2015 IEEE Winter Conference on Applications of Computer Vision

A General Framework for Fast 3D Object Detection and Localization Using an Uncalibrated Camera

Andrés Solís Montero Jochen Lang Robert Laganière
School of Electrical Engineering and Computer Science
University of Ottawa
asolismon@uottawa.ca {jlang,laganiere}@eecs.uottawa.ca

Abstract

In this paper, we present a real-time approach for 3D object detection using a single, mobile and uncalibrated camera. We develop our algorithm using a feature-based method based on two novel naive Bayes classifiers for viewpoint and feature matching. Our algorithm exploits the specific structure of various binary descriptors in order to boost feature matching by conserving descriptor properties (e.g., rotational and scale invariance, robustness to illumination variations and real-time performance). Unlike state-of-the-art methods, our novel naive classifiers only require a database with a small memory footprint because we store efficiently encoded features. In addition, we also improve the indexing scheme to speed up the matching process. Because our database is built from powerful descriptors, only a few images need to be 'learned' and constructing a database for a new object is highly efficient.

1. Introduction

Vision systems for object detection, localization and tracking enable numerous applications in human-computer interaction, mobile applications and surveillance. In our system we aim to both, detect the presence of an object as well as locate it in the image. Mobile applications place restrictions on compute resources and memory footprint of an application. These restrictions motivate us to improve performance and reduce database size of our object detection and tracking system compared to the state-of-the-art.

Many different strategies have been proposed for object detection including divide-and-conquer search [14, 15], edge matching [4, 19], gradient matching [20, 10], recognition by parts [11], and feature-based methods [21, 23, 18, 27]. In this paper we present a 3D object detection and localization framework that mixes feature-based approaches, binary descriptors and classifiers and is capable of detecting moving objects from a moving uncalibrated camera (see Figure 1). We exploit the most relevant characteristics of the particular descriptor employed in our framework (i.e., invariant to rotation, scale, illumination, noise, etc.) while improving the classification rates. Our database has a small memory footprint (i.e., $\mathcal{O}(n)$ where n is the number of keypoints stored in the database) compared to existing classifiers (i.e., Random Forests [9], Ferns [18]) because we assume a classic naive Bayesian model with independent binary features. Also, we efficiently store encoded features derived from a set of images of the object to reduce memory usage. Our improved indexing scheme speeds up the matching process and outperforms pattern-based indexing [23]. Finally, our framework uses a Kalman filter for object prediction and tracking while reducing noise in the estimations [28].

The contributions of our paper are a framework for real-time 3D object detection and localization (Section 3), two novel naive Bayes classifiers for viewpoint and feature matching and a fast indexing scheme for searching (Section 4). We show experimentally (Section 5) that assuming complete independence between binary features generates competitive classification rates while reducing memory usage.

2. Related Work

Feature-based object recognition systems have been extensively studied in the literature [21, 23, 18, 27]. They strongly rely on feature matching schemes to recognize and localize objects in different views. Different matching schemes and similarity measures have been created. The two main approaches in the literature are keypoint descriptors and classifiers.

A keypoint descriptor is a function that is applied to the neighborhood of a keypoint location, creating a compact signature that describes the keypoint in an image. A descriptor is an 'unique' representation of a keypoint which can match to another keypoint in a different image based on their similarity (as measured by, e.g., L1-norm, Euclid-

978-1-4799-6683-7/15 $31.00 © 2015 IEEE
DOI 10.1109/WACV.2015.122

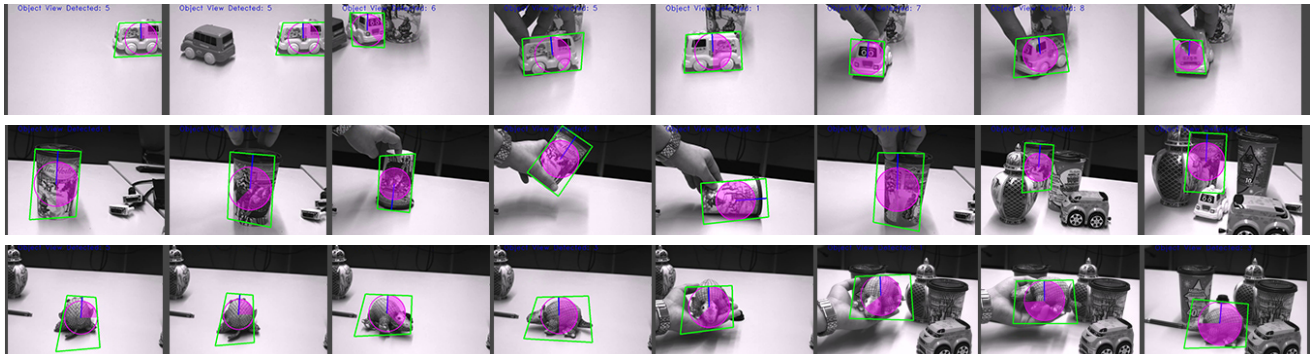

Figure 1. Object detection examples. The different objects in each row from top to bottom were trained with 8, 6 and 10 views, and the recognition rate was 91%, 93%, and 80% respectively. The circle inside the detection box represents the ratio of training views over the total training views used in the detection (full circle corresponds to all training views used).

