \neq [\mathbf{q}^{k+1}] \subseteq \text{int}[\mathbf{q}^k]$ is verified, the box $([\mathbf{x}], [\mathbf{q}^{k+1}])$ is proved to satisfy Property (P1), and hence so does $([\mathbf{x}], [\mathbf{q}])$ since the former is included in the latter. However, because Algorithm 1 has to bisect the domain $[\mathbf{q}]$ for insuring convergence by separating the different commands associated to the same pose,[7] this test fails in practice in most situations. This issue was overcome in [11], in the restricted context of constraints of the form $\mathbf{x} = \mathbf{f}(\mathbf{q})$, by computing $[\mathbf{q}^{k+1}] := [H]([\mathbf{q}^k])$ in (7), i.e., removing the intersection with $[\mathbf{q}^k]$, in order to allow inflating and shifting $[\mathbf{q}^{k-1}]$ if necessary.[8] As a result, the Hansen-Sengupta acts as a rigorous local search routine allowing the sequence to converge towards the aimed solution set. An inflation factor $\tau$ also has to be applied before the Hansen-Sengupta operator so as to ease the strict inclusion test after each iteration. Hence, the computation of $[\mathbf{q}^{k+1}]$ is as follows:

$$[\tilde{\mathbf{q}}^k] := \text{mid}[\mathbf{q}^k] + \tau([\mathbf{q}^k] - \text{mid}[\mathbf{q}^k]) \quad \text{and} \quad [\mathbf{q}^{k+1}] := [H]([\tilde{\mathbf{q}}^k]). \tag{8}$$

Then the condition $\emptyset \neq [\mathbf{q}^{k+1}] \subseteq \text{int}[\tilde{\mathbf{q}}^k]$ also implies Property (P1) and is likely to succeed as soon as $([\mathbf{x}], [\mathbf{q}])$ is small enough and close enough to some nonsingular solution, which eventually happens thanks to the bisection process. A typical value for the inflation factor is $\tau = 1.01$; It would have to be more accurately tuned for badly conditioned problems, but it is not the case of usual robots.

*Regularity test.* In order to satisfy the regularity constraints, the interval evaluation of each Jacobian $\mathbf{F_x}$ and $\mathbf{F_q}$ over the box $([\mathbf{x}], [\mathbf{q}])$ has to be regular. Testing the regularity of interval matrices is NP-hard, so sufficient conditions are usually used instead. Here, we use the strong regularity of a square interval matrix $[A]$, which consists in checking that $C[A]$ is strongly diagonally dominant, where $C$ is usually chosen as an approximate inverse of the midpoint of $[A]$ (see [21]).

### 4.3 Connected Component Computation

In order to distinguish boxes in $\mathcal{S}$ belonging to one specific aspect from the rest of the paving, we use transitively the relation between linked boxes defined by Property (P3), i.e., we have to compute connected components with respect to the links in $\mathcal{N}$. This is done in three steps:

Step 1. Compute *neighborhood relations* between boxes, i.e., determine when two boxes share at least one common point;
Step 2. Certify aspect connectivity in neighbor boxes, i.e., check Property (P3);
Step 3. Compute connected components with respect to the certified links.

[7] In [8], only problems where parameters have one solution were tackled, hence allowing successfully using the parametric existence test (7).

[8] This was already used in [9] in a completely different context related to sensitivity analysis, and in a recently submitted work of the authors dedicated to the projection of a manifold.

**Step 1. Computing Neighborhood Relations** Two boxes $([\mathbf{x}], [\mathbf{q}])$ and $([\mathbf{x}'], [\mathbf{q}'])$ are *neighbors* if and only if they share at least one common point, i.e., $([\mathbf{x}], [\mathbf{q}]) \cap ([\mathbf{x}'], [\mathbf{q}']) \neq \emptyset$. The neighborhood relations between boxes $\mathcal{N}$ are obtained during the branch and prune computation: After the current box has been pruned (line 4 of Alg. 1), its neighbors are updated accordingly (it may have lost some neighbors); also, the boxes produced when splitting the current box (line 9 of Alg. 1) inherit from (some of) the neighbors of the current box, and are neighbors to one another. One delicate point in managing neighborhood comes from the fact that pose or command parameters are often angles whose domains are restricted to a single period, e.g., $[-\pi, \pi]$; the periodicity of these parameters has to be taken into account: Boxes are neighbors when they share a common point modulo $2\pi$ on their periodic dimensions.

