Real-time Incremental J-linkage for Robust Multiple Structures Estimation

Roberto Toldo and Andrea Fusiello
Department of Computer Science - University of Verona
Strada le grazie 15, Verona - Italy
{roberto.toldo | andrea.fusiello}@univr.it

Abstract

This paper describes an incremental, real-time implementation of J-linkage, a procedure that can detect multiple instances of a model from data corrupted by noise and outliers. The method is incremental, as it exploits the information extracted in the previous steps and processes the data as they become available. It works in real-time, thanks to several approximations that have been introduced to get around the quadratic complexity of the original algorithm. Tests have been carried out both with synthetic data and real data.

1. Introduction

The problem of estimating the parameters of a mathematical model from a set of observed data which contains outliers is very frequent and well studied in Computer Vision. The RANSAC [7] algorithm is the most common solution, both for its capability to handle an high fraction of outliers and for its simplicity of implementation. In the recent past, some efforts have been devoted – separately – to enhance the efficiency for real-time application and to improve the robustness against multiple models. However, no endeavor have been made to cope with both issues simultaneously.

As for the multiple models issue, this work builds on the J-linkage [17] approach, which proved to be very effective, but whose complexity is quadratic on the number of points, thus making it unsuitable for real-time. The aim of this work is to reduce the complexity of the J-linkage method in order to be able to run it with real-time applications. Furthermore, the proposed algorithm is incremental, for it processes the data as it arrives in time and exploits the information collected in the previous steps. To the best of our knowledge, this is the first algorithm specifically tailored for robust multiple models estimation that can run in a very time efficient way.

The paper is organized as follows. In Sec 2 a brief survey of the state of the art will be carried out. In Sec. 3 we will shortly introduce the original J-linkage algorithm. In Sec. 4 we will describe the amendments made to the method to decrease its time of execution and make it incremental. In Sec. 5 the results with both synthetic and real tests are shown, while in Sec. 6 conclusions are drawn.

2. Related work

The recent development of computer hardware have made it possible to perform an increasing number of tasks in real-time. Among the latter, robust model fitting is often a crucial operation. As a consequence, a number of efforts have been made in order to increase the efficiency of the standard RANSAC. Some improvements exploit the idea that bad hypotheses can be discarded in an early stage, without the need of verifying them against all data points. In [2] a model is preliminarily tested using a subset of d randomly selected points. The remaining set of points is evaluated only if the first d points are inliers to the generated hypothesis. This pre-verification test allows a boost in the efficiency even if more hypotheses need to be generated.

Further exploiting this idea, Capel [1] proposed a formula to approximate the probability that the current generated hypothesis, evaluated only on a subset of points, is better than the best generated model in the previous steps. Most recently, [4, 11] described an optimal randomized verification strategy based on the theory of sequential decision making.

Sometimes a similarity function between points is available, such as in the case of correspondences between two or more images, or can be estimated. The Progressive Sample Consensus (PROSAC) algorithm [3] is designed to use this similarity score and generate the hypothesis with high similarity first. Assuming that points with high similarity are more likely to be inliers than points with low similarity, the good hypotheses should be generated first.

The problem of obtaining the best guess in a fixed amount of time has been faced for the first time by the preemptive RANSAC procedure [13]. A fixed number of hypotheses are generated and compared against each other in parallel. Using a *breadth-first* approach the hypothesis are

tested on a subset and only a fraction of them are evaluated on the next subset.

A combination of techniques based on pre-verification are used to evaluate a small subset of points giving an estimation of the inlier fraction. This preemption scheme is the basis of the recent Adaptive Real-Time Random Sample Consensus (ARRSAC) algorithm [14].

Real-time applications usually also requires on-line processing, meaning that data is not available as a batch but it arrives progressively. Thus, a good algorithm not only needs to be fast, but also *incremental*, i.e., it must process the data as they become available, exploiting the results of the previous steps. In [16] an incremental scheme is applied to the Preemptive RANSAC in a vehicle relocation problem. The new features are used to generate new hypotheses and updated accordingly with the preemption scheme.

If multiple instances of the same structure are present in the scene, the robust estimator must tolerate both gross outliers and *pseudo-outliers*. The latter are defined in [15] as "outliers to the structure of interest but inliers to a different structure". Algorithms designed for the single structure task (like RANSAC) fail in this case.

A naive approach to multiple models fitting would be to sequentially apply RANSAC and remove the inliers from the data set as each model instance is detected. This has been proven to be non optimal. The multiRANSAC algorithm [19] define a RANSAC procedure that detects a known number of models in a parallel fashion. The method is effective, but fails when multiple intersecting models are present and requires the number of models to be known beforehand.