ian norm, Hamming distance, etc.). Descriptors of a keypoints may be invariant to changes such as rotation, illumination, noise, scale, etc. Descriptors can be classified as binary (i.e. BRIEF [8], ORB [22], FREAK [24], BRISK [16], HIP [23], etc.), or gradient-based (i.e. SIFT [17], SURF [2], HOG [10], etc.). Unfortunately gradient based descriptors, although considered to be the gold standard, are computationally expensive and less appropriate for low-powered embedded platforms in their original form. On the other hand binary descriptors demonstrate accuracy while achieving fast performance, low memory requirements and are simple to implement. They can take advantage of CPU and GPU specific instructions for their implementation and matching [8], making them even faster. Binary descriptors have been effectively used in mobile platforms for planar object detection [23, 3, 1].

Classifiers treat the matching/similarity problem as one of classification [6, 5, 18]. They improve the classification rates while reducing computation after the feature database is learned in an off-line training phase. Some classifiers such as Random Forrest and Ferns learn the feature distribution of a database by means of randomization [5]. The off-line learning step ensures a better encoding of the most relevant features and their structure for very significant perspective changes. One of the leading approaches in feature classifiers is Ferns [18, 27, 26].

Fern classifiers learn the distribution of binary features around a relevant location using multiple views of the same keypoint. The features are binary comparisons between images intensities at random positions around the keypoint. Fern solves the classifying problem as the maximization of the posterior probability given a set of binary features observation around the keypoint. Ozuysal et al. [18] propose a Semi-Naive Bayes Classifier where they assume that groups of binary features are conditionally independent as a trade off between complexity and performance. Ferns are created assuming dependency between binary features themselves and by randomly grouping the features (by choosing the size of a Fern group and the number of groups to describe a keypoint). Finally, the conditional distributions for each group (Ferns) are learned in an off-line phase by warping the image patch around each keypoint using multiple random transformations. The learned distributions are used as a lookup table at run-time to speed up the classification problem. The drawbacks of Ferns are a large memory footprint and the off-line learning phase. The memory requirement is exponential in the Fern size, i.e., $2^{fernSize} \times nFerns \times nKeypoints \times sizeof(data)$. For example, if we train for 1000 keypoints ($nKeypoints$) using 50 ferns ($nFerns$) of size 11 ($fernSize$) with 4 bytes floats ($sizeof(data)$), storing their conditional probabilities results in a 400 MB database. The random selection of binary features is time consuming because the training phase needs to generate all possible transformations to be able to classify from arbitrary viewpoints.

Some modifications of the original Ferns have enabled their use on mobile platforms [26, 27]. Most of these modifications target the memory consumption by tuning the Fern size, the number of ferns and the data used to store the conditional probabilities but Ferns are inherently exponential in their size. Also, the improvements do not address the off-line learning phase. Daniel et al. [27] combine PhonySIFT and PhonyFerns for a mobile platform, by executing both in the same pipeline for object detection. Other approaches to adapt Ferns for mobile platforms execute computations on an remote server. But these approaches suffer from low performance due to restricted bandwidth, and high power consumption because of the required network communications [27, 25]. Ventura et al. [25] associate 2D locations in the video frame with 3D locations in the real world which requires building and updating a 3D model.

Our work combines descriptors and classifiers in one single matching scheme. We show through our experimental results that a full Naive Bayes classifier combined with bi-

nary descriptors, reduces the training time and memory usage, and improves the matching rate of the descriptors while performing as well as the original Ferns.

3. Framework Overview

We create a two-phase framework: an off-line feature training and a runtime-matching scheme for object detection and localization (similar to Ferns [18], Taylors and Drummond [23], and Akhoury [1]). During the training phase (Section 4.1) we learn the model of the 3D object of interest based on a set of images of the object from different viewpoints. Each image is randomly warped several times for learning the most relevant keypoints and their binary descriptors. We then use the most stable keypoints for all views and their set of descriptors to train our Naive Bayes Classified Descriptors (NBCD) (Section 4.2). A tree-like index data structure from the classifiers is created to speed up classifications (Section 4.3). We compute the keypoint correspondences between the images using a RANSAC approach and a likelihood test to validate the geometrical constraints (Section 4.4). During the runtime object detection and localization phase, we extract the most relevant keypoints from a video stream. Once the match between the database and the video frame is created, we have sets of matching points for all close database views. These matched keypoints are used to select the closest view in the database with another Bayes classifier (Section 4.5). Finally, the closest view is projected back (Section 4.6) to the video input and tracked using a Kalman Filter (Section 4.7).

