Hashmod: A Hashing Method for Scalable 3D Object Detection

Wadim Kehl[1]
kehl@in.tum.de
Federico Tombari[12]
federico.tombari@unibo.it
Nassir Navab[1]
navab.cs.tum.edu
Slobodan Ilic[3]
slobodan.ilic@siemens.com
Vincent Lepetit[4]
lepetit@icg.tugraz.at

[1] Computer-Aided Medical Procedures, TU Munich, Germany
[2] Computer Vision Lab (DISI), University of Bologna, Italy
[3] Siemens AG Research & Technology Center Munich, Germany
[4] Institute for Computer Graphics and Vision, TU Graz, Austria

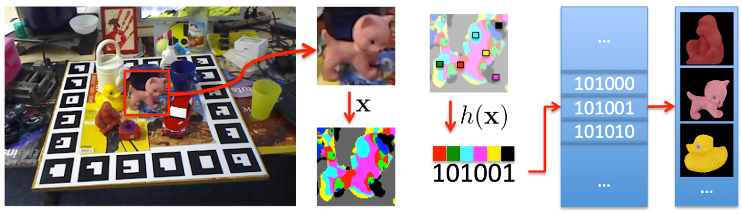

Figure 1: A sliding window evaluates learned hash functions h computed on extracted LineMOD features x to efficiently index into subsets of candidate views for further matching.

We present a scalable method for detecting objects and estimating their 3D poses in RGB-D data. To this end, we rely on an efficient representation of object views and employ hashing techniques to match these views against the input frame in a scalable way. While a similar approach already exists for 2D detection, we show how to extend it to estimate the 3D pose of the detected objects. In particular, we explore different hashing strategies and identify the one which is more suitable to our problem. We show empirically that the complexity of our method is sublinear with the number of objects and we enable detection and pose estimation of many 3D objects with high accuracy while outperforming the state-of-the-art in terms of runtime, as depicted in Figure 2.

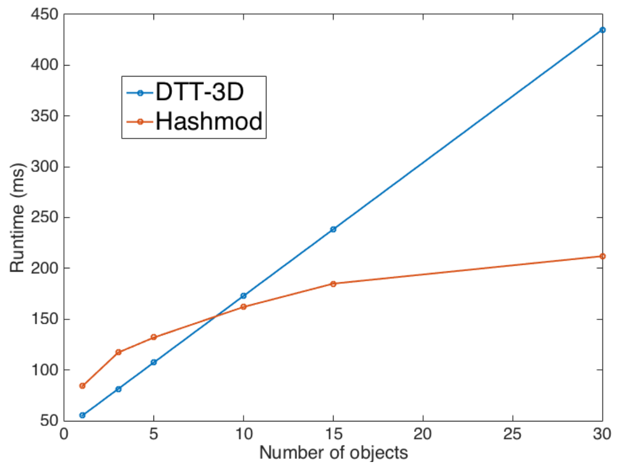

Figure 2: Runtime comparison with the state-of-the-art template matching DTT-3D [2]. Our method overtakes due its sublinear time complexity.

Descriptor computation Given a database of M objects, we synthetically create N views for each object from poses regularly sampled on a hemisphere of a given radius. From this, we compute a set $\mathcal{D}$ of d-dimensional binary descriptors

$$\mathcal{D} = \{\mathbf{x}_{1,1}, \dots, \mathbf{x}_{M,N}\}, \tag{1}$$

where $\mathbf{x}_{i,j} \in \mathbb{B}^d$ is the descriptor for the i-th object seen under the j-th pose. We use LineMOD [1] in practice to compute these descriptors and concatenate the binary representation to obtain the binary strings $\mathbf{x}_{i,j}$.

Hash learning We learn several hashing functions whose purpose is to immediately index into a subset, often called a "bucket", of $\mathcal{D}$ when applied to a descriptor $\mathbf{x} \in \mathbb{B}^d$ during testing (see Figure 1). These buckets are filled with descriptors from $\mathcal{D}$ with the same hash value so that we can restrict our search for the nearest neighbor of $\mathbf{x}$ to the bucket retrieved via the hashing function instead of going through the complete set $\mathcal{D}$. It is very likely, but not guaranteed, that the nearest neighbor is in at least one of the buckets returned by the hashing functions. In practice, a careful selection of the hashing functions is important for good performance. Since the descriptors $\mathbf{x}$ are already binary strings, we design our hashing functions $h(\mathbf{x})$ to return a short binary string made of b bits directly extracted from $\mathbf{x}$. This is a very efficient way of hashing and we will refer to these short strings as hash keys. We evaluated multiple strategies and settled for the one below that is inspired by greedy tree growing for Randomized Forests:

Starting with a set of descriptors at the root, we determine the bit that splits this set into two subsets with sizes as equal as possible, and use it as the first bit of the key. For the second bit, we decide for the one that splits those two subsets further into four equally-sized subsets and so forth. We stop if b bits have been selected or one subset becomes empty. This procedure alone yields a balanced tree with leafs of similar numbers of elements. Each hash key can be regarded as a path down the tree and each leaf represents a bucket. Note that such a balanced repartition ensures retrieval and matching at a constant speed. We now further adapt the strategy to our problem: to improve detection rates we favor similar views of the same object to go into different branches. The idea behind this strategy is to reduce misdetections due to noise or clutter in the descriptors, leading to different buckets during testing. Formally, the j-th bit B of the key is selected by solving:

$$\arg\min_B \frac{1}{|N_i|} \sum_i \left| |\mathcal{S}_L^B(N_i)| - |\mathcal{S}_R^B(N_i)| \right| + \frac{1}{|N_i|^2} \left(P(\mathcal{S}_L^B(N_i)) + P(\mathcal{S}_R^B(N_i)) \right) , \tag{2}$$

where $N_i \subset \mathcal{D}$ is the set of descriptors contained by the i-th node at level j, and $\mathcal{S}_{\{L,R\}}^B(N_i)$ are the two subsets of N_i that go into the left and right child induced by splitting with B. The first term in above equation balances equal splits whereas the second penalizes similar views of the same object falling into the same side of the split.

Results We ran our method on the LineMOD ACCV12 dataset [1] consisting of 15 objects. We are able to consistently detect at around 95% – 96% accuracy on average which is slightly worse than LineMOD and DTT-3D. However, we are always faster than LineMOD and overtake DTT-3D at around 8 objects where our constant-time hashing overhead becomes negligible and the methods' time complexities dominate. This is important to stress since real scalability comes from a sublinear growth.

[1] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradsky, K. Konolige, and N. Navab. Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes. In *ACCV*, 2012.

[2] R. Rios-Cabrera and T. Tuytelaars. Discriminatively trained templates for 3D object detection: A real time scalable approach. In *ICCV*, 2013.