Local Geometric Consensus: A General Purpose Point Pattern-Based Tracking Algorithm
Liming Yang, Jean-Marie Normand, Guillaume Moreau

To cite this version:

HAL Id: hal-01276767
https://hal.archives-ouvertes.fr/hal-01276767
Submitted on 20 Feb 2016

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.
Local Geometric Consensus: a general purpose point pattern-based tracking algorithm

Liming Yang, Jean-Marie Normand, and Guillaume Moreau

Abstract—We present a method which can quickly and robustly match 2D and 3D point patterns based on their sole spatial distribution, but it can also handle other cues if available. This method can be easily adapted to many transformations such as similarity transformations in 2D/3D, and affine and perspective transformations in 2D. It is based on local geometric consensus among several local matchings and a refinement scheme. We provide two implementations of this general scheme, one for the 2D homography case (which can be used for marker or image tracking) and one for the 3D similarity case. We demonstrate the robustness and speed performance of our proposal on both synthetic and real images and show that our method can be used to augment any (textured/textureless) planar objects but also 3D objects.

Index Terms—Augmented reality, Tracking, Point-pattern matching, Pose computation.

1 INTRODUCTION

Object tracking is the corner stone of vision based augmented reality (AR) and “tracking by matching keypoints” techniques have proven successful in recent years. However, most of the methods only work with local feature descriptors based on textures, such as SIFT [12], SURF [2], BRIEF [4] etc. Nevertheless, tracking only based on points spatial layout without using any texture information offers other important benefits. Most of all, it allows for tracking models that are textureless or which textures may vary, e.g. augmented maps [29]. Besides, as pure points can be retrieved from standard textured images with the help of feature point detection, this kind of method could also be used to “augment everything” [24].

In this paper, we focus on the pure point pattern tracking problem, more specifically on “tracking by matching” (as opposed to “tracking by tracking” [24]) since the former can more easily recover from tracking failure and ensures that there is no “drift” during tracking. In other words, tracking by matching consists of performing matchings at 20 - 30Hz and compute camera pose from those matchings.

We reformulate the matching problem as follows: consider having \( M \) model point sets \( (P_1, P_2, \ldots, P_M) \), each point set \( P_i \) containing uniformly randomly distributed \( m_i \) points in \( d \) (2, 3) dimensions. Let \( T : \mathbb{R}^d \rightarrow \mathbb{R}^d \) be a geometric transformation on \( d \) dimensions of which we only know its type (similarity, homography, etc.). Let \( Q \) be a set of observed scene points, some of which belong to \( T(P'_i) \) where \( P'_i \) is a subset of \( P_i \) in which each point may have undergone a small translation (i.e. jitter). Other points of \( Q \) are extra points, i.e. acquisition noise. Our objective is to find \( P_i \) amongst the \( M \) known model point sets and to determine the transformation \( T \) between \( P_i \) and \( Q \).

The difficulty of the pure point pattern matching (PPM) over traditional texture-based keypoint matching is that the available information for matching is extremely limited: only the geometrical distribution of points. Moreover, due to the presence of jitter, the only usable information becomes biased. Given the poorness of the quality of the information, robust matching algorithms are generally time consuming. This is the reason why PPM algorithms in the literature are either fast but not robust against point jitter [15], or robust but not fast enough [29] for tracking purposes in complex AR applications.

Our main contribution is a general algorithm for pure point pattern matching under any transformation in 2D or 3D in a fast and reliable way. The main purpose of this work is not to compete with existing texture-based tracking algorithms but rather to address a problem where current methods would fail or require a lot of preprocessing, which may include:

- Model-based augmentation with little texture information (e.g. CAD drawings)
- Augmenting different paper maps with GIS [29]
- Industrial environments in which placing textures (or classical black and white markers) may be an issue
- Registering 3D models acquired in different lighting conditions
- Robust initial pose estimation for edge-based tracking.

The remainder of this paper is organized as follows: Section 2 presents related works and the differences between our approach and the existing literature. Section 3 describes the general framework which can be easily adapted to many different transformations. Then, detailed implementations for the two most frequently used transformations (i.e. 2D perspective and 3D similarity) are presented in Section 4. Section 5 gives the evaluation results of the method for the 2D homography case with synthetic point sets and tests with 3D similarity model registration. Section 6 presents two applications using our method while Section 7 motivates some choices before concluding.

2 RELATED WORKS

There exist numerous methods to solve PPM problems, many of them being general methods that can be used with various transformations. In this section, we first focus on some typical general methods before moving to some specific methods that relate to our goal: tracking.

General methods can be divided into two categories, direct methods and iterative ones. Geometric Hashing (GH) [28] is a classical direct method. It uses subsets of points to construct coordinate bases, calculate other points’ coordinates under such bases and then vote for each basis. The basis which receives the highest number of votes gives the final result. GH is robust against extra/missing points and point jitter but its major drawback is the huge computational complexity in \( O(b^d) \), where \( b - 1 \) is the number of points needed to form a basis [28]. Generalized Hough Transformation [1] or pose clustering [16] are computationally intensive when the dimensions of the parameter space increase. Branch and bound algorithms [14] recursively compute parametric sub-spaces and calculate the upper/lower bounds for each sub-space to improve performance. RANSAC [7] is another direct approach for PPM. First one subset of correspondences is randomly chosen among all possible point correspondences in order to estimate a first “hypothesis” of the transformation. Then other correspondences are used to check whether this hypothesis is valid. The “hypothesis” that gives rise to the highest number of inliers is chosen as final output. RANSAC is an efficient method when the inlier ratio (in the sense of correspondence) is higher than 50%. However, when applied to pure PPM problems, since the number of all combinations \( (p \in P, q \in Q) \) is very large, RANSAC can hardly work. If a point
correspondence reliability metrics is available, PROSAC [5] can be an efficient method for PPM.