Some methods exploit the fact that random sampling yields clusters of models around the real ones in the parameter space. A popular method of this class is the Randomized Hough Transform (RHT) [18]. A histogram is built over the parameter space and filled with the votes of random sampled models. Peaks in the histogram correspond to emerging models. Despite its intuitiveness, the method has limited accuracy and low computational efficiency. Additionally, defining a proper discretization of the parameter space is not trivial. Some of the drawbacks of RHT can be overcome if mean-shift [5] is applied to find density peaks in parameters space.

In [17] random sampling is used to produce a *conceptual representation* of points that are subsequently clustered with a tailored procedure, called J-linkage. This method has been proved to be very effective. However, in its original formulation, the complexity is quadratic on the number of points, thus making it unusable for real-time applications.

3. J-linkage algorithm

In this section the original J-linkage algorithm will be described and analyzed in terms of computational complexity. For further details please refer to [17].

The first step, as in RANSAC, consists in random sampling, i.e. randomly choose M minimal sample set to estimate the model hypothesis. As pointed out in [12, 8, 19], if the assumption that inliers tend to be closer to one another than outliers is verified, the sampling strategy may be modified. J-linkage selects all but the first points of each sample following an exponential probability. Namely, if a point $\mathbf{x}_i$ has already been selected, then $\mathbf{x}_j$ has the following probability of being drawn:

$$P(\mathbf{x}_j|\mathbf{x}_i) = \begin{cases} \frac{1}{Z}\exp -\frac{||\mathbf{x}_j-\mathbf{x}_i||^2}{\sigma^2} & \text{if } \mathbf{x}_j \neq \mathbf{x}_i, \\ 0 & \text{if } \mathbf{x}_j = \mathbf{x}_i. \end{cases} \quad (1)$$

where Z is a normalization constant and σ is chosen heuristically. This requires $O(n^2)$ time, where n is the number of points, since the pairwise Euclidean distance between all the points must be computed.

When the pool of hypothesis has been generated, the *preference set* (PS) of each point can be computed, i.e., the set of hypothesis a point has given consensus to, or – in other words – the set of hypothesis it *prefers*. Each point will be represented by its PS. The key observation is that points belonging to the same structure have similar PSs, hence they are close in the *conceptual space* $\{0,1\}^M$. Therefore, J-linkage uses an agglomerative clustering procedure to find models. At the beginning every point belongs to a separate cluster. At each step the two clusters with the minimum pairwise distance are merged. The preference set of a merged cluster is equal to the intersection of the original clusters. The distance between two clusters is defined as the *Jaccard distance* between their preference sets: Given two sets A and B, the Jaccard distance is

$$d_J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}. \quad (2)$$

The Jaccard distance ranges from 0 (identical sets) to 1 (disjoint sets). The *cut-off* value is set to 1, which means that the algorithm will only link together elements whose preference sets overlap. Outliers emerge as small clusters. The general agglomerative clustering requires $O(n^2)$ time for construction and $O(n^2)$ time to compute the clusterization with an underlying heap data structure.

4. Real-time J-linkage

The original J-linkage algorithm works in batch, i.e., it processes all the data at once, and its computational complexity does not suite real-time constraints. This section will describe how we modified it in order to make it work *on-line* in *real-time*. This means that the data becomes available in pieces at each time step, and that it must be processed to yield partial results before the next time step.

This new *Real-time J-linkage* is *incremental* (it processes the data as it becomes available and reuses the information extracted in the previous time step) and improves the *time efficiency* of J-linkage.

4.1. Random sampling

A non-uniform sampling strategy for the points of the sample other than the first, as explained in Section 3, is important in most of the problems. However, a continuous probability function defined over the distances between all the points is too expensive to compute for a real-time application. We propose to simplify Eq. 1 so that the probability of choosing a point $\mathbf{x}_j$ depends on whether or not it is in the k-neighborhood of the already selected point $\mathbf{x}_i$, namely:

$$P(\mathbf{x}_j|\mathbf{x}_i) = \begin{cases} P_h & \text{if } \mathbf{x}_j \in k\text{−Neighborhood}(\mathbf{x}_i) \\ P_l & \text{if } \mathbf{x}_j \notin k\text{−Neighborhood}(\mathbf{x}_i) \end{cases} \quad (3)$$

with $P_h >> P_l$. The k-neighborhood can be computed efficiently by storing the data points in a KD-tree data structure. Inserting or removing a node in the tree requires $O(\log(n))$ time, while querying requires $O(\log(n^{\frac{2}{3}} + k))$ time in the case of three-dimensional data.