4. Implementation

4.1. Training Features

We use a training set of images covering different viewpoints of the 3D object of interest. Our approach wraps each reference image using random transformations from different viewpoint bins and scales similarly to [1, 23] and [18]. Each warped keypoint selected for voting is back-projected to the original reference image. Stable keypoints $\{k_1, k_2, k_3, \ldots, k_n\}$ are selected as the points with more votes [18]. In our training stage we use binary descriptors instead of images patches. Also we do not have to create multiple viewpoint 'bins' to train thanks to the invariant properties of the descriptors. For each artificially warped image, we extract a binary descriptor (e.g., BRIEF, ORB, BRISK, FREAK) of its relevant keypoints. The descriptor is a binary string $[f_1, f_2, \ldots, f_L]$ of length L, denoted $D = f^{(1:L)}$ (we consider each bit of the string as a feature). Once the keypoints are back-projected and merged with the reference image keypoints, each keypoint contributes with a descriptor D extracted in the warped image. Typically, 1000 images are generated for each viewpoint bin. The use of binary features instead of image patches considerably reduces the amount of memory needed while creating the dataset of stable keypoints before training the classifier. A gray scale image patch of 32x32 pixels with 8-bit depth has a memory usage of 1 Kb compared to 32 bytes (128 bits) in case of ORB, or 64 bytes (512 bits) in case of FREAK, or BRISK. The number of viewpoints bins used in training is related to the properties of the binary descriptor. In previous solutions [23, 18, 1], the reference images needed to be trained from all possible viewpoints but with rotation-invariant descriptors there is no need to train for the 360° in-plane rotations around the keypoint. The same idea applies when the descriptor is scale-invariant. In fact, all our examples in this paper use only one viewpoint bin with perspective transformations around the center of the image frame. Aside from the numbers of bins, we use the same perspective warping during training as in [1] (i.e., we generate 1000 viewpoints inside a single bin with yaw and pitch of $[-45°, 45°]$ each, a roll of $[-30°, 30°]$, and 3 scale factors of $\{0.6, 1, 1.6\}$.

4.2. Naive Bayes Classified Descriptors

Once we computed the set of most stable keypoints and their associated descriptors, we want to learn the distribution of the binary features describing a keypoint. Formally, we want to be able to assign the most likely keypoint k_i given a binary descriptor D, $\arg\max_i P(K = k_i | f^{(1:L)})$. The event $f^{(i)} \epsilon \{0, 1\}$ is the binary value of the descriptor at i^{th} index.

Assuming independence between the descriptor features, an uniform prior $P(K)$, our problem reduces to the maximum likelihood classification problem. We assume all keypoints are equally likely for simplicity but it is possible to learn the prior distribution while training. In contrast to Ferns [18] we do not assume dependencies between the binary features combining them into groups. Our experiments demonstrate correct classification of binary descriptors into classes while removing the exponential memory use of Ferns. Each binary feature of the descriptor contributes a specific weight to the classification giving a strong *unique* representation. Repeated observations with invariant descriptors reinforce a feature. For example, a rotation invariant descriptor should have the same value for each index under any rotation but in practice the binary feature is not completely stable. In our Bayes classifier, we describe the likelihood of every single bit inside the descriptor without knowing nor learning the relation with other features but at the same time benefiting from the descriptor's invariance.

Each binary feature's conditional probability is learned as the maximum likelihood estimation of the mean with a scale factor to ensure that the model is not invalidated when there is not enough training data or one of the features has zero probability [13].

$$P(f^{(i)} = 1 | K = k_i) = \frac{\sum_{j=1}^{M} f_j^{(i)} + N_c}{M + C \times N_c} \qquad (1)$$

Where $\sum_{j=1}^{M} f_j^{(i)}$ is the sum of the i^{th} bit values of all the descriptors associated with keypoint k_i and M is the total number of descriptors belonging to keypoint k_i. In practice, the value of N_c does not influence the results, we use the values of $N_c = 1$ and $C = 2$ because we have two possible classifications $f^{(i)} \epsilon \{0, 1\}$.