Iterative methods have also been developed. ICP [3] assigns to each model point its nearest neighbor in the scene point set and minimizes the sum of correspondence distances in each iteration until a threshold is reached. Robust point matching [8] uses a soft assignment method to iteratively find correspondences instead of only relying on the nearest neighbors in each step. Inspired by image correlations, Tsin et al. [23] developed the Kernel Correlation method, which defines a cost function to describe the registration of two point sets and finds the registration by minimizing the cost function.

In AR, RANSAC-like methods are often used in conjunction with texture-based key point algorithm, which can often greatly reduce the number of possible correspondences. Other general point registration methods, being not initially designed for tracking purpose, prove to be practically unusable for AR because of their computation cost.

Herling et al. proposed the Universal Feature Tracking [9] (UFT) method for tracking 3D point sets. They first generate 2D point patterns of the 3D point set in different virtual camera poses. During the initialization, the 2D pattern which matches best the scene point set is selected. Then the real camera pose is calculated by iteratively finding an accurate transformation between the selected 2D pattern and the scene point set. Tracking is done with previous frame information. Theoretically their method could work with real 3D point data but only planar objects are used in their experiments. Moreover the method also has constraints related to object occlusion as well as the camera-object distance.

As to the special case of 2D homography, LLAH [15] uses local geometry instead of whole point sets to achieve real time tracking. It constructs surface-ratios to create index tables for each point by using its nearest neighbors so that points can be efficiently matched. AR applications are developed by using this method such as Random Dot Markers (RDM) [25], augmenting city maps [26], and general planar objects [24]. However, this method is sensitive to extra/missing points within the point pattern as well as to point jitter, as shown in [29] and later in this paper (cf. Section 5.1).

Our previous work, RRDM [29], also uses surface-ratios in points’ neighborhoods. Since it constructs a more robust descriptor, to overcome problems encountered by LLAH, augmenting unprepared maps with GIS data becomes possible. Unfortunately, time complexity of this method is quadratic. When the point set becomes large, real-time tracking is impossible, as described in the map experiment of [29].

Our proposal is inspired by the work of [28, 15, 29, 7]. We rely on Geometric Hashing (GH) to be robust against noise, work on local point sets to speed up the process and geometric consensus between neighbor point sets ensures a termination with reduced computations. Even though the refiner (Section 3.4) of Local Geometric Consensus (LGC) remains similar to the recovery phase of [29], LGC has the following advantages compared to the state of the art:

- It is much faster than GH, RDM and RRDM
- It is more robust than RDM and RRDM
- It handles several models, unlike RRDM (unclear for UFT)
- It handles any known transformation, unlike RDM and UFT
- It deals with large occlusion, does not restrict the distance to objects nor needs previous frames information unlike UFT
- It can handle much lower inlier ratios than RANSAC
- It has more use cases than texture-based tracking techniques.

3 General algorithm

Our algorithm takes several model point sets \(P_1, P_2, \ldots, P_M\), a query scene point set \(Q\) and the type of transformation as inputs. The output is the matched model number \(i\) and the transformation \(T\) between \(P_i\) and \(Q\), with \(T\) belonging to the given transformation type (cf. Fig. 1).

The core idea is based on the consensus of local geometry: correspondences information between two small subsets of \(P_i\) and \(Q\) is used as an indication to find their neighboring subsets’ correspondences, thus all the other correspondences between the two sets can be found. The very first correspondences information between two subsets is found by geometric hashing, in which local coordinate systems, being invariant to local transformations, are constructed to perform the matching.

Some important definitions and a brief description of LGC are given in Section 3.1 while the next sections will give detailed explanations.

3.1 Definitions and brief description of the algorithm

The algorithm works with local geometric features. By assuming that points are randomly distributed, the relative positions between points in a local point set is very characteristic (demonstrated in [15]). For each model point \(p\) or scene point \(q\), \(p\) (or \(q\)) and its \(k\) nearest neighbors (\(k\) being a parameter of the algorithm) are used as a local geometric feature, we call it \(p\)-patch (resp. \(q\)-patch). Two patches (\(p\)-patch, \(q\)-patch) with known point correspondences is called a paired-patch (\(p, q\)) (cf. Fig 2). With the point correspondences, one can also find the transformation of the paired-patch. As some transformations (such as homographies) are not linear, according to Taylor expansion (cf. Section 7.2), they can be approximated by other linear ones (such as affinities) in a small local area. A local transformation \(T\) is used in local patches as a linear approximation of \(T\).

Let \(m_i\) be the number of points in model \(P_i\) and \(V_i\) be the space enclosed by \(P_i\)’s convex hull. Then \(p_i = \sqrt{m_i/V_i}\) is the point density of \(P_i\), where \(d\) is the number of dimension of point sets. As a consequence, \(l_i = p_i^{-1}\) is a measure of inter-point distance. Without losing generality, we assume that all point sets are normalized beforehand so that the center of mass of each point set is at origin and \(l_i = l\) is now the same for all models. We assume also that the jitter follows an uncorrelated multivariate normal distribution with mean \(\Theta\) and variance \(\Sigma = \eta \Theta\) for all models, where \(\eta\) is a parameter of our algorithm called jitter factor. From now on, all point sets are normalized ones unless otherwise stated.

Our algorithm is composed of three modules: hypotheses generator, hypotheses validator, and results refiner (cf. Fig. 4). \(M\) containers are used to store correspondences coming from each model respectively.

