In-Depth Evaluation of Popular Interest Point Detectors on Video Streams

Steffen Gauglitz Tobias Höllerer

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

Technical Report 2009-08

ABSTRACT

We present an in-depth evaluation of popular interest point detectors, which, in contrast to existing evaluations, is targeted towards the application in visual tracking and augmented reality. In particular, candidate algorithms, testbed, and performance criteria are chosen with respect to the application of visual tracking. We evaluate the impact of individual algorithm parameters and present results in terms of repeatability, number of features detected, and computation time. We also describe our method to semi-automatically generate ground truth in detail.

1 INTRODUCTION

Visual tracking is a core component for visual odometry [20, 26, 6], visual Simultaneous Localization and Mapping (SLAM) [7] and augmented reality (AR) [13]. While some of the underlying techniques and algorithms have been known for a longer time, visual tracking at the framerates needed by AR is a fairly young endeavor, enabled by the rapid increase in computation power of modern hardware and the availability of compact and cheap cameras.

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 [29, 22, 21, 25] 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 most visual tracking applications in general and AR in particular.

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 and similarity measures. 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 existing literature on evaluations of interest point detectors and explains the differences to our work. Section 3 reviews the detectors that are evaluated in this report. Section 4 describes the setup for the evaluation in detail and Section 5 presents the obtained results. Finally, Section 6 concludes.

Detector	Scale invariant	Subpixel accurate	Output	Ref.
Harris	no	no	(x,y)	[9]
Shi-Tomasi	no	no	(x,y)	[31]
DoG	yes	yes	(x,y,σ)	[17]
Fast Hessian	yes	yes	(x,y,σ)	[3]
FAST	no	no	(x,y)	[28]

Table 1: Tested interest point detectors. (x,y) = location, σ =scale.

2 EXISTING EVALUATIONS

Schmid et al. [29] compared interest point detectors on two still images for changes in rotation, viewpoint and illumination, as well as with artificially added image noise. They found that the Harris Corner Detector [9] outperformed other existing approaches in 2000. Mikolajczyk et al. [22] compared affine invariant detectors. The evaluation of Moreels and Perona [25] contains two novel contributions: they explored the performance of combinations of detectors and descriptors, and their testbed consists 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 that are tested. Execution time, a criterion crucial for designing real-time systems, receives no [29, 21] or only little [22, 25] attention: Moreels and Perona [25] mentioned execution times in the order of one second to one minute per image[1]; Mikolajczyk et al. [22] listed execution times between 0.5 seconds (MSERs of Matas et al. [19]) and and several minutes (salient regions of Kadir et al. [12]) per image[2], 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: the detectors and evaluated are suitable for use in real-time applications, the performance measures are chosen with respect to the application of visual tracking. Most notably, the testbed consists of video streams affected by noise and motion blur rather than high-resolution, low-noise photographs.

3 INTEREST POINT DETECTORS

The interest point detectors that are evaluated in this work are listed in Table 1. Many other detectors have been proposed in the literature. The reasons for the selection above are as follows: these detectors have been widely used, they were previously shown to outperform other detectors [29, 28, 3], and they are used in state-of-the-art visual tracking systems.

[1] 3 GHz PC, image size 1024x178 [25]
[2] Pentium 4 2GHz, image size 800x640 [22]

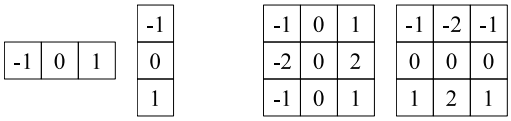


Figure 1: Filter kernels to compute image gradients I_x, I_y. Left: most elementary kernel, right: Sobel kernel.

In the following, we will briefly review these algorithms. In particular, we will explain the role of each of the algorithms' parameters, as their impact on the algorithm's performance is individually evaluated.

3.1 Harris Corner Detector

Based on Moravec's corner detector [24], Harris and Stephens [9] 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 [9] propose to use a Gaussian window:

$$w(u,v) = \exp\left\{-(u^2+v^2)/2\sigma^2\right\}$$

Matrix M is called the second-order moment matrix [25] and is an approximation to the auto-correlation function of image I (cf. Appendix A of [29]). 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, the following corner score $c(x,y)$ is used, which is derived based on λ_1, λ_2, but can be expressed without them:

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

Subsequently, 8-neighborhood non-maximum suppression is applied and all pixels with a response of less than a predefined threshold of the maximum response encountered, $c(x,y) < \text{threshold} \cdot \max_{x,y}\{c(x,y)\}$, are ignored.

Apart from the threshold and parameters k and $w(u,x)$, one also has to choose a kernel to compute the image gradients I_x and I_y. Figure 1 shows two possible choices: the most elementary kernel, which is suggested to use by Harris and Stephens [9], and the Sobel kernel (used, for example, in the OpenCV implementation). Furthermore, if using a per se infinite Gaussian window for $w(u,x)$, the kernel has to be truncated for practical reasons. As the decrease in influence is determined by σ, the kernel size is normally given in σ's.

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 [31] 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

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

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

3.3 Difference of Gaussians

As part of the Scale Invariant Feature Transform (SIFT), Lowe [16, 17] proposed to use the local extrema of the image filtered with differences of Gaussians (DoG): 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:

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

Doubling σ corresponds to a scale change of one *octave*, k determines the "scale resolution" of the image pyramid, i.e. the number of *levels* that are used per octave ($k=2 \rightarrow$ one level, $k=\sqrt{2} \rightarrow$ two levels etc.).

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., $|D(\hat{x})| < th_{contr}$, 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 [9], 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 th_{edge}, are rejected as well.

As the input image is smoothed before any feature points are detected, the highest spatial frequencies are ignored. To overcome this, the algorithm optionally upsamples the image by a factor of two at the beginning. The tracking systems that use DoG [15, 30] omit this step, as the execution time increases significantly.

3.4 Fast Hessian

Bay et al. [3] developed Speeded Up Robust Features (SURF) based on the observation that some approximations can drastically decrease the time of computation without sacrificing too much accuracy, if any. A key element to speed up computation is the usage of integral images as introduced by Viola and Jones [32]:

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

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.

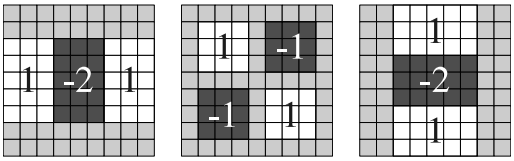


Figure 2: Filters composed of box filters as used by SURF as approximations to second order derivatives of Gaussians. Left to right: filters for obtaining D_{xx}, D_{xy}, D_{yy} at the lowest scale ($\sigma = 1.2$). Weights of black and white regions as denoted, grey regions have weight zero. Figure adapted from Bay et al. [3].

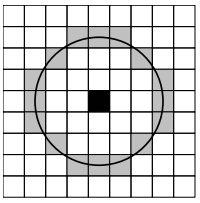


Figure 3: Bresenham circle. The black point is the current candidate point p, the 16 grey points are the discretized approximation of the outlined circle around it. Figure adapted from Rosten and Drummond [28].

