Dataset and Evaluation of Interest Point Detectors for Visual Tracking

Steffen Gauglitz Tobias Höllerer Matthew Turk

Department of Computer Science
University of California, Santa Barbara
{sgauglitz,holl,mturk}@cs.ucsb.edu

Technical Report 2010-06 *

Abstract

In this report, we present an extensive dataset of 96 video streams with ground truth. It includes various geometric changes, lighting conditions, and levels of motion blur, and can be used as testbed for a variety of algorithms in the context of visual tracking. We then use this dataset for a detailed quantitative evaluation of popular interest point detectors, which, in contrast to existing evaluations, is geared towards visual tracking in all relevant factors of the evaluation design.

1 Introduction

Visual tracking is a core component for many applications including visual odometry [6, 24], visual Simultaneous Localization and Mapping (SLAM) [7] and augmented reality (AR) [12].

Although some visual odometry systems work with optical flow, most tracking applications use feature-based visual tracking. In this case, interest point detection and feature description are the first steps of the system. Many algorithms have been proposed to tackle these tasks, and existing visual tracking systems use different approaches.

Existing evaluations are geared towards object recognition and image retrieval: they use low-noise, high-resolution, still images, large databases to match features against, and/or potentially very expensive algorithms; hence their results have limited validity for visual tracking. In particular, we are not aware of any work that compares the respective algorithms on video streams, which is the setup of interest for visual tracking.

In contrast, our work evaluates existing interest point detectors and with respect to their application in real-time visual tracking. Our testbed consists of video streams with several thousand frames total that exhibit different motion patterns; the algorithms are tested for both consecutive frames of smooth motion and randomly shuffled frames, revealing performance in the presence of translational movement, in-plane and out-of-plane rotation, scale changes and motion blur. We also evaluate the impact of individual algorithm parameters. For this evaluation, a setup to semi-automatically obtain stable ground truth for video streams is proposed and explained in detail.

Outline

This report is structured as follows: Section 2 discusses literature on datasets and existing detector evaluations. Section 3 reviews each detector that is included in the evaluation. Section 4 details the testbed and the setup for evaluation. Section 5 presents the obtained results, and finally, Section 6 draws conclusions.

*This report extends and superseeds our Technical Report 2009-08.

2 Existing Datasets and Evaluations

2.1 Datasets

In many domains, certain datasets have successfully been established as de-facto standards and used to compare and evaluate state-of-the-art algorithms; for example, the Middlebury datasets for multi-view reconstruction [28] and optical flow [3]. The testdata of Mikolajczyk et al. [21] has been used for various comparisons of local invariant detectors and descriptors. However, these sets consists of only a few still, high-resolution images per sequence with rather larger baseline distances, which limits their usefulness (and the significance of obtained results) for visual tracking.

Zimmermann et al. [31] collected image sequences of three different objects with approximately 12,000 images and made them available including ground truth. Lieberknecht et al. [15] presented a dataset of 40 sequences featuring 8 different textures in 5 different motion patterns with 1200 frames each. This dataset might be the most similar to ours in terms of purpose and scope, but unfortunately the ground truth is not made available. Moreover, their motion patterns all combine several motions (much like our pattern "unconstrained," see Section 4.2) and are, effectively, rather similar to each other. Hence, as with Zimmermann et al. [31]'s sequences, they do not allow a detailed analysis for different geometric distortions, lighting conditions or levels of motion blur.

2.2 Evaluations

Schmid et al. [27] compared interest point detectors on two still images under changes in rotation, viewpoint and illumination, as well as with artificially added image noise. Mikolajczyk and Schmid [19] and Mikolajczyk et al. [21] compared affine invariant detectors. Moreels and Perona [23] explored the performance of combinations of detectors and descriptors with a testbed of three-dimensional objects rather than flat pictures.

However, all comparisons mentioned above are geared towards object recognition and image retrieval rather than tracking. This becomes clear from the chosen testbeds, the performance measures chosen to evaluate the algorithms, and the set of detectors and descriptors that are tested. Execution time, a criterion crucial for designing real-time systems, receives very little or no attention, and reported execution times are in the order of seconds to several minutes, which is intractable for real-time tracking.

In contrast, the evaluation in this work aims at visual tracking in all of the factors mentioned above. Most notably, the performance measures are chosen with respect to the application of visual tracking and the testbed, which will be detailed in the next section, consists of video streams with several thousand frames affected by noise and motion blur rather than a few high-resolution, low-noise still images.

3 Interest Point Detectors

Due to the high dimensionality of image data, tracking every single pixel is computationally prohibitive and incorporates a lot of redun-

dancy, as pixels do not move independently of each other. Instead, a sparse set of features is extracted from the image. Although work has been done to detect and integrate other features (e.g., edges [8, 13]), features that build upon or around single points are most commonly used for tracking.

In the following, we will briefly review the detectors that are included in the evaluation. In particular, we will explain the role of each of the algorithms parame- ters, as their impact on the algorithms performance is individually evaluated.

3.1 Harris Corner Detector

Based on Moravec's corner detector [22], Harris and Stephens [10] developed the following algorithm: Given an image I, the algorithm first computes matrix M for every pixel (x,y):

$$M(x,y) = \begin{bmatrix} \sum_{u,v} w_{u,v} \cdot [I_x(x_r,y_r)]^2 & \sum_{u,v} w_{u,v} \cdot I_x(x_r,y_r) I_y(x_r,y_r) \\ \sum_{u,v} w_{u,v} \cdot I_x(x_r,y_r) I_y(x_r,y_r) & \sum_{u,v} w_{u,v} \cdot [I_y(x_r,y_r)]^2 \end{bmatrix} \quad (1)$$

I_x and I_y denote the derivatives of image I, $(x_r,y_r) := (x+u,y+v)$, and $w(u,v)$ is a window and weighting function. In the simplest case, $w(u,v)$ can be a binary rectangular filter. Harris and Stephens [10] propose to use a Gaussian window $w(u,v) = \exp\{-(u^2+v^2)/2\sigma^2\}$.

Matrix M is called the second-order moment matrix [23] and is an approximation to the auto-correlation function of image I (cf. Appendix A of [27]). Depending on the eigenvalues λ_1, λ_2 of M, the region can be classified as either flat (λ_1 and λ_2 small), edge region (one small, one large eigenvalue) or corner region (both large). To avoid the explicit computation of λ_1, λ_2, which is costly, Harris and Stephens [10] propose the following corner score, which is derived based on λ_1, λ_2, but can be expressed without them:

$$\begin{aligned} c(x,y) &= \lambda_1\lambda_2 - k \cdot (\lambda_1+\lambda_2)^2 \\ &= \det(M(x,y)) - k \cdot [\mathrm{trace}(M(x,y))]^2 \end{aligned} \quad (2)$$

The score c is large when both eigenvalues are large. The second term suppresses candidate pixels with only one large eigenvalue, and the larger the factor k, the more pixels with two medium eigenvalues are preferred over pixels with one small and one large eigenvalue. Subsequently, 8-neighborhood non-maximum suppression is applied and all pixels with a response of less than a predefined percentage θ of the maximum response encountered, $c(x,y) < \theta \cdot \max_{x,y}\{c(x,y)\}$, are ignored.

3.2 Shi-Tomasi's "Good Features To Track"

Based on a theoretical analysis of which features will be "good to track," Shi and Tomasi [29] derive an image motion model for affine motion and pure translation, which they use for tracking and monitoring the tracked features. For tracking, they suggest using the translation model, where the matrix involved is equivalent to M (Eq. (1)). With the same reasoning as above, the eigenvalues λ_1, λ_2 of M are computed and a candidate point is accepted if

$$c(x,y) = \min(\lambda_1,\lambda_2) > \lambda_\theta := \theta \cdot \max_{x,y}\{c(x,y)\} \quad (3)$$

Compared to the Harris score (Eq. (2)) this requires an additional square root operation per pixel.

3.3 Difference of Gaussians (DoG)

The approach of detecting local extrema of the image filtered with differences of Gaussians (DoG) was introduced by Lowe [17, 18] as part of SIFT.

