A Generic Interval Branch and Bound Algorithm for Parameter Estimation
Bertrand Neveu, Martin de La Gorce, Pascal Monasse, Gilles Trombettoni

To cite this version:

HAL Id: hal-01942306
https://hal-enpc.archives-ouvertes.fr/hal-01942306
Submitted on 3 Dec 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
A Generic Interval Branch and Bound Algorithm for Parameter Estimation

Bertrand Neveu · Martin de la Gorce · Pascal Monasse · Gilles Trombettoni

Received: date / Accepted: date

Abstract In engineering sciences, parameter estimation is a challenging problem consisting in computing the parameters of a parametric model that fit observed data. The system is defined by unknown parameters and sometimes internal constraints. The observed data provide constraints on the parameters. This problem is particularly difficult when some observation constraints correspond to outliers and/or the constraints are non convex. The RANSAC randomized algorithm can efficiently handle it, but is non deterministic and must be specialized for every problem. This paper presents the first generic interval Branch and Bound algorithm that produces a model maximizing the number of observation constraints satisfied within a given tolerance. This tool is inspired by the IbexOpt Branch and Bound algorithm for constrained global optimization (NLP) and is endowed with an improved version of a relaxed intersection operator applied to observations. Experiments have been carried out on two different computer vision problems. They highlight a significant speedup w.r.t. Jaulin et al.’s interval method in 2D and 3D shape recognition problems (having 3 parameters). We have also obtained promising results on a stereo vision problem where the essential matrix (5 parameters) is estimated exactly at a good accuracy in hours for models having a thousand points, a typical size for such problems.

B. Neveu
LIGM, École des Ponts ParisTech, Université Paris–Est, France
E-mail: Bertrand.Neveu@enpc.fr

M. de la Gorce
LIGM, École des Ponts ParisTech, Université Paris–Est, France
E-mail: martin.delagorce@gmail.com

P. Monasse
LIGM École des Ponts ParisTech, Université Paris–Est, France
E-mail: Pascal.Monasse@enpc.fr

G. Trombettoni
LIRMM, University of Montpellier, CNRS, France
E-mail: Gilles.Trombettoni@lirmm.fr
1 Introduction

Parameter estimation is a difficult problem widely studied by engineering sciences. It consists in determining the \( n \) numerical parameters of a model based on \( m \) observations. Calibration or geolocation can be viewed as specific parameter estimation problems.

A parameterized model is defined by an implicit equation \( f(x, p) = 0, \ p = (p_1, \ldots, p_n) \) being the \( n \)-vector of parameters to be determined. An observation \( o_i \) is a \( d \)-dimensional vector of observed data (values) for \( x \). Given a finite set of observations \( \{o_1, \ldots, o_i, \ldots, o_m\} \), we search for a parameter vector that fits the observations, i.e. that satisfies the observation constraints \( f(o_i, p) = 0 \).

Because the model is not perfect (i.e., \( f \) is an approximation of the real model) or due to uncertainties on the observations \( o_i \), there exists generally no model fitting all the observation constraints exactly. If \( f \) is linear, it is possible to compute analytically the parameters minimizing \( \sum_i f(o_i, p)^2 \). This least square method, or other minimization criteria, can also be applied to parameter estimation in which \( f \) is non linear. In this case, numerical local methods are generally used to estimate the parameters, such as the Levenberg-Marquardt algorithm, a quasi-Newton method of choice in computer vision.

The least square method is relevant when the errors on the observations are small, i.e. when the errors are bounded or when the actual observation values follow a probability law. Unfortunately, the answer of this least square method is poor in presence of outliers. Outliers can have numerous origins, including extreme values of the noise, erroneous measurements and data reporting errors.

In this paper, we consider a different approach where the observations are partitioned into two groups:

- An observation \( o_i \) is an inlier (i.e., compatible with the parameter vector \( p \)) when it satisfies an observation constraint, given by the implicit equation above but within a tolerance value \( \tau \):

\[
-\tau \leq f(o_i, p) \leq +\tau.
\]

- An observation \( o_i \) is an outlier if it does not satisfy the corresponding observation constraint.

Thus, the problem addressed in this paper is to compute a model fitting a maximum number of observations. In a variant, we search for parameter vectors that fit at least \( q \) of these observations (with \( n \leq q \leq m \), where \( n \) is the number of parameters to estimate).

More formally, the consensus set \( Consensus(p) \) is the set of observations compatible with \( p \):

\[
Consensus(p) = \{ o_i | -\tau \leq f(o_i, p) \leq +\tau \}. \tag{1}
\]

The parameter estimation problem can be formulated as a numerical constraint satisfaction problem (resolution problem) with \( n \) variables \( p = (p_1, \ldots, p_n) \).
having a real interval domain, and a constraint stating that at least \( q \) observations are compatible with the model:

\[
\text{card}(\text{Consensus}(p)) \geq q. \quad (2)
\]

The optimization version of this problem simply consists in maximizing the cardinality of the consensus set, i.e.

\[
\arg \max_p (\text{card}(\text{Consensus}(p))). \quad (3)
\]

More generally, a parameterized model can be defined by the observation constraints and internal constraints between parameters \( C(p) \), so that the estimation parameter problem handled by the interval Branch and Bound proposed in this paper will be defined as follows:

\[
\arg \max_p (\text{card}(\text{Consensus}(p))) \quad \text{s.t.} \quad C(p). \quad (4)
\]

The random sample consensus algorithm (RANSAC) [12] has become a state-of-the-art tool to achieve a parameter estimation robust to outliers. This stochastic algorithm proceeds by randomly sampling observations for determining a model (\( n \) observation constraints for determining \( n \) parameters), before checking the number of other observations compatible with this model. However, RANSAC cannot guarantee that the computed model fits a maximum number of observations, and the sampling phase is dedicated to the parameter estimation problem studied. For instance, in the stereo vision problem tested in our experiments, computing a first model which fits 5 observations requires a symbolic computation known as the “five points method” [28].

In the landscape of exact approaches, only a few works have been proposed. In addition, the existing deterministic algorithms are ad-hoc and are strongly dependent on the application domain.

This paper proposes a first generic Branch and Bound algorithm to parameter estimation for which the main user inputs are the observation constraint expression \( f \) and the associated tolerance \( \tau \) (see (1)).

This approach extends and improves Jaulin et al.’s work based on interval constraint programming [17,16]. This is a combinatorial algorithm that explores the parameter space using an interval method and uses a relaxed intersection operator between boxes to better contract the parameter space in presence of outliers without losing solutions.

After a background on RANSAC, interval methods and Jaulin et al.’s approach in Section 2, we describe in Section 3 an interval branch and bound algorithm that can compute the model satisfying a maximum number of observation constraints. The algorithmic improvements can bring significant gains in performance compared to the basic approach, as shown in experiments on shape detection and stereo vision problems (see Section 4).
2 Background

This section includes the principles of the RANSAC algorithm and the deterministic interval approach proposed by Jaulin et al. We also present an interval Branch and Bound (B & B) for constrained global optimization from which our generic interval parameter estimator is inspired.

2.1 The RANSAC algorithm

The RANdom SAmple Consensus algorithm (RANSAC) [12] has become a state-of-the-art tool to achieve a parameter estimation robust to outliers. In addition to the observed dataset, the input of RANSAC is the tolerance $\tau$ in the observation constraints (see (1)) and, in the resolution version of parameter estimation, a number $q$ of inliers. RANSAC is a stochastic and non exact method that searches for the best parameter vector by repeating the following steps. $q$ is a fixed threshold in the resolution version. In the optimization version maximizing the consensus cardinal, $q$ is initialized to $n$ (see below) and takes the maximum cardinal of the consensus set found during the process.

