Chapter 1
Computer Vision for Mobile Augmented Reality

Matthew Turk and Victor Fragoso

Abstract Mobile augmented reality (AR) employs computer vision capabilities in order to properly integrate the real and the virtual, whether that integration involves the user's location, object-based interaction, 2D or 3D annotations, or precise alignment of image overlays. Real-time vision technologies vital for the AR context include tracking, object and scene recognition, localization, and scene model construction. For mobile AR, which has limited computational resources compared with static computing environments, efficient processing is critical, as are consideration of power consumption (i.e., battery life), processing and memory limitations, lag, and the processing and display requirements of the foreground application. On the other hand, additional sensors (such as gyroscopes, accelerometers, and magnetometers) are typically available in the mobile context, and, unlike many traditional computer vision applications, user interaction is often available for user feedback and disambiguation. In this chapter, we discuss the use of computer vision for mobile augmented reality and present work on a vision-based AR application (mobile sign detection and translation), a vision-supplied AR resource (indoor localization and post estimation), and a low-level correspondence tracking and model estimation approach to increase accuracy and efficiency of computer vision methods in augmented reality.

1.1 Introduction

Augmented reality (AR) provides a live experience of the physical world with computer-generated augmentation appropriate to the location and particular task at hand. The augmentation is often specific textual information (e.g., the name of a nearby person or the date of a building's construction), location or geometric information (e.g., outlining or marking the destination building or door), or a virtual entity

M. Turk (✉)
University of California, Santa Barbara, USA
e-mail: mturk@cs.ucsb.edu

V. Fragoso
West Virginia University, Morgantown, USA
e-mail: victor.fragoso@mail.wvu.edu

G. Hua and X.-S. Hua (eds.), *Mobile Cloud Visual Media Computing*,
DOI 10.1007/978-3-319-24702-1_1

(e.g., an animated character or an advertisement placed appropriately in the scene). This information may be delivered by several different modalities or channels, such as audio (speech or sound directed individually to the user), video (through a display screen or another way of projecting imagery to the user), haptics (touch-based interaction), or other means. While AR may include a wide range of technologies, modalities and devices, the most typical AR systems aim to provide visual information through a transparent (optical see-through) or video see-through display—perhaps delivered via a smartphone or a head-mounted device.

In order to properly deliver spatial information, an AR system needs to know the location of the user and device either coarsely or precisely, depending on the application and the type of augmentation. In Sect. 1.3, we present research in pose estimation and localization for indoor environments. Many AR systems have used easily recognizable visual markers placed in the scene to aid tracking and localization (e.g., [25, 49]). However, this limits AR to structured environments, and most recent work in the field has sought to avoid this restriction. In the most demanding case, the precise position of the camera sensor is required, along with an accurate geometric and photometric model of the user's environment, in order to deliver artifact-free annotations that appear well integrated with the visual scene. Real-time, artifact-free mobile augmented reality with nontrivial models for augmentation is still a significant research challenge. Small errors can easily translate to significant misalignments, which are especially noticeable over time as a graphical overlay jitters with respect to the underlying scene. In some AR applications, apparent jitter can be reduced by using thick lines, temporal filtering, good annotation design, and other mechanisms, but misalignment remains a limiting factor in most augmented reality systems.

Markerless AR systems rely on low-level tracking and modeling techniques [20] to build 3D models and compute the camera position and orientation with respect to a known coordinate system. Typical approaches start with feature detection and description, then match features from frame to frame, using known geometric constraints to build a (often sparse) model comprising 3D locations of keypoints and the pose of the camera with respect to the model. While these are all areas that have been long studied in the computer vision field, augmented reality brings a different set of constraints and demands to the problem, which has led to practical solutions that are well-matched to the AR context (e.g., [26, 50]). In Sect. 1.4, we present work on keypoint correspondences and model estimation that aims to improve the accuracy, speed, and robustness of vision-based tracking and modeling.

To create a model of a full workspace or large area, low-level tracking and modeling techniques must handle issues that arise when synthesizing multiple portions of a scene, when combining rotational motion with more general (rotational + translational) motion, when closing the loop on a scene (returning to a portion previously modeled), and other challenging issues. In recent years, much progress has been made in systems that provide SLAM (simultaneous localization and mapping) capabilities for mobile robots, micro aerial vehicles, and AR applications. While beyond the scope of this chapter, our work in live tracking and mapping for both rotation-only and general motion [21] may be helpful in merging models and avoiding undesired calibration procedures in consumer AR applications.