The hypotheses generator is a two-stage procedure: offline registration (cf. Fig. 3) and online generation. During the offline stage, every model patch is registered in the generator. During the online stage, a scene point \(q\) is randomly chosen and the \(q\)-patch is fed to the hypotheses generator to build several paired-patches. These generated paired-patches are called “hypotheses” in the following. The validator takes each hypothesis as input and verifies them. If a hypothesis is valid, the validator produces a list of correspondences and adds
them to one of the containers. The “generation-validation” process is looped until either one of the two following conditions is satisfied: (1) one container has more than $N_{\text{large}}$ correspondences or (2) $N_{\text{max}}$ scene points have been sent to the generator. We explain these conditions and their values in Section 3.5. Finally, the result refiner finds and improves the final result with the largest correspondences set among $M$ containers (cf. Fig. 4). Note that only the hypotheses generator needs to work on all models whereas the hypotheses validator and the result refiner only use the scene point set $Q$ and the model $P_i$ from which model points of the input hypothesis come.

The time complexity of the method is about $O(m + n)$, where $m = \sum_{i=1}^{M} m_i$ is the number of total model points, and $n$ is the number of scene points. It will be explained in each of the following sections.

3.2 Hypotheses generator

The hypotheses generator is basically a geometric hashing module. During the offline registration, each $p$-patch (i.e. a set of $k + 1$ points) is registered into a hash table with its model number. During online generation, the input $q$-patch is considered as a query set and the corresponding $p$-patches that may belong to different models are retrieved.

We chose to use geometric hashing because it can drastically reduce the search space, it works well with moderate point jitter and in the presence of extra/missing points, and it can deal with many different types of transformations. By restricting it to local patches, the number of points in each $p$-patch is very small so the matching can be processed very quickly.

Geometric hashing uses a local basis. Let $f(T_L)$ be the degree of freedom of the linear transformation $T_L$. By using $b = \text{floor}(f(T_L)/d) + 1$ non-degenerated points, one can construct a local affine right-hand basis $B$ as follows: let the first point be the origin of the local basis, each following point defines the direction of one axis for the local basis. Then the last point’s position can be uniquely expressed by a geometric descriptor $X$, which is a $D = bd - f(T_L)$ dimensional vector and represents the last point coordinates in $B$. The local basis is constructed so that $X$ is invariant to transformation $T_L$. If $X_{B_1}$ represents $X$ in $B_1$ basis formed by $b$ points, we have:

$$X_{B_1} = X_{B_2} \iff \exists! T_L \text{ s.t. } B_2 = T_L(B_1)$$

where $\exists!$ represents the unique existential quantification operator meaning “there is one and only one”.

Jitter can influence the value of $X$ as well. If the geometric descriptor of basis $B$ is $X$, and if $B'$ giving rise to a geometric descriptor $X'$ is a jittered version of $B$, then $X'$ is a random variable. Since $T_L$ is a linear transformation, we can analytically find $\Sigma = \{ \Sigma_1, \Sigma_2, \ldots, \Sigma_D \}$ so that $X' \sim N(X, \Sigma^2)$ when the variance of jitter $\Sigma$ is comparably small to inter-point distances $l$. Note that for simplicity, $\Sigma$ is not a covariance matrix but a vector of length $D$. As a consequence we say that $B$ and $B'$ with descriptors $X$ and $X'$ respectively, are matched under jittered conditions if equation (2) is satisfied:

$$|X_i - X'_i| \leq 2\Sigma_i$$

Given $b$ points, there are many ways to construct a basis. For a $p$-patch, we choose $p$ as the first point (i.e. the origin), other points are chosen so that $X$ lies in a bounded region $R_X = \{ X : |X_i| \leq 1, 1 \leq i \leq D \}$. This region is uniformly partitioned into $100^b$ bins (meaning that $[-1,1]$ in each dimension is equally divided in 100 segments) to serve as hash entries. To avoid inefficient record in the hash table, $\Sigma_i$ is used instead of $\Sigma_i$ in Alg. 1, with $\Sigma_i = \max(\Sigma_i, 0.05)$.

**Algorithm 1 Offline $p$-patch registration for model $P_i$**

```
for $B \in \{$ All combination of $b$ points in $p$-patch containing $p$ $\}$ do
    Compute geometric descriptor $X$ and its variance $\Sigma$
    Register (model number $i$, $B$) into all bins that intersect $X \pm 2\Sigma$
end for
```

The generation step begins by creating a list of $(p_j$-patch, $q$-patch). They are called pre-hypotheses since points correspondences are unknown in these two patches. Each pre-hypothesis has a local voting table for their neighbor point correspondences. Then the voting schema detailed in Alg. 2 is applied. Finally, point correspondences are estimated from voting results, thus giving rise to hypotheses.

Since the generator is a multi-model geometric hashing module, the matching complexity for each $q$-patch is roughly $O(m(k+1)^b)$, where $m$ is the total number of model points, $k$ is the number of nearest neighbors used and $b$ is the number of points in $B$, which only depends on the transformation type.

3.3 Hypotheses validator

The core work of the validator is to check whether the local transformation $T_L$ of an input hypothesis is an approximation of $T$. If a paired-patch contains point correspondences of the real transformation, it is an in-paired-patch and its local transformation is a good approximation of $T$. Local transformations of two neighboring in-paired-patches should be similar as they are both approximations of $T$ in the same neighborhood. Such a relationship cannot be established for out-paired-patches. We use this consensus between neighbor paired-patches’ transformations to validate hypotheses. Obviously, the more
paired-patches share the consensus with $T_L$, the more likely $T_L$ is an approximation of $T$. The measurement of consensus is $T_L$-dependent, it is addressed in concrete implementations, cf. Section 4.