1. Random sampling: A subset of the observations is randomly selected. The cardinality of the sample is the smallest sufficient to determine the model parameters: $n$ observation constraints are generally required to determine the $n$ parameters.
2. A parameter vector $p$ corresponding to a model that fits the selected sample is computed by solving the selected system of $n$ observation constraints.
3. Consensus: One checks if at least $q$ elements of the entire observation dataset are inliers given the tolerance parameter $\tau$.
4. If a consensus is obtained, a second and better model may be computed by using the whole consensus set. $q$ is updated accordingly in the optimization version.

In some problems, e.g. shape detection, the goal is to compute all the valid models fitting at least $q$ observations. In this case, a RANSAC version like the one presented in [30] finds the different models in a greedy way: when a model has been found, it removes the observation constraints involved in the consensus set before searching for a next model. There is of course no guarantee that this approach can find all the valid models.

2.2 Complete interval constraint programming approach by Jaulin et al.

A parameter estimation method based on interval constraint programming that is robust to outliers was first described in [17]. Intervals are the first ingredient of the approach.
2.2.1 Interval arithmetic and contractors

We denote by \([x_i] = [x_{i1}, x_{i2}]\) the interval/domain of the real-valued variable \(x_i\), where \(x_{i1}, x_{i2}\) are floating-point numbers. A Cartesian product of intervals like the domain \([x] = [x_1] \times \ldots \times [x_N]\) is called a (parallel-to-axes) box. The width of an interval \([x_i]\) denotes the size or width \(\overline{x_i} - \underline{x_i}\) of this interval. The width of a box is given by the width \(\overline{x}\). The perimeter of a box, given by \(\sum_i \overline{x_i} - \underline{x_i}\), is another size measurement used in this paper.

Interval methods also provide contracting operators (called contractors), i.e. methods that can reduce the variable domains involved in a constraint or a set of constraints without loss of solutions. Let us consider one observation \(o_i\), and a subspace of the parameter space given by a box \([p]\). Given an observation \(o_i\), we denote \(V_i\) the set of parameter vectors that are compatible with that observation, i.e.

\[
V_i = \{p \mid -\tau \leq f(o_i, p) \leq +\tau\}.
\]  

The set of parameter values within that box that are compatible with \(o_i\) is given by \(V_i \cap [p]\). A contractor is able to reduce the input box \([p]\) while keeping all the points in \(V_i \cap [p]\), i.e. without losing any solution. The main contractor used in this paper is the well-known HC4-revise \([3, 21]\), also called forward-backward. This contractor handles a single constraint and obtains a (generally non optimal \([11]\)) contracted box including all the solutions of that constraint. Let us give the principles of HC4-revise on the constraint \(f(o_i, p) \leq +\tau\).

The forward phase traverses the expression tree of the constraint bottom up using interval arithmetic. In the example, the unknowns \(p\) are replaced by their intervals \([p]\) in the expression and interval arithmetic evaluates \([f](o, [p])**.  

For instance, if \(f(o_1, p) = o_1 + p_1 + p_2\) with \(o_1 = 2.5\) and \([p] \in [-2, 3] \times [0, 1]\), we have \([f](o_1, [p]) = [2.5, 2.5] + [-2, 3] + [0, 1] = [0.5, 6.5]\). The image interval is then intersected with \([−\infty, \tau]\) because the constraint is an inequality. If the tolerance \(\tau\) is \(1\times 10^{-1}\), then the intersection is empty. The contractor terminates with an empty contracted box, which proves that no vector inside the initial domain \([p]\) satisfies the constraint. If the resulting interval \([r]\) is not empty, e.g. the tolerance is 5, then the computed interval is \([r] = [0.5, 5]\) and one can run the second phase.

In the backward phase, the expression tree is traversed top-down, and interval arithmetic is applied on so-called inverse operators. Without detailing, this phase amounts to evaluating all the (inverse) functions isolating every variable occurrence. Consider for instance the inverse function \(f(p)\) used to contract \(p_1: f(p_1, o_1, p_2) = r - o_1 - p_2\). We evaluate \(f(p)\) using interval arithmetic, which produces the following contraction: \([p_1] := [p_1] \cap [f(p)]([o_1], [p_2]) = [-2, 3] \cap ([0.5, 5] - [2.5, 2.5] - [0, 1]) = [-2, 3] \cap [-3, 2.5] = [-2, 2.5]\).

To contract a system of constraints, the HC4 algorithm performs a propagation loop applying iteratively the HC4-Revise procedure introduced above on each constraint individually until a quasi fixpoint is obtained in terms of contraction.
2.2.2 Jaulin et al.’s combinatorial interval method for parameter estimation

Jaulin et al. proposed in [18] a simple deterministic algorithm based on interval methods to handle inverse problems:

- A search tree is built to exhaustively explore the parameter space. \( \mathbf{p} \) is recursively subdivided: one variable \( p_i \) in \( \mathbf{p} \) is selected, its domain \([p_i]\) is bisected into two sub-intervals and the two corresponding sub-boxes are explored recursively. The combinatorial process stops when a precision is reached, i.e. when the width of the current box is inferior to \( \epsilon_{sol} \) (the box is thus a leaf of the search tree).

- At each node of the tree, the current box is contracted w.r.t. the observation constraints, and sometimes eliminated, using the forward-backward contractor described above.

Compared to ransac, the main advantage of this approach is that all the valid model instances can be produced in an exhaustive way. In addition, bounded errors can also be taken into account in the observed data: a measurement with a bounded error can be modelled by a (constant) box \([\mathbf{o}]\) and not a vector. The main drawback is that outliers lead to an empty contracted box (i.e., no model instance exists in the box) since the parameter box contracted using an outlier observation constraint does not intersect the other ones.

The second ingredient used in the parameter estimation tool based on interval constraint programming was also proposed by Jaulin et al. to deal with outliers (see [16]). Like ransac, the approach assumes that at least \( q \) observations are inliers, the other ones being viewed as potential outliers.

This idea is implemented by the \( q \)-intersection operator. This operator relaxes the previous intersection of \( m \) boxes (corresponding in our inverse problem to the contraction of \([\mathbf{p}]\) w.r.t. all the \( m \) observation constraints) by the union of all the intersections obtained with \( q \) boxes. More formally:

Definition 1 Let \( S \) be a set of boxes. The \( q \)-intersection of \( S \), denoted by \( \cap^q S \), is the box of smallest perimeter that encloses the set of points of \( \mathbb{R}^n \) belonging to at least \( q \) boxes.

For instance, the box in dotted lines in Fig. 1–b is the 4-intersection of the \( m = 10 \) two-dimensional boxes (in plain lines).

The parameter estimation tool proposed by Jaulin et al. has been extended to cope with outliers using a \( q \)-intersection operator. The contraction phase is replaced by a more sophisticated routine called \texttt{contractAndQinter} in this paper:

1. A forward-backward contraction is achieved on each of the \( m \) observation constraints, which produces a set \( S \) of boxes. Note that no (standard) intersection is achieved on these boxes, i.e. no propagation loop is achieved to contract all the constraints simultaneously.