To achieve invariance against changes in scale, the detector builds a pyramid of images by convoluting the image I with differences of Gaussians at different scales σ:

$$\begin{aligned} DoG_{k,\sigma}(x,y) &= G(x,y,k\sigma) - G(x,y,\sigma) \\ &= \frac{1}{2\pi(k\sigma)^2} \exp\left\{-\frac{x^2+y^2}{2(k\sigma)^2}\right\} - \frac{1}{2\pi\sigma^2} \exp\left\{-\frac{x^2+y^2}{2\sigma^2}\right\} \end{aligned} \quad (4)$$

In practice, this is done by first convoluting with the Gaussian kernels $G(\sigma)$ and then computing differences of the resulting images:

$$\begin{array}{lllll} D_0 &= I * DoG_{k,\sigma_0} &= I * G(k\sigma_0) &- I * G(\sigma_0) \\ D_1 &= I * DoG_{k,k\sigma_0} &= I * G(k^2\sigma_0) &- I * G(k\sigma_0) \\ &\vdots \end{array}$$

Lowe [18] shows that DoG_σ is an approximation to the Laplacian of Gaussians $(\sigma\nabla)^2 G$, which creates the ideal scale space [16]. As feature points, the algorithm selects local extrema, which are found by comparing each sample to its eight neighbors in the current image D_n and the 18 neighbors "above" (in D_{n-1}) and "below" (in D_{n+1}). The feature point locations are then refined to subpixel accuracy by fitting a parabola to the sample point and its immediate neighbors [5]. Feature points with low contrast, i.e., $|D(\hat{x})| < \theta_{\mathrm{contr}}$, where $\hat{x} = (x,y,\sigma)^T$ is the refined extremum location, are rejected. The ratio of the principal curvatures are estimated using the eigenvalue approach from Harris and Stephens [10] (cf. Section 3.1), and feature points with an "edge response," i.e., where the ratio of the two principal curvatures is larger than a threshold θ_{edge}, are rejected as well.

3.4 Fast Hessian

Bay et al. [4] developed Speeded Up Robust Features (SURF) based on the observation that some approximations can drastically decrease the time of computation without sacrificing much accuracy, if any. A key element to speed up computation is the usage of integral images as introduced by Viola and Jones [30] :

$$I_{int}(x,y) = \sum_{i=0}^{x}\sum_{j=0}^{y} I(i,j) \quad (5)$$

Once I_{int} is computed, filtering the image with a box filter takes, at any given point, only four additions regardless of the size of the filter.

SURF's detector, named Fast Hessian [4], is based on the determinant of the Hessian matrix, which at scale σ is defined as follows:

$$H(x,y,\sigma) = \begin{bmatrix} \frac{\partial^2}{\partial x^2} G(\sigma) * I(x,y) & \frac{\partial}{\partial x}\frac{\partial}{\partial y} G(\sigma) * I(x,y) \\ \frac{\partial}{\partial x}\frac{\partial}{\partial y} G(\sigma) * I(x,y) & \frac{\partial^2}{\partial y^2} G(\sigma) * I(x,y) \end{bmatrix} \quad (6)$$

As convolution with the Gaussian second order derivatives is very costly especially for higher scales, Bay et al. approximate them by filters that are composed of simple box filters and can therefore be computed in constant time using the integral image. The computed candidate score then is

$$\begin{aligned} c(x,y,\sigma) &= D_{xx}(\sigma) \cdot D_{yy}(\sigma) - (0.9 D_{xy}(\sigma))^2 \\ &\approx \det[H(x,y,\sigma)] \end{aligned} \quad (7)$$

where D_{xx}, D_{xy} and D_{yy} are the results of convoluting the image with the filters depicted in Fig. 1, and the factor 0.9 helps approximate $\det[H(x,y,\sigma)]$ more closely. 3x3x3-neighborhood non-maximum suppression and subpixel refinement are then applied as for the DoG detector. Likewise, candidates with c below a certain threshold are rejected. To speed up the computation, one may optionally increase the sampling intervals, i.e., compute c only for every nth pixel [4].

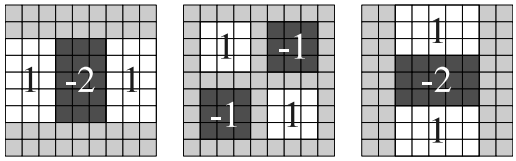


Figure 1: Filters composed of box filters as used by Fast Hessian as approximations to second order derivatives of Gaussians. Left to right: filters for obtaining D_{xx}, D_{xy}, D_{yy}. Weights of black and white regions as denoted, grey regions have weight zero. Figure adapted from Bay et al. [4].

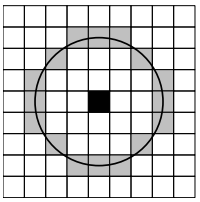


Figure 2: Bresenham circle. The black point is the current candidate point p, the 16 grey points are the discretized approximation of the outlined circle around it. Figure adapted from Rosten and Drummond [26].

3.5 Features from Accelerated Segment Test (FAST)

Rosten and Drummond [25, 26] developed a high-speed corner detector which they coined FAST, for Features from Accelerated Segment Test. The algorithm operates on a discretized circle around a candidate point p as shown in Fig. 2. p is classified as a corner if there exists a contiguous arc of at least nine pixels that are all brighter or all darker than p by a threshold t. The algorithm was further accelerated by training a decision tree to test as few pixels as possible for classifying a candidate pixel as corner or non-corner. With this decision tree, only 2.26 pixels are tested for each candidate on average, whereas with the naïve algorithm, 2.8 are tested [26].

In contrast to all aforementioned detectors, detection with the FAST algorithm does not inherently provide a measure of the "strength" of a feature. In order to apply non-maximum suppression, the following score is computed for each candidate point:

$$c(p) = \max\left\{\sum_{q\in S_+} |I_q - I_p| - t, \sum_{q\in S_-} |I_q - I_p| - t\right\} \qquad (8)$$

where S_+ is the subset of pixels on the circle that are brighter than p (by t) and S_- the subset of pixels that are darker than p (by t) [26].

3.6 Center-Surround Extrema (CenSurE)

Like Lowe [18]'s DoGs, the filters designed by Agrawal et al. [1] aim at approximating a Laplacian, though simplified further: In the first step, the filter is reduced to a bi-level filter, i.e., with filter values -1 and 1. Approximating a Laplacian then yields a torus-shaped filter kernel as depicted in Fig. 3 left. As this filter is computationally rather expensive, three approximations are proposed, each getting less symmetric, but easier to compute (Fig. 3 right).

As mentioned above, box filters can be efficiently computed at any scale using an integral image. For octagons and hexagons, Agrawal et al. [1] propose the use of *slanted* integral images, where the sum stored at each pixel (x,y) represents the sum of the trapezoidal area above (x,y). Octagons and hexagons can then be decomposed in a few trapezoids and thus can also be efficiently computed at any scale.

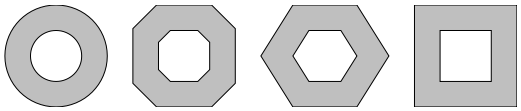


Figure 3: CenSurE's Bi-Level Filters. The circle (left) is the ideal, fully symmetric bi-level approximation of the Laplacian; from left to right, the approximations are coarser (less symmetric), but easier to compute: octagon, hexagon, box. Figure adapted from Agrawal et al. [1].

3x3x3-neighborhood non-maximum and edge response suppression are applied in a similar fashion as in the DoG and Fast Hessian detectors. Agrawal et al. [1] conclude that the octagon filter represents the best trade-off between stability and performance.

4 Testbed Design & Evaluation Setup

4.1 Establishing ground truth

To evaluate algorithms on images taken with a moving camera, ground truth information is needed, specifying which point x_j in frame j corresponds to point x_i in frame i. For general 3D scenes, this is very difficult to obtain without a 3D model of the scene. Therefore, most existing evaluations [20, 21, 26, 27, 31] use planar or near-to-planar scenes, where x_i and x_j are related by a homography $H_{ij}(q) \in \Re^{3x3}$ [11, 27]:

$$x_j = H_{ij}(q) \cdot x_i \qquad (9)$$

Here, $x_{i/j}$ are in homogeneous coordinates: $x_i = (x,y,1)^T$. Solving for H_{ij} may be done by projecting a known pattern onto a static scene [27] or indicating reference points manually [26, 31].

One option to obtain a reference coordinate frame is using fiducial markers [9]. However, they occlude a significant part of the image, rendering it unavailable for the evaluation, require a minimum viewing angle for detection, and their detection uses algorithms similar to the ones to be evaluated, which potentially biases the result.