Some augmented reality applications focus on objects in the camera's field of view than on (or in addition to) general scene geometry, providing information about objects of interest or giving a user the ability to interact with a virtual representation of the object. For example, someone playing an augmented game may indicate an object for the virtual character to go to, or a consumer may select an object to get additional information (such as vendor and price) that may float above the object in the AR display. In Sect. 1.2, we describe a system for automatic sign translation, which augments the scene by replacing the text of a sign by its translation, while displaying the appropriate sign geometry and background colors.

Given the impressive advances in recent years in mobile computing hardware and devices and in computer vision algorithms for tracking, modeling, and recognition, in addition to the rapid maturity of mobile computing ecosystems and a tremendous consumer demand for mobile devices and applications—not to mention the captivating futuristic portrayals in film and television—the field of mobile augmented reality has captured the imagination of many and is poised to become a mainstream technology for entertainment, productivity, learning, and other important areas. However, much progress is still needed in order to deliver the high-quality experience that is envisioned. This chapter describes a few efforts toward this goal of improved vision-based AR technologies to support compelling user experiences.

1.2 Sign Translation

One compelling augmented reality application is the translation of text in natural scenes (or sign translation) using a mobile device; see Fig. 1.1 for an illustration. This application, besides being useful when traveling abroad, imposes interesting and challenging mobile computer vision problems: text detection, visual tracking, and character recognition, among others. To guarantee a satisfactory user experience, the application must solve these problems as efficiently and quickly as possible.

With these constraints in mind, we developed TranstlatAR [17], a translation system that uses the camera and the touchscreen of a mobile device. The system identifies the words of interest from the live camera stream and presents the translation as an AR overlay which seamlessly replaces the original text in the live camera stream, matching background and foreground colors estimated from the source images. In the following sections, we describe the translation system as well as an automatic text detector tailored for this system.

1.2.1 Overview of the System

TranslatAR's architecture, shown in Fig. 1.2, was designed such that all the expensive operations run in the background thread, while the system maintains interactive frame rates for tracking and augmentation. In the following sections, we describe several

Fig. 1.1 *Top row* TranslatAR in operation. The user detects the text he or she wishes to translate and taps on it (*top left*). The system automatically detects the extent of the text, extracts the letters via OCR, and produces a translation, which is then presented as a live augmented reality (AR) overlay (*top right*). *Bottom row* TranslatAR used in two other situations

components in the system, such as the text detection algorithm, text extraction and recognition, the translation, the visual tracking, and the translation overlay process.

1.2.2 Text Detection

The goal of the text detection component is to compute an accurate bounding box enclosing the sign to translate. This computed bounding box is important to initialize the visual tracker as well as to extract the text via OCR.

The original text detection algorithm implemented in the system required the user to tap on the text of interest; the user's input enabled the text detection process to be efficient given the computational resources of a mobile device. Thus, given a point c onto which the user tapped, the system first finds the bounding box around the text, then the exact location and orientation of the text within. This process is illustrated in Fig. 1.3 and is explained in the following sections.

Bounding box. To find approximate upper and lower text boundaries, first the image gradients I_x and I_y are computed. A short horizontal line segment s_h around the input point c is then moved vertically upward and downward, respectively, until the following criterion is met (for δ_y consecutive scanlines):