2. The box returned by \texttt{contractAndQinter} is the \( q \)-intersection of these contracted boxes: \( \cap^q S \).
The $q$-intersection of boxes is a difficult problem and has been proven DP-complete in [6] where an exact algorithm based on the search of $q$-cliques has been proposed. In this paper, we simply resort to a non optimal $q$-intersection operator often used in parameter estimation. Computing a reasonable but non optimal enclosure of $\cap^q S$ may be satisfactory, provided it can be done in polynomial time, since the $q$-intersection operator is typically used to filter the search space at each node of a search tree. It appeared that the exact $q$-intersection algorithm was too costly for the problems studied in this paper.

This algorithm, called here $q$-proj, solves the problem on each dimension independently. For each dimension $i$, the algorithm computes the projection $S[i]$ of the boxes and solves the $q$-intersection problem on $S[i]$. Since $S[i]$ is a set of intervals, each $\cap^q S[i]$ can be computed in polynomial time by sorting the lower and upper bounds using a method first introduced in [8]. The overall complexity of $q$-proj is $O(nm \log(m))$. Figure 2 illustrates the algorithm.
Jaulin et al.’s interval-based approach has been used in several parameter estimation applications, including geolocation. More recently, a 3D reconstruction problem has been handled by estimating the coefficients of the transformation matrix between two poses [4].

2.3 IbexOpt: a global optimization code based on interval techniques

The parameter estimation code proposed in this paper (see Section 3) provides an optimization tool finding a parameter vector that maximizes the number of observation constraints satisfied. Its overall architecture is inspired by the IbexOpt global optimization code available as a plugin of the open source C++ Interval Based EXplorer IBEX library [7]. Let us provide a brief overview of IbexOpt. Details can be found in [32,26]. IbexOpt handles continuous global optimization under inequality constraints defined by:

\[
\min_{x \in [x]} f(x) \quad s.t. \quad g(x) \leq 0
\]

where \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) is the real-valued objective (non convex) function and \( g : \mathbb{R}^n \rightarrow \mathbb{R}^m \) is a vector-valued (non convex) function. \( x = (x_1, ..., x_i, ..., x_n) \) is a vector of variables varying in a domain (i.e., a box) \([x]\). A point \( x \) is said to be feasible if it satisfies the constraints.

The algorithm is launched with the vector of constraints \( g \), the objective function \( f \) and with the input domain initializing a list \( \text{Boxes} \) of boxes to be handled. \( \epsilon_{\text{obj}} \) is the absolute or relative precision required on the objective function and is used in the stopping criterion. Therefore, the algorithm computes a feasible point of cost \( \tilde{f} \) such that no other solution exists with a cost lower than \( \tilde{f} - \epsilon_{\text{obj}} \). IbexOpt adds a variable \( x_{\text{obj}} \) in the problem (to the vector \( x \) of variables) corresponding to the objective function value, and a constraint \( f(x) = x_{\text{obj}} \).

IbexOpt is a B&B algorithm that maintains two main types of information during the iterations:

- \( \tilde{f} \): the value of the best feasible point \( x_{\tilde{f}} \) found so far, and
- \( f_{\min} \): the minimal value of the lower bounds \( x_{\text{obj}} \) of the nodes \([x]\) to explore, i.e. the minimal \( x_{\text{obj}} \) in the list of open nodes and in the list of the discarded small nodes/boxes.

In other terms, in every box \([x]\), there is a guarantee that no feasible point exists with an objective function value lower than \( x_{\text{obj}} \).

Starting with the initial box, the B&B algorithm selects iteratively one node to handle until the list of open nodes becomes empty. The selected box \([x]\) is split into two sub-boxes along one dimension (selected by any branching strategy) that are handled by a Contract&Bound procedure.

The Contract&Bound procedure contracts the handled box without loss of feasible part [7]. In other words, some infeasible parts at the bounds of
the domain are discarded by constraint programming (CP) [33,3,25] and convexification [31,22] algorithms. This contraction works on the extended box including the objective function variable $x_{obj}$ and on the associated constraint, so improving $x_{obj}$ amounts to improving the lower bound of the objective function image (lower bounding).

The procedure also carries out upper bounding. It calls one or several heuristics searching for a feasible point $x_f$ that improves the best cost $\tilde{f}$ found so far. If the upper bound of the objective value is improved, it removes from the set of open nodes those having $x_{obj} > \tilde{f} - \epsilon_{obj}$. Note that each time a new upper bound $\tilde{f}$ is found, a constraint $x_{obj} \leq \tilde{f} - \epsilon_{obj}$ is added to the problem for decreasing the upper bound of the objective function in the box. Imposing $\tilde{f} - \epsilon_{obj}$ as a new upper bound (and not only $\tilde{f}$) aims at finding a solution significantly better than the current best feasible point.

Finally, Contract&Bound pushes the two sub-boxes into the set Boxes of open nodes. However, if the size of a box reaches a given precision, the box will be no more studied (i.e., bisected) and the lower bound of the objective function $f_{min}$ is updated with the value of the objective function in this small discarded box.

**Node selection strategy**

Several strategies are available in IbexOpt to select a node in the list of open nodes. In a standard approach, the search tree is traversed in best-first order by selecting at each iteration the node having the lowest $x_{obj}$ value. This strategy may require an exponential memory to store the list of open nodes.

An efficient variant calls a probing procedure named feasible diving at each node of this best-first search strategy to better intensify the search (see [26]). The adaptation of this FeasibleDiving procedure to our B&B method for parameter estimation is detailed in the next section.

### 3 A Parameter Estimation Algorithm Fitting a Maximum Number of Inliers

We have designed an interval B&B algorithm for parameter estimation that computes a model maximizing the consensus, i.e. maximizing the number of valid observations (inliers) of a parameterized model. It appeared that the standard best-first search strategy has difficulties finding a good solution and that a depth first search strategy gets stuck in a local optimum area. Therefore we have used a mixed strategy based on best-first search where, at each node, a feasible diving procedure [26] is performed.

The EstimB&B algorithm presented in Algorithm 1 implements a Best-First search algorithm using Feasible Diving for computing a parameterized model that maximizes the number of inliers. The algorithm returns the best solution found ($\text{best}$ is the node where the best solution has been found), with a number
\( q_{\min} \) of validated inliers, and an interval \([q_{\min}, q_{\max}]\). Two main cases can occur. Either the precision \( \Delta_{\text{obj}} \) required on the number of inliers is obtained, i.e.
\( q_{\max} - q_{\min} \leq \Delta_{\text{obj}} \), or \( q_{\max} - q_{\min} > \Delta_{\text{obj}} \), which means that “small” boxes of width inferior to \( \epsilon_{\text{sol}} \) remain open (unexplored) with a number of inliers at most equal to \( q_{\epsilon_{\text{sol}}{\max}} \). In other terms, \( q_{\epsilon_{\text{sol}}{\max}} \) is a maximum number of inliers in a small box discarded by the \( \epsilon_{\text{sol}} \) parameter. The parameter \( q_{m} \) is a minimum number of inliers imposed by the user in a solution (\( q_{m} \geq n + 1 \)), \( q_{m} = n + 1 \) being a default value. It is useful for finding a significant first current solution at the beginning of the solving process. The algorithm is also called with the

\[ \text{EstimB&B} \left( p, [p]_{\text{init}}, C, f, o = \{ o_1, \ldots, o_t, \ldots, o_m \}, \tau, \epsilon_{\text{sol}}, \Delta_{\text{obj}}, q_{m} \right) \]