We reduce memory and computation by storing the logarithmic values $logP(f^{(i)} = 0|K = k_i)$ and $logP(f^{(i)} = 1|K = k_i)$ for bit i in a single byte. Each logarithmic probability is stored using 4 bits and hence values in the range of 0 to 15. The logarithm of a probability value is a negative number but we just save the absolute rounded integer value (see figure 2). Finally, for each keypoint we use a lookup table with L $bytes$ where L is the size of the binary descriptor in bits. Where the i^{th} byte encodes the negative conditional log-probability of the feature $f^{(i)}$ given the keypoint. For each byte the 4 most significant bits encodes the probability of feature i being 0 given keypoint k_i and the least significant bits the probability of the binary feature being 1.

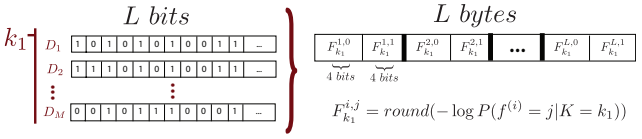

Figure 2. Given a point k_1 and its associated descriptors $\{D_j, j = 1, 2, ..., M\}$, we encode $F_{k_1}^{(i,j)} = round(-\log P(f^{(i)} = j|K = k_1))$ as the negative round value of the log conditional probability. We use only 4 bits (i.e. values from 0 to 15) to store each conditional probability, making a byte for each binary feature in the descriptor.

4.3. Indexing

We do not use standard pattern-based indices because they are not rotationally-invariant, e.g., the patterns used in [23] suffer deformations under different keypoint orientation which causes the same keypoint to generate different indices depending on orientation. Computationally expensive solution such as sub-pixel interpolation are not fast enough for indexing but high repeatability is nevertheless required. We propose a different index computation based on certain pattern positions around the keypoint, we select a list of binary features indices from the computed binary descriptor. This ensures that two keypoints with the same descriptors will have the same index value. Also, this scheme allows us to have a variable index size L according the database size and we select the proper index size (i.e. number of bits) accordingly.

As in [23], we assign an integer index value to each binary descriptor corresponding to the most relevant keypoint k_i by combining their binary values. Finally, we select the most repeated indices for each keypoint. Every index value will be pointing to a list of keypoint classifiers containing the same index (e.g., for a 13-bit index we will have a maximum of $2^{13} - 1$ index values). Each index will have approx. $\frac{n}{2^L} \times Z$ keypoint classifiers where Z is the amount of indices per keypoint added to the database, L is the number of bits of the index and n the total amount of keypoints in the database. This makes our indexing search order $O(\frac{n}{2^L})$, and in practice it is fast with an appropiate choice of L.

4.4. Probabilistic Model for Fundamental Matrix Verification

Our fundamental matrix verification step uses the likelihood-ratio test of Brown and Lowe [7]. In the test n_m is the number of matches and n_i is the number of inliers defined by RANSAC. The event of a correct/incorrect Fundamental Matrix is represented by the binary variable $F_C \epsilon \{0, 1\}$. The event that the i^{th} keypoint match $K_M^{(i)} \epsilon \{0, 1\}$ is an inlier/outlier is assumed to be independent Bernoulli, so that the total number of inliers is Binomial.

$$P(K_M^{(1:n_m)}|F_C = 1) = B(n_i; n_m; p_1) \quad (2)$$

$$P(K_M^{(1:n_m)}|F_C = 0) = B(n_i; n_m; p_0) \quad (3)$$

Where p_1/p_0 is the probability a keypoint is an inlier given a correct/incorrect Fundamental Matrix. The set of keypoint matches is denoted $K_M^{(1:n_m)}$ and $B(.)$ is the Binomial distribution. The fundamental matrix is accepted if $P(F_C = 1|K_M^{(1:n_m)}) > p_{min}$ which leads to the likelihood-ratio test

$$\frac{B(n_i; n_m; p_1)P(F_C = 1)}{B(n_i; n_m; p_0)P(F_C = 0)} \gtrless_{reject}^{accept} \frac{p_{min}}{1 - p_{min}} \quad (4)$$

Following Brown and Lowe [7], we choose values of $P(F_C = 1) = 10^{-6}$, $p_{min} = 0.999$, $p_1 = 0.6$ and $p_0 = 0.1$ leads to $n_i > 8.0 + 3.0n_m$.

4.5. Naive Bayes Classifier for View Matching

Given an observation of keypoints, view matching finds the closest view in our database to a frame in the video sequence. Let $V = \{v_j, j = 1, 2, ..., v\}$ be the set of views in our database, each view $v_j = k^{(1:n)}$ is composed of a set of most stable keypoints from the training phase. The event that the k_i keypoint belongs to view v_j is represented by a binary variable $K_j^i \epsilon \{0, 1\}$ that we assume to be independent Bernoulli.