The idea of consensus between neighboring paired-patches was first seen in RRDM [29], however our approach is different. In RRDM, transformations of neighboring paired-patches are first estimated independently before being compared, which consumes a lot of time. Furthermore, with large noise, point correspondences can hardly be found independently between model patches and scene patches. In this method, if the transformation of one paired-patch is found, we use it as an initial guess to estimate transformations of its neighboring paired-patches. This allows us to improve the possibility and the speed of finding similar transformations between neighboring paired-patches. This method, by relying on the consensus and on a proper termination criterion $N_{\text{large}}$, can almost find a correct estimation of $T$, which will be further improved by the refiner.

Alg. 3 describes the validator module. Recall that a paired-patch contains correspondences between two patches, thus it can be seen as a small correspondence list as well. The validator takes an hypothesis paired-patch $(p_0, q_0)$ as input, finds its neighboring paired-patches having a similar local transformation and adds them to supporterlist. Then removes the same thing with these new added paired-patches. This process continues until no paired-patch can be added anymore. The resulting supporterlist is a list of correspondences which are located in the same subregion and share the same transformation $T$. If the local transformation $T_L$ of the input hypothesis is considered a good approximation of $T$, supporterlist is registered into the $i$-th container, $i$ being the model number of point $p_0$. If the $i$-th container is not empty, supporterlist and the stored list are merged to give the largest correspondences list sharing the same $T$.

Algorithm 3 Validator

**Input:** One hypothesis: paired-patch $(p_0, q_0)$

**Output:** A list of correspondences: supporterlist

```
for each $(p, q) \in$ supporterlist do
  $T_L = \text{local transformation of paired-patch } (p, q)$
  for all $(p', q') \in$ paired-patch $(p, q)$ do
    Build paired-patch $(p', q')$ and estimate its local transformation $T_L'$ based on $T_L$ (cf. for details)
    if $T_L \simeq T_L'$ then
      supporterlist $\leftarrow$ supporterlist $\cup \{(p', q')\}$
  end if
end for
```

Step $(\ast)$ of Alg. 3 finds point correspondences between $p$’-patch and $T_{L-1}(q'$-patch) using nearest neighbors. These correspondences lead to an estimation of the local transformation $T_L'$ and are used to create a paired-patch $(p', q')$.

Time complexity of the validator depends on the number of input hypotheses. It is proportional to the number of model points $m$ and to $N_{\text{max}}$. So the time complexity is about $O(mnN_{\text{max}})$.

### 3.4 Result refiner

After loops of “generator-validator”, many correspondences are filled into containers. If the $i$-th container has the most correspondences, then model $P_i$ is considered to be found. The refiner takes the correspondence list in $i$-th container as input, noted as $sl$. It works on the scene point set $Q$ and the model point set $P_i$ to find the transformation $T$. The overall refining process is similar to RRDM [29], but Delaunay triangulations and a threshold $d_t$ are used to improve its performance.

First of all, Delaunay triangulations are generated for both point sets, so that each point is connected with its mesh neighbors by the Delaunay mesh (preferred to nearest neighbors, cf. Section 7.1). From $sl$, a transformation $T$ can be estimated. Then we try to find matching points for all mesh neighbors of points in $sl$. When a new matching is found, the new correspondence is added to $sl$. With these correspondences, the transformation can be better estimated. This process continues until $sl$ stops growing. Note that at the end of Alg. 4, $T$ is the transformation between normalized $P_i$ and $Q$. However, since point correspondence between original sets can be derived from $sl$, it is very easy to calculate the transformation between them.

Algorithm 4 Refiner

**Input:** Largest correspondence list: $sl$

**Output:** Transformation: $T$

```
repeat
  Estimate $T$ using $sl$
  for all $(p, q) \in sl$ do
    Add all mesh neighbors of $q$ in potentialComers
    Add all mesh neighbors of $p$ in tmpSet
    In $Q$, find nearest neighbors of each $T(p_j), p_j \in \text{tmpSet}$
    Add these nearest neighbors into potentialComers
  end for
  for all $q' \in$ potentialComers do
    if $\exists p' \in P_i : |p' - T^{-1}(q')| \leq d_t$ then
      $sl = sl \cup \{(p', q')\}$
    end if
  end for
  Estimate $T$ using new $sl$
  Erase $(p, q) \in sl$ if $|p - T^{-1}(q)| > 3\sigma$
  until $sl$ stops growing
  Return: $T$
```

In Alg. 4, $d_t$ is a threshold which describes how far could a projected scene point $T^{-1}(q')$ be away from its model corresponding point. This value can vary according to the position of $q'$. Let $C$ be the convex hull of scene points in $sl$. If $q'$ is inside the region enclosed by $C$, we can take roughly $d_t = 2\sigma$ thanks to Gaussian distribution. When $q'$ is outside the region, the farther it is from the region, the larger $d_t$ will be (cf. Fig. 5). Assuming $q$ is the center of $C$ and $C$ intersects $qoq'$ at $q_C$, we then take $d_t = 2\frac{\text{area } C}{\text{area } \sigma}$. The number of iterations in the outside loop is linear to $n$ (i.e. the number of points in $Q$) in the worst case. Thus the time complexity is about $O(n)$.

### 3.5 Parameters

All parameters used in the algorithm will be discussed here so that they can be more easily adjusted according to special use.