SURF's detector, named Fast Hessian [3], 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 Figure 2, and the factor 0.9 helps approximate $\det[H(x,y,\sigma)]$ more closely. 26-neighborhood non-maximum suppression and subpixel refinement are then applied as for the DoG detector. Likewise, candidates with c below a predefined threshold are rejected. To speed up the computation, one may optionally increase the sampling intervals, i.e. compute c only for every second, third... pixel [3].

3.5 Features from Accelerated Segment Test (FAST)

Rosten and Drummond [27, 28] 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 Figure 3. p is classified as a corner if there exists a contiguous arc of at least 9 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, whereas with the naïve algorithm, 2.8 are tested [28].

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:

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

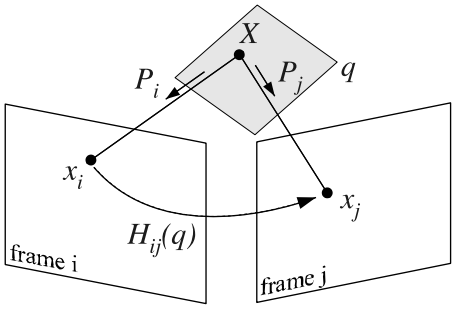


Figure 4: $x_i = P_i \cdot X$ and $x_j = P_j \cdot X$ are the projections of world point X onto camera frames i and j, respectively. For all X on a planar surface q, x_i and x_j are related by the homography $H_{ij}(q)$ [10, 29].

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) [28]. Note that FAST therefore only has a single parameter, threshold t.

4 Evaluation Setup

4.1 Ground truth

To evaluate the algorithms' performance 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 [21, 22, 28, 29] use planar or near-to-planar scenes, where x_i and x_j are related by a homography $H_{ij}(q) \in \Re^{3x3}$ [10, 29] (cf. Figure 4):

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

where $x_{i/j}$ are in homogeneous coordinates: $x_i = (x,y,1)^T$. Existing comparisons solve for H_{ij} by either projecting a known pattern onto a static scene [29] or aligning a small dataset of photographs by hand [28]. Neither is feasible for non-static video streams with several thousand frames. Moreels and Perona [25] describe a setup that is not restricted to planar scenes, but their setup is not feasible for arbitrary camera motion.

One option to obtain a reference coordinate frame is using fiducial markers (e.g., ARTag [8]). However, they occlude a significant part of the image, rendering it unavailable for the evaluation, and require a minimum viewing angle for detection. For this work, we used small ($\varnothing = 15mm$) bright red balls as markers (which look the same from any direction) and implemented the following semi-automatic algorithm to detect them in the images:

1. The user indicates the position of the balls to be tracked in the first frame of the sequence.

2. The computer initializes an adaptive color model in HSV color space (cf. Bradski [4] and the skin color model in Lee and Höllerer [14]). Applied to a new frame, this color model produces a "probability map" that a given pixel belongs to the colored ball (Figure 5 middle).

3. A template of the object to find (e.g., a "white" circle of appropriate size) is convoluted with the probability map. Together with distance constraints, the most probable positions of the balls are identified.

4. The adaptive color model learns 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.

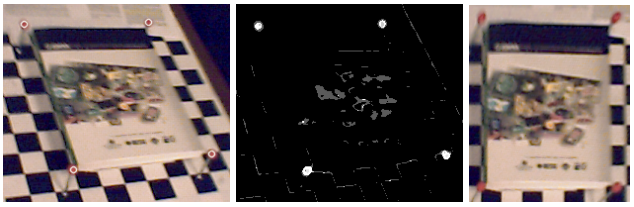


Figure 5: Adaptive color model. The image in the middle shows the "probability map" that the adaptive color model generated for the image on the left. The small white circles in the left image indicate the estimated positions of the balls. The image is then warped into a canonical frame in which the balls form a rectangle (right image).

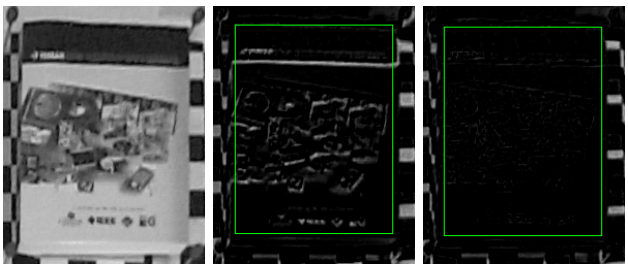


Figure 6: Image alignment. Images from left to right: warped current frame, difference to reference frame before image alignment, difference to reference frame after image alignment. The area outside the green rectangles is ignored both during alignment and when evaluating the detectors, as the computed homography is invalid here. The alignment was substantially improved, as the image inside the rectangle turned nearly completely black, indicating a very small image difference. The residuals are due to change in appearance (lighting effects), sensor noise and interpolation artifacts.

5. Using the positions of four balls, a homography is computed that warps the current image into a canonical frame (Figure 5 right).

6. The warp is refined using Lucas-Kanade image alignment [18, 1] between the first and the current frame (Figure 6).

The image alignment step works as follows: Given a template $T(x,y)$ and an image $I(x,y)$, it tries to find the parameters p for a warp $W(x,y;p)$ that minimize the following expression [2]:

$$\sum_{x,y}[I(W(x,y;p))-T(x,y)]^2 \tag{10}$$

For this problem, there exists a family of similar iterative algorithms which Baker and Matthews [2] classify in "forwards" vs. "inverse" and "additive" vs. "compositional". The original algorithm by Lucas and Kanade [18] ("forwards-additive" in above taxonomy) starts with an estimate p_0 of the parameters and iteratively solves for increments Δp: $p_i \leftarrow p_{i-1}+\Delta p$. The iteration stops if Δp or the error of Eq. (10) fall below predefined thresholds. Using a first-order Taylor expansion on $I(W(x,y;p+\Delta p))$ for the minimization of Eq. (10), the algorithm involves computation of the image gradients as well as the Jacobian $\partial W/\partial p$ of the warp [2]:

$$\frac{\partial W(x,y;p)}{\partial p} = \begin{pmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \cdots \end{pmatrix} \tag{11}$$

The homography H in Eq. (9) can be parametrized as non-linear warp W as follows [11]:

$$W_H(x,y;p) = \frac{1}{p_3x+p_6y+1}\begin{pmatrix} p_1x+p_4y+p_7 \\ p_2x+p_5y+p_8 \end{pmatrix} \tag{12}$$

Using this parametrization and the abbreviations

$$\xi = p_1x+p_4y+p_7 \tag{13}$$
$$\eta = p_2x+p_5y+p_8 \tag{14}$$
$$\zeta = p_3x+p_6y+1 \tag{15}$$

Eq. (11) becomes [11]:

$$\frac{\partial W_H(x,y;p)}{\partial p} = \frac{1}{\zeta}\begin{pmatrix} x & 0 & -x\frac{\xi}{\zeta} & y & 0 & -y\frac{\xi}{\zeta} & 1 & 0 \\ 0 & x & -x\frac{\eta}{\zeta} & 0 & y & -y\frac{\eta}{\zeta} & 0 & 1 \end{pmatrix} \tag{16}$$

With this Jacobian, the algorithm outline can be found in any of [18, 1, 2].