$$\max_{(x,y)\in s_h} |I_x(x, y)| < \varepsilon, \tag{1.1}$$

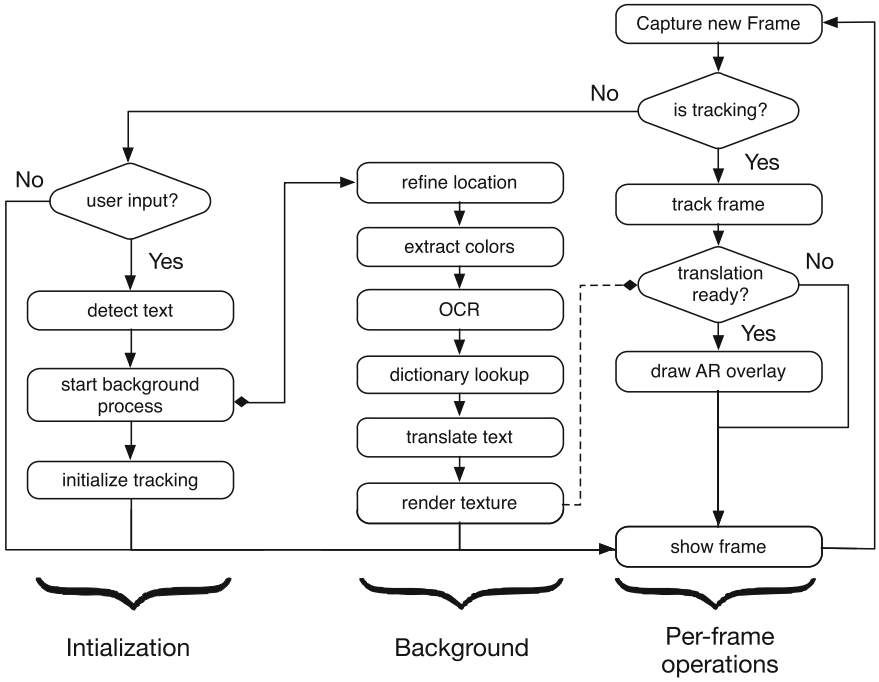


Fig. 1.2 Architecture of TranslatAR: Initialization and per-frame operations run in the main thread, while the rest of the operations are executed in the background

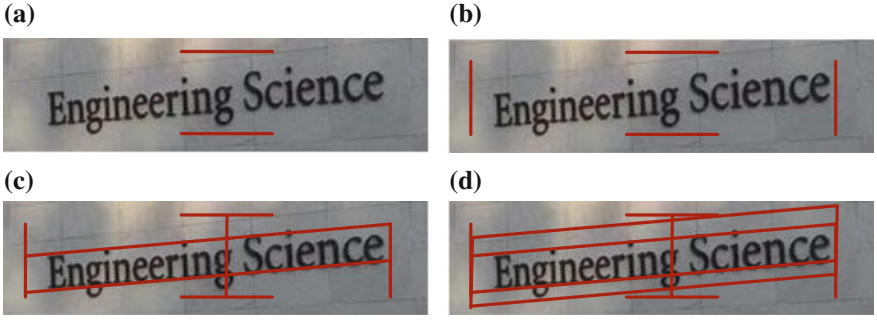


Fig. 1.3 Text detection in operation after the user's tap. First, the vertical extent of the text is determined (**a**). Subsequently, using the assumed text height, the horizontal extent is estimated (**b**). A constrained and modified Hough transform is used to estimate the baseline and orientation (**c**), and finally, the area is expanded to account for ascenders and descenders (**d**)

that is, until the segment s_h does not cross any vertical edges. The example in Fig. 1.3a shows the final upper and lower location of s_h. The same process is applied to compute the left and right boundaries, sweeping a vertical line segment s_v over I_y. The algorithm uses knowledge obtained in the first step by making the length of

s_v relative to the distance between upper and lower s_h (i.e., the estimated text height). Here, the required width of the "gap" δ_x is set slightly larger so that the algorithm does not stop between letters. The result of this process is shown in Fig. 1.3b. Values for ε, δ_x, δ_y, and the lengths of s_h and s_v were obtained experimentally.

Though fast and simple, this approach is able to detect an approximate bounding box in many conditions. However, it is susceptible to fail for very nonuniform backgrounds.

Location and orientation refinement. To detect the exact location and orientation of the upper and lower "baselines" of the text, the algorithm applies a constrained and modified Hough transform as follows: First, only pixels within the bounding box are considered, and only lines that cross the vertical line through c at an angle of $\pm 15°$ are taken into account. This reduces the computational cost considerably, ensures that only "reasonable" lines are taken as candidates for baselines, and leverages the assumption/limitation that the user will hold the phone roughly parallel to the text.

Second, the algorithm optimizes the voting scheme for the task of finding text baselines as follows: horizontal edges (i.e., in I_y) vote for lines passing through the respective point (vote with positive weight), while vertical edges (in I_x) vote against them (vote with negative weight). This is designed so that the ideal line goes along horizontal edges while cutting few or no vertical edges. The result can be seen in Fig. 1.3c. Finally, lines are moved vertically until no edge intersections are detected to account for ascenders and descenders (Fig. 1.3d). The resulting quadrilateral region of interest is warped into a rectangle, correcting any perspective distortion and showing the text as if seen orthogonally.

Text extraction, recognition, and translation. The system uses the computed warped image as described earlier to perform background and foreground color estimation and to "read" the text via OCR.

We begin describing the color estimation. The algorithm assumes that the letters have a single constant color with a reasonable amount of color contrast to the background, i.e., that there are two dominant clusters in color space that represent foreground and background. They are extracted from the subsampled rectified image using K-Means [1] with $k = 2$. To differentiate between foreground and background, the algorithm retrieves a few labeled samples along the left and right borders and assumes that the background color is the one with the majority of the collected labels; this is justified as the detection algorithm automatically includes a small margin.