#### 3.5.1 $k$ and $\eta$

$k$ defines the size of local patches, the bigger it is, the more robust the algorithm is against extra/missing points. Since our method is based on Geometric Hashing, the efficiency related to $k$ in the generator is $O(k^3)$, where $b$ is the number of points in a basis $B$ (cf. Sec. 3.2).
The first termination condition on $P$ is very difficult to obtain even with pre-known point jittered point pattern matching problem since beyond that value, pre-correspondence is from $C$ list in the algorithm is against acquisition noise. But descriptor (cf. Sec. 3.2). We found that $\eta_k$ large the point set in the scene, one needs only to do at most $41$ simple $50\%$ inliers, we have $N_{max}$ where one has $10^{max}$ models to track and each model has $10$ points, an in-paired-patch has a probability of $(\lambda)$ of no result from $T$. This is a direct inspiration from $[11]$. In a conservative estimation (see Section 9 for demonstration)

$$\exists \text{max } |E|$$

is a random set. (i.e. $\eta_i$ has the mean measured $\lambda$ against acquisition noise. But $\eta$ also impacts the generator efficiency at about $O(\eta^p)$, where $D$ is the dimension of geometric descriptor (cf. Sec. 3.2). We found that $\eta \approx 10\%$ is the limit for the jittered point pattern matching problem since beyond that value, precise transformation is difficult to obtain even with pre-known point correspondences (cf. Fig 9). Fortunately, we found $\eta = 5\% - 7\%$ to be sufficient for many applications.

### 3.5.2 $N_{large}$ termination rule

The first termination condition on $N_{large}$ means that we are almost sure to have found the model so there is no need to test more hypotheses. A good choice should give very few false alerts. Given a model $P_i$ and the scene $Q$, let us consider those two random events $E_0$ and $E_1$:

$$E_0 = \{Q \text{ is a random set. (i.e. } Q \text{ does not come from } P_i)\}$$

$$E_1 = \exists N_{large} \text{ correspondences in a local region agreeing on } T$$

Reducing false alerts means reducing the probability of $E_0$ knowing $E_1$, i.e. $P(E_0|E_1)$. We can show that $P(E_0|E_1) \approx 0$ with a very small $N_{large}$. In a conservative estimation (see Section 9 for demonstration) where one has $10^6$ models to track and each model has $10^6$ points, $P(E_0|E_1)$ is smaller than $10^{-6}$ with $N_{large} = 20$. However, this result cannot guarantee that the method can achieve an accurate estimation of $T$ when a model is present, because the refiner may find false inliers.

### 3.5.3 $N_{max}$ termination rules

The second condition on $N_{max}$ says that we are almost sure that no model exists for the point set $Q$. This is a direct inspiration from RANSAC. If $Q$ has $t$ inliers, an in-paired-patch has a probability of $p_1$ chance to be proposed to the validator and a probability of $p_2$ to pass the validation phase, then the probability ($\lambda$) of no result from the validator after trying $N$ image points is:

$$\lambda = (1 - \frac{t}{n}p_1p_2)^N$$

So, if we want $\lambda < \lambda_{max}$, we have:

$$N_{max} = \log(\lambda_{max}) / \log(1 - \frac{t}{n}p_1p_2)$$

From our experiences, $p_1$ can be small in very noisy situations but $p_2$ is very large, let us say $p_1 = 0.2$ and $p_2 = 0.7$. In a scene with $50\%$ inliers, we have $N_{max} = 41$ with $\lambda_{max} = 0.05$. So no matter how large the point set in the scene, one needs only to do at most $41$ simple geometric matching queries with $k+1$ points. In all our experiments, $N_{max}$ is set to be $45$. When scene set contains more outliers, one can adjust $N_{max}$ to a more proper value according to equation (4).

### 4 Specific Implementations

In this section, we present the detailed implementation of the algorithm for 2D homography and 3D similarity. 2D homography is used to match coplanar point sets (markers) while 3D similarity matching is very useful for model registration. Given the general algorithm, one needs three more pieces of information for a detailed implementation:

(a) How to construct the $b$-basis and compute the geometric descriptor
(b) How to measure the similarity between $T_i$s.
(c) How to find $T_i$ or $T$ with known point correspondences. We detail them for both cases.

### 4.1 2D homography

A 2D homography is basically a perspective transformation, and each point has only two coordinates. So $T$ belongs to 2D perspective transformation and $T_L$ belongs to 2D affine transformations.

(a) $b$-basis is ordered as $(p_0, p_1, p_2, p_3)$ so that $\Delta p_0 p_1 p_2 p_3$ has the largest surface among all triangles containing $p_0$, $p_0 p_1 \times p_0 p_2 > 0$. This gives us a local right-hand basis and $|X_1|, |X_2| \leq 1$ (cf. Fig. 6), same as in $[11]$.

Thus, if $p^{(i)}$ stands for the $i$-th coordinate of point $p$:

$$\begin{pmatrix} X_1 \\ X_2 \\ 1 \end{pmatrix} = A \begin{pmatrix} p_1^{(i)} \\ p_2^{(i)} \\ p_3^{(i)} \end{pmatrix}$$

$$A = (a_{ij}) = \begin{pmatrix} p_1^{(1)} - p_0^{(1)} & p_1^{(2)} - p_0^{(2)} & p_1^{(3)} - p_0^{(3)} \\ p_2^{(1)} - p_0^{(1)} & p_2^{(2)} - p_0^{(2)} & p_2^{(3)} - p_0^{(3)} \end{pmatrix}^{-1}$$

$$\Sigma_1 = \Sigma_2 = (X_1 + X_2 - 1)^2 + X_1^2 + X_2^2 + 1)\sigma (\alpha_1^2 + \alpha_2^2)$$

(b) 2D affinity is a $2 \times 3$ matrix. The left $2 \times 2$ square matrix can have a singular decomposition, where a rotation $\theta$ and two scale factors $\alpha(1), \alpha(2)$ can be retrieved. By using the result from $[29]$, we say two affine transformation are similar if the difference of their rotation is less than $10^\circ$ and the ratio of each scaling is less than $1.3$.