**Step 2. Certifying Connectivity Between Neighbors** Once the branch and prune algorithm has produced the paving $\mathcal{S}$ and its neighboring information $\mathcal{N}$, a post-process is applied to filter from $\mathcal{N}$ the links that do not guarantee the two neighbor boxes cover the same aspect: it may happen that two neighbor boxes share no common point satisfying the kinematic relation $\mathbf{f} = 0$, e.g., if they each cover a portion of two disjoint, but close, aspects. Asserting neighborhood Property (P3) requires again a certification procedure: For any neighbor boxes $\big(([\mathbf{x}], [\mathbf{q}]), ([\mathbf{x}'], [\mathbf{q}'])\big) \in \mathcal{N}$, we verify

$$\exists \mathbf{q} \in ([\mathbf{q}] \cap [\mathbf{q}']), \mathbf{f}(\mathrm{mid}([\mathbf{x}] \cap [\mathbf{x}']), \mathbf{q}) = 0. \tag{9}$$

Indeed, for connectivity to be certified, it is sufficient to prove that the intersection of neighbor boxes share at least one point from the same aspect. Because neighbor boxes $([\mathbf{x}], [\mathbf{q}])$ and $([\mathbf{x}'], [\mathbf{q}'])$ are in $\mathcal{S}$, they satisfy Property (P1) and Property (P2), i.e., $\forall \mathbf{x} \in ([\mathbf{x}] \cap [\mathbf{x}']), \exists \text{ a unique } \mathbf{q} \in [\mathbf{q}], \mathbf{f}(\mathbf{x}, \mathbf{q}) = 0$ and $\exists \text{ a unique } \mathbf{q}' \in [\mathbf{q}'], \mathbf{f}(\mathbf{x}, \mathbf{q}') = 0$. We need to check these unique values $\mathbf{q}$ and $\mathbf{q}'$ are actually the same for one value $\mathbf{x}$ inside $[\mathbf{x}] \cap [\mathbf{x}']$, e.g., $\mathbf{x} = \mathrm{mid}([\mathbf{x}] \cap [\mathbf{x}'])$. Using the certification procedure described in Section 4.2 allows proving Equation (9). Each link is certified this way. If the certification fails for a given link, it is removed from $\mathcal{N}$.

**Step 3. Computing Connected Components** Given the set $\mathcal{N}$ of certified connections between certified boxes in $\mathcal{S}$, a standard connected component computation algorithm (e.g., [13]) is applied in order to obtain a partition of $\mathcal{S}$ into $\mathcal{S}_k$, each $\mathcal{S}_k$ covering a single aspect of the considered robot.

## 5 Experiments

We present experiments on four planar robots with respectively 2 and 3 degrees of freedom, yielding respectively a 2-/3-manifold in a 4/6 dimensional product space.

**Robot Models** Robot RPRPR (resp. RRRRR) has two arms, each connecting an anchor point ($A$, $B$) to its end-effector ($P$), each composed of a revolute joint, a prismatic (resp. revolute) joint and again a revolute joint in sequence. The end-effector $P$ lies at

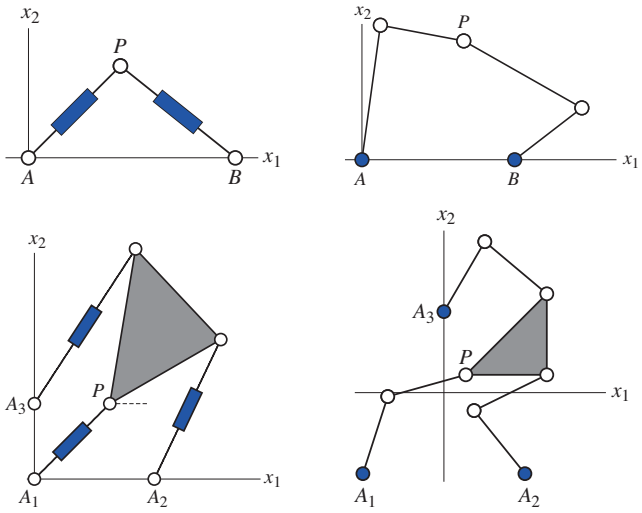



**Fig. 3.** R<u>P</u>R<u>P</u>R (top-left), <u>R</u>RRR<u>R</u> (top-right), 3-R<u>P</u>R (bottom-left) and 3-<u>R</u>RR (bottom-right).