4.2. On-line processing

The Real-time J-linkage performs an agglomerative clustering every time period. The time between two consecutive time steps is spent in three activities, as illustrated in Fig. 1: the agglomerative clustering, the insertion/removal of new/old data points and the update of the hypotheses pool. In the following of this section we will describe these three stages.

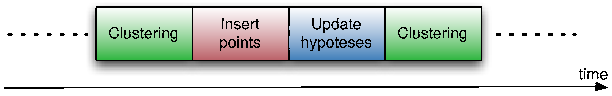


Figure 1. The on-line processing time is spent in three stages: clustering step, point insertion and update of hypotheses.

4.2.1 Clustering

The agglomerative clustering consists in iteratively choosing the closest two elements to be merged until the smallest Jaccard distance is 1, as already explained in Sec. 3. The bottleneck is determining – at each iteration – the two closest elements. At a first glance, storing the pairwise distances in a heap seems the best choice, since the structure can be built in $O(n)$ time and the minimum key can be extracted in constant time. In our application, however, the keys are updated at every step, or eliminated (when the Jaccard distance is equal to 1) and each of these operations would cost $O(\log n)$ time. In contrast, a double linked ordered list requires $O(n \log(n))$ time to be constructed, but removing an element from the list can be performed in constant time. Every time a distance update occurs the list should be sorted again, but a lazy update approach can be implemented. The minimum updated distance is kept and compared against the first element of the list: only when the latter is greater the sorting takes place, and this occurs a very few times during the clusterization, because distances are usually incremented during the updates. For these reasons, we employed a double linked list.

Furthermore, an approximation is introduced that stems from the same observation underlying the local random sampling: inliers tend to be closer to one another than outliers. Exploiting this fact, one can conjecture that the distance of two points in the conceptual space tends to be lower if the points are closer in the Euclidean space. Thus, the algorithm does not calculate the pairwise distance between all the points in conceptual space, but only for points in the Euclidean k-neighborhood. The latter is readily available from the KD-tree that was built previously. When two clusters are merged together, the neighborhoods of the points are merged accordingly: The k-neighborhood of a cluster is defined as the union of the k-neighborhoods of its members. (Fig. 2).

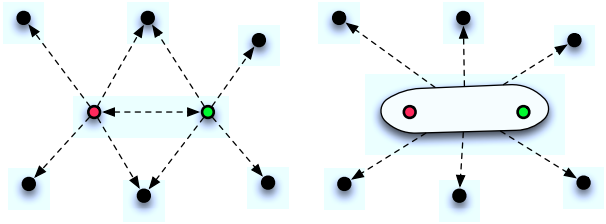


Figure 2. When the red and green points (left) are merged in a cluster, the k-neighborhood of the resulting cluster (right) is the union of the k-neighborhoods of the two points.

4.2.2 Point insertion

In an incremental framework not only points are processed (i.e., clustered) as soon as they become available, but they are also discarded when a certain event occurs (e.g. a buffer size is reached, a fixed time is elapsed, or the point is not visible any more).

In principle, when a point is inserted the Real-time J-linkage needs to compute its preference set and add it to the agglomerative clustering as a singleton. In order to satisfy the real-time constraint, incoming points are stored in a FIFO buffer and fetched for insertion in batches of constant size.

When a point is removed, instead, the algorithm updates the preference set of the cluster it belonged to (which is the intersection of the preference sets of the points that belong to the cluster).

4.2.3 Update of the hypotheses

Whereas in the batch version of the J-linkage all the hypotheses are generated at once, in this on-line version a fixed number of hypotheses is generated at each time step and added to the pool, replacing the oldest ones. When a new hypothesis is added, the preference set of all the points must be updated accordingly, and the preference sets of the clusters must be updated as well.

As a consequence, a cluster may broken apart during this operation, due to the fact that the intersection of the preference sets may be empty. In this case the points that belonged to the clusters are re-inserted as singletons.

Then the agglomerative clustering step is run again on the updated set of clusters.

Algorithm 1 REAL-TIME J-LINKAGE

Input: points stored in a queue Q.
Output: detected geometric primitives R.

1. Fetch n points from Q and add them to the active points set X and to the clustering C as singletons.
2. Generate k hypotesis by randomly sampling the points in X and add them to the hypotesis pool H, thereby substituting the k oldest hypotesis.
3. Update the preference set of each cluster in C.
4. Perform agglomerative clustering on C (Sec. 4.2.1).
5. Fit a geometric primitive to each cluster of C and output them.
6. Remove from X and C points that are no longer visible.
7. Goto step 1.