(c) With known point correspondences, $T_L$ can be found by the least square method while $T$ can be found in various ways including OpenCV’s $\text{findHomography}$ method.

### 4.2 3D similarity

Similarity is a linear transformation itself, so $T_L = T$.

(a) $b$-basis is ordered as $(p_0, p_1, p_2)$ so that $|p_0 p_1 p_2|$ defines the positive direction of second axis of this local basis (cf. Fig. 7). Thus the coordinates of $p_2$ are:

$$X_1 = \frac{p_0 p_2 - p_0 p_1}{\|p_0 p_1\|^2}, X_2 = \sqrt{\|p_0 p_2\|^2 - \|p_0 p_1\|^2} - X_1, X_3 = 0$$

It is easy to verify that $|X_1|, |X_2| \leq 1$ and thus the variance:

$$\Sigma_1 = \Sigma_2 = \sqrt{\|p_0 p_2\|^2 + \|p_0 p_1\|^2 - \|p_0 p_1\|^2}, \Sigma_3 = 0$$

(b) A 3D similarity can be decomposed into a rotation matrix $R$ and a scale $\alpha$. $R$ can be represented as a quaternion $q$. We say two similarities are equal if the difference of rotation is less than $10^\circ$ measured by $\Phi_3$ mentioned in $[10]$ and the ratio of scales is less than $1.3$, that is:
Fig. 7. Local $b$-basis $(p_0, p_1, p_2)$ for 3D similarity. $p_0$ is the origin. $\hat{x}_i$ are axis of the local coordinate system, with $\hat{x}_i \perp \hat{x}_j$, ($i \neq j$). $X_i$ are the coordinates of $p_2$ under this local coordinate system, with $X_i \equiv 0$.

\[
2 \arccos(\|q_0 \cdot q_1\|) \leq 10^\circ, \max(\|\alpha_1\|, |\alpha_2|) \leq 1.3
\]  

(8)

(c) We use the method from [27] to estimate the similarity $T$ with known point correspondences.

5 RESULTS ON SYNTHETIC POINT SETS

In this section, we present results obtained under different conditions. Section 5.1 focuses on performance in noisy conditions. While this first section takes the 2D homography transformation as a basis and performs extensive tests, Sections 5.2 and 5.3 show that the algorithm has the potential to work well in 3D and can benefit from additional information. All experiments are performed on a PC with an Intel Xeon E3 1240@3.40GHz CPU and 16GB of RAM.

5.1 Speed and robustness study

The performance of our method is compared with other methods using synthetic data. In the graphs, RDM stands for Uchiyama’s Random Dot Markers [25] (original implementation is used), RRDM for our previous work [29], LGC for our current proposal and GH for traditional Geometric Hashing [28]. Default parameters of these methods are used. The homography transformation is set to a 30° perspective transform unless stated otherwise. Fig. 8 to Fig. 13 present single model matching, i.e. the scene set $Q$ is matched against only one model set $P$, while Fig. 14 shows multi-model matching. We use formula (14) in [29] to judge whether reprojections are precise.

We first investigate the performance with different number of points in the model set. Fig. 8 first shows the result in an ideal condition, that is without jitter, extra or missing points and then the result with $\beta = 15\%$ extra points and $\eta = 3\%$. The second case is more realistic and can be seen as a simulation of a real scene. As shown in the figures, GH is too slow to provide results for more than 60 points. RRDM has a clear quadratic behavior. RDM works well in the ideal condition but finds too few precise results in the second case. LGC achieves the best performance with a linear behavior.

Fig. 9 shows results of robustness studies. We increase the jitter factor $\eta$ to study robustness to acquisition noise which can be due to image processing artifacts, camera calibration errors, etc. Here, we stick to point sets containing 100 points with no extra or missing points. Clearly, RDM can handle some noise, but is outperformed by both RRDM and LGC. LGC is the fastest and most robust. KPC represents results directly found with pre-Known Point Correspondences, which can be seen as the result obtained using ground-truth point correspondences. Note that $\eta = 10\%$ is about the limit of achieving a precise matching even with ground-truth point correspondences for jitered point pattern matching. LGC sometimes outperforms KPC (cf. $\eta = 7\%$ and $8\%$), because LGC does not keep correspondences when the reprojection distance between the scene and the model points is greater than $3\sigma$ (cf. Alg. 4). Thus LGC may use less noisy point correspondences than KPC thus leading to a more accurate homography.

Then, we present results with extra points (such as outliers that could be detected by image processing techniques) in Fig. 10, results with randomly removed points (which simulates underdetections) in Fig. 11, results when points in a region are masked (i.e. occlusion) in
5.2 Test: 3D model registration

Most 3D models are composed of 3D points, forming small surfaces which represent their envelope. We show in this section that our method implemented in 3D, allows for registering models undergoing similarity transformations. Since, 3D models are often made of dense point clouds which may contain thousands or millions of points, it is almost impossible to register such amount of points directly. In order to reduce the number of points that need to be matched, we use 3D interest point detectors on the 2 models to be matched to find geometrical keypoints which are fed to our algorithm to find the similarity transformation between them. We only use point coordinates for matching, neither color information nor normal direction is used. For illustration purpose, we use the famous Stanford Bunny with Zhong’s Intrinsic Shape Signatures [30] (ISS) since it has a good repeatability [22, 6]. η is set to 3%. We first create the model point set with ISS key points: for the 35947 Bunny points, we obtain 480 keypoints. Then, random similarity transformations are applied on the bunny model to create 5 instances for matching. Results of registration are listed in Table 1. We can conclude that LGC has a potential to be also used for 3D similarities estimation.