Formally, we are looking for the most likely view given an observation of keypoints in the video frame, $\arg\max_{v_j} P(V = v_j|K_j^{(1:n)})$.

Using Bayes Formula and assuming an uniform prior $P(V)$, neglecting the constant denominator, our problem reduces to finding the maximum likelihood $\arg\max_{v_j} \prod_{i=1}^{n} P(K_j^i|V = v_j)$. Because we assumed K_j^i

to be independent Bernoulli given v_j, the total number of inliers becomes binomial.

$$\prod_{i=1}^{n} P(K_j^i | V = v_j) \;=\; B(n_i, n, \omega_j) \tag{5}$$

Where ω_j is the probability of a keypoint corresponding to view v_j, n is the total number of keypoints in a view and n_i is the amount of inliers and $B(.)$ is the Binomial distribution. Note that if we assume ω_j to be the same for every view v_j, the most likely view from our model database will be the one with the highest number of inliers.

4.6. Object Detection and Localization

Once we know the fundamental matrix between the closest view in the database and the video frame, we proceed by finding the location of the object. However, we neither have 3D object information nor the camera intrinsic parameters. Instead of devising a computationally expensive camera auto-calibration process, we project the object's boundary in the closest view from the database to the video frame. The approximate object boundary has been previously identified by the user during the training stage. We simply assume that there is a plane that contains the object in the 3D space and compute a homography between the two views. The relationship between this homography H_ϕ and the Fundamental Matrix F is given as $F = [e']_X H_\phi$ where $[e']_X$ is a skew-symmetric matrix defined by Hartley and Zisserman [12]. The homography is computed using the RANSAC inliers of the Fundamental Matrix. The final projection of the objects boundary is forced to be on the epipolar lines by moving it to the closest point on its corresponding epipolar line.

4.7. Kalman Filter for Object Prediction

Each corner of the object bounding box is independently processed by a Kalman filter [28] at each video frame to produce a statistical estimate of the underlying system. The corner measurements are in pixel coordinates and passed through the filter each time we accept the likelihood-ratio 4.4. In case of a rejection we use the filter's predictions to narrow the search of the object in the next frame, which reduces the search between the keypoints in the video frame and database. If the object is not detected in the predicted area, the region is expanded to the frame size. The main reason of using a Kalman filter is to compensate for the noise associated with keypoint classifiers, for the noise in the Fundamental Matrix computation, and also to stabilize the detection results. We do not attempt to model the object's dynamics accurately and hence we simply use a constant velocity model for each corner separately.

5. Experimental Results

We compare our framework against Ferns and Binary Descriptors in terms of performance, memory usage, and running time. Our framework uses the binary descriptors' implementations of OpenCV (i.e., BRIEF, ORB, BRISK, FREAK) with their default parameter configurations. The Ferns implementation was the authors' [18] from their webpage. We use it unmodified except for adapting their training intervals to show the impact of training on their solution and ours.

5.1. Memory Usage

First, we compare the memory usage between binary descriptors without a training phase for Ferns and our framework. It is easy to see that binary descriptors are the most compact with a memory usage of $bits \times K$, where $bits$ is the size in bits of the descriptor and K the amount of the keypoints in the database. On the other hand, the memory footprint of the Ferns database grows exponentially with the Ferns' size S, i.e., $2^S \times M \times byteS \times K$, where M is the number of Ferns and $byteS$ is the size of bytes used to store the conditional probabilities. The original Fern implementation uses $S = 11$, $M = 30$ and $byteS = 4$ (float). However, the memory needed for our database is $8 \times bits \times K$, i.e., it is $O(K)$ just as the binary descriptors. We use a byte to represent every bit of the descriptor. For example, 1000 keypoints using the BRIEF descriptor will require 31.25Kb for storage, Ferns will require 234Mb, and ours 250Kb. Or put differently, adding an extra keypoint to the Ferns database will require 240Kb, almost as much as another 1000 points in our representation.

Figure 1 shows three tracking results with our framework. The car object in the first row uses a database with 8 views for a total of 450Kb, the cup in the second row with 6 views uses 350Kb, and the toy in the third row uses 10 views and 550Kb. Each view contains $\sim$200 keypoints.

5.2. Performance

Next, we evaluate the performance of each algorithm under different images transformations. We synthetically generate perspective transformations of the planar object (see Figure 3) at different scales and different positions in the image with changes in contrast and brightness (see Table 1). The background is filled with white noise. See Figure 3 for some example transformations.