Applied to a whole image, this algorithm is too slow for real-time processing and can only correct for reasonably small image distortions, but it is perfectly suited as a refinement step in addition to the tracking system described above. Figure 6 illustrates the effect of this addition.

If any of the above steps produces a suspicious result (e.g., large jumps or differences in position or color distribution, the image alignment algorithm has not converged), the user is asked for assistance. Overall, this semi-automatic tracking system produced stable warped video streams. Examples of its output are depicted in Figures 6 and 7. For evaluation of the detectors, only the planar area inside the markers is used (green rectangle in Figure 6 right), not including the markers (which violate the assumption of an unprepared environment) and the surrounding area (for which the homographic warp is incorrect, as may be seen from the high intensity differences in Figure 6 right along the right border).

4.2 Testbed

The testbed consists of 36 different video streams, showing six different planar textures in six different motion patterns each, all recorded with a unibrain Fire-i camera with a resolution of 640x480 pixels. The textures are shown in Figure 8, 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. Figure 9a shows a reconstruction of one of the flight paths, Figure 7 shows a few frames together with the warped images that are used as ground truth (6x 300 frames).

- "translation": hovering about 1m above the object of interest, the camera slowly rotates perpendicular to its optical axis, effectively causing the object to move from the left to the right of the frame with very little distortion (6x 50 frames).

- "in-plane rotation": position roughly 1m above the object of interest, the camera is rotating around its optical axis from 0° to 90°, resulting in in-plane rotation of the object (6x 50 frames).

- "out-of-plane rotation": starting above the object, the camera goes down in an arc, resulting in out-of-plane rotation of the object, cf. the flight path shown in Figure 9b (6x 50 frames).

- "zoom": the camera moves perpendicularly away from the object (6x 50 frames).

- "motion blur": the camera sways rather quickly above the object, resulting in heavy motion blur (6x 50 frames).

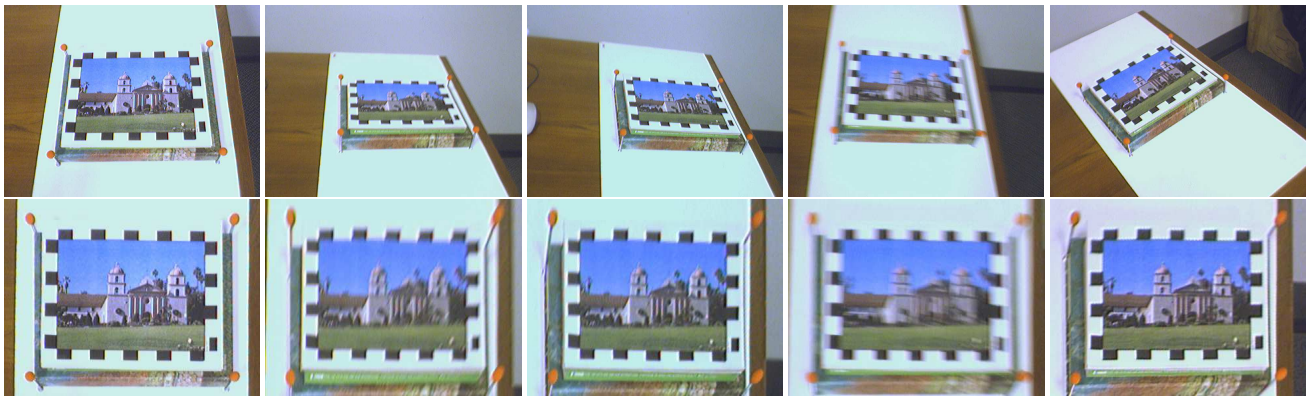
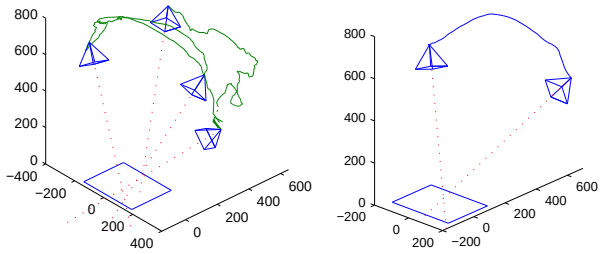


Figure 7: Top row: a few frames of one of the video streams; bottom row: the same frames, warped to the reference frame. This sequence illustrates some of the challenges the detectors have to face: scale change, rotation, motion blur, small viewing angle. The black-and-white pattern on the border was added to help the image alignment algorithm (cf. Section 4.1). For the evaluation, only the area inside is used.

Figure 8: Used textures. From left to right: “wood”, “bricks”, “build”, “paris”, “mission”, “sunset”.



(a) “unconstrained” (b) “out-of-plane rotation”

Figure 9: Flight paths of selected video streams, all axes in millimeters.

It should be noted that all motions were conducted with a handheld camera and are therefore approximate, i.e. the motion pattern “rotation” will also contain a certain amount of translational movement. As these conditions are exactly the same for all algorithms and we desire robustness against all kind of motions, this does not affect the comparison. The camera movement is reconstructed from the known position of the markers for the purposes of illustration (Figure 9) and binning the results according to the change in camera positions.

The algorithms’ performance was measured between consecutive frames, simulating continuous tracking during smooth motion, as well as between randomly chosen frames of a sequence, simulating tracking recovery after failure or re-visiting a previously mapped scene.

It turned out that the contrast of the texture “wood” was too low so that none of detectors was capable of finding repeatable features (to repeatedly detect the weak features, the respective thresholds had to be set so low that they detected a lot of noise and became unusable especially for the other textures), so it was not used for the evaluation. From the other streams, a total of 39 frames were ignored because the ground truth algorithm (cf. Section 4.1) failed to converge. Therefore, all algorithms were run on 2711 different frames, and—including the randomized order—evaluated for about 30,000 frame pairs.

4.3 Performance measures

The criterion that is most relevant for visual tracking as well as for other domains [3] is repeatability [29][3]:

$$\text{repeatability} = \frac{|\{(x_i, x_j) \,|\, \|H_{i1} \cdot x_i - H_{j1} \cdot x_j\| < \varepsilon\}|}{|S_i|} \qquad (17)$$

where S_i, S_j are the sets of points detected in frames i and j, respectively, $x_i \in S_i, x_j \in S_j$, and H_{i1} is the homography between the ith and the first frame (Eq. (9)). Even with perfect ground truth, a re-detected point will not be at exactly the same position as in the old frame [28]: 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. For the comparisons in Section 5, ε is set to 2.