For this work, we fabricated a precisely milled acrylic glass frame which holds the planar texture and four bright red balls (15 mm diameter) as markers (chosen as markers as they look the same from any direction), placed such that their center is in the plane of the texture. The markers are 13x10.5" apart, situated outside of the area for the texture, which is 11x8.5" (standard letter format). The area of the texture itself is 9.5x7", of which a margin of 0.75" is subtracted to avoid border effects. This leaves an area of 8x5.5" to be detected/tracked by the algorithms under evaluation.

We then implemented the following semi-automatic algorithm to detect and track markers and texture in the videos:

1. The position and size of the balls are manually indicated in the first first frame of the sequence.

2. An adaptive color model in HSV color space is initialized, which, applied to a new frame, produces a "probability map" that a given pixel belongs to the colored ball (Fig. 4 middle). The most probable positions of the balls are then identified using template matching with distance constraints.

3. The color model is adapted to the appearance of the balls in the new frame. For subsequent frames, a mixture model using both the model from the first and the previous frame is used to avoid long-term drift.

4. The position of each ball individually is refined using "inverse-compositional" image alignment [2] with 3 degrees of freedom (x, y, scale) between the current and the previous frame, and the homography between the current image and a canonical reference frame is computed (Fig. 4 right).

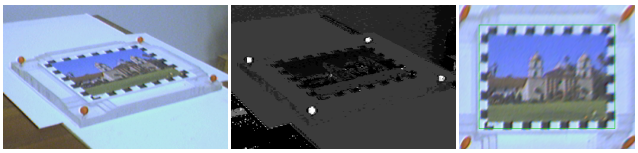


Figure 4: Adaptive color model: The image in the middle shows the "probability map" that the adaptive color model generated for the image on the left. The image is then warped into a canonical frame in which the balls form a rectangle (right image).

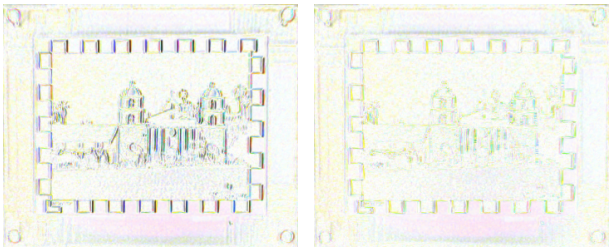


Figure 5: Result of image alignment: Difference between current frame and reference frame before (left) and after image alignment (right). Images are shown inverted (i.e. white = image difference 0) and with increased contrast. The alignment was substantially improved. The residuals are due to change in appearance (lighting effects, motion blur), sensor noise and interpolation artifacts.

5. Finally, the homography is refined using image alignment between the reference frame and the current frame (Fig. 5).

This algorithm was embedded into an interactive tool allowing the user to inspect the result and correct and re-start the algorithm if needed.

Overall, this semi-automatic tracking system produced very stable warped video streams despite extreme viewing angles and motion blur with a manageable amount of manual labor. Examples of its output are depicted in Fig. 6.

4.2 Dataset

The testbed consists of 96 video streams, showing six different planar textures in 16 different motion patterns each, all recorded with a Unibrain Fire-i camera with a resolution of 640x480 pixels. The textures are shown in Fig. 7, and the motion patterns are as follows:

- **Unconstrained**: free movement of a hand-held camera, unconstrained except that the object of interest has to stay in the field of view. The motion is mostly smooth, some parts exhibit quick movements and motion blur. Fig. 8a shows a reconstruction of one of the flight paths (6x 500 frames).
- **Panning**: located about 1m from the object of interest, the camera slowly rotates perpendicular to its optical axis, effectively causing the object to move from sideways with very little distortion (6x 50 frames).
- **Rotation**: located about 1m from the object of interest, the camera rotates around its optical axis from 0° to 90°, resulting in in-plane rotation of the object (6x 50 frames).
- **Perspective distortion**: starting roughly perpendicular above the object, the camera goes down in an arc, resulting in perspective distortion (out-of-plane rotation) of the object, cf. the flight path shown in Fig. 8b (6x 50 frames).
- **Zoom**: the camera moves perpendicularly away from the object, from 60 cm to 130[±10] cm (6x 50 frames).

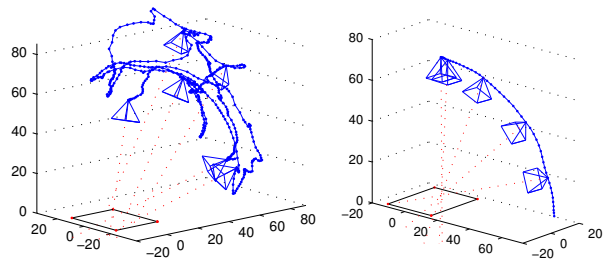


Figure 8: Flight paths of selected video streams, all axes in centimeters. (a) unconstrained (with texture "building"), (b) perspective distortion (with texture "bricks").

- **Motion blur**: mounted to a pan-tilt unit to precisely control its speed, the camera pans sideways with nine different speed settings. The speeds are the 1- to 9-folds of $0.02778°/s$, or, equivalently, 1- to 9-folds of about 5.1 pixels per frame (6x 9 sequences of length 13–89, varying length due to the varying time it takes for the object to disappear).
- **Static lighting**: the camera is statically mounted on a tripod and observes the scene under four different lighting conditions. The transition from one condition to the next is *not* included (6x 4x 20 frames).
- **Dynamic lighting**: the camera is statically mounted on a tripod and observes the scene transitioning from bright lighting to dark (a screen being moved in front of a soft lamp) and back (6x 100 frames).

The motion patterns "panning" through "zoom" were conducted with the camera mounted to an appropriately mechanically guided, but manually operated contraption, hence they are not exactly the same among the different textures and contain certain amounts of motion blur and jitter. As these conditions are exactly the same for all algorithms and we desire robustness against all kinds of motions, this does not affect algorithm comparison. All videos are encoded with the lossless HUFFYUV codec. In total, the dataset consists of 6889 frames.

The camera movement is reconstructed from the position of the target texture for the purposes of illustration (Fig. 8) and binning algorithm performance according to the relative change in camera positions.

To evaluate algorithms, the sequences can be used in consecutive order, simulating continuous tracking during smooth motion, as well as in randomly (or otherwise) sampled order, thus evaluating robustness against larger baseline distances. With respect to tracking, this simulates tracking recovery after failure or re-visiting a previously mapped scene.

4.3 Performance measures

The performance measures and characteristic values evaluated in this work are execution time, repeatability and number of features detected by a particular algorithm. Execution time is relevant as visual tracking usually is performed as a real-time task. While autonomous robots might be able to adapt their speed to ensure reliable tracking at lower frame rates [6], AR applications demand frame rates of around 30 Hz [12, 14].

As motivated earlier, the criterion that is most relevant for visual tracking as well as for other domains [4] is repeatability [27]:

$$\text{repeatability} = \frac{|\{(x_a \in S_i, x_b \in S_j) \,|\, \|H_i \cdot x_a - H_j \cdot x_b\| < \varepsilon\}|}{|S_i|} \quad (10)$$

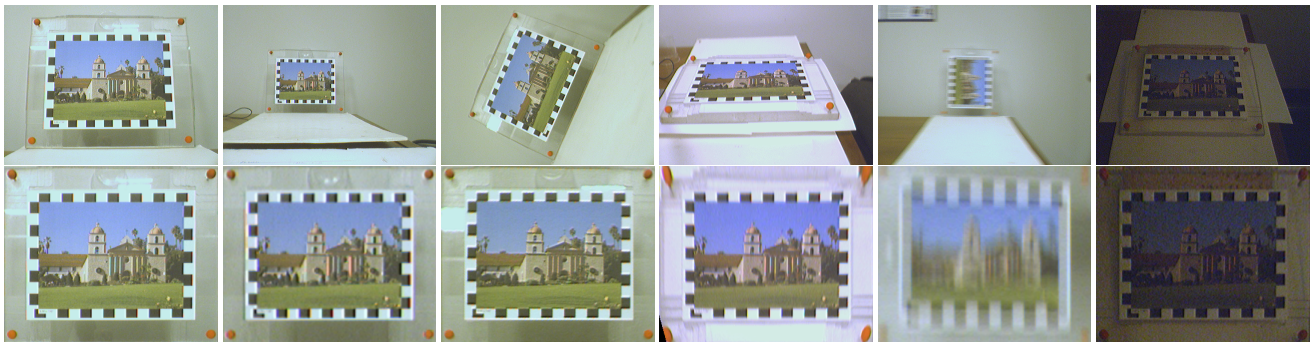


Figure 6: Top row: a few examples of the 6889 frames in the testbed; bottom row: the same frames, warped to the reference frame. These examples illustrate the challenges that the dataset encompasses: scale changes (first two images), rotation, perspective distortion, motion blur (here: fastest setting), lighting (here: darkest condition). The black-and-white pattern on the border was added to improve the image alignment result (cf. Section 4.1), algorithms to be evaluated may use only the area inside.