In considering Table 1, it is important to take training into account. The parameters for the transformation during the feature learning (Section 4.1) with our Naive Bayes Classified Descriptors (NBCD) are the same for all objects with rotations of roll of $[-30, 30]$, pitch and yaw of $[-45, 45]$, and with three scale levels $\{0.6, 1, 1.6\}$. The average training time for each view is $\sim$25 seconds. Although the range of training transformation is considerably

Algorithm	Rotation*	Scale×	Illumination+
BRIEF	9%	43%	62%
NBCD + BRIEF	15%	62%	64%
ORB	75%	74%	89%
NBCD + ORB	92%	91%	**90**%
BRISK	79%	75%	82%
NBCD + BRISK	**94**%	**92**%	84%
FREAK	69%	70%	79%
NBCD + FREAK	84%	89%	81%
Ferns	14%	46%	61%
Ferns*	89%	88%	68%
* roll = $[-180°, 180°]$, pitch and yaw = $[-70°, 70°]$			
× scale = $[0.2, 2]$			
+ $\alpha \times I(x, y) + \beta$, $\alpha = [.3, 3]$, $\beta = [-100, 100]$			

Table 1. Comparison of detection rates between matching with only binary descriptors, with Ferns and with our Naive Bayes Classified Descriptors (NBCD) when combined with binary descriptors. The detection rates were tested under random rotation, scale and illumination variations and averaged. The detection rates of Ferns are listed with similar training range to ours and in the line Ferns* with complete training.

less than the object transformation for the detection test, our NBCD with BRISK or ORB has the highest detection rate in all three tests as shown in Table 1. Our NBCD exploits properties of the descriptor, e.g., BRIEF lacks rotation invariance and hence our NBCD with BRIEF also lacks this feature but our NBCD with ORB demonstrates rotational invariance because it is a feature of ORB. Ferns use affine transformations in their training. We compare to them with two set of transformations in training. The set Ferns in Table 1 uses the same training parameters than for our NBCD and the set Ferns* uses a full range of rotations and a wider scale range of 0.5 to 1.5.. The average training time for Ferns and Ferns* was ∼170 and ∼360 seconds, respectively. The results in Table 1 demonstrate the dependency of Ferns on training for all possible view transformations to successfully detect an object. In contrast, our NBCD with the corresponding descriptor is successful even with limited training.

Figure 3. Example of an object viewed under different rotations, scales and illuminations.

We also compare Ferns and our NBCD with ORB on the mouse pad example of Ozuysal et al. [18]. Both methods achieve a similar detection rate but ours uses 200x less memory with training that is 14.5x faster (see Table 2). We

Algorithm	Fps	Detection	Memory	Time
Ferns	25.1	**87.4**%	16Mb	348s
NBCD + ORB	**26.5**	86.9%	**80**Kb	**24**s

Table 2. Comparison between Ferns and our framework (i.e., NBCD + ORB) for the mousepad example of Ozuysal et al. [18]. We compare fps (i.e., frame per seconds), detection rate, memory usage and construction time of the model.

also compare both approaches using 200 frames of a video of a more complex 3D object (see Figure 5). Our detection rate is 88% compared to 72% for Ferns because the planarity assumption in Ferns is violated by 3D objects. Running times for Ferns and NBCD are similar but we find that our NBCD is independent of keypoint detection and feature extraction (see Figure 6). Our framework can therefore take advantage of improvements in speed of new binary descriptors or of hardware implementations.

Figure 4. Mousepad example of Ozuysal et al. [18]. Frames 131, 171, 219 are shown: Ferns in first row and our NBCD with ORB in the second row (see Table 2).

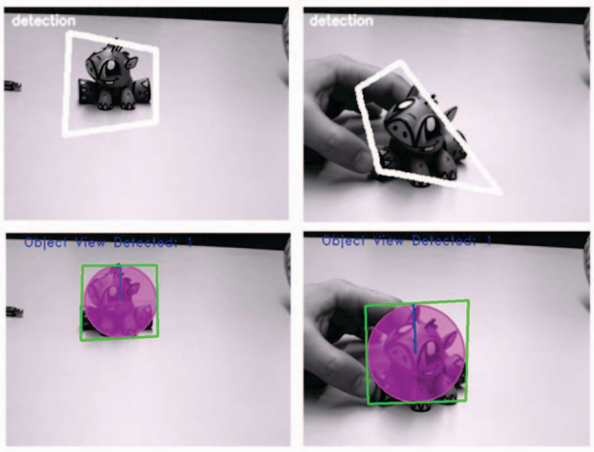

Figure 5. The planarity assumption in Ferns does not perform well with more complex non planar objects (e.g., top images). The use of the fundamental matrix is stable and can be used with planar and non-planar object.