Unfortunately, the repeatability criterion is biased in favor of algorithms that return many keypoints, as will be visualized in Figure 15a: as a trivial example, an algorithm that simply “detects” every single pixel has a repeatability of 100%. Alternative measures have been proposed [23], but they are specific to corner detectors, rely on subjective decisions and are not relevant for visual tracking as an application. In some studies, the algorithm parameters are adjusted so that all detectors return the same number of points (e.g., [3]), however, this assumes that all detectors work equally well for any number of features, which is not the case as Section 5 will show: The optimal number of points can differ by an order of magnitude for different detectors. Moreover, the number of features

[3] In the definition of Schmid et al. [29], the denominator of Eq. (17) is defined as $\min\{|S_i|, |S_j|\}$. This however leads to a seemingly perfect repeatability of 1 if the detectors returns 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.

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.

4.4 Implementation

For the evaluated algorithms, the implementations of the original authors were used where available:

- **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 [28]. 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 (SIFT):** The original SIFT implementation is only available as a binary executable[4] which does not allow the flexibility needed for this evaluation. Instead, two publicly available SIFT implementations were used, from Rob Hess[5] and Andrea Vedaldi[6]. Due to the very complex algorithm and subtleties e.g. in the location refinement (cf. Section 3.3), the implementations are not exactly equivalent. Initially, Hess' code was faster due to more efficient convolution functions. After using the same functions in Vedaldi's code, this implementation was slightly faster than Hess'. Also, its repeatability is slightly higher, so it was used for the comparison.

- **Fast Hessian (SURF):** The original implementation was provided as compiled library.

- **FAST:** The original implementation is available as source code[7].

The software framework for this evaluation was implemented in C++ and compiled with gcc 4.3.2. All experiments were conducted under Linux on an IBM Thinkpad T60 with 1.83 GHz Dual Core CPU. Execution time is measured only for the core part of the algorithms, that is, without the unifying interface and any initialization steps, e.g. format conversions, algorithm initialization and memory allocations.

5 Results

5.1 Algorithm parameters

First, a total of 20 parameters of the five 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 five "unconstrained" video streams, repeatability was evaluated for all consecutive frame pairs as well as 5x 3000 randomly selected frame pairs (3000 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 Figures 10-14, the parameters that were evaluated are listed in Table 2 along with the initial values and the chosen final values.

[4] http://www.cs.ubc.ca/~lowe/keypoints/
[5] http://web.engr.oregonstate.edu/~hess/
[6] http://vision.ucla.edu/~vedaldi/code/siftpp/siftpp.html
[7] http://svr-www.eng.cam.ac.uk/~er258/work/fast.html

Detector	Parameter	Initial value[a]	Final value[b]
Harris	k	0.06[c]	
	th	0.01[c]	
	$w(u,v)$	$\exp\left\{\frac{-(u^2+v^2)}{2\sigma^2}\right\}$	
	σ	2[c]	1.5
	kernel size	2σ[d]	1.5σ
	grad. kernel	[-1 0 1]	Sobel
Shi-Tomasi	th	0.01	0.05
		other parameters as for Harris	
DoG	octaves	*varies*[e]	2
	levels	3	
	σ_0	1.6	
	th_{edge}	10	
	th_{contr}	0.03	0.1
Fast Hessian	octaves	4	2
	threshold	4	24
	sampling	2	1
FAST	threshold	20[f]	40

[a] the initial value is the default value as given in the respective original paper, unless denoted differently.
[b] if a field is left blank, the respective initial value is used.
[c] no value provided in original paper, value from Schmid et al. [29].
[d] no value provided in original paper, value from implementation used in Rosten and Drummond [28].
[e] the default number of octaves is derived from the resolution of the image, see [17] for details. For image of size 640x480, the default value is 5.
[f] see http://svr-www.eng.cam.ac.uk/~er258/work/README-win32.

Table 2: Parameter values.

5.1.1 Strategy for choosing the "final" parameters

As a result of the thorough parameter analysis we faced the problem of choosing one set of parameters for the comparison. With three different criteria and several different conditions (image resolution, consecutive vs. randomized frames), there is no single optimal value. Instead, several points in parameter space can be considered Pareto-optimal. We followed the following guidelines to choose the values:

1. we want one set of parameters to work well for all conditions (i.e. both image resolutions and both consecutive and ranomized frames);

2. repeatability is considered the most important criterion, i.e. gains in repeatability justify longer execution time unless the gain is very small compared to the increase in time;

3. we assume that the default parameters have been carefully chosen (even if this might have happened with respect to a different application), hence they get a certain "inertia" to remain unchanged: we only change the values if the benefit on at least one of the criteria is obvious (less so if the default value is not from the original paper, for example, σ for Harris and Shi-Tomasi).

It should be noted that the times reported in Figure 13 for the Fast Hessian detector are *not* representative: comparison with execution times on Microsoft Windows XP revealed that all other execution times were very similar across the different platforms and