\[ \text{node.box} \leftarrow [p]_{\text{init}} \]
\[ C_{\text{obs}} \leftarrow \{ -\tau \leq f(o_i, p) \leq +\tau \mid o_i \in o \} \] // Observation constraints

\[ \text{node.q}_{\min} \leftarrow q_{m} - \Delta_{\text{obj}} - 1 \]

\[ \text{best} \leftarrow \text{node} \]
\[ q_{\epsilon_{\text{sol}}{\max}} \leftarrow 0 \] /* The highest cost of the nodes having reached the \( \epsilon_{\text{sol}} \) precision. */

\[ (\text{node}, \text{best}, q_{\epsilon_{\text{sol}}{\max}}) \leftarrow \text{Contract&Bound} \left( \text{node}, C, \epsilon_{\text{obj}}, \Delta_{\text{obj}}, \text{best}, q_{\epsilon_{\text{sol}}{\max}} \right) \]
\[ \text{nodeHeap} \leftarrow \{ \text{node} \} \]

\[ \text{while} \ \text{nodeHeap} \neq \emptyset \ \text{do} \]
\[ \text{node} \leftarrow \text{Pop} \left( \text{nodeHeap} \right) \]
\[ (\text{best}, \text{nodeHeap}, q_{\epsilon_{\text{sol}}{\max}}) \leftarrow \text{FeasibleDiving} \left( \text{node}, C, C_{\text{obs}}, \text{best}, \text{nodeHeap}, \epsilon_{\text{sol}}, q_{\epsilon_{\text{sol}}{\max}} \right) \]
\[ q_{\max} \leftarrow \max(q_{\epsilon_{\text{sol}}{\max}}, q_{\min} + \Delta_{\text{obj}}) \]

\[ \text{return} \ \{ \text{best.solutions}, \text{best.q}_{\min}, q_{\max} \} \]

**Algorithm 1:** Interval-based B&B calling Feasible Diving at each node

The model as parameter: the parameters \( p \) of the model, i.e. the variables to be determined using the internal constraints \( C \) and the observation constraints \( C_{\text{obs}} = \{ -\tau \leq f(o_i, p) \leq +\tau \} \) (see Section 1).

The algorithm maintains a set of open nodes \( \text{nodeHeap} \) implemented by a heap data structure having at its top the node with the largest number of possible inliers (i.e., an upper bound \( q_{\max} \) of the objective function value to maximize). Several records are associated to a node: the corresponding domain (box), the maximum set \( \text{possibleCS} \) (CS stands for consensus set) of observation constraints in the node (i.e., the constraints that have not been discarded by the \( q \)-intersection algorithm), and the set of observation constraints \( \text{validCS} \) guaranteed to be satisfied in the box and its cardinality \( q_{\min} = \text{card}(\text{validCS}) \) corresponding to a lower bound of the number of inliers guaranteed to belong to the box, if a feasible point has been found in the box.

The loop in Algorithm 1 performs a best-first search calling at each iteration the \text{FeasibleDiving} function. From the selected node, \text{FeasibleDiving}, described in Algorithm 2, builds a tree in depth-first order and keeps only
Algorithm 2: The Feasible Diving procedure run at each node of our interval B&B algorithm.

```
FeasibleDiving(node, C, C_{obs}, best, nodeHeap, \epsilon_{sol}, q_{max}^{sol})
while node.box \neq \emptyset and width(node.box) > \epsilon_{sol} and not node.isLeaf do
    (node1, node2) ← Bisect(node) // Bisection of node.box
    (node1, best, q_{max}^{sol}) ← Contract&Bound(node1, C, C_{obs}, \epsilon_{obj},
    \Delta_{obj},\epsilon_{sol}, q_{max}^{sol})
    (node2, best, q_{max}^{sol}) ← Contract&Bound(node2, C, C_{obs}, \epsilon_{obj},
    \Delta_{obj},\epsilon_{sol}, q_{max}^{sol})
    (nodebetter, nodeworse) ← Sort(node1, node2)
    if nodeworse.box \neq \emptyset and width(nodeworse.box) > \epsilon_{sol} and not isLeaf then
        nodeHeap ← Push(nodeworse, nodeHeap)
    node ← nodebetter
return (nodeHeap, best, q_{max}^{sol})
```

the most promising node at each iteration. More specifically, the box in the node is bisected along one dimension and the two sub-boxes are handled by the Contract&Bound procedure. The sub-box with the largest $q_{max}^{sol}$ is handled in the next step while the other sub-box is pushed into the heap nodeHeap of open nodes. Thus, contrary to a standard depth-first search, FeasibleDiving does not perform any backtracking. That is why we call it a greedy or diving algorithm.

Algorithm 3: Contraction of a node box and computation of valid inliers

```
Contract&Bound(node, C, C_{obs}, \epsilon_{sol}, \Delta_{obj}, best, q_{max}^{sol})
node ← ContractAndQinter(node, C, C_{obs}, best.q_{min} + \Delta_{obj} + 1)
if node.box \neq \emptyset then
    node ← ValidateBox(node, C, C_{obs})
    if node.q_{min} > best.q_{min} then // New best solution found
        best ← node
        nodeHeap ← FilterOpenNodes(nodeHeap, best.q_{min} + \Delta_{obj})
    if width(node.box) < \epsilon_{sol} then
        q_{max}^{sol} ← max(q_{max}^{sol}, node.q_{max})
    else
        if node.q_{min} = node.q_{max} then
            node.isLeaf ← true
return (node, best, q_{max}^{sol})
```
Algorithm 3 describes the Contract&Bound function handling every open node. It contracts the node box and tries to find more than $best.q_{min} + \Delta_{obj}$ valid (guaranteed) satisfied observation constraints inside the box. Recall that the best node found so far has at least $q_{min}$ valid inliers, so that the next best node (if any) is searched for with a cost at least equal to $best.q_{min} + \Delta_{obj} + 1$.

**The ContractAndQinter contraction procedure**

This procedure calls standard interval contraction procedures and a $q$-intersection projection procedure (see the $q$-proj algorithm in Section 2.2.2).

1. The box is first contracted by the constraints $C$ between parameters, if any: first, the HC4 constraint propagation algorithm (see Section 2.2.1) is performed; compared to Jaulin et al.’s algorithm, we add calls to a linear programming solver for improving variable interval bounds, using a linear relaxation of the constraints based on affine arithmetics [27].

2. The $q$-intersection projection algorithm, using $q = best.q_{min} + \Delta_{obj} + 1$ is then called on the parameter directions, plus one. Indeed, $q$-proj is also called on an additional direction for which the contraction expected is better. Details can be found in [24].

3. An upper bound $node.q_{max}$ of the number of inliers in the node is given by the minimum over all dimensions of the maximum number of intersected intervals found by the $q$-intersection projection procedure in a dimension.

4. If $node.q_{max} < q$, then the node box becomes empty and the corresponding branch in the tree will be pruned. Also, $node.q_{max}$ may be inferior to the number of possible observations in the box (i.e., $\text{card}(node.\text{possibleCS})$), in particular if the box contains several valid models (solutions).

**The ValidateBox procedure for finding a better feasible point**

To improve the lower bound, one has to find a feasible point in the current box, i.e. a so-called valid or guaranteed point that satisfies all the constraints $C$, if any, and a greater number of observation constraints (inliers) than the current solution. This is the aim of the ValidateBox procedure.

When $C$ is empty, one can simply compute the number of inliers at one point in the current box, for example at the middle of the box.