Figure 7: Used textures. From left to right: "wood," "bricks," "build," "paris," "mission," "sunset."

where S_i, S_j are the sets of points detected in frames i and j, respectively, and H_k is the homography between the kth frame and the reference frame. Even with perfect ground truth, a re-detected point will not be at exactly the same position as in the old frame: This is most obvious in the case of detectors that do not work with subpixel refinement, where the detected point has to "jump" to the next pixel at some point when the object is moved (a more detailed list of reasons may be found in [26]). For our evaluations, ε is set to 2.

It should be noted that Schmid et al. [27] define the denominator of Eq. (10) as $\min\{|S_i|, |S_j|\}$. However, this leads to a seemingly perfect repeatability of 1 if the detector returns, for example, 100 points in the first frame and only one point in the second frame, as long as this single point is among the 100 detected earlier, which is misleading. Our choice of denominator comes at the price of a non-symmetric performance measure, but ensures that it reflects the way that a tracking system operates: it tracks *from* one frame *to* the next, hence a non-symmetric performance measure seems acceptable.

Unfortunately, the repeatability criterion is biased in favor of algorithms that return many keypoints: as a trivial example, an algorithm that simply "detects" every single pixel has a repeatability of 1. One possibility is to adjust the parameters for the comparison so that different detectors return the same number of points (see, e.g., [4]), however, this assumes that all detectors work equally well for any number of features, which is not the case as the results in this work will show. Additionally, the number of features that a detector returns might be an important factor in the decision of which detector to use in a specific application, so instead of trying to equalize this value, we will report this number together with the repeatability.

Moreover, the repeatability criterion as given in Eq. (10) obviously allows the $|S_i|$ to be arbitrarily small: A single, perfectly stable point will produce a repeatability of 1. However, if we want to use the detector for a tracking system that estimates a pose with d degrees of freedom, we need at least $d/2$ correctly re-detected (and identified) points. In our setup, we require 4 points to correctly obtain the homography. Therefore, we amend Eq. (10) and set the score to 0 if the number of re-detected points is smaller than 4. This is especially important during the first stage of our evaluations, in which we determine the parameter configuration to be used (Section 5.1): The above correction ensures that maximizing repeatability also maximizes the chance of tracking success.

4.4 Implementation

The software framework for this evaluation was implemented in C++. partially using the OpenCV library[1]. For the evaluated detectors, the implementations were used:

- **Harris and Shi-Tomasi:** Implementations of these algorithms are provided with the OpenCV library. Additionally, Edward Rosten made available his implementation of the Harris detector, which was used in the comparison in [26]. The interface of the implementation in OpenCV only allows binary rectangular windows for $w(u,v)$ (see Section 3.1), so the code was modified to reveal hidden parameters. After this change, both implementations were found to be equivalent (i.e. detect the same points). For the comparison, OpenCV's implementation was used, as it was slightly faster.

- **DoG:** The original DoG+SIFT implementation is only available as a binary executable[2] which does not allow the flexibility needed for this evaluation. Instead, two publicly available SIFT implementations were used, from Rob Hess[3] and Andrea Vedaldi[4]. Due to the very complex algorithm and subtleties e.g. in the location refinement (cf. Section 3.3), the implementations are not exactly equivalent. The comparison shows results for Vedaldi's implementation, which was found to perform slightly faster and better.

- **Fast Hessian:** The original implementation of Bay et al. [4] is available as compiled library[5] and was used for this evaluation.

[1] http://sourceforge.net/projects/opencvlibrary/
[2] http://www.cs.ubc.ca/~lowe/keypoints/
[3] http://web.engr.oregonstate.edu/~hess/
[4] http://vision.ucla.edu/~vedaldi/code/siftpp/siftpp.html
[5] http://www.vision.ee.ethz.ch/~surf/

- **FAST:** The original implementation is available as source code[6].
- **CenSurE:** We use an implementation of WillowGarage, which is "[...] based on CenSurE with some modifications for increased speed and stability" [7].

For each of the algorithms, an interface was implemented that was derived from a common virtual base class. Implementing these interfaces required significant work in some cases, for example to convert image formats, reveal hidden algorithm parameters and re-parametrize implementations to make the parameters comparable between implementations (for different implementations of the same algorithm) or to the respective original paper. With these interfaces, the algorithms can be used interchangeably without changing the code. Execution time is measured only for the core part of the algorithms, that is, without the unifying interface needed for the evaluation or any initialization or memory allocation steps that could be moved to a pre-processing step.

Reported timings refer to an IBM Thinkpad T60 with a 1.83 GHz Dual Core CPU (only one core used for the computations) running Linux.

5 Results

5.1 Algorithm parameters

First, a total of 25 parameters of the six detector algorithms were evaluated in terms of their impact on the performance of the algorithm, namely, execution time, number of detected features and repeatability. We used the six "unconstrained" video streams, repeatability was evaluated for all consecutive frame pairs as well as 6x 5000 randomly selected frame pairs (5000 per texture). For each algorith, we set all parameters to their respective default values (as given by the original authors, if available), and then varied one parameter at a time across a large range of values. We made two exceptions from this rule: 1. for Harris and Shi-Tomasi, we first evaluated (all combinations of) gradient kernel and $w(u,x)$, and then evaluated the remaining parameters using the chosen new values, 2. for Fast Hessian, we first evaluated the number of octaves and the sampling step, and then evaluated the threshold for the new values. The complete results are shown in Figs. 9–14, the parameters that were evaluated are listed in Table 1 along with the chosen final values.

Detector	Parameter	Value
Harris	k	0.15
	threshold θ	0.001
	$w(u,v)$	$\exp\{-(u^2+v^2)/2\sigma^2\}$
	σ	2
	kernel size	2σ
	gradient kernel	Sobel
Shi-Tomasi	threshold θ	0.022
	$w(u,v)$	$\exp\{-(u^2+v^2)/2\sigma^2\}$
	σ	1.5
	kernel size	1.5σ
	gradient kernel	Sobel
DoG	octaves	4
	levels	3
	σ_0	1.6
	θ_{edge}	10
	θ_{contr}	0.02
Fast Hessian	octaves	4
	threshold	4
	sampling step n	1
	initLobe	3
FAST	threshold t	20
CenSurE	scales	6
	response threshold	6
	line th. (binarized)	10
	line th. (projected)	10

Table 1: Parameter values for the interest point detectors.

5.2 Comparison

Fig. 15 presents the obtained detector comparison results, specifically, the repeatability under various conditions. For all plots, repeatability was measured for all consecutive frames and/or a set of randomly chosen frames per texture. For the application of tracking, the former simulates continuous tracking, the latter tracking failure of a few frames or re-visiting a previous mapped scene (loop closure).

Fig. 15a shows the repeatability vs. the number of detected points (in the textured region of interest) achieved by the detectors by varying the respective thresholds. For all following experiments, the threshold was set to the value given in Table 1 (indicated by the respective marker in Fig. 15a).

Fig. 15b explore the tradeoff repeatability vs. execution time, both in the case of consecutive frames and random frame pairs. For the latter, the repeatability of all detectors decreases significantly, however, the performance of the "center-oriented" detectors, especially Fast Hessian, drops much farther. This result (averaged for all frame pairs in Fig. 15b) is broken down by texture in Fig. 16, and by the relative baseline distance in Fig. 15c. The trend can be analyzed further using the isolated motion patterns Fig. 15d–Fig. 15g: while perspective distortion is the biggest challenge for all detectors, the corner detectors maintain higher repeatability. As perspective distortion occurs frequently in many camera paths, including our pattern "unconstrained," this result is important.

The difficulty of the corner detectors to cope with motion blur Fig. 15h and the "wood" texture (Fig. 16) is apparent.