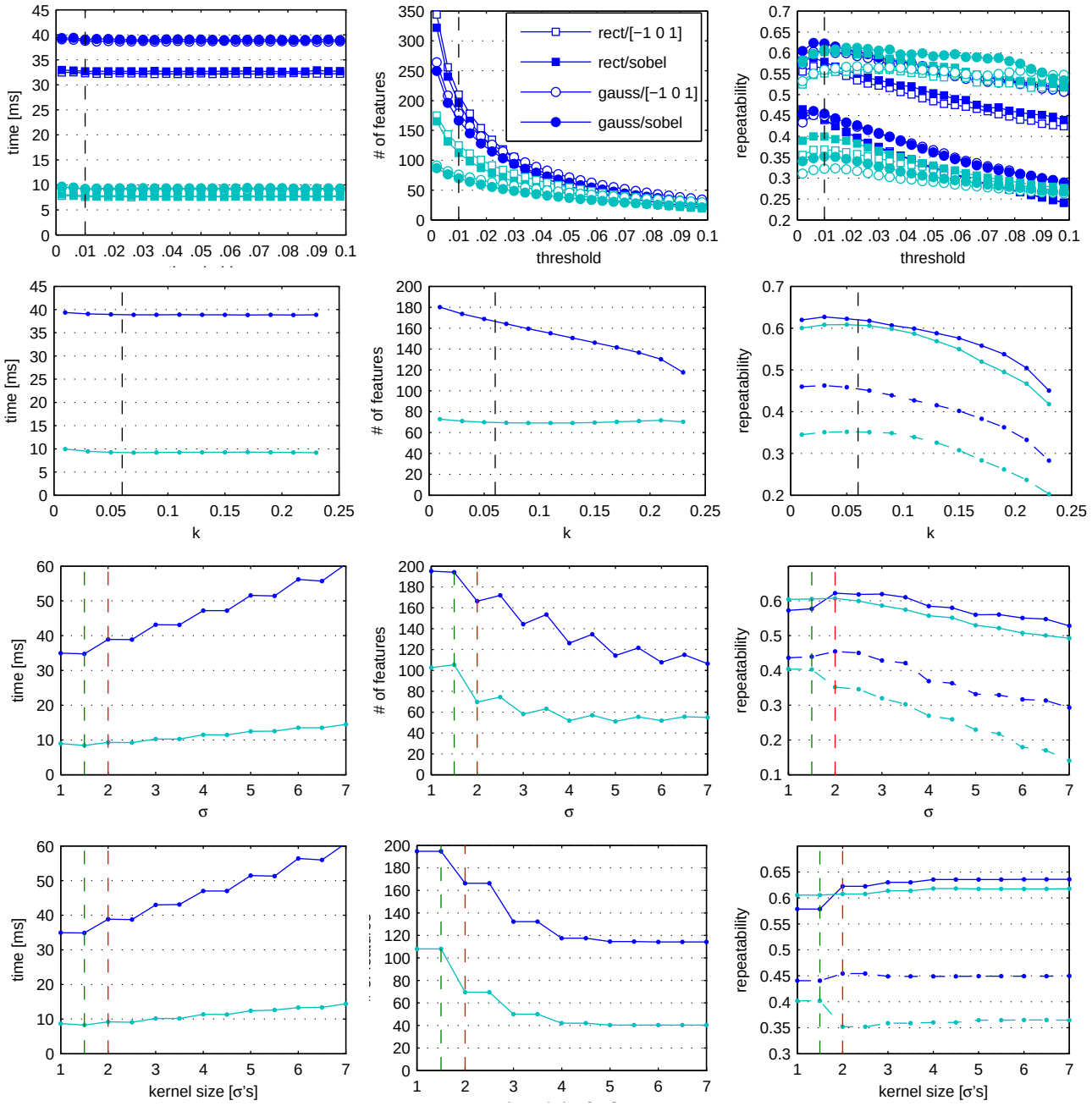


Figure 10: Harris: impact of varying threshold t, $w(u,v)$ and gradient kernel (top row), k (second row), σ (third row), and kernel size (in σ's, last row), on computation time (left), number of features (middle), and repeatability (right). **Legend for Figures 10-14**: Blue: image resolution 640x480, light cyan: resolution 320x240. Dashed red line indicates initial value (cf. Table 2), dashed green indicates final value (dashed black if initial value=final value). In the last column, solid lines show repeatability for consecutive frames, dashed lines show repeatability for 5x 3000 random frame pairs.

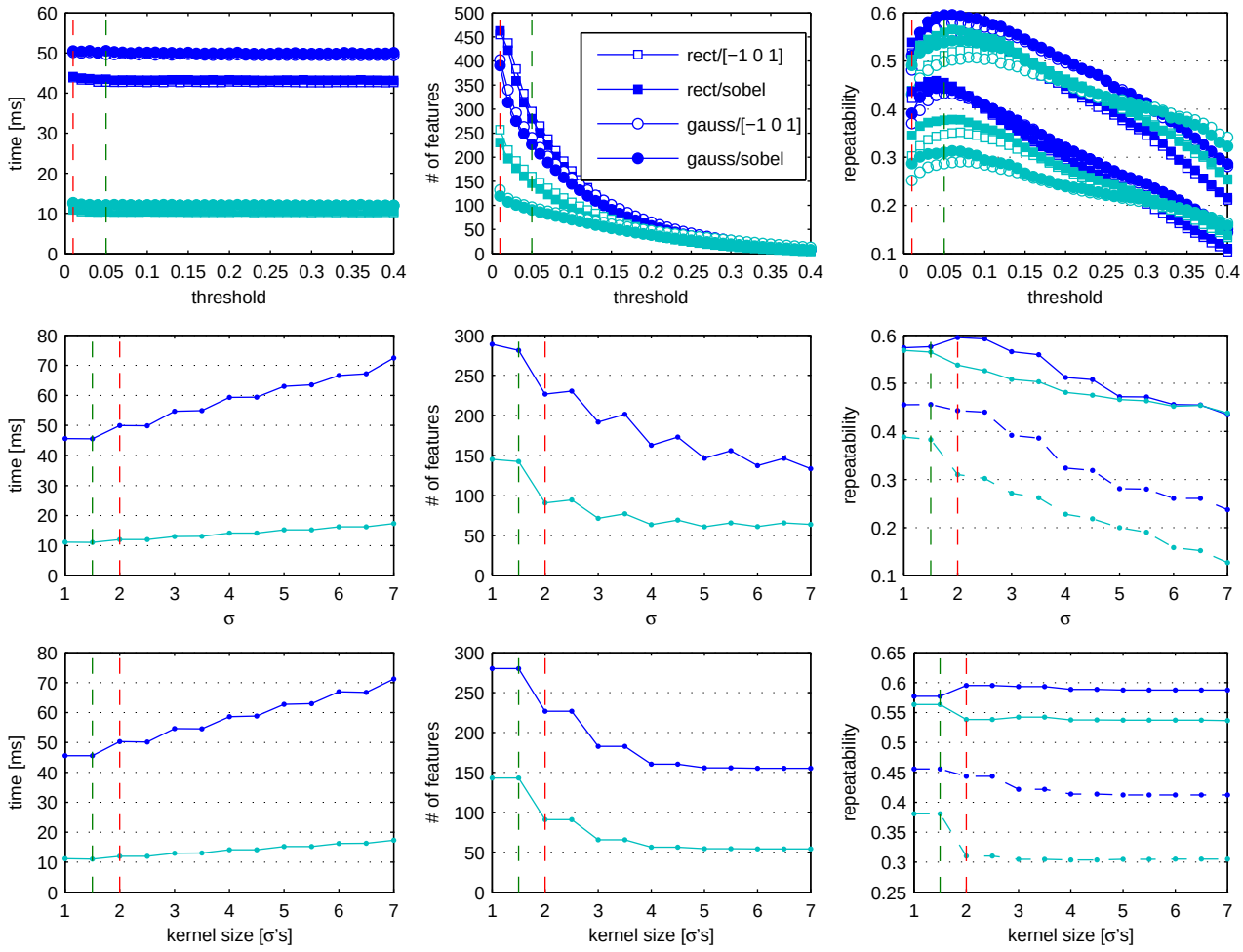


Figure 11: Shi-Tomasi: impact of varying threshold t, $w(u,v)$ and gradient kernel (top row), σ (second row), and kernel size (in σ's, last row), on computation time (left), number of features (middle), and repeatability (right). *Legend see caption to Figure 10.*