When the parameters have constraints between them (i.e., $C$ is not empty), the search of the solution validating inliers is a difficult task per se, since the selected vector must also satisfy the constraints in $C$. To find this feasible point, we use an algorithm similar to the algorithm finding feasible points in IbexOpt (see [1]). Every equation $h(p) = 0$ in $C$ is replaced by two inequalities $-\epsilon_{eq} \leq h(p) \leq +\epsilon_{eq}$ (e.g., $\epsilon_{eq}$ is fixed to $1\cdot10^{-4}$ in our experiments). We will then try to guarantee with interval-based calculations that the solution found will satisfy these relaxed constraints, as follows. First, we linearize these constraints in order to build an inner polytope inside the box where all the points
satisfy the constraints $C$. Second, if an inner polytope has been found, we use a linear programming approach to pick a point (vertex) of this polytope. Third, we compute the number of inliers at this point and update the new current solution if the number improves the best number of inliers found so far (in the best node). Different inner polytopes may be built and, for a given polytope, different vertices may be selected. Therefore several random runs of this routine are performed. We experimentally fixed the maximum number of runs to 10, stopping the procedure as soon as an empty inner polytope is built.

If a new best solution is found by ValidateBox, then the FilterOpenNodes procedure removes from the nodeHeap set of open nodes those for which $node.q_{\max}$ is inferior or equal to $best.q_{\min} + \Delta_{obj}$.

Note that Jaulin et al.'s method on which we have built our tool does not have any ValidateBox procedure to guarantee the computed model. Only small boxes are returned in which one expects (with no guarantee) to have a real vector checking the observation constraints.

Towards a generic code for parameter estimation

With the implementation of our EstimBB code, we are close to providing a generic tool for parameter estimation taking into account outliers. More specifically, it is straightforward to provide our tool with most of the code parameters:

- the parameterized model (model parameters and internal constraints),
- the observation constraints made of the $f$ expression, the $\tau$ relaxation value of the equality constraints, and the $o_i$ observations, and
- the $\Delta_{obj}$ and $\epsilon_{sol}$ precision parameters.

The contractor part could be made more generic, i.e. could be specified outside the C++ code. It is for now encoded in the IBEX C++ library following the contractor programming principles [7]. For example, for the essential matrix estimation described in the next section, the IBEX code specifies that HC4 is first called on the internal constraints $C$, and then the Affine relaxation technique (ART) contracts the same system before the $q$-intersection contractor ($q$-proj) uses the observation constraints. This means that if users want to solve a new problem, either they must either copy-paste the C++ code encoding the same sequence of contractors, or modify it to change the contractors involved.

4 Experiments

The algorithms were implemented in IBEX. We made preliminary experiments on the shape detection problem studied in [24], replacing the search of all

\footnote{Note that this algorithm is heuristic in that it is possible that no such inner polytope is found.}
solutions with at least \( q \) inliers by the optimization problem that searches for a solution maximizing the number of inliers. Second, we tried to solve the more challenging problem of essential matrix estimation in stereo vision.

4.1 Shape Detection

The combination of several improvements to Jaulin’s et al. method brings a significant speedup of two orders of magnitude on each tested instance of 3D plane and 2D circle detection problems. The different improvements are described in [24] and let us briefly summarize the principle of the most important one. The \texttt{ContractAndQinter} procedure has been extended to better contract the boxes/nodes, using a new dimension. Once the \( q \)-proj algorithm ends, a new procedure linearizes all the observation constraints and projects them on a new axis, along the mean normal direction. On this new dimension the intervals are expected to be smaller and \( q \)-intersecting them will more likely reduce the resulting box.

These problems have a parameter space of dimension \( n = 3 \) and have no additional (internal) constraints. First experiments dealing with the search of all planes or circles are described in [24]. They suggested that our interval B&B algorithm could also guarantee a model maximizing the number of inliers while ensuring a good performance. We have verified that we obtain the same speedups for all the improvements when we search for the best model in number of inliers. The Branch & Prune algorithm in [24] was replaced by the B&B algorithm presented in Section 3.

For plane detection, we have generated 9 artificial test cases, with different numbers of points, numbers of planes to detect, and inlier rates (i.e., percentage of points belonging to a plane). All the 3D points belong to a bounded box. The inlier points belonging to each plane have been placed near the plane: two coordinates are random, and the third one is computed to satisfy the thick equation of the plane. The remaining points have been uniformly randomly generated in the bounding box. In order to be close to the real case described further, the planes are made of 2 sets of orthogonal planes (e.g., 2 vertical and 2 horizontal planes for the instances \( P_1 \) to \( P_3 \)). Table 1 presents the characteristics of the test cases: the number of points \( m \), the number of planes, the minimum inlier rate \( r \) and the maximum inlier rate. We run our algorithm with the following parameters \( q_m = r \cdot m, \tau = 0.001 \) and \( \epsilon_{sol} = 0.00001 \). The time out (TO) was set to 10,000 seconds.

<table>
<thead>
<tr>
<th>Test case</th>
<th>( P_1 )</th>
<th>( P_2 )</th>
<th>( P_3 )</th>
<th>( P_4 )</th>
<th>( P_5 )</th>
<th>( P_6 )</th>
<th>( P_7 )</th>
<th>( P_8 )</th>
<th>( P_9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>#points</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>4000</td>
<td>4000</td>
<td>4000</td>
</tr>
<tr>
<td>#planes</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>minimum inlier rate</td>
<td>10%</td>
<td>5%</td>
<td>4%</td>
<td>2%</td>
<td>1.5%</td>
<td>1%</td>
<td>2%</td>
<td>1.5%</td>
<td>1%</td>
</tr>
<tr>
<td>maximum inlier rate</td>
<td>20%</td>
<td>10%</td>
<td>10%</td>
<td>2%</td>
<td>1.5%</td>
<td>1%</td>
<td>2%</td>
<td>1.5%</td>
<td>1%</td>
</tr>
</tbody>
</table>
We have also tested our algorithm on two real 3D point clouds, issued from parts of scene1 of velodyndata\(^2\) [19]. We selected from this urban scene the 529 points labeled house40, and the 3611 points labeled house44, two specific buildings in the scene. These problems are named \(H_{40}\) and \(H_{44}\) in the tables.

We run our algorithm with the parameters \(\tau = 0.001\), \(\epsilon_{sol} = 0.0001\), \(q_m = 21\) for \(H_{40}\) and \(q_m = 75\) for \(H_{44}\).

We made finally an experiment for circle detection in a 2D photo. The data are issued from a buoy detection problem [16] amounting to finding a circle in a submarine photo. We used the data of the paper with 614 points \((C_1)\), and created a (more) noisy problem \(C_2\) with 1228 points, randomly adding to the original problem 614 outliers. The parameters are the center coordinates \((x, y)\) and the radius \(r\) of the circle to be detected. An observation \(o_i\) is given by the coordinates \(ox_i\) and \(oy_i\) of the point in the image. The equation is then

\[
(x - ox_i)^2 + (y - oy_i)^2 = r^2.
\]

We run our algorithm with the parameters \(q_m = 57\), \(\tau = 11\) and \(\epsilon_{sol} = 0.02\).