the shared extremal revolute joint and is described as a 2D point $(x_1, x_2) \in [-20, 20]^2$. The prismatic (resp. initial revolute) joint in each arm is actuated, allowing to vary the arms lengths (resp. angles). The arm lengths (resp. angles) are considered to be the command $(q_1, q_2) \in [2, 6] \times [4, 9]$ (resp. $[-\pi, \pi]^2$) of the robot. Using the architecture parameters defined in [3] (resp. [6]), their kinematic equations are:

$$\begin{aligned} x_1^2 + x_2^2 - q_1^2 &= 0, \\ (x_1 - 9)^2 + x_2^2 - q_2^2 &= 0, \end{aligned}$$

and, respectively

$$\begin{aligned} (x_1 - 8\cos q_1)^2 + (x_2 - 8\sin q_1)^2 - 25 &= 0, \\ (x_1 - 9 - 5\cos q_2)^2 + (x_2 - 5\sin q_2)^2 - 64 &= 0. \end{aligned}$$

Robot 3-R<u>P</u>R (resp. 3-<u>R</u>RR, with the restriction that it has only a fixed orientation, i.e., $x_3 = 0$, its free orientation variant being too complex for the current implementation of the method) has three arms, each connecting an anchor point $(A_1, A_2, A_3)$ to its end-effector $(P)$, each composed of a revolute joint, a prismatic (resp. revolute) joint and again a revolute joint in sequence. The end-effector is a triangular platform whose vertices are attached to the extremal revolute joints of the arms. The position parameters $(x_1, x_2, x_3)$ represent the coordinates $(x_1, x_2) \in [-50, 50]^2$ of one vertex of the platform, and the angle $x_3 \in [-\pi, \pi]$ between its basis and the horizontal axis. The prismatic (resp. initial revolute) joint in each arm is actuated, allowing to vary the arm lengths (resp. angles). The arm lengths (resp. angles) are considered to be the command $(q_1, q_2, q_3) \in [10, 32]^3$ (resp $[-\pi, \pi]^3$) of the robot. Using the architecture parameters defined in [7] (resp. [3]), their kinematic equations are:

$$\begin{aligned} x_1^2 + x_2^2 - q_1^2 &= 0, \\ (x_1 + 17\cos x_3 - 15.9)^2 + (x_2 + 17\sin x_3)^2 - q_2^2 &= 0, \\ (x_1 + 20.8\cos(x_3 + 0.8822))^2 + (x_2 + 20.8\sin(x_3 + 0.8822) - 10)^2 - q_3^2 &= 0, \end{aligned}$$

and, respectively

$$\begin{aligned}(x_1 - 10 - 10\cos q_1)^2 + (x_2 - 10 - 10\sin q_1)^2 - 100 = 0,\\ (x_1 + 10\cos x_3 - 10 - 10\cos q_2)^2 +\\ (x_2 + 10\sin x_3 - 10 - 10\sin q_2)^2 - 100 = 0,\\ (x_1 + 10\sqrt{2}\cos(x_3 + \pi/4) - 10\cos q_3)^2 +\\ (x_2 + 10\sqrt{2}\sin(x_3 + \pi/4) - 10 - 10\sin q_3)^2 - 100 = 0.\end{aligned}$$

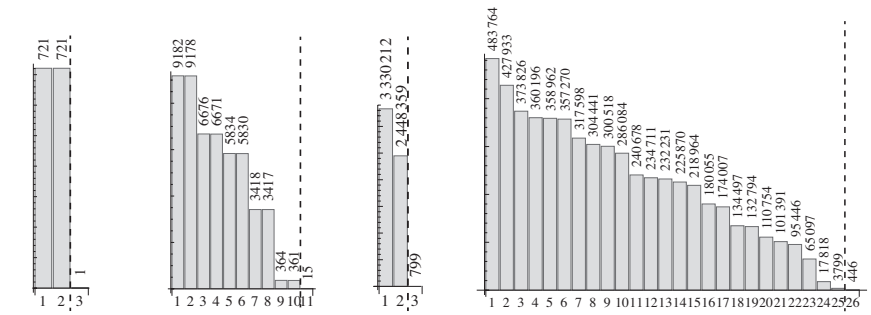
**Implementation** We have implemented the proposed method described in Section 4 using the Realpaver library [12] in C++, specializing the classes for the different routines in the branch and prune algorithm. Given an NCSP that models a robot and a prescribed precision $\epsilon$, the implementation outputs certified boxes grouped by certified connected components as explained in Section 4. Hence we can count not only the number of output boxes but also the number of output certified connected components. The experiments were run using a 3.4GHz Intel Xeon processor with 16GB of RAM.