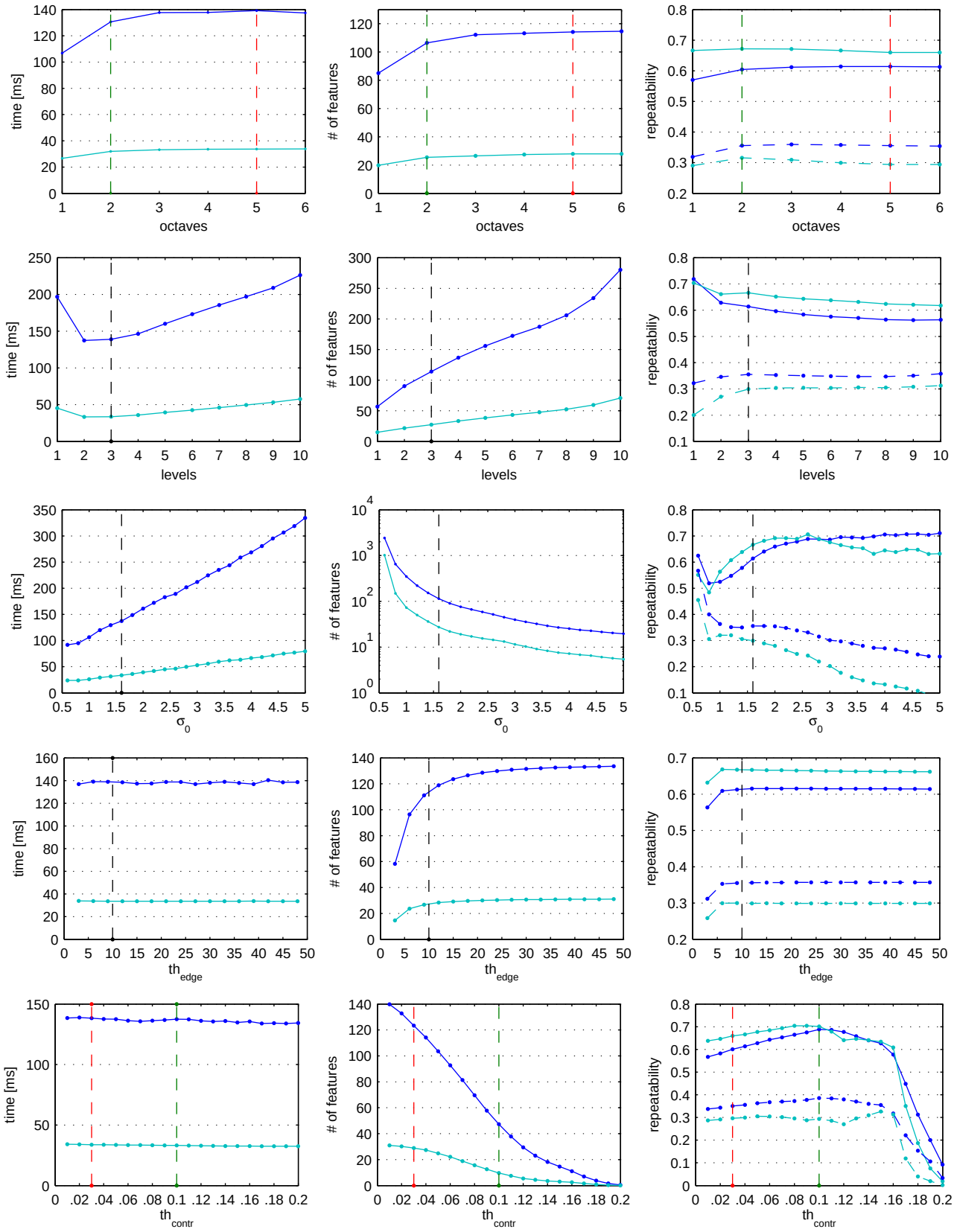


Figure 12: DoG: impact of varying the number of octaves (top row), number of levels per octave (second row), σ_0 for smoothing the first image (third row), th_{edge} (fourth row), and th_{contr} (bottom row), on computation time (left), number of features (middle), and repeatability (right). *Legend see caption to Figure 10.*

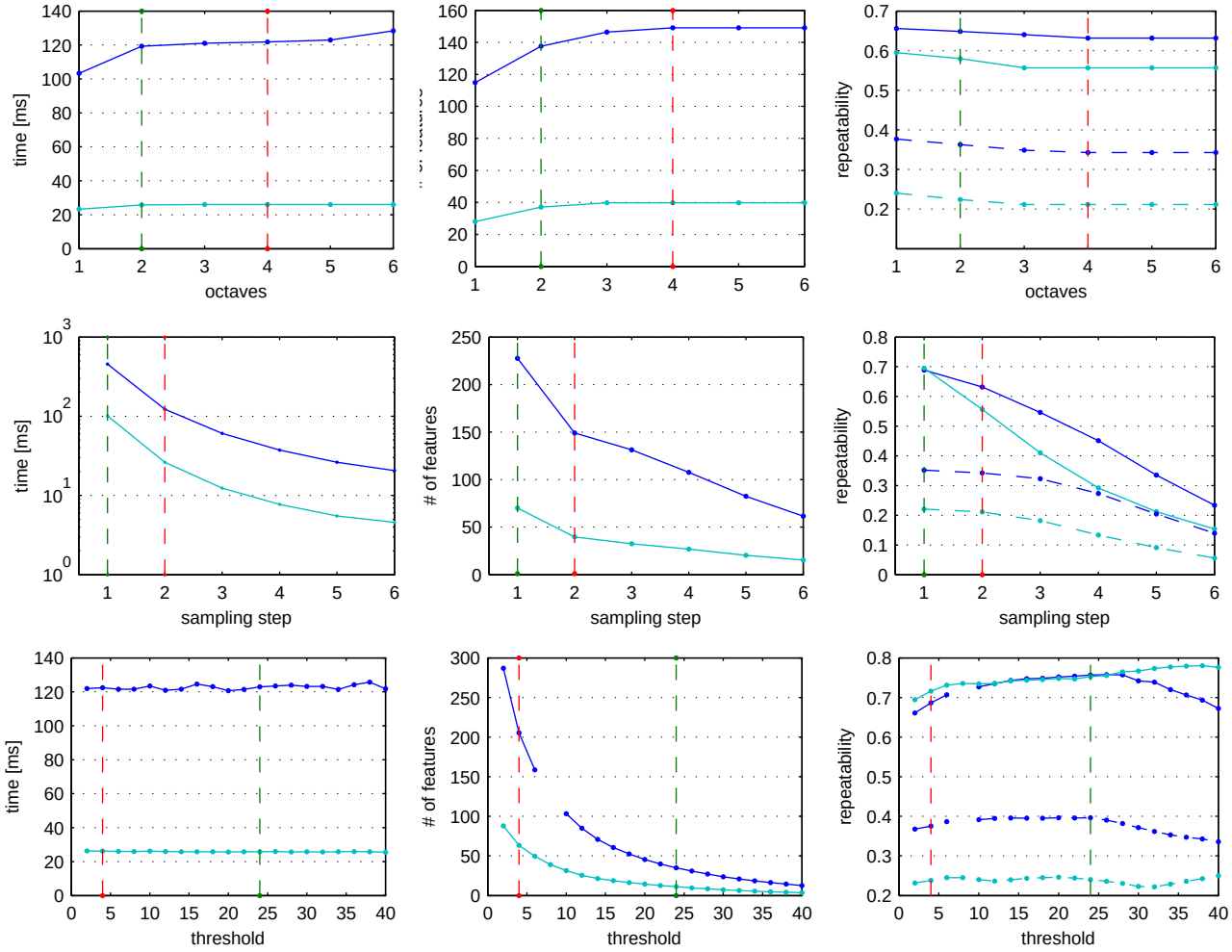


Figure 13: Fast Hessian: impact of varying the number of octaves (top row), size of sampling step (second row), on computation time (left), number of features (middle), and repeatability (right). *Legend see caption to Figure 10.* Please see comments in Section 5.1 on the times reported in this figure.