### Table 2 Maximum number of inliers and CPU time results on plane and circle detection test cases

<table>
<thead>
<tr>
<th>Test case</th>
<th>(P_1)</th>
<th>(P_2)</th>
<th>(P_3)</th>
<th>(P_4)</th>
<th>(P_5)</th>
<th>(P_6)</th>
<th>(P_7)</th>
<th>(P_8)</th>
<th>(P_9)</th>
<th>(H_{40})</th>
<th>(H_{44})</th>
<th>(C_1)</th>
<th>(C_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#inliers</td>
<td>202</td>
<td>103</td>
<td>103</td>
<td>20</td>
<td>22</td>
<td>16</td>
<td>94</td>
<td>16</td>
<td>35</td>
<td>38</td>
<td>171</td>
<td>58</td>
<td>58</td>
</tr>
<tr>
<td>QInterEstim</td>
<td>1.0</td>
<td>2.2</td>
<td>3.6</td>
<td>17.8</td>
<td>34.5</td>
<td>102.7</td>
<td>60.4</td>
<td>86.2</td>
<td>206.4</td>
<td>2.9</td>
<td>638.5</td>
<td>7.7</td>
<td>24.2</td>
</tr>
<tr>
<td>BasicEstimBB</td>
<td>7.9</td>
<td>61.7</td>
<td>62.6</td>
<td>4579</td>
<td>8121</td>
<td>TO</td>
<td>TO</td>
<td>TO</td>
<td>TO</td>
<td>TO</td>
<td>28.3</td>
<td>89.4</td>
<td></td>
</tr>
<tr>
<td>EstimBB</td>
<td>0.2</td>
<td>0.5</td>
<td>0.5</td>
<td>11.0</td>
<td>16.9</td>
<td>40.0</td>
<td>40.1</td>
<td>62.0</td>
<td>126.6</td>
<td>1.3</td>
<td>193.6</td>
<td>1.3</td>
<td>6.4</td>
</tr>
</tbody>
</table>

We report in Table 2 the maximum number of inliers found (and proven) by our algorithm, the CPU times required (in second) by the QInterEstim algorithm described in [24] for computing all models (planes, circles), by the basic B&B algorithm using q-intersection named BasicEstimBB, without all improvements detailed in [24], and by our EstimBB algorithm that includes all these improvements. We can note that, as expected, finding and proving an optimal solution is faster than finding all solutions. The QInterEstim results are slightly different from those presented in [24], since the algorithm has been rerun in the current version of IBEX to make the comparison fair.

Observation: We have compared the application of our generic \texttt{EstimBB} code to shape recognition with a mixed-integer program for 2D line detection in the plane. The MIP model was solved by CPLEX.\(^3\) It turns out that the MIP model is competitive with our interval approach only for a small number of points or a large inlier rate. The MIP approach does not scale up either to a parameter space of dimension 3 (plane detection in 3D space). The MIP model, adapted to find a plane with a maximum number of inliers, could solve

\(^2\) See sites.google.com/site/kevinlai726/datasets

\(^3\) Thanks to Leo Liberti for suggesting the MIP model.
to optimality the problem $P_1$ with only 200 points and the same inlier rate in 1320 seconds, whereas our EstimBB code solved it in 0.07 second.

4.2 Essential Matrix of a Stereo Image Pair Estimation

In computer vision, the essential matrix is a $3 \times 3$ matrix having some internal constraints, which relates corresponding points in stereo images assuming that the cameras satisfy the pinhole camera model.

4.2.1 Problem presentation

The essential matrix, discovered by Longuet-Higgins [20], encodes the relative position and attitude of two cameras following the (ideal) pinhole model. To each camera, called arbitrarily left and right, is attached an orthonormal coordinate frame, whose $z$-axis is the view direction. A 3D point, of coordinates $M = (X Y Z)^T$ in the right camera frame, is only observed up to scale as its projection into the image plane $m_R = \lambda(M)M$ with $\lambda(M) = f_R/Z$, with $f_R > 0$ the focal length of the right camera (see Figure 3 left). The point is observed in the left image at $m_L = \mu(M)(RM + T)$ with $R$ and $T$ the rotation and translation matrices mapping from right to left coordinate frames. Substituting $M$ as a function of $m_R$ in the latter equation yields:

$$m_L = \mu(M)/\lambda(M) Rm_R + \mu(M) T,$$

which amounts to saying that vectors $m_L$, $Rm_R$ and $T$ are linearly dependent. This dependency can be rewritten into the epipolar equation:

$$\det (m_L T Rm_R) = m_L^T (T \times R) m_R = 0,$$

where the essential matrix $E = T \times R$ denotes the $3 \times 3$ matrix composed of the columns of $R$ multiplied on the left by $T$ with the vector product.

Inversely, through a factorization of $E$, the pair $(R,T)$ can be recovered, yielding relative pose and attitude of the stereo system of cameras. Each matching point pair $(m_L^i, m_R^j)$ gives a single homogeneous constraint on $E$, so that $E$ can only be computed up to an unknown scale. This means that only the direction of $T$ can be estimated, a phenomenon known as scale ambiguity.

The epipolar equation (8) can be interpreted, algebraically, as an orthogonality relation between the vector $m_L$ and the vector $Em_R$, or, geometrically, as the point $m_L$ being in the plane through $O_L$ orthogonal to the vector direction $Em_R$. Intersecting this plane with the image plane $Z = f_L$ yields a line in the image, called the epipolar line (in left image) associated to $m_R$, and $m_L$ is on this line. Symmetrically, $m_R$ is in the plane through $O_R$ orthogonal to direction $E^Tm_L$. The vectors yielding no epipolar line are $e_L = \mu T$ and $e_R = \lambda R^{-1}T$ characterized by $Ee_R = E^T e_L = 0$. These points, called epipoles, designate the direction of one camera center relative to the coordinate frame of the other camera (see Figure 3 right).
The estimation of $E$ is done through matching point pairs $(m^i_L, m^i_R)$ representing an observation $o_i$ via a procedure known as the 5-point algorithm [28], exploiting the non-linear constraints on matrix $E$:

$$
\|E\| = 1 \quad 2EE^T - \text{trace}(EE^T)E = 0 \quad \det(E) = 0,
$$

where the norm constraint on $E$ prevents the trivial solution $E = 0$ of the other two equations. Since automatically computed matching pairs $(m^i_L, m^i_R)$ may be outliers for some $i$, the RANSAC algorithm [12] (see Section 2.1) is traditionally used: five pairs are selected to estimate $E$, then the number of other pairs, called inliers, compatible with this estimation (up to the threshold $q$) is counted; the procedure is repeated and the estimation of $E$ with a maximal number of inliers is returned. However, the five-point procedure is complex due to its non-linear constraints (9). Therefore they are often relaxed in favor of the estimation of the fundamental matrix $F = (K_L^{-1})^T E K_R^{-1}$, where $K_L$ and $K_R$ map direction vectors $m_L$ and $m_R$ to the image points $x_L$ and $x_R$ in homogeneous coordinates [15]. The fundamental matrix satisfies the epipolar equation linked to image points $x_L^T F x_R = 0$, and the only constraint on the $3 \times 3$ matrix $F$ is $\det(F) = 0$. From matching image points $(x^i_L, x^i_R)$, the simpler eight-point algorithm can be used in a RANSAC procedure and the determinant constraint enforced a posteriori. Also, the slightly more complex seven-point algorithm, embedding this constraint, can be used [23]. To summarize, the cardinality of a RANSAC sample can be chosen as $n = 5$ (direct essential matrix estimation), $n = 7$ (via the fundamental matrix with singularity constraint), or $n = 8$ (the same but with the singularity constraint enforced a posteriori).