Fast Hessian performs best for small baseline distances for almost every texture (Fig. 16), as well as for panning motion (Fig. 15e) and motion blur (Fig. 15h).

For the different static lighting conditions, all detectors cope almost equally bad with the increased noise level (Fig. 15i). Fig. 15j is the only figure that shows repeatability over time, namely, while the lighting is changed (note that the output is smoothed with a median filter to make the figure legible). While all detectors show the same behavior on average, detailed analysis of lighting changes for each texture reveal differences. Interestingly, there are two textures in which Fast Hessian outperforms all others, but two others where it fails to find any features and is outperformed by all others.

6 Conclusions

We presented a comprehensive evaluation of six interest point detectors with respect to their application in real-time visual tracking. We used a testbed relevant to visual tracking, namely, video streams affected by motion blur and noise, with several thou- sand frames total, and explicitly evaluated a total of 25 algorithm parameters as well as comparing the six detectors to each other.

We believe that the results of our parameter evaluation are immensely helpful in "tuning" a particular algorithm for any particular application (note that the optimal parameters might be very different from the parameters we chose for the comparison), but also to gain insight into how the algorithms work. The comparison provides quantitative support for the decision of which detector to choose for designing new tracking applications.

[6] http://svr-www.eng.cam.ac.uk/~er258/work/fast.html
[7] http://pr.willowgarage.com/wiki/Star_Detector

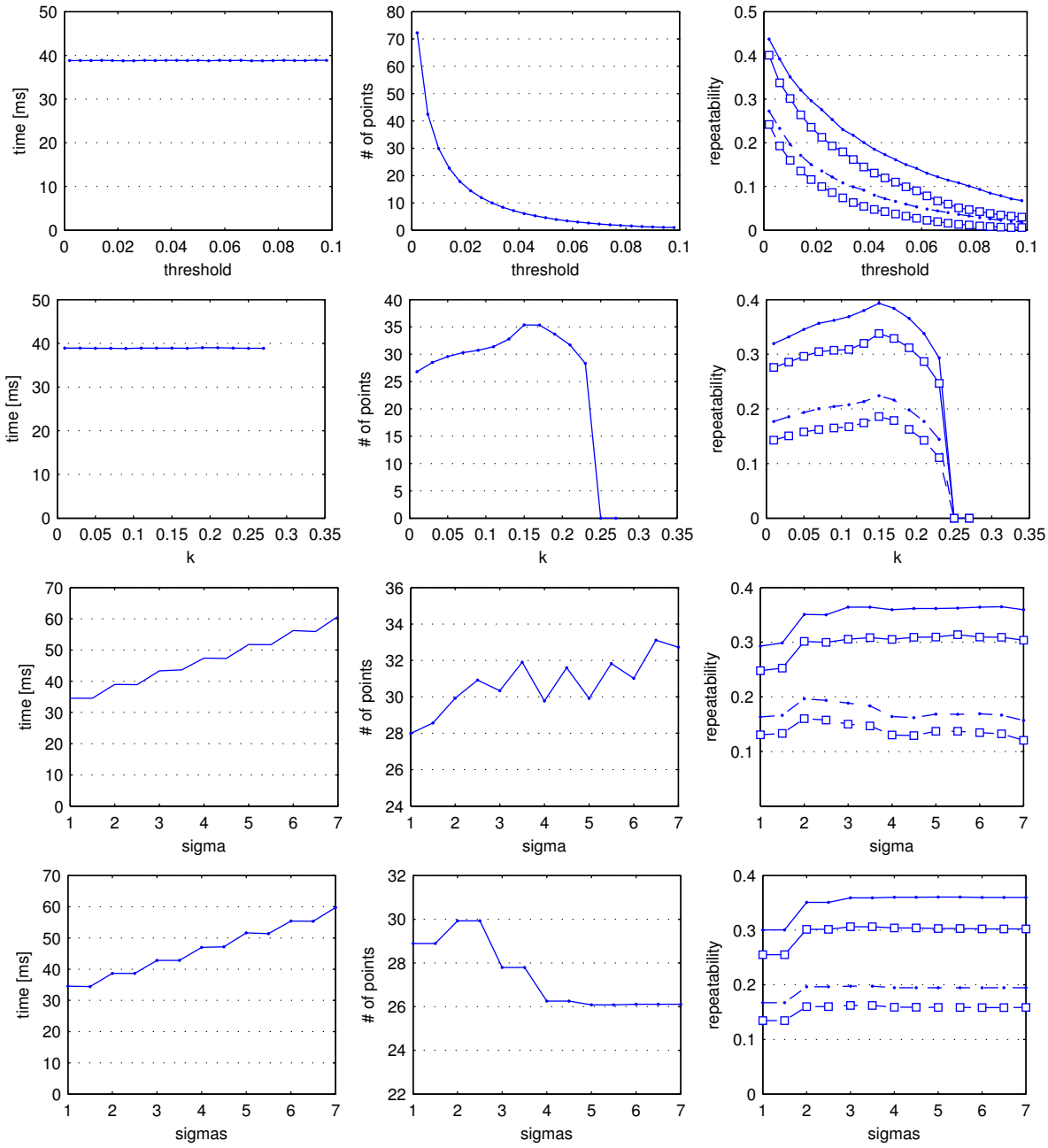


Figure 9: **Harris:** Impact of varying algorithm parameters on execution time (left), number of detected points (middle) and repeatability (right). In the right column, solid lines depict the repeatability of consecutive frames, whereas dashed lines depict the repeatability between random frame pairs. Dots depict the repeatability irrespective of number of points, and squares depict the repeatability after the correction described in the last paragraph of Section 4.3.

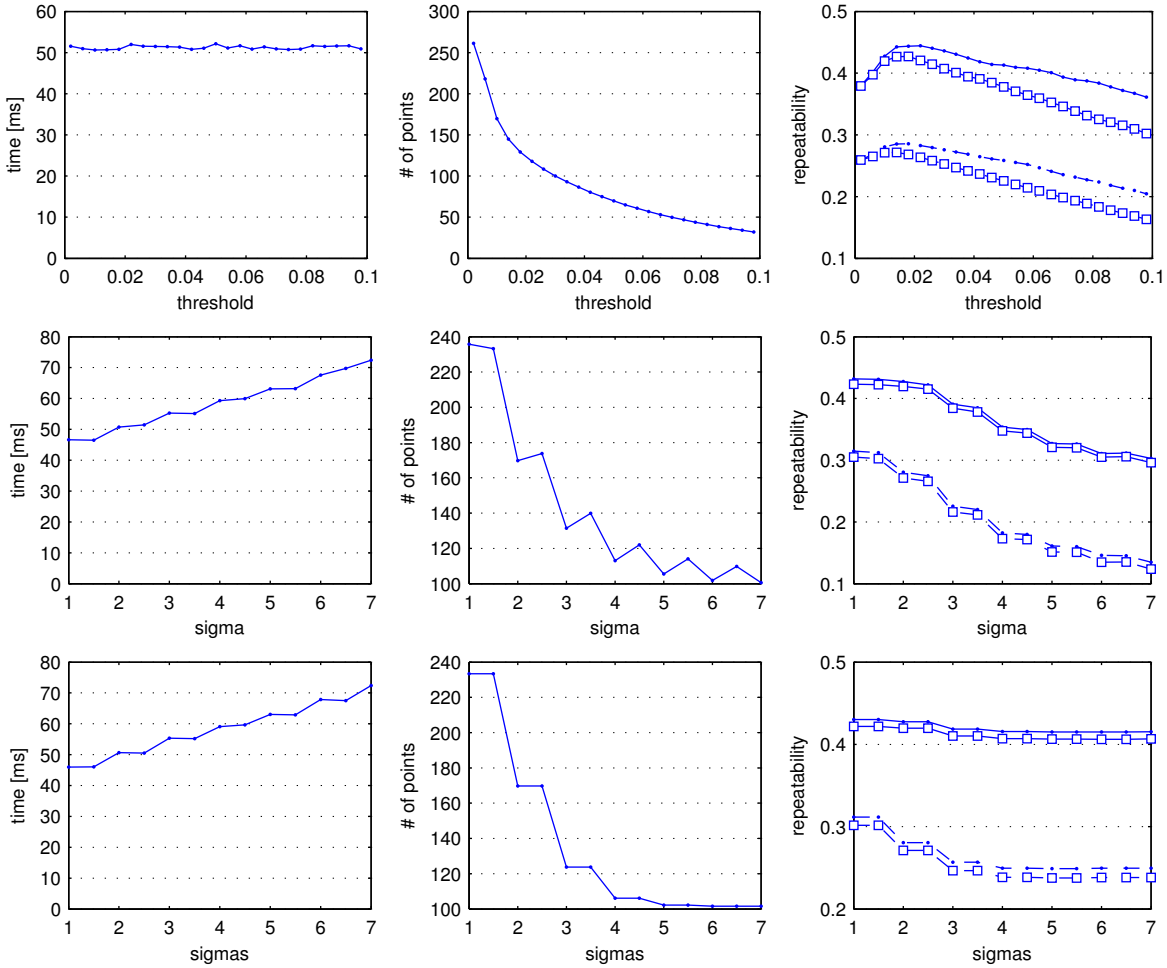


Figure 10: **Shi-Tomasi:** Impact of varying algorithm parameters on execution time (left), number of detected points (middle) and repeatability (right). Legend to right column see Fig. 9.