This approach estimates both colors very accurately and fast when the assumptions are met. It can fail for very nonuniform backgrounds when there are significant specularities on the letters. However, in such cases, one of the other components (detection, OCR) is likely to fail, and though improving the user experience, the color estimation is not crucial to the operation.

The system relies on a standard OCR system for extraction and recognition of the letters and uses the warped image containing the word of interest for this task. The system used Tesseract [45], as it is freely available and was easy to integrate. As bad text detection frequently causes the OCR to return spurious, non-alphanumeric

characters (such as punctuation marks), the system computes the ratio of alphanumeric characters to all characters in the string as a rough indicator of successful extraction.

The following (optional) step was motivated by preliminary tests with the OCR which showed that single letters were frequently misrecognized. With a string returned from the OCR, the system searches through a dictionary of valid words to identify the nearest neighbor with respect to the Levenshtein distance [32]. The Levenshtein distance to the found string is computed for each dictionary word within ± 2 of the length of the found string, and the word with the smallest distance is taken as replacement for the original string returned by the OCR. This implementation clearly does not scale to large dictionaries and is only meant as proof-of-concept add-on.

With the extracted string, the system uses Google Translate API,[1] an existing free online translation service, to do the actual text-to-text translation. The input language is detected automatically by Google Translate, and the desired output language can be selected by the user in our GUI.

1.2.3 Visual Tracking and AR Overlay

Visual tracking enables the system to keep track of the word of interest in the live video stream and to present the translation in a live AR-style overlay. Fortunately, several circumstances make tracking in our application easier than in the general case: (1) it can be assumed that the text is displayed on a near-to-planar surface, (2) as the region of interest consists of text, it is automatically well-textured and contains features with high contrast, which is important for tracking, (3) the system is only required to track over short periods of time (as long as it takes the system to obtain the translation and the user to read it), (4) the system assumes a "cooperative" user who will not move the phone jerkily.

The application implemented a tracking system based on ESM [4], in which an image region is tracked by iteratively minimizing the difference between a reference frame (the template) and the current frame over a warp transformation. In other words, the tracker computes a warp that aligns the template image onto the current frame. Though costly for large intra-frame movements and/or large image templates, in our case (due to the above constraints), it provides sufficiently fast and robust tracking even for a relatively small template.

Based on the transformation computed by the tracker, a graphical augmentation is rendered onto the live video screen; first the bounding box is displayed while the text is being translated, and then, as soon as it becomes available, the translation itself is seamlessly augmented in the live stream.

[1] https://developers.google.com/translate/.

1.2.4 Implementation Details

The system was implemented on the Nokia N900 smartphone, which is based on a TI OMAP 3430 SoC with a 600 MHz ARM Cortex A8 CPU and runs the Linux-based operating system Maemo. The application was developed in C++, using OpenCV and libCVD for computer vision tasks (processing frames of size 320 × 240), GStreamer for frame capture, and Qt for the GUI, which consists of a large viewfinder and a few buttons for configuration (e.g., language selection).

The ESM tracker was implemented from scratch using libCVD. It uses a downsampled grayscale version of the warped rectangular text bounding box as a template and the respective previous frames homography as initial estimate for the 8-degree-of-freedom alignment. The graphical augmentation was implemented in OpenGL ES 2, leveraging the device's GPU; the translated text is rendered with OpenCV and then passed to the vertex shader along with the transformation estimated by the tracker, and finally the fragment shader renders the texture onto the current frame. HTTP requests to and responses from Google's online translation service are handled with the curl library,[2] a library for transferring data using various protocols.

1.2.5 Evaluation

Runtime. Table 1.1 presents an overview of the execution times of the main system components on the N900. As the expensive steps are offloaded into a background thread, the system maintains interactive frame rates for tracking and live feedback throughout the computation. The application achieved a frame rate of about 26 fps.

Text detection. To evaluate the text localization method [17], we used the ICDAR 2003 detection dataset. This dataset contains 251 images of varying size with at least one word in each image. Ground truth is provided in the form of a horizontal bounding box for each word.

As the algorithm was designed to work with video frames of a fixed size, the images were resized to 320 × 240 pixels. To conduct automated evaluation, the experiment simulated the required user input: the starting point c. This point was calculated/simulated as the center of the rectangle provided by the dataset, and it was adjusted properly to the new dimensions. As the dataset only provides an enclosing horizontal rectangle, and since the algorithm computes the