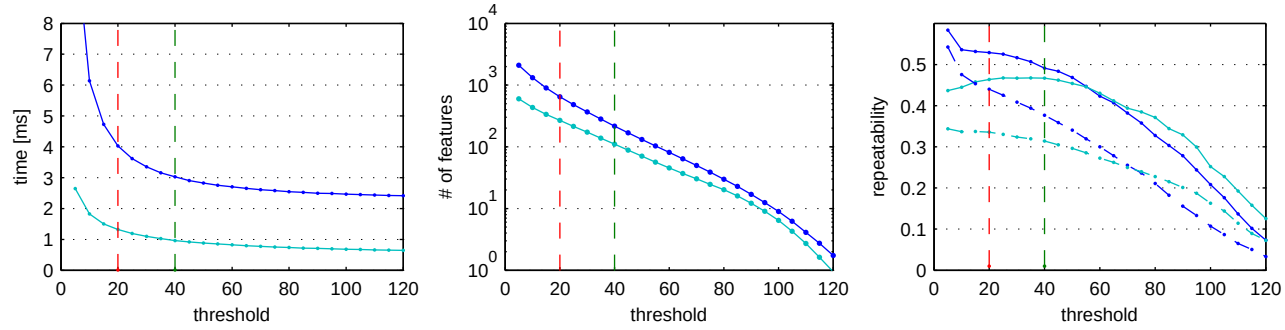


Figure 14: FAST: impact of varying the threshold t on computation time (left), number of features (middle), and repeatability (right). *Legend see caption to Figure 10.*

compilers while Fast Hessian was surprisingly slow. We assumed that this was due to a non-optimized library (note that Fast Hessian is the only algorithm for which we do not have the source code) and requested a new compilation, which was indeed much faster (while being exactly equivalent in terms of detected points). We used this version for the comparison and will report more realistic times in the next section, but considered it not necessary to repeat the parameter evaluations, as the times here are only used to compare different versions of Fast Hessian to each other.

5.2 Comparison

Figure 15 shows results for the motion pattern "unconstrained": Figure 15a shows the repeatability vs. the number of detected points (in the region of interest) achieved by the detectors by varying the respective threshold parameter. For comparison, the dashed gray line shows the "repeatability" of *randomly selected points*[8], thus visualizing the criterion's bias (cf. Section 4.3). Although the repeatability of FAST keeps to increase the more points it returns, it becomes clear from the comparison with the random points' repeatability that the average "quality" of the features actually decreases beyond a certain threshold. For all following evaluations, the threshold was fixed to the position of the respective marker.

For consecutive frames of (mostly) smooth motion, Figure 15b shows a clear trade-off between fast execution, but mediocre repeatability (FAST) and slow execution, but high repeatability (DoG, Fast Hessian). Downsampling the image, while resulting in the expected substantial speedup, does not decrease the For the DoG detector, downsampling is equivalent to setting the first sampling octave to 1 (instead of 0), as we validated experimentally. For Fast Hessian, setting the sampling step to 2 results in a similar speed-up, but worse performance, as may be seen in the parameter evaluation in Figure 13.

For randomly selected frame pairs (Figure 15c), the repeatability of all detectors is considerably lower, however, the performance of DoG and Fast Hessian dropped much further. This is confirmed in Figure 16a: For small baseline distances between the camera positions of the two frames, DoG and Fast Hessian perform best, but their repeatability drops below the repeatability of the corner detectors as the distance increases. This effect is investigated further using the isolated motion patterns:

- **near-translational movement** (Figure 16b): the repeatablity is fairly constant on high levels for all detectors.
- **scale changes** (Figure 16c): In terms of the set of detected points, neither DoG nor Fast Hessian can confirm the claimed scale invariance (note that this does not make any statemet about wether or not the scale information returned by each keypoint is accurate and/or useful).
- **in-plane rotation** (Figure 16d): DoG and Fast Hessian perform better than the corner detectors, although the latter also show a certain degree of rotational invariance in that the repeatability is very constant for 10-90.
- **out-of-plane rotation** (Figure 16e): the performance of DoG and Fast Hessian quickly falls below the performance of the corner detectors. This result is important, as out-of-plane rotation and the resulting perspective distortion are very common for many camera paths. As it occurs in the "unconstrained" motion too, we attribute the trend shown in Figure 16a mainly to the trend here.
- **motion blur** (Figure 16f): Fast Hessian copes best with heavy motion blur.

[8] Values were experimentally obtained using the exact same methods as for the detector algorithms.

6 Conclusions

We presented a comprehensive evaluation of five popular 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 thousand frames total, and explicitly evaluated a total of 20 algorithm parameters as well as comparing the five detectors to each other.

When comparing the detectors, we found that the more expensive detectors DoG and Fast Hessian do on average have a higher repeatability than the corner detectors Harris, Shi-Tomasi, and FAST *if* evaluated for smooth motion. However, for increasing perspective distortion (which is very likely to occur frequently in any visual tracking application), they quickly lose that advantage. Especially for Harris and Shi-Tomasi, downsampling the image proves a very effective way of reducing the computation time without decreasing repeatability.

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.

References

[1] 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.

[2] S. Baker and I. Matthews. Lucas-Kanade 20 years on: A unifying framework: Part 1. Technical Report CMU-RI-TR-02-16, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 2002.

[3] 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.

[4] G. R. Bradski. Computer vision face tracking for use in a perceptual user interface. *Intel Technology Journal*, Q2:15, 1998.

[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.

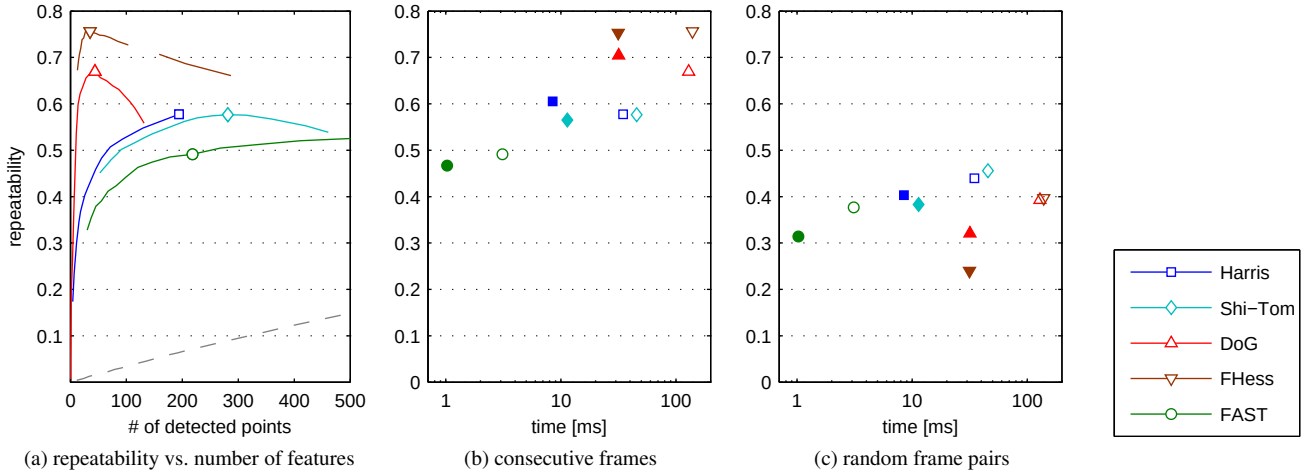
[7] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007.