5.3. Running Time

Binary descriptors use the Hamming distance as a similarity measure because it can be computed efficiently count-

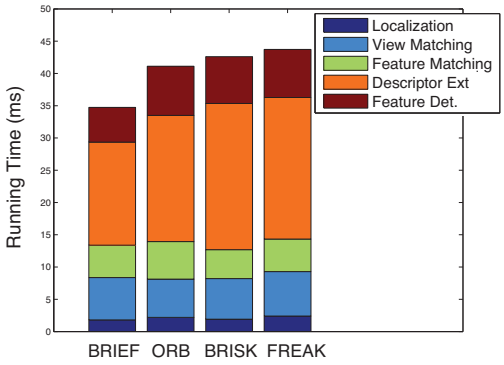

Figure 6. Running time of our framework using different binary descriptors. The running time is separated by keypoint detection, descriptor extraction, keypoint matching, view matching, and localization of the object. Our framework is independent of the keypoint extraction and descriptor selection.

ing bits in parallel [29]. Solutions exist to count the number of ones in a 32 bits integer with less than 15 operations in C. Ferns use a lookup table approach where the S bits (Ferns size) are packed together and used as the index for M lookup values (number of Ferns). As a result Ferns similarity is linear in the number of bits and the number of Ferns. Using S=8 or 16 creates better performance because the bits are already grouped like primitive data types. Our keypoint classification is also a lookup table and it depends on the amount of bits of the descriptor, each descriptor bit is translated to a log probability value. Overall they are all linear in terms of the number of bits O(bits). However, our classifier and Ferns are capable of creating a better classification than the Hamming distance similarity measure (see Figure 7);

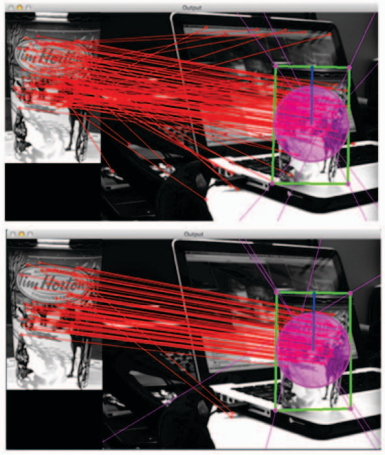

Figure 7. Matches between the database view and the video feed. The matches in the top image are using Hamming distance (118 matches: 92 inliers and 26 outliers). The bottom shows our classifier (84 matches: 82 inliers and 2 outliers). All matches in both images have the same Hamming distance. Ferns show similar results to ours.

5.4. Indexing

Finally we compare pattern-based indexing [23] to our indexing scheme. We find that pattern-based indexing is highly dependent on the keypoint location and orientation. We check if the same keypoint from different viewpoints produce the same index value because this is indicative of rotation and scale invariance. In our experiments, we rotate (i.e., from 0 to 180) and scale (i.e., from 0.3 to 2) the object's views and compute the keypoint matches between the original and the transformed views. Matches are all those points that have more than 90% of descriptors bits with the same value. Finally, we compute their index value in both locations and compute the repetition rate out of all matches in Fig. 8. In our index scheme (IS), we use a 13-bit index of randomly selected bits from the descriptor. For the pattern-based, we extracted a 5-bit (i.e., $P5$) and a 13-bit (i.e., $P13$) indices as described in [23]. In the case of FREAK which does not provide a feature detection; we use a combination of ORB feature detector plus FREAK descriptor to match the keypoints. BRIEF was removed from the comparison because it did not perform well under the transformations. See Fig. 8.

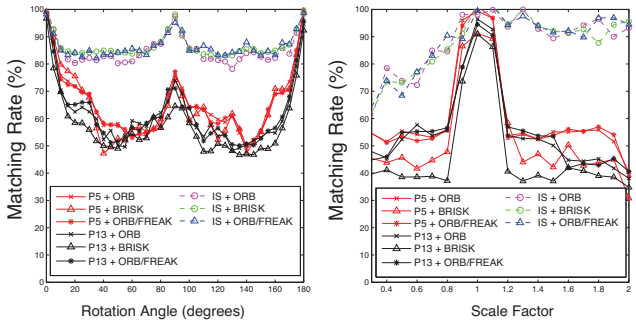

Figure 8. The diagrams show the index matching rates under different rotation and scale transformations. We compare pattern-based index schemes (i.e., $P5$ and $P13$) and our proposed index scheme (IS) using binary descriptors. The matching rate shows the repetitive rate indices from the matched keypoints.