4.2.2 Related Work

The classical methods, based on RANSAC, are not able to guarantee that the solution found is optimal, i.e. are not able to maximize the number of inliers. Some works, in solving different computer vision optimization problems, tried to find the maximum number of inliers, using a B&B method.
One work [34] studies the estimation of the essential matrix and follows an approach close to ours: it maximizes the number of inliers using a B&B method. The differences lie in the essential matrix parameterization and in the angular reprojection error criterion used in their approach. This extends the earlier global optimization approach proposed in [14], where a B&B algorithm is used to directly search for an essential matrix minimizing the maximum reprojection error, without identifying outliers. Note that these approaches rely on ad-hoc considerations about the essential matrix problem, while our proposed method is a direct application of the generic algorithm.

Another work [9,10] solves different computer vision problems as homography estimation, linearised fundamental matrix, or affine registration, finding the maximum number of inliers. The B&B approach used builds a search tree, branching directly on the inliers set. The optimization at each node handles a pseudo-convex problem. Good results are obtained for problems with a low outlier rate.

Rotational homography with unknown focal length is studied in [2]. A specific parameterization is used for the 4 parameters (rotation angles and focal length).

The camera pose and feature correspondence problem is solved by a B&B method in the 6D space of camera poses [5]. Specific computations are proposed for finding a tight upper bound.

4.2.3 Model implementation

We implemented the essential matrix estimation problem as a parameter estimation problem:

- Observations: \( \mathbf{o}_i = (m^i_L, m^i_R) := (K^{-1}_L w^i_L, K^{-1}_R w^i_R) \). Calibration matrices \( K_L \) and \( K_R \) are known (depending on focal lengths \( f_L \) and \( f_R \), and the positions of principal points, orthogonal projections of camera centers \( O_L \) and \( O_R \) on their associated image plane), their last row is \((0 \ 0 \ 1)\) and points are in homogeneous pixel coordinates \( w = (u \ v \ 1)^\top \).

- Model: the essential matrix \( E \) is represented by a \( 3 \times 3 \) matrix with 9 variables, linked by the non-linear constraints presented above: normalization, determinant, essential matrix constraints. Overall, these constraints remove 4 degrees of freedom. Our parameter vector is thus \( \mathbf{p} = (E_{11}, E_{12}, \ldots, E_{33}) \).

The internal constraints \( C(\mathbf{p}) \) are the 11 scalar equations (9).

- Criterion: maximization of the number of inliers, with a given tolerance \( \tau \). An inlier is an observation \( \mathbf{o} = (m_L, m_R) \) such that

\[
-\tau \leq f(\mathbf{o}, \mathbf{p}) = m^\top_L E m_R \leq \tau. \tag{10}
\]

However, for a point near the epipole, such that \( m^\top_L E \sim 0 \) or \( E m_R \sim 0 \), the preceding test can succeed whatever the corresponding point, respectively \( m_R \) or \( m_L \). Therefore, we add the following requirement for an inlier, depending on a threshold \( \epsilon_{\text{epi}} \):

\[
\min(\| E^\top m_L \|_\infty, \| E m_R \|_\infty) > \epsilon_{\text{epi}}. \tag{11}
\]
For estimating this essential matrix model with constraints, we use the generic EstimBB described by Algorithm 2 that carries on a Best First Search strategy performing feasible diving (FD) at each node. The bisection strategy used is the classical Largest First strategy available in IBEX, where at each node the variable with the largest interval is chosen to be bisected. The other details of the algorithm, the contraction procedures and the policy for validating inliers have been presented in Section 3.

4.2.4 Experimental results

We tested our method on the 6 real instances for essential matrix estimation from USAC library [29], named test1 to test6, with between \( m = 534 \) and \( m = 2245 \) pairs of 2D points.

The tolerance \( \tau \) was estimated for each instance as the result of a variant of the RANSAC algorithm, that determines this parameter based on statistical considerations [23]. The parameters of the algorithm reported in the left part of Tables 3 and 4 are the size limit \( \epsilon_{sol} \) of bisected boxes and the precision \( \Delta_{obj} \) on the number of inliers.

The results reported on the right part of the tables are an interval \([a, b]\) on the number of inliers, meaning that a solution with \( a \) inliers has been validated and that no solution with a number of inliers greater than \( b \) exists. We also reported the CPU time in second, the number of nodes, and the memory used in MBytes on an x86-64 CPU at 1600 MHz. An entry \( > 100,000 \) means that the timeout of 100,000 seconds was reached. The test2, test3, test4 and test5 instances are solved to optimality with a 1% precision on the number of inliers w.r.t the total number of correspondences. In Table 3, we can see that our search algorithm is always significantly faster than the traditional Best First Search (BFS). It also quickly finds good solutions and its memory consumption is smaller than BFS (see the memory column).

Depth First Search (DFS) can only solve test2 and is 2.5 times slower than FD. For the other instances, DFS remains stuck in a local optimum area, and is unable to give an upper bound, or terminates with a larger precision (test5).

Figure 4 shows the 427 inliers found among the 534 point-point correspondences of both images of test2. Each point is displayed with a portion of the epipolar line issued from its matching point and the estimated essential matrix. Most points are on, or close to, their epipolar line. Visual inspection (no ground truth is provided for these datasets) checks that all are correct, except a few mismatches that stand out: 3 points on the right half of the left image, and 1 point on the brick wall of the right image. These cannot be real since the corresponding zones are out of the frame of the alternate image. Still, they are classified as inliers because they are close to their epipolar line by chance. Indeed, the epipolar test is a point-to-line distance, so 1D in nature. Knowing the image is 800 pixels high and epipolar lines are mostly horizontal, with a 1-pixel tolerance, a pair of points randomly chosen independently in both images can be misclassified as an inlier with a probability of 1/400. Given the number of input pairs, it should not be surprising that a few mismatches are
not rejected. The detected outliers are shown in Figure 5. They are represented by red points. Portions of the corresponding epipolar lines are in light brown while the altitude vectors, in cyan, show the error. A couple of correspondences are a close call (near their epipolar lines), and could have been classified as inliers with a slightly looser tolerance.

Table 4 reports the results on *test1* and *test6* instances. Since these instances were not able to be solved to optimality, we also tested partial instances with only $m = 320$ correspondences and computed the number of inliers validated by the essential matrix result on these partial instances.

The *test1* instance is the most difficult one, since the tolerance is smaller and one has to go deeper in the search tree to validate the solution. The algorithm did not terminate in 100,000 seconds, even with a partial instance of 320 correspondences. We computed for this partial instance an essential matrix that has 2009 inliers when we validate it on the entire instance. This is better than the 2005 inliers directly computed on the entire instance.

The *test6* instance is also difficult because the percentage of inliers is small. We solved a partial instance with 320 correspondences to optimality. The essential matrix computed on this partial instance validates 546 inliers on the
Table 3: Experimental results on essential matrix estimation on the USAC test2, test3, test4, and test5 instances.