**Results of the Method** Table 1 provides some figures on our computations. Its columns represent the different robots we consider. Line "# aspects" provides the theoretically established number of aspects of each robot provided in [3, 6, 7]; This value is unknown for the 3-RRR robot. Line "precision" gives the prescribed precision $\epsilon$ used in the computation. Lines "# boxes" and "# CC" give respectively the number of boxes and the number of connected components returned by our method. Line "time" gives the overall computational time in seconds of the method, including the post-processes.

**Table 1.** Experimental results.

| | PRRP | RPRPR | RRRRR | 3-RPR | 3-RRR |
|---|---|---|---|---|---|
| # aspects | 4 | 2 | 10 | 2 | unknown |
| precision | 0.1 | 0.1 | 0.1 | 0.3 | 0.008 |
| # boxes | 38 | 2 176 | 69 612 | 13 564 854 | 11 870 068 |
| # CC | 4 | 4 | 1 767 | 44 220 | 56 269 |
| # $CC_{filtered}$ | 4 | 2 | 10 | 2 | 25 |
| time (in sec.) | 0.003 | 0.36 | 38 | 12 700 | 10 700 |

Note that despite the quite coarse precisions we have used, the number of output boxes can be very large, due to the dimension of the search space we are paving. The number of connected components is much smaller, but still it is not of the same order as the theoretically known number of aspects, implying numerous disjoint connected components does in fact belong to the same aspect. This is explained by the numerical instability of the kinematic equations of the robots in the vicinity of the aspect boundaries, which are singularities of the robot. Indeed, in these regions, the numerical certification process cannot operate homogeneously, resulting in disconnected subsets of certified boxes, separated either by non-certified boxes or by non-certified links.




(a) RPRPR (b) RRRRR (c) 3-RPR (d) 3-RRR

**Fig. 4.** Number of boxes in each connected component. Each bar corresponding to a connected component shows the number of contained boxes (ordered largest first).

Under this assumption, each aspect should consist of, on the one hand, one large connected component comprising wide boxes covering the regular inner part of the aspect, and smaller and smaller boxes close to its boundary; And, on the other hand, numerous small connected components comprising tiny boxes gathered on its boundary. As a result, the number of boxes in the "boundary" components should be many times smaller than that of the regular component in an aspect. It should thus be possible to filter out these spurious tiny "boundary" components, based on the number of boxes they contain, as they have no practical use in robotics.

In order to distinguish the relevant components from the spurious ones, we use such a filtering post-process on the output of our method: Output connected components are ordered by decreasing number of constituting boxes; The largest ratio, in number of constituting boxes, between two consecutive components in this order is computed, and used as a separation between relevant and spurious components. Applying this heuristic post-process, the number of obtained connected components, reported at Line "# $\mathrm{CC}_{filtered}$" in Table 1, reaches the theoretically known number of aspects in the case of the robots we considered. Figure 4 illustrates the number of boxes of the connected components retained after filtering (the dashed lines represent the computed heuristic thresholds). This seems to indicate that our assumption is correct for the considered robots, i.e., that the major part of each aspect is indeed covered with a single large, regular, connected component.

The retained connected components projected onto the $\mathbf{x}$ subspace are depicted in Figures 5 and 6.[9] They graphically correspond to the aspects of the robots for which they are theoretically known (e.g., see [3, 4, 6, 7]). Note that the red boxes, that enclose the singularity curves, seem to cross the aspects due to the projection, while they of course do not cross in the product space.

Because the computation requires an exponentially growing time and space with respect to the prescribed precision $\epsilon$, we need to tweak it for an efficient and reliable aspect determination. For the first three robots, the precision $\epsilon = 0.1$ gave precise enough

[9] These figures are also available at http://www.dsksh.com/aspects/.

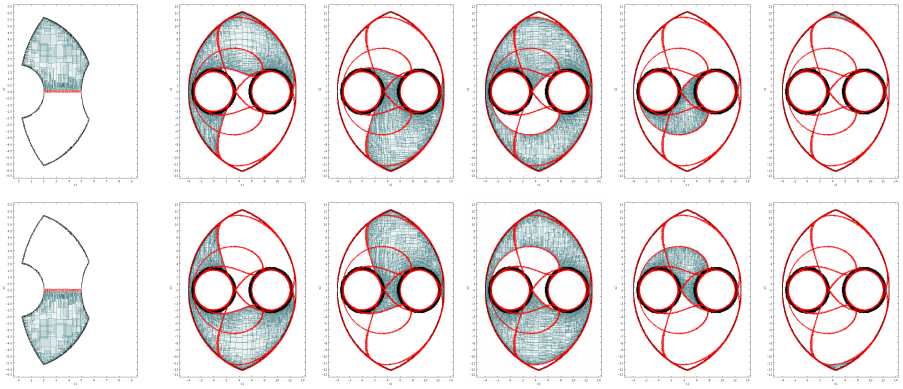

**Fig. 5.** Computed 3D workspace of of 3-RPR (after filtering). First figure shows the undecided boxes that cover the surface of the workspace. Second and third figures show the certified connected components corresponding to the two aspects.