6. Conclusion

We present a framework for 3D object detection and tracking from an uncalibrated mobile camera at frame rates higher than 24 fps. Our NBCD framework achieves high recognition rates and can take advantage of the properties of the used keypoint detector and binary descriptor. We introduce an improved indexing scheme for speeding up the keypoint classification that is rotation and scale invariant. NBCD successfully combines binary descriptors and classifiers for keypoint classification, leading to increased performance as compared to binary descriptors by themselves and to reduced memory footprint in comparison to Ferns classifiers. Unlike Ferns, the framework detects a non-planar 3D

object from different views. As a result, 3D objects are detected even if they are partially occluded and in different orientations. Experimental results confirm the effectiveness of the proposed solution.

Acknowledgment

This research was partly funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) under the Engage program with You.i Labs as the industrial partner.

References

[1] S. S. Akhoury and R. Laganière. Training binary descriptors for improved robustness and efficiency in real-time matching. In *Proc. of Int'l Conference on Image Analysis and Processing (ICIAP)*, volume 8157, pages 288–298. Springer, 2013.

[2] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, June 2008.

[3] D. Bekele, M. Teutsch, and T. Schuchert. Evaluation of binary keypoint descriptors. In *Proc. IEEE International Conference on Image Processing (ICIP)*, pages 3652–3656, Melbourne, Australia, 2013. IEEE.

[4] A. C. Berg, T. L. Berg, and J. Malik. Shape matching and object recognition using low distortion correspondence. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 26–33, 2005.

[5] A. Bosch, A. Zisserman, and X. Munoz. Image classification using random forests and ferns. In *Proc. of IEEE International Conference on Computer Vision (ICCV)*, pages 1–8. IEEE, 2007.

[6] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[7] M. Brown and D. G. Lowe. Automatic panoramic image stitching using invariant features. *Int'l Journal of Computer Vision*, 74(1):59–73, Aug. 2007.

[8] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: binary robust independent elementary features. In *Proc. of European Conference on Computer Vision (ECCV)*, pages 778–792, Berlin, Heidelberg, 2010. Springer-Verlag.

[9] A. Criminisi and J. Shotton. *Decision Forests for Computer Vision and Medical Image Analysis*. Springer Publishing Company, Incorporated, 2013.

[10] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 886–893, France, June 2005. IEEE.

[11] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[12] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[13] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[14] C. H. Lampert. An efficient divide-and-conquer cascade for nonlinear object detection. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1022–1029. IEEE, 2010.

[15] C. H. Lampert, M. B. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2008.

[16] S. Leutenegger, M. Chli, and R. Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. *IEEE Int'l Conference on Computer Vision (ICCV)*, 0:2548–2555, 2011.

[17] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int'l Journal of Computer Vision*, 60(2):91–110, Nov. 2004.

[18] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *IEEE Trans. Pattern Analysis and Machine Intelligence (TPAMI)*, 32(3):448–461, Mar. 2010.

[19] N. Payet and S. Todorovic. From contours to 3d object detection and pose estimation. *IEEE Int'l Conference on Computer Vision (ICCV)*, 0:983–990, 2011.

[20] X. Ren and D. Ramanan. Histograms of sparse codes for object detection. *Computer Vision and Pattern Recognition (CVPR)*, 0:3246–3253, 2013.

[21] F. Rothganger, S. Lazebnik, C. Schmid, and J. Ponce. 3d object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints. *Int'l Journal of Computer Vision*, 66(3):231–259, Mar. 2006.

[22] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *Proc. of IEEE Int'l Conference on Computer Vision (ICCV)*, pages 2564–2571. IEEE Computer Society, 2011.

[23] S. Taylor and T. Drummond. Multiple target localisation at over 100 fps. In *Proc. of British Machine Vision Conference (BMVC)*. British Machine Vision Assoc., 2009.

[24] P. Vandergheynst, R. Ortiz, and A. Alahi. Freak: Fast retina keypoint. *Computer Vision and Pattern Recognition (CVPR)*, 0:510–517, 2012.

[25] J. Ventura, C. Arth, G. Reitmayr, and D. Schmalstieg. Global localization from monocular slam on a mobile phone. In *Proc. of IEEE Virtual Reality*. IEEE, April 2014.

[26] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Pose tracking from natural features on mobile phones. In *Proc. of the IEEE/ACM Int'l Symposium on Mixed and Augmented Reality (ISMAR)*, pages 125–134. IEEE Computer Society, Sept. 2008.

[27] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Real-time detection and tracking for augmented reality on mobile phones. *IEEE Trans. on Visualization and Computer Graphics*, 16(3):355–368, 2010.

[28] G. Welch and G. Bishop. An introduction to the kalman filter. Technical report, Chapel Hill, NC, USA, 1995.

[29] M. V. Wilkes, D. J. Wheeler, and S. Gill. *The Preparation of Programs for an Electronic Digital Computer (Charles Babbage Institute Reprint)*. The MIT Press, 1984.