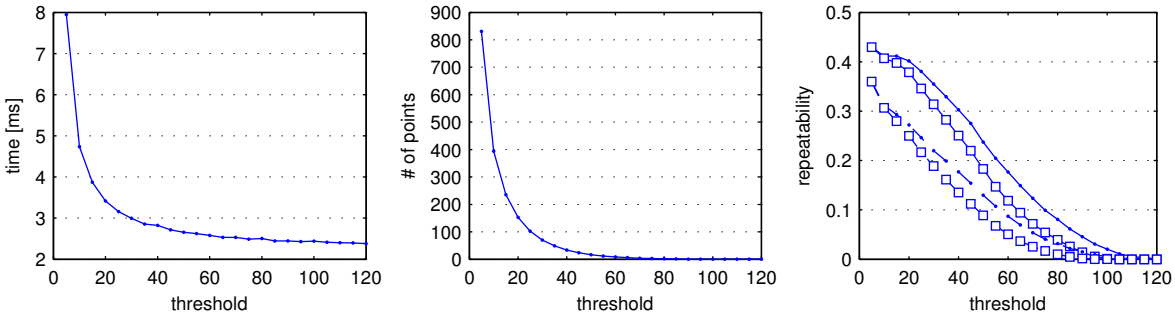


Figure 11: **FAST:** Impact of varying the threshold on execution time (left), number of detected points (middle) and repeatability (right). Legend to right column see Fig. 9.

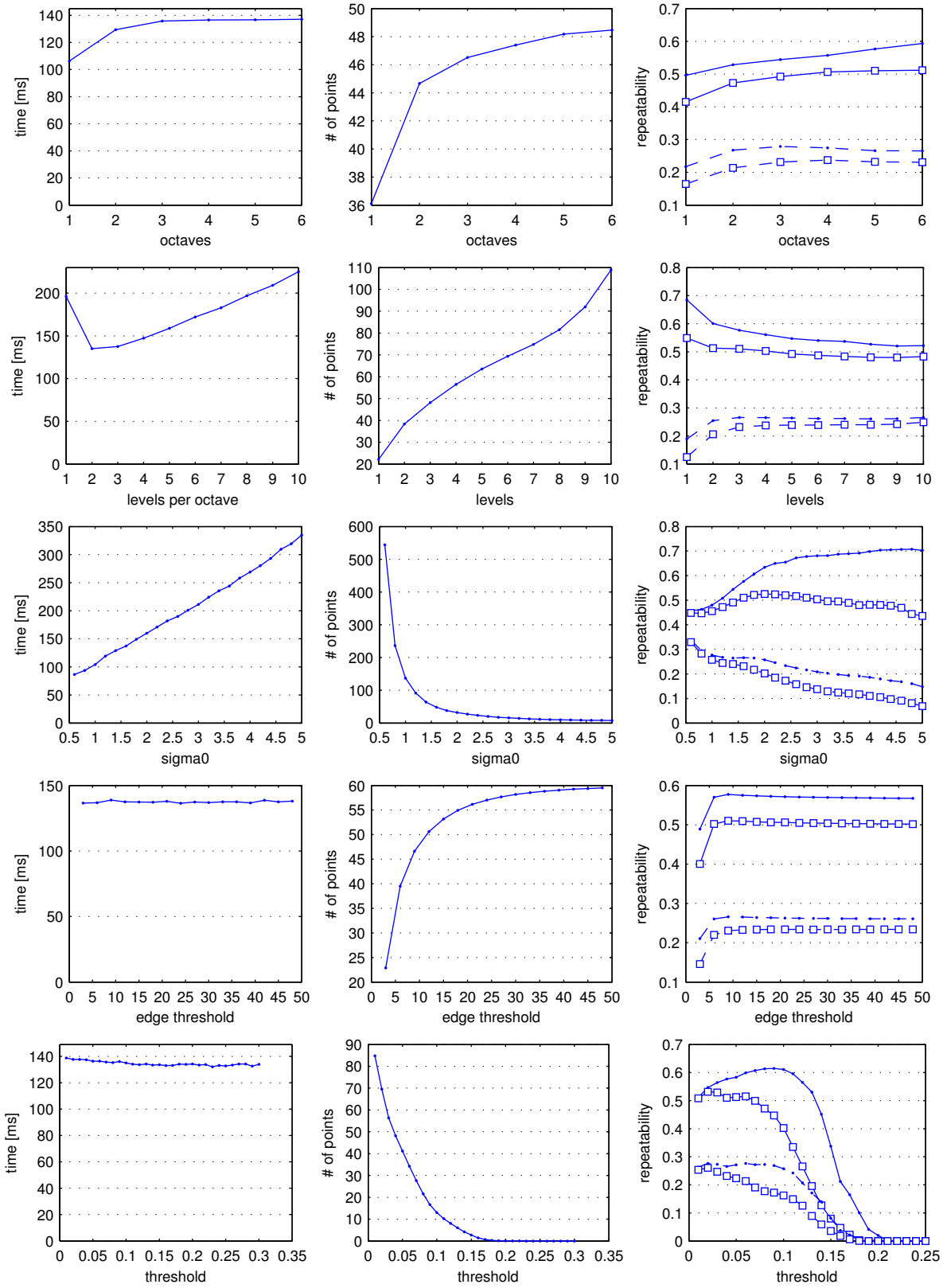


Figure 12: **Difference-Of-Gaussians:** Impact of varying algorithm parameters on execution time (left), number of detected points (middle) and repeatability (right). Legend to right column see Fig. 9.