5.3 Test: LGC with additional information

So far, we have only considered pure point matching problems where each point is indistinguishable from another. However, points may have their own properties including color or local feature in textures. We show here that such properties can be used in our algorithm.

Some traditional feature point matching methods (e.g. SIFT, SURF or BRIEF) are based on these different properties of points. They first find rough correspondences by using point properties before applying RANSAC-like methods. Since such methods need high inlier ratios, the rough correspondences should contain a lot of inliers. This implies that point properties should be discriminant enough, which results in high dimensional descriptors. We show that using our method instead of RANSAC greatly reduces the descriptors’ dimension.

In our experiment we use ORB [19]. Since its descriptor is composed of a bit sequence generated from random pixel comparisons, it is very easy to get lower dimension descriptors from the original ones. Image points are no longer fed to the generator randomly but according to their nearest Hamming distances to model points. Let the length

Fig. 12. Occlusion experiment (model P contains 100 points). β=0%, η=3%, missing=0%, perspective=30°.

Table 1. Registration results

<table>
<thead>
<tr>
<th>Instance ID</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key point size</td>
<td>485</td>
<td>485</td>
<td>479</td>
<td>488</td>
<td>491</td>
</tr>
<tr>
<td>Inliers found</td>
<td>459</td>
<td>455</td>
<td>447</td>
<td>457</td>
<td>439</td>
</tr>
<tr>
<td>Rotation difference (°)</td>
<td>0.10</td>
<td>0.06</td>
<td>0.00</td>
<td>0.10</td>
<td>0.10</td>
</tr>
<tr>
<td>Scale difference (%)</td>
<td>0.09</td>
<td>0.04</td>
<td>0.05</td>
<td>0.05</td>
<td>0.09</td>
</tr>
</tbody>
</table>
Fig. 14. Discrimination performance. Top: with different number of models. Each model contains 100 points. Bottom: with different jitter factor. 50 models are used with 100 points in each model. Full boxes stand for success, empty boxes for good model found with wrong transform.

of the descriptor in bits be $L_{ORB}$. In the generator, we do not process $(p_j$-patch, $q$-patch) which Hamming distance is larger than $L_{ORB}/2$ in Alg. 2, thus reducing the number of false hypotheses. The condition in Alg. 4 is also relaxed to find more corresponding points: we add $(p'_i, q'_i)$ to the solution list if the Hamming distance between $p'_i$ and $q'_i$ is less than $L_{ORB}/3$ and if $|p'_i - \mathbf{T}^{-1}(q'_i)| \leq 3d_i$. We use these values for illustration purpose, they are not optimized.

The original 32 bytes ORB are chopped to have lower dimensional descriptors which are used to compare our method with USAC [18]. Results are presented in Fig. 15 where we can see that LGC works with much smaller descriptors than USAC.

6 Applications

6.1 Tracking ordinary planar objects

Corners in real textured images are assumed to be randomly distributed. As demonstrated in [24], these corners can be used for tracking. We compare our method with RDM and SURF in this section on real planar targets. However, our purpose is not to compete with textural-based trackers in these situations. SURF serves as a reference. We aim at illustrating the flexibility of our approach to deal with traditional targets and showing its robustness on real targets.

We use the OpenCV “goodFeaturesToTrack” method [21] as interest point detector in this experiment which performs at about 10ms/frame. maxCorners and minDistance parameters are selected in favor of RDM according to [24] (qualityLevel=0.13 and blockSize=12 for stable point detection). All remaining parameters are set to their default values. As to SURF, a FLANN based matcher in OpenCV is used to build a library of descriptors for tracking.

We use a Logitech C270 camera with resolution set to $640 \times 480$.

The experiment is as follows: first eight different coplanar objects (cf. Fig. 16) pass individually in front of the camera. The same top view of each object is registered using three algorithms (RDM, LGC and SURF). Then, each object is tracked by the three algorithms separately.

Similar experiments are repeated twice for a total of 1753 frames and visual assessments are provided for all estimated homographies. Both SURF and LGC match at about 15-18 ms/frame while RDM matches at around 50 ms/frame. Fig. 16 shows a summary of the results for each model during tracking. LGC outperforms RDM in all cases. Although SURF performs better than LGC in general, it works badly on crossword and random dot since these two models contain less textures. As RDM can be used for “augmenting everything”, we claim our method is a better alternative to RDM for this purpose.

6.2 Augmenting engineering drawings

Engineering drawings are largely used in mechanical engineering and architectural design. They are different from ordinary texture-rich models because they are most of the time 2D representations that contain only geometric information (such as straight lines and circles) of 3D CAD models. Traditional texture-based key point tracking methods cannot deal with such cases. Fig. 17 presents results of augmenting engineering drawings with their 3D models. We extract in real-time the intersections of the drawing which are mapped to previously created model point sets.

7 Discussion

In this section, we discuss some aspects of the algorithm.

7.1 Neighbors

We use nearest neighbors to create local patches but use mesh neighbors in the refiner. We tried to use the same type of neighbors in the algorithm but neither performs better. Although mesh neighbors are more robust against perspective distortions [13], they are very sensitive to extra/missing points, leading to a less robust generator. On the other hand, when points are gathered into two or more local groups, nearest neighbors may give rise to “important edges” in the resulting network (cf. Fig. 18). If these edges disappear due to extra/missing points, two subregions are disconnected. Thus by visiting nearest neighbors in the refiner, inliers are often confused to a small subregion, which gives an imprecise estimation of the transformation.
7.2 Transformation $T$

Theoretically, the algorithm works with any transformation $T$, but some conditions on point sets have to be satisfied.