5. Experiments

We tested both the time efficiency and the accuracy of the proposed Real-time J-linkage on the task of 3D planes extraction (data are 3D points and models are planes).

We run three categories of experiments: simulation with synthetic data, simulation with real data coming from a batch structure-and-motion pipeline and experiments with PTAM[1], a real-time software for tracking and mapping in small environments [9, 10].

[1]Freely available at http://www.robots.ox.ac.uk/ gk/PTAM/

The algorithm have been coded C++ code and carefully optimized. The bitwise operations are carried out using the BitMagic library[2], that implements several performance optimization for Intel platforms.

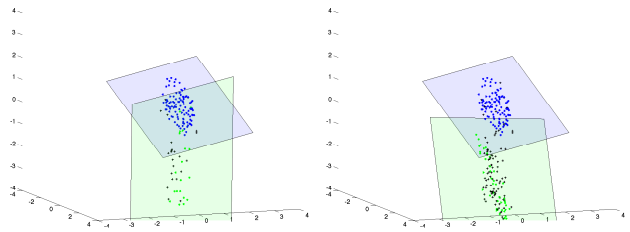
5.1. Synthetic data

Two examples have been synthetically generated. In the first one, two planes are present, each one composed of 100 points corrupted by a small amount of Gaussian noise. Additionally, 50 pure outliers were introduced. Real-time J-linkage was fed with 10 points and generated 1000 hypothesis per time step. The original J-linkage was repeatedly run on the partial data.

Two sample results are shown in Fig. 3, while a comparison with the original J-linkage algorithm in terms of accuracy and time is reported in Tab. 1. The *accuracy* is defined by considering this as the problem of classifying inliers vs outliers, so it is (# true positives + # true negatives) / # of points.

In the second example, four planes have been generated, each one composed of 100 points corrupted by a small amount of Gaussian noise and 50 pure outliers withal. Real-time J-linkage was fed with 10 points and generated 2000 hypothesis per time step. The size of the hypothesis pool was 10000 for both examples.

Two sample results are shown in Fig. 4, while a comparison with the original J-linkage algorithm in terms of accuracy and time is reported in Tab. 2. The reader might notice how the loss in accuracy with respect to J-linkage is balanced out by a pronounced speed-up in execution time.



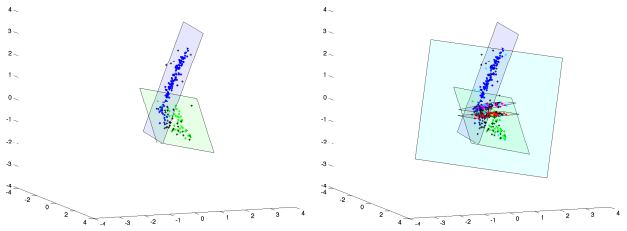
(a) Frame 15. (b) Frame 25.

Figure 3. Planes extracted at different times in the 2-planes example.

Table 1. Comparison of J-linkage (JL) vs Real-time J-linkage (RTJL) in the 4-planes example.

	Time [s]	Average fps	Speed up	Accuracy
JL	70.1	0.35	1	1.0
RTJL	3.1	8	22.61	0.72

[2]Freely downloadable from http://bmagic.sourceforge.net



(a) Frame 20. (b) Frame 45.

Figure 4. Planes extracted at different times in the 4-planes example.

Table 2. Comparison of J-linkage (JL) vs Real-time J-linkage (RTJL) in the 4-planes example.

	Time [s]	Average fps	Speed up	Accuracy
JL	459.85	0.097	1	1.0
RTJL	8.82	5.1	52.13	0.7125

5.2. Real data from SaM

We tested Real-time J-linkage with data coming from SAMANTHA, a batch Structure-and-Motion (SaM) pipeline [6, **?**]. In order to simulate a real-time setting, points were stored in a queue from which Real-time J-linkage fetched subsets of ten at each time step.

Two data-set have been tested: *Castelvecchio* and *Valbonne*. The first one is composed of 871 points, and it took 23.36 seconds to run all the 88 steps. Two results are shown in Fig. 5. The second example is composed of 673 points and it took 14.69 seconds to run all the 63 steps. Two sample results are shown in Fig. 6.

(a) Frame 15. (b) Frame 88.

Figure 5. Extracted planes at different times from *Castelvecchio* data-set. Different colors encode different planes.

5.3. Real data from PTAM

In order to test the proposed algorithm in a real environment, we plugged it to the output of the PTAM (Parallel Tracking and Mapping for Small Augmented Reality Workspaces) software. Although PTAM is mainly used for augmented reality tasks, it can provide a rough three-dimensional reconstruction of points in real-time.