results to determine the correct number of aspects after filtering out the spurious components. For 3-RPR, we needed the coarser precision $\epsilon = 0.3$ to compute without causing over-consumption of the memory. In the computation of 3-RRR, the threshold between the regular and the spurious components was not as clear as for the other robots, even though we improved the precision up to $\epsilon = 0.008$. Enumerating the obtained components from the largest ones, we assume that this robot (with fixed-orientation) has 25 aspects, the following components being likely to be a part of another component, i.e., spurious. This remains to be formally demonstrated.

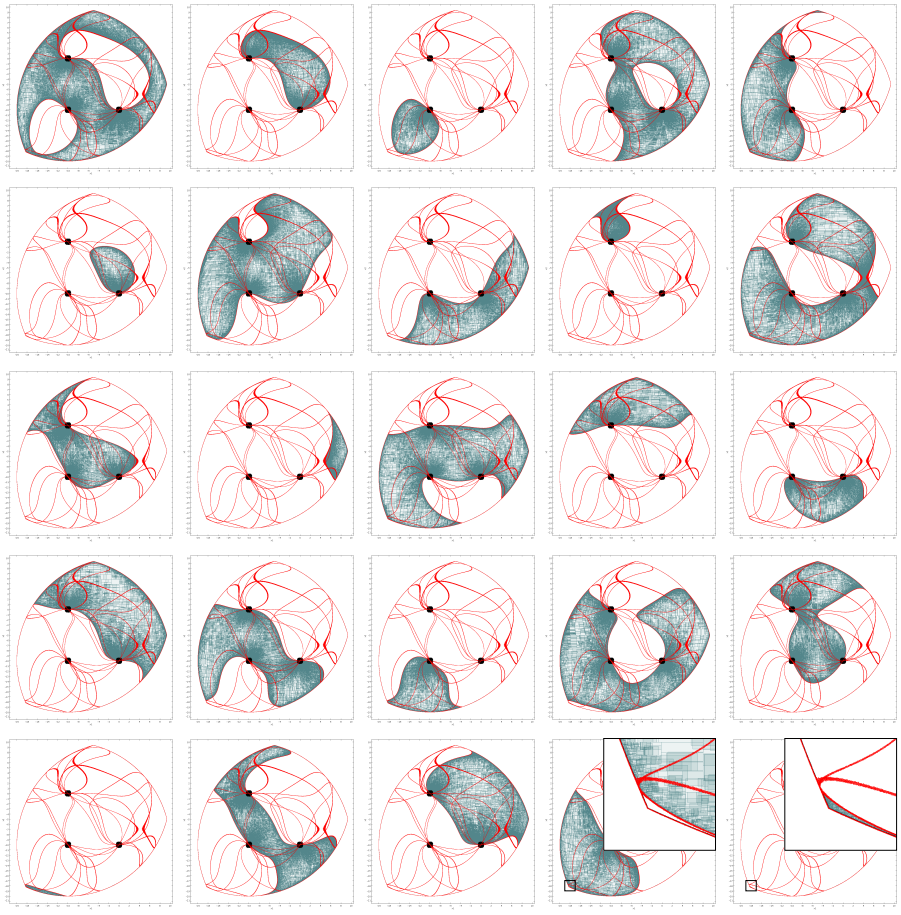
## 6 Conclusion

The computation of aspects, i.e., singularity free connected sets of configurations, is a critical task in the design and analysis of parallel robots. The proposed algorithm uses numerical constraint programming to fully certify this computation. It is worth noting that this is the first algorithm that automatically handles such a large class of kinematic models with fully certifying the configuration existence, non-singularity and connectivity: The only restriction of the algorithm is its computational complexity, which is obviously exponential with respect to the number of degrees of freedom of the robot. The presented experiments have reported the sharp approximations of aspects for some realistic models: The correct number of aspects was computed for well-known planar robots with two and three degrees of freedom. The more challenging 3-RRR, whose number of aspects is still an open question, remains out of reach because of the complexity of the computation, though we have obtained some promising results fixing the orientation of its moving-platform. Tuning the propagation and search strategies of the algorithms should allow fully analyzing it in the future. Finally, although experiments have shown that the proposed method computes approximations of all aspects of well-known robots, it cannot be used for actually rigorously counting the aspects, a challenge we will address in the future on the basis of this method.