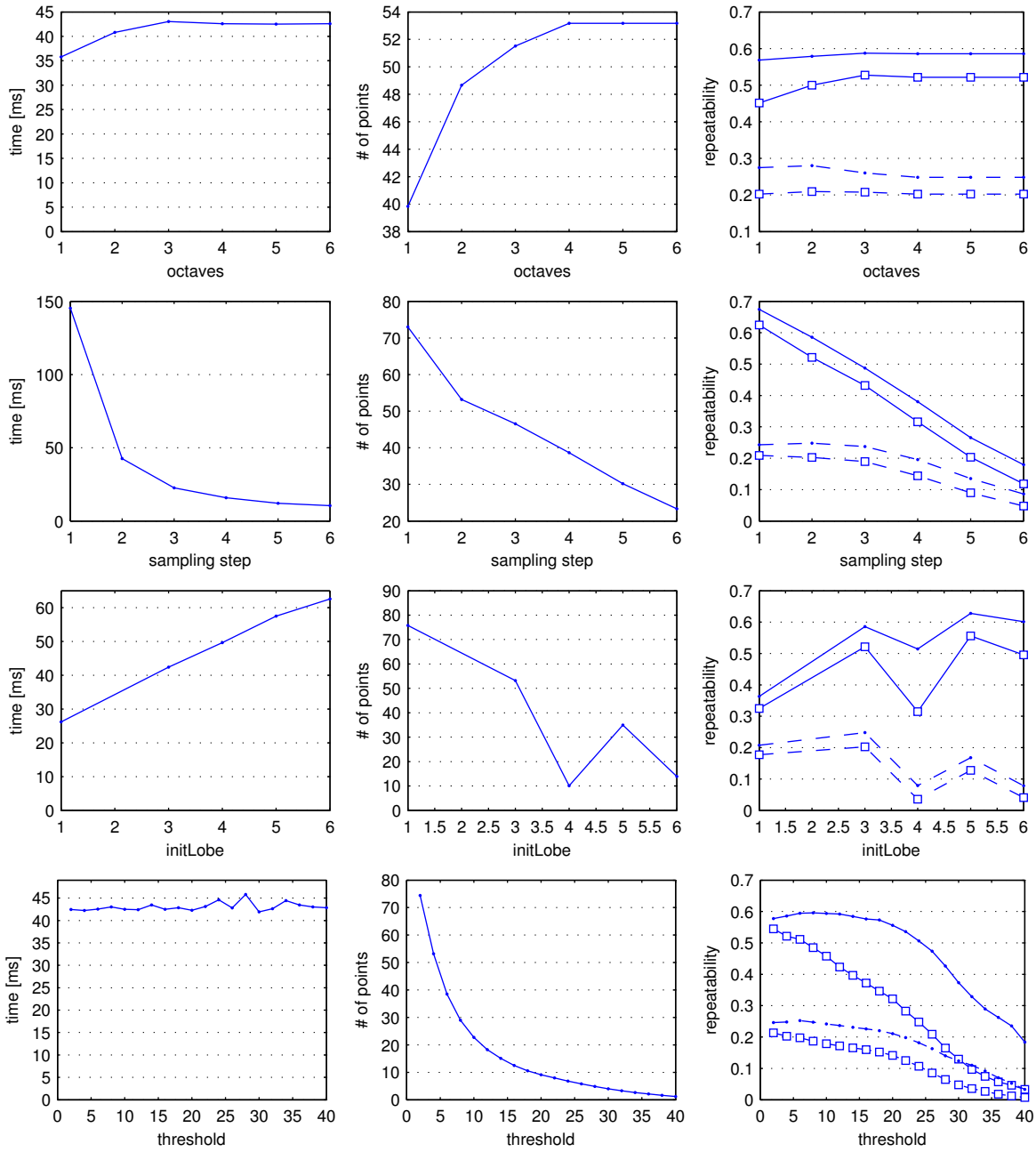


Figure 13: **Fast Hessian:** Impact of varying algorithm parameters on execution time (left), number of detected points (middle) and repeatability (right). Legend to right column see Fig. 9.

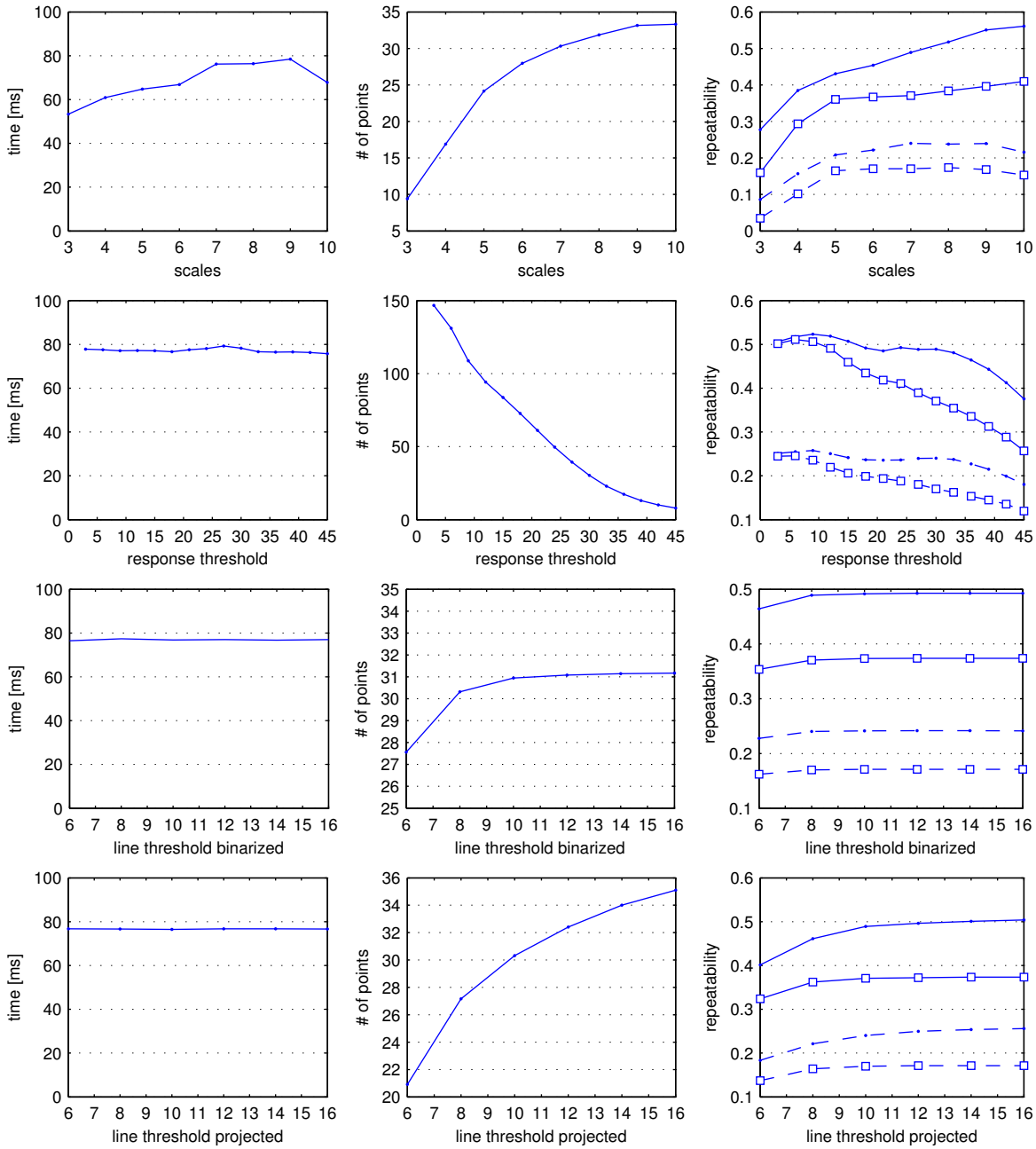
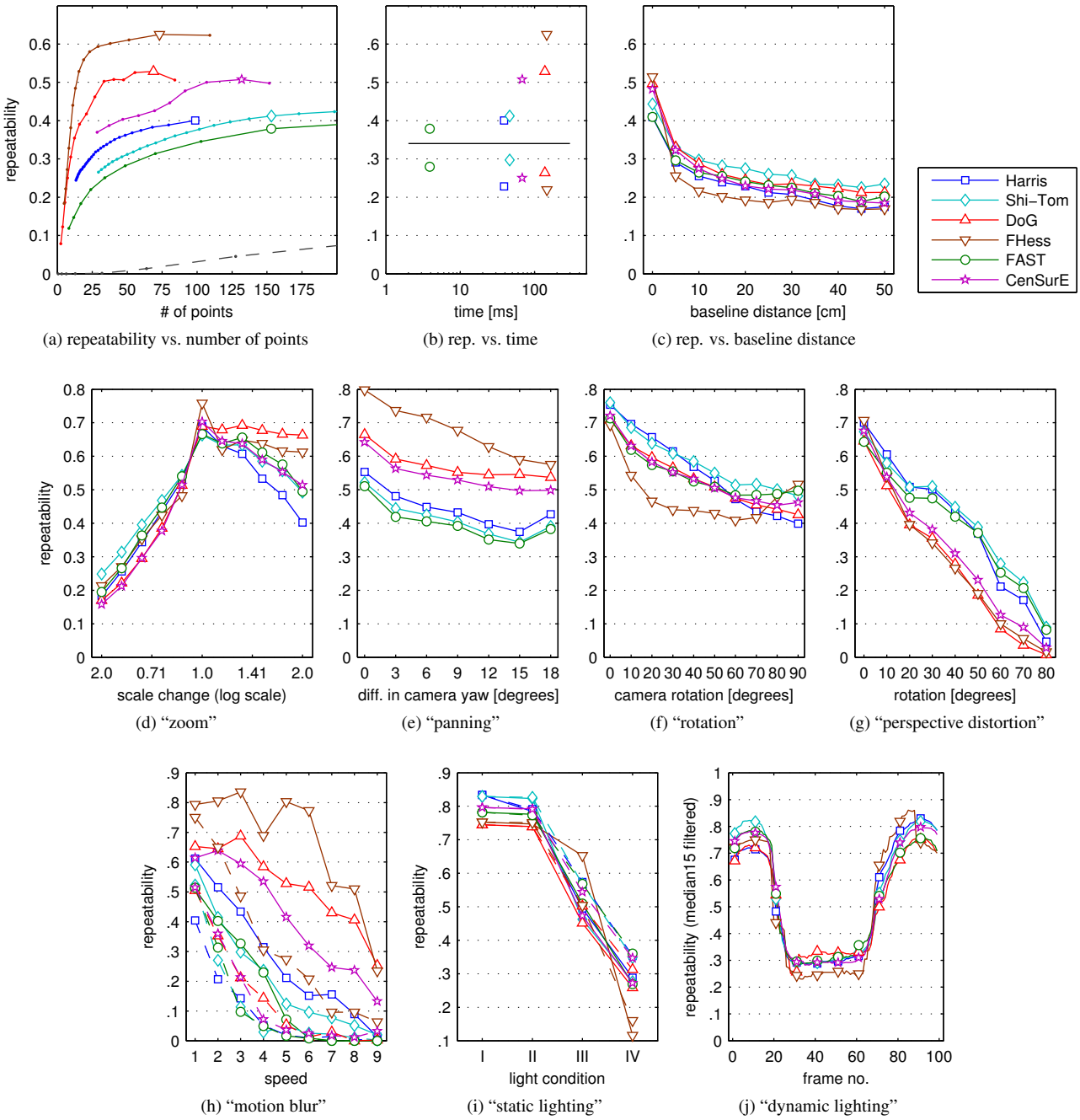