Some output examples are shown in Fig. 7. Points belonging to the same plane share the same color. Provided that the data is reliable enough, our method is capable of detecting planes in real-time with good accuracy. Points were inserted as soon as they were detected. Real-time J-linkage generated 1000 hypothesis per time step (in this case the clock is given by the frame rate of the input video), with a pool of hypothesis of size 10000. Planes with less than 20 points were discarded.

A video of the system in action is available for download [3].

(a) Frame 40. (b) Frame 63.

Figure 6. Extracted planes at different times from *Valbonne* data-set. Different colors encode different planes.

Figure 7. Different frames of the real-time simulation with PTAM software. Different colors encode different planes.

[3] http://profs.sci.univr.it/~fusiello/demo/jlk/

6. Conclusions

In this paper we have described an algorithm capable of detecting multiple instances of a model from data corrupted by noise and outliers in real-time. Starting from the original J-linkage algorithm, several adaptations have been made to make it incremental and faster. The trade-off between time efficiency and accuracy can be controlled parametrically by changing the number of points or hypothesis processed at each step, the total length of the hypothesis pool or the number k of neighbors to take into account. The proposed Real-time J-linkage can be applied to a wide range of applications.

References

[1] D. Capel. An effective bail-out test for RANSAC consensus scoring. In *Proc. British Machine Vision Conf., Oxford*, volume 10, 2005. 1

[2] O. Chum and J. Matas. Randomized RANSAC with $T_{d,d}$ test. In *Proc. British Machine Vision Conference*, pages 448–457, 2002. 1

[3] O. Chum and J. Matas. Matching with PROSAC-progressive sample consensus. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005. CVPR 2005*, volume 1, 2005. 1

[4] O. Chum and J. Matas. Optimal randomized RANSAC. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(8):1472–1482, 2008. 1

[5] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002. 2

[6] M. Farenzena, A. Fusiello, and R. Gherardi. Structure-and-motion pipeline on a hierarchical cluster tree. In *Proceedings of the IEEE International Workshop on 3-D Digital Imaging and Modeling*, ICCV Workshops, pages 1489–1496, Kyoto, Japan, October 3-4 2009. 5

[7] M. Fischler and R. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Morgan Kaufmann Readings Series*, pages 726–740, 1987. 1

[8] Y. Kanazawa and H. Kawakami. Detection of planar regions with uncalibrated stereo using distribution of feature points. In *British Machine Vision Conference*, pages 247–256, 2004. 2

[9] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007. 4

[10] G. Klein and D. Murray. Improving the agility of keyframe-based SLAM. In *Proc. 10th European Conference on Computer Vision (ECCV'08)*, pages 802–815, Marseille, October 2008. 4

[11] J. Matas and O. Chum. Randomized RANSAC with sequential probability ratio test. In *Tenth IEEE International Conference on Computer Vision, 2005. ICCV 2005*, volume 2, 2005. 1

[12] D. Myatt, P. Torr, S. Nasuto, J. Bishop, and R. Craddock. NAPSAC: High noise, high dimensional robust estimation. *BMVC02*, pages 458–467, 2002. 2

[13] D. Nister. Preemptive RANSAC for live structure and motion estimation. *Machine Vision and Applications*, 16(5):321–329, 2005. 1

[14] R. Raguram, J. Frahm, and M. Pollefeys. A Comparative Analysis of RANSAC Techniques Leading to Adaptive Real-Time Random Sample Consensus. In *Proceedings of the 10th European Conference on Computer Vision: Part II*, pages 500–513. Springer, 2008. 2

[15] C. V. Stewart. Bias in robust estimation caused by discontinuities and multiple structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(8):818–833, 1997. 2

[16] K. Tanaka and E. Kondo. Incremental RANSAC for online relocation in large dynamic environments. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 68–75, 2006. 2

[17] R. Toldo and A. Fusiello. Robust Multiple Structures Estimation with J-Linkage. In *Proceedings of the 10th European Conference on Computer Vision: Part I*, pages 537–547. Springer, 2008. 1, 2

[18] L. Xu, E. Oja, and P. Kultanen. A new curve detection method: randomized Hough transform (RHT). *Pattern Recognition Letters*, 11(5):331–338, 1990. 2

[19] M. Zuliani, C. S. Kenney, and B. S. Manjunath. The multi-RANSAC algorithm and its application to detect planar homographies. In *Proceedings of the IEEE International Conference on Image Processing*, Genova, IT, September 11-14 2005. 2