[8] 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.

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

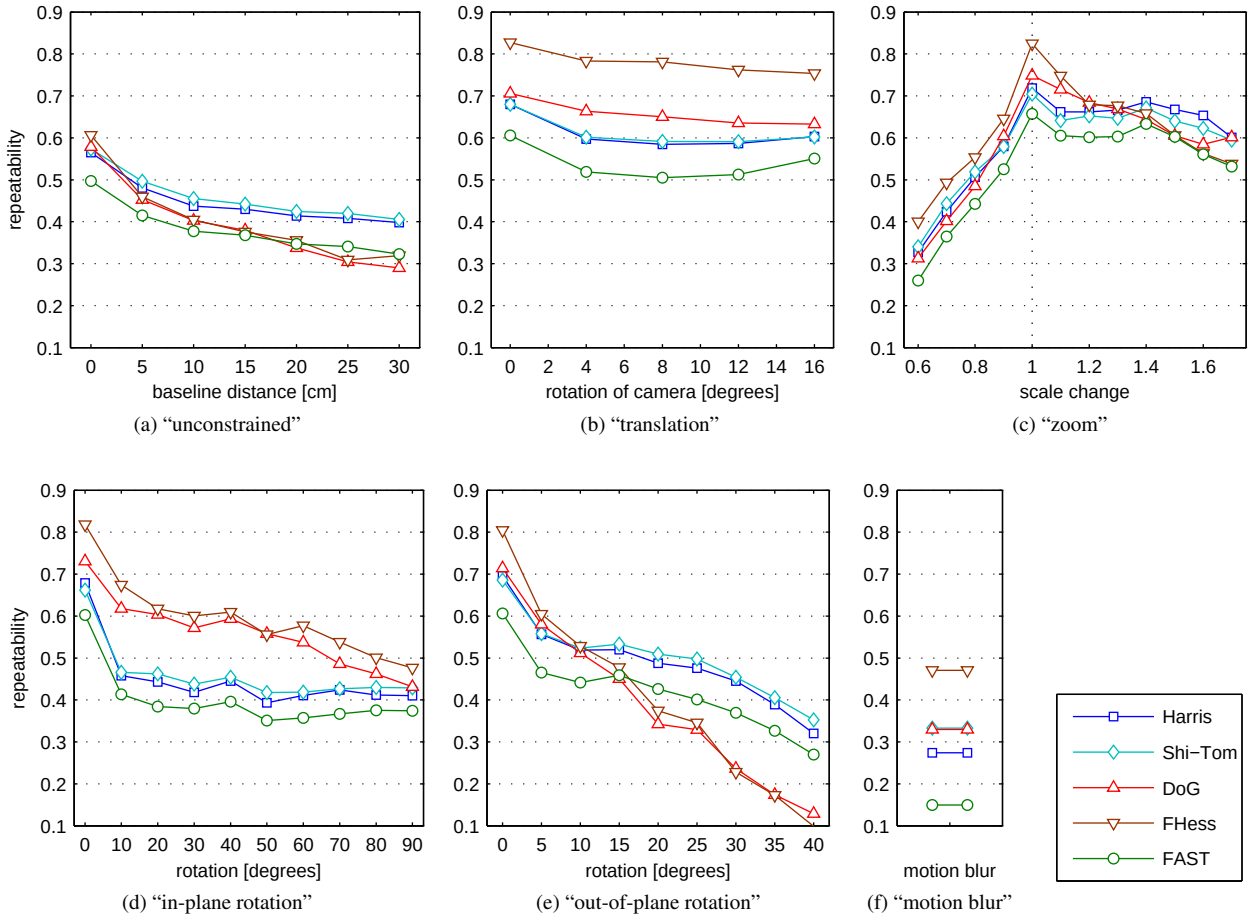
[10] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[11] H. S. Hong and M. J. Chung. Expansion of hager belhumeur inverse additive algorithm to homographies. *Proc.*



(a) repeatability vs. number of features (b) consecutive frames (c) random frame pairs

Figure 15: Repeatability of detectors (motion pattern "unconstrained", averaged for all frame pairs and textures). (a) Repeatability vs. number of detected points (in the planar region of interest), varying the respective threshold parameter. For comparison, the dashed gray line shows the repeatability of randomly selected points. (b) Repeatability vs. execution time for consecutive frames, for frames of 640x480 pixels (empty markers) and 320x240 pixels (filled). (c) Repeatability vs. execution time for 3000 pairs of randomly selected frames for each texture.



(a) "unconstrained" (b) "translation" (c) "zoom"

(d) "in-plane rotation" (e) "out-of-plane rotation" (f) "motion blur"

Figure 16: Repeatability as function of the baseline distance (a) and for specific camera motions (b-f). For (a), repeatability was evaluated for 5x 3000 random frame pairs of the motion pattern "unconstrained" (3000 pairs per texture) and subsequently binned and averaged according to the relative change in the camera's position between the two frames. For (b)-(f), 5x 500 random frame pairs of the specified motion pattern were used.

9th Intl. Conf. on Control, Automation, Robotics and Vision (ICARCV'06), pp. 1–4, Dec. 2006.

[12] T. Kadir, A. Zisserman, and M. Brady. An affine invariant salient region detector. In *Proc. 8th European Conf. on Computer Vision (ECCV'04)*, pp. 228–241, 2004.

[13] 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.

[14] T. Lee and T. Höllerer. Handy AR: Markerless inspection of augmented reality objects using fingertip tracking. In *Proc. 11th IEEE Intl. Symposium on Wearable Computers*, pp. 83–90, Oct. 2007.

[15] 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.

[16] 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.

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

[18] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision (ijcai). In *Proc. 7th Intl. Joint Conf. on Artificial Intelligence (IJCAI'81)*, pp. 674–679, April 1981.

[19] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *Proc. British Machine Vision Conf. (BMCV'02)*, pp. 384–393, 2002.

[20] L. Matthies and S. A. Shafer. Error modeling in stereo navigation. *IEEE Journal of Robotics and Automation*, 3(3):239–248, June 1987.

[21] 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.

[22] 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.

[23] F. Mohanna and F. Mokhtarian. Performance evaluation of corner detectors using consistency and accuracy measures. *Computer Vision and Image Understanding*, 102(1):81–94, April 2006.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

[30] S. Se, D. Lowe, and J. Little. Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks. *The Intl. Journal of Robotics Research*, 21(8):735–758, 2002.

[31] 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.

[32] 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.