We assume that $T$ can be expanded by the Taylor theorem in the neighborhood of $u_0$:

$$T(u) = T(u_0) + \frac{\partial T(u_0)}{\partial u} \Delta u + \frac{\partial^2 T(u_0 + \lambda \Delta u)}{\partial u^2} (\Delta u)^2$$  \hspace{1cm} (9)

with $\Delta u = u - u_0$ and $\lambda \in [0, 1]$. The first two terms on the right are the linear local transformation $T_L(u)$ that we mentioned before, the last term is a small quantity of the same order as $(\Delta u)^2$.

If the difference between $T(u)$ and $T_L(u)$ is not very important, the last term can be easily managed by considering that it represents the “jitter”. The difference can be expressed as:

$$\| T(u) - T_L(u) \| \leq \| \frac{\partial^2 T(u_0 + \lambda \Delta u)}{\partial u^2} \| l^2$$  \hspace{1cm} (10)

$\| \frac{\partial^2 T(u_0 + \lambda \Delta u)}{\partial u^2} \|$ depends only on transformation $T$, not on inter-point distance $l$. So if the inter-point distance $l$ is small enough, this quantity can be smaller than $0.1l$ to be managed by the algorithm.

7.3 Repetitive structures

As our method relies on discriminating local geometric structures, repetitive patterns are an issue. They impact both the “generator” and the “validator” since they work on local patterns, but do not affect the “refiner”. If the hypothesis (cf. Sec. 3.3) is generated inside a repetitive pattern, the algorithm will probably give a false result; otherwise both “generator” and “validator” have enough discriminant information to estimate a true correspondence. As the generator selects scene points randomly, the algorithm’s performance is roughly proportional to the percentage of points in repetitive patterns. Our experiments show that the algorithm fails with exclusively regular patterns such as a chessboard, but can successfully handle small repetitive structures inside a more global irregular one, indeed only 2 frames out of 317 fail in the “office design” case (cf. Fig. 19). In the latter example, we rely on [17] to extract intersections and junctions.

8 Conclusion

In this paper, we have presented LGC, an algorithm based on Local Geometric Consensus that can be used for several transformation types, both in 2D and 3D. In its original version, it only uses the spatial distribution of points that we can find in randomly distributed texture-less point sets but it can also be adapted to textured images by using local features information when available. Therefore, our algorithm could be used to augment everything as claimed in [24]. We have experimentally shown that LGC outperforms both RDM [24] and our previous work [29] as far as robustness and speed are concerned. Even though LGC handles small regular patterns, its performance degrades while these patterns go larger. Further work will imply going beyond those limitations and extensive testing of the 3D case.
To have a conservative estimation, we want to find the upper bound of \( P(E_0 | E_1) \), if this upper bound is small enough, it is less likely to have false alerts. \( P(E_1 | E_0) \) and \( P(E_1 | E_0) \) are two events independent of \( P(E_0) \). So \( P(E_0 | E_1) \) will be larger if \( P(E_0) \) is smaller. \( P(E_0) \) can be seen as the probability that model \( T \) appears in the scene. When there are more models to track, \( P(E_0) \) will be smaller. Considering we have \( 10^8 \) models to track, which is a huge number, the probability that \( T \) appears in the scene is roughly \( 10^{-4} \). In equation (11), \( P(E_1 | E_0) \) is very large, say \( P(E_1 | E_0) \approx 1 \). We have to find \( P(E_1 | E_0) \).

Before solving \( P(E_1 | E_0) \), let us consider a simpler question: in a \( N_b \) random binary string where 1 appears at each bit with probability \( p_b \), how long is the largest sequence containing only 1s? Schilling et al. [20] show that the expectation and variance of \( L_b \) can be approximated by the following equations:

\[
E(L_b) = \log_1/p_b (N_b q_b) \quad \text{and} \quad \Sigma(L_b) = \frac{\pi^2}{6} \ln(1/p_b) \tag{12}
\]

where \( q_b = 1 - p_b \). \( E(L_b) \) is proportional to \( \log N_b \), it varies little with the length of the sequence. \( L_b \) is independent of \( N_b \).

Event \( E_1 | E_0 \) and “the largest sequence containing only 1s in a random binary string” have the same nature, since both of them describe the possibility of several individual random events having the same result being geometrically agglomerated. However, we do not have exactly the same situation since all correspondences in this local region are not inliers. Equations (12) can only give a rough estimation.

For a given transformation \( T \), a point correspondence is considered coherent with \( T \) if the scene point is projected within a \( \Theta \) radius neighborhood of the model point (cf. Alg. 4). If the scene point has a random position, the probability of the coherence is about \( (3 \Theta / l)^2 \). Remember that \( l \) is the “inter-point” distance defined in Section 3.1. We choose the limit of jittered point pattern matching, \( \sigma = 0.1l \) (cf. Section 3.5). If at least every two correspondences contain one inlier and \( d = 2 \), which is a conservative estimation, the results in \( p_b \approx 0.2 \). Let \( N_b = 10^6 \) points, we have \( E(L_b) \approx 8.4 \) and \( \Sigma(L_b) \approx 0.8 \).

If the transformation \( T \) needs \( B_T \) points as basis which gives us \( B_T \) “free” points in \( N_{\text{large}} \), we have:

\[
P(E_1 | E_0) = P(L_b \geq N_{\text{large}} - B_T)
\]

Take as example a 2D homography (i.e. \( B_T = 4 \)), \( P(E_1 | E_0) = P(L_b \geq E(L_b) + 9.52 \Sigma(L_b)) \leq 10^{-10} \) by setting \( N_{\text{large}} = 20 \). Thus \( P(E_0 | E_1) \approx 10^{-6} \) and it is a conservative upper bound estimation.

REFERENCES