*Acknowledgments* This work was partially funded by the French agency ANR project SIROPA (PSIROB06_174445) and JSPS (KAKENHI 23-3810). The machine used for the experiments was supported by Prof. Kazunori Ueda.



(a) RPRPR (b) RRRRR



(c) 3-RRR

**Fig. 6.** Projections into the 2D workspace of the computed aspects (after filtering). Green boxes are certified; red and black boxes are undecided (i.e., do not satisfy Properties (P1) and (P2), respectively).

## References


1. Amine, S., Tale-Masouleh, M., Caro, S., Wenger, P., Gosselin, C.: Singularity conditions of 3T1R parallel manipulators with identical limb structures. ASME Journal of Mechanisms and Robotics 4(1), 011011–1011011–11 (2012)
2. Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.F.: Revising hull and box consistency. In: Proc. of International Conference on Logic Programming. pp. 230–244 (1999)
3. Chablat, D.: Domaines d'unicité et parcourabilité pour les manipulateurs pleinement parallèles. Ph.D. thesis, École Centrale de Nantes (1998)
4. Chablat, D.: Joint space and workspace analysis of a two-dof closed-chain manipulator. In: Proc. of ROMANSY 18 Robot Design, Dynamics and Control. pp. 81–90. Springer (2010)
5. Chablat, D., Moroz, G., Wenger, P.: Uniqueness domains and non singular assembly mode changing trajectories. In: International Conference on Robotics and Automation. pp. 3946–3951 (2011)
6. Chablat, D., Wenger, P.: Working Modes and Aspects in Fully Parallel Manipulators. In: International Conference on Robotics and Automation. vol. 3, pp. 1964–1969 (1998)
7. Coste, M.: A simple proof that generic 3-RPR manipulators have two aspects. Tech. rep., Institut de Recherche Mathématique de Rennes (IRMAR) (2010)
8. Goldsztejn, A.: A Branch and Prune Algorithm for the Approximation of Non-Linear AE-Solution Sets. In: Proc. of ACM SAC 2006. pp. 1650–1654 (2006)
9. Goldsztejn, A.: Sensitivity analysis using a fixed point interval iteration. Tech. Rep. hal-00339377, CNRS-HAL (2008)
10. Goldsztejn, A., F.Goualard: Box Consistency through Adaptive Shaving. In: Proc. of ACM SAC 2010. pp. 2049–2054 (2010)
11. Goldsztejn, A., Jaulin, L.: Inner approximation of the range of vector-valued functions. Reliable Computing 14, 1–23 (2010)
12. Granvilliers, L., Benhamou, F.: Algorithm 852: Realpaver: an interval solver using constraint satisfaction techniques. ACM TOMS 32(1), 138–156 (2006)
13. Hopcroft, J., Tarjan, R.: Algorithm 447: efficient algorithms for graph manipulation. Commun. ACM 16(6), 372–378 (1973)
14. Kong, X., Gosselin, C.: Type Synthesis of Parallel Mechanisms. Springer (2007)
15. Lhomme, O.: Consistency Techniques for Numeric CSPs. In: Proc. of IJCAI 1993. pp. 232–238 (1993)
16. Merlet, J.P.: Parallel robots. Kluwer, Dordrecht (2000)
17. Merlet, J.P.: A formal-numerical approach for robust in-workspace singularity detection. IEEE Trans. on Robotics 23(3), 393–402 (2007)
18. Montanari, U.: Networks of constraints: Fundamentals properties and applications to picture processing. Information Science 7(2), 95–132 (1974)
19. Moore, R.: Interval Analysis. Prentice-Hall (1966)
20. Moroz, G., Rouiller, F., Chablat, D., Wenger, P.: On the determination of cusp points of 3-RPR parallel manipulators. Mechanism and Machine Theory 45(11), 1555 – 1567 (2010)
21. Neumaier, A.: Interval Methods for Systems of Equations. Cambridge University Press (1990)
22. Van Hentenryck, P., Mcallester, D., Kapur, D.: Solving polynomial systems using a branch and prune approach. SIAM Journal on Numerical Analysis 34, 797–827 (1997)