<table>
<thead>
<tr>
<th>instance</th>
<th>size</th>
<th>tolerance</th>
<th>algorithm</th>
<th>$\epsilon_{sol}$</th>
<th>$\Delta_{obj}$</th>
<th>inliers</th>
<th>time</th>
<th>nodes</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>test2</td>
<td>534</td>
<td>0.002</td>
<td>FD</td>
<td>0.0001</td>
<td>4 [427 432]</td>
<td>13,244</td>
<td>5,944,460</td>
<td>289</td>
<td></td>
</tr>
<tr>
<td>test2</td>
<td>534</td>
<td>0.002</td>
<td>BFS</td>
<td>0.0001</td>
<td>4 [427 431]</td>
<td>17,556</td>
<td>7,330,800</td>
<td>874</td>
<td></td>
</tr>
<tr>
<td>test2</td>
<td>534</td>
<td>0.002</td>
<td>DFS</td>
<td>0.001</td>
<td>4 [427 432]</td>
<td>31,904</td>
<td>9,829,778</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>test3</td>
<td>1026</td>
<td>0.001</td>
<td>FD</td>
<td>0.0001</td>
<td>9 [778 785]</td>
<td>15,270</td>
<td>4,253,024</td>
<td>202</td>
<td></td>
</tr>
<tr>
<td>test3</td>
<td>1026</td>
<td>0.001</td>
<td>BFS</td>
<td>0.001</td>
<td>9 [779 788]</td>
<td>21,391</td>
<td>5,417,166</td>
<td>803</td>
<td></td>
</tr>
<tr>
<td>test3</td>
<td>1026</td>
<td>0.001</td>
<td>DFS</td>
<td>0.001</td>
<td>9 [24 1026]</td>
<td>&gt; 100,000</td>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>test4</td>
<td>877</td>
<td>0.0005</td>
<td>FD</td>
<td>0.0001</td>
<td>8 [598 606]</td>
<td>15,051</td>
<td>4,981,658</td>
<td>197</td>
<td></td>
</tr>
<tr>
<td>test4</td>
<td>877</td>
<td>0.0005</td>
<td>BFS</td>
<td>0.0001</td>
<td>8 [599 607]</td>
<td>24,658</td>
<td>7,313,670</td>
<td>961</td>
<td></td>
</tr>
<tr>
<td>test4</td>
<td>877</td>
<td>0.0005</td>
<td>DFS</td>
<td>0.001</td>
<td>8 [85 877]</td>
<td>&gt; 100,000</td>
<td>47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>test5</td>
<td>1083</td>
<td>0.002</td>
<td>FD</td>
<td>0.0001</td>
<td>10 [657 667]</td>
<td>43,918</td>
<td>10,892,426</td>
<td>486</td>
<td></td>
</tr>
<tr>
<td>test5</td>
<td>1083</td>
<td>0.002</td>
<td>BFS</td>
<td>0.0001</td>
<td>10 [657 667]</td>
<td>53,453</td>
<td>12,448,860</td>
<td>794</td>
<td></td>
</tr>
<tr>
<td>test5</td>
<td>1083</td>
<td>0.002</td>
<td>DFS</td>
<td>0.01</td>
<td>10 [650 721]</td>
<td>&gt; 100,000</td>
<td>47</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Experimental results on essential matrix estimation on the USAC test1 and test6 instances.

<table>
<thead>
<tr>
<th>instance</th>
<th>size</th>
<th>tolerance</th>
<th>algorithm</th>
<th>$\epsilon_{sol}$</th>
<th>$\Delta_{obj}$</th>
<th>inliers</th>
<th>time</th>
<th>nodes</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>test1</td>
<td>320</td>
<td>0.0002</td>
<td>FD</td>
<td>0.001</td>
<td>2 [280 295]</td>
<td>&gt; 100,000</td>
<td>2198</td>
<td></td>
<td></td>
</tr>
<tr>
<td>test1</td>
<td>2245</td>
<td>0.0002</td>
<td>FD</td>
<td>0.001</td>
<td>21 [2005 2102]</td>
<td>&gt; 100,000</td>
<td>1009</td>
<td></td>
<td></td>
</tr>
<tr>
<td>test6</td>
<td>320</td>
<td>0.002</td>
<td>FD</td>
<td>0.001</td>
<td>2 [96 98]</td>
<td>72,321</td>
<td>55,626,252</td>
<td>1569</td>
<td></td>
</tr>
<tr>
<td>test6</td>
<td>2201</td>
<td>0.002</td>
<td>FD</td>
<td>0.001</td>
<td>21 [526 949]</td>
<td>&gt; 100,000</td>
<td>3777</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The entire instance with 2201 correspondences in 72,321 seconds, i.e. a greater number than the 526 inliers obtained with the algorithm running in 100,000 seconds on the entire instance.

We also performed experiments on the two image pairs from the Corridor (Figure 6) and Valbonne (Figure 7) data sets also tested in [34]. 89 SIFT matches were generated for each pair using the online SIFT code. The intrinsic camera parameters for the Valbonne church come from [13]. In these figures, inliers are marked with green points while outliers are linked by red lines. For each outlier, a portion of the epipolar lines is displayed in pale brown while the altitudes, marking the distance to the epipolar line, are shown in cyan. A close inspection of these results shows that there are two false negatives for the Corridor dataset: one is on the appliance at head level fixed to the wall, while the other corresponds to a point very close to the epipole in the left image. The latter is a true match but the proximity to the epipole induces an imprecise epipolar line, leading to a point-line distance of 7 pixels. There are also two false positives, one obviously on the pipe on the up-right part of the left image, the other on the corner of the letter O on the floor in the left image. Concerning the Valbonne dataset, all points are correctly classified.

The results are reported in Table 5. The Valbonne pair was easier to solve because of its smaller outlier rate. An objective comparison with the results

4 http://www.robots.ox.ac.uk/~vgg/data/data-mview.html
5 http://demo.ipol.im/demo/82/
of [34] is not possible because the input SIFT points were not provided, and different SIFT implementations differ notoriously in their results.

Table 5 Experimental results on essential matrix estimation on Corridor and Valbonne instances

<table>
<thead>
<tr>
<th>instance</th>
<th>size</th>
<th>tolerance</th>
<th>algorithm</th>
<th>$\epsilon_{sol}$</th>
<th>$\Delta_{obj}$</th>
<th>inliers</th>
<th>time (sec.)</th>
<th>nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corridor</td>
<td>89</td>
<td>0.002</td>
<td>FD</td>
<td>0.0001</td>
<td>0</td>
<td>[58 59]</td>
<td>5228</td>
<td>5,589,406</td>
</tr>
<tr>
<td>Corridor</td>
<td>89</td>
<td>0.002</td>
<td>FD</td>
<td>1.e-8</td>
<td>0</td>
<td>[58 58]</td>
<td>5973</td>
<td>6,331,172</td>
</tr>
<tr>
<td>Valbonne</td>
<td>89</td>
<td>0.002</td>
<td>FD</td>
<td>0.0001</td>
<td>0</td>
<td>[76 76]</td>
<td>1344</td>
<td>1,492,470</td>
</tr>
</tbody>
</table>

We also tried another criterion with a more geometric sense, the Sampson error, to decide whether a matching is an inlier. This criterion is more costly to compute and it is more difficult for the $q$-proj procedure to discard the outliers. The results are for the moment limited to a partial instance of test2 with 20 correspondences, where a result of [17 20] inliers is found in 28,900 seconds. The current solver cannot handle either the fundamental matrix estimation problem useful in stereo vision (dimension 7).

5 Conclusion

This paper has presented an exact interval B&B approach implemented in IBEX for a parameter estimation taking into account outliers. The corresponding code is rather generic and has the potential to become an IBEX plug-in. First experiments on two computer vision problems (shape detection in dimension 3 and essential matrix estimation in dimension 5) suggest that the current interval B&B algorithm provides good results in medium dimension,
i.e. up to 4 or 5 parameters (depending on the inlier rate), whereas most exact approaches cannot cope with dimension 3.

References