Figure 14: **CenSurE:** Impact of varying algorithm parameters on execution time (left), number of detected points (middle) and repeatability (right). Legend to right column see Fig. 9.



(a) repeatability vs. number of points (b) rep. vs. time (c) rep. vs. baseline distance

(d) "zoom" (e) "panning" (f) "rotation" (g) "perspective distortion"

(h) "motion blur" (i) "static lighting" (j) "dynamic lighting"

Figure 15: **Repeatability of detectors under various conditions**, all results averaged for all respective frame pairs and textures.

(a-c) on motion pattern "unconstrained," (a) Rep. on consecutive frames vs. number of detected points (in the region of interest), varying the respective threshold parameter. For comparison, the dashed gray line indicates the repeatability of randomly selected points, thus clearly visualizing the criterion's bias. (b) Rep. vs. execution time, for consecutive frames (above line) and 5000 pairs of randomly selected frames per texture (below line). (c) Rep. on random frame pairs as a function of the baseline distance between the two frames. (d-g) Rep. as function of geometric changes. For each figure, 5x 500 random frame pairs (500 per texture) of the specified motion pattern were evaluated and then binned and averaged according to the relative change in the camera's position between the two frames. (h) Rep. in the case of motion blur, both during motion (solid lines) and compared to the first, still frame (dashed lines). For absolute values of the speeds 1-9 refer to Section 4.2. (i) Rep. for different light conditions, both within one light condition (solid lines) and compared to the first condition (dashed lines). (j) Rep. for dynamic light changes. Here, the repetability is shown over time (i.e. frames) and filtered with a median filter of length 15 to make the figure legible (note that (j) is the only figure that shows repeatability for single frames).

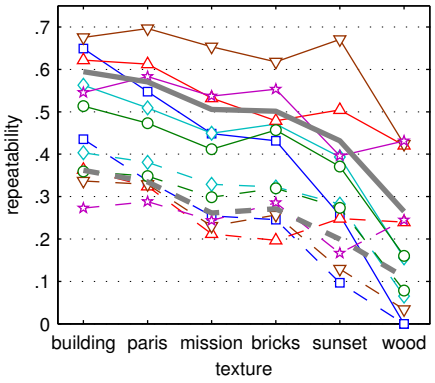


Figure 16: Repeatability on the different textures, motion pattern "unconstrained." Here, solid lines indicate performance for consecutive frames, dashed lines indicate performance for randomly selected pairs (5000 per texture). The bold gray lines indicate the average over all detectors, further legend as in Fig. 15.

REFERENCES

[1] M. Agrawal, K. Konolige, and M. R. Blas. CenSurE: Center surround extremas for realtime feature detection and matching. In D. A. Forsyth, P. H. S. Torr, and A. Zisserman, editors, *ECCV (4)*, vol. 5305 of *Lecture Notes in Computer Science*, pp. 102–115. 2008.

[2] S. Baker and I. Matthews. Equivalence and efficiency of image alignment algorithms. In *Proc. 2001 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR'01)*, vol. 1, pp. 1090 – 1097, December 2001.

[3] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. In *ICCV*, pp. 1–8. 2007.

[4] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features. In *Proc. 9th European Conf. on Computer Vision (ECCV'06)*, pp. 404–417, Graz, Austria, May 2006.

[5] M. Brown and D. Lowe. Invariant features from interest point groups. In *Proc. 2002 British Machine Vision Conf. (BMVC'02)*, 2002.

[6] Y. Cheng, M. W. Maimone, and L. Matthies. Visual odometry on the mars exploration rovers - a tool to ensure accurate driving and science imaging. *IEEE Robotics & Automation Magazine*, 13(2):54–62, 2006.

[7] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007.

[8] E. Eade and T. Drummond. Edge landmarks in monocular SLAM. In *Proc. 17th British Machine Vision Conf. (BMVC'06)*, vol. 1, pp. 7–16, Edinburgh, September 2006.

[9] M. Fiala. ARTag, a fiducial marker system using digital techniques. In *Proc. 2005 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, pp. 590–596, Washington, DC, USA, 2005.

[10] C. Harris and M. Stephens. A combined corner and edge detector. In *Proc. 4th ALVEY Vision Conf.*, pp. 147–151, 1988.

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

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

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

[14] T. Lee and T. Höllerer. Hybrid feature tracking and user interaction for markerless augmented reality. In *Proc. 2008 IEEE Virtual Reality Conf. (VR'08)*, pp. 145–152, March 2008.

[15] S. Lieberknecht, S. Benhimane, P. Meier, and N. Navab. A dataset and evaluation methodology for template-based tracking algorithms. In *ISMAR*, 2009.

[16] T. Lindeberg. Scale-space theory: A basic tool for analysing structures at different scales. *Journal of Applied Statistics*, 21(2):224–270, 1994. (Supplement on Advances in Applied Statistics: Statistics and Images: 2).

[17] D. G. Lowe. Object recognition from local scale-invariant features. In *Proc. 1999 IEEE Intl. Conf. on Computer Vision (ICCV'99)*, pp. 1150–1157, Corfu, 1999.

[18] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Intl. Journal of Computer Vision*, 60(2):91–110, November 2004.

[19] K. Mikolajczyk and C. Schmid. Scale & affine invariant interest point detectors. *Intl. Journal of Computer Vision*, 60(1):63–86, 2004.

[20] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, Oct. 2005.

[21] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. van Gool. A comparison of affine region detectors. *Intl. Journal of Computer Vision*, 65(7):43 – 72, November 2005.

[22] H. Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical Report CMU-RI-TR-80-03, Robotics Institute, Carnegie Mellon University, September 1980.

[23] P. Moreels and P. Perona. Evaluation of features detectors and descriptors based on 3D objects. *Intl. Journal of Computer Vision*, 73 (3):263–284, 2007.

[24] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry. *Proc. 2004 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR'04)*, 1:652–659, July 2004.

[25] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *Proc. 2005 IEEE Intl. Conf. on Computer Vision (ICCV'05)*, vol. 2, pp. 1508–1511, October 2005.

[26] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *Proc. 2006 European Conf. on Computer Vision (ECCV'06)*, vol. 1, pp. 430–443, May 2006.

[27] C. Schmid, R. Mohr, and C. Bauckhage. Evaluation of interest point detectors. *Intl. Journal of Computer Vision*, 37(2):151–172, 2000.

[28] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 1:519–528, 2006.

[29] J. Shi and C. Tomasi. Good features to track. In *Proc. 1994 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR'94)*, pp. 593–600, 1994.

[30] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR'01)*, vol. 1, p. 511, Los Alamitos, CA, USA, 2001.

[31] K. Zimmermann, J. Matas, and T. Svoboda. Tracking by an optimal sequence of linear predictors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31:677–692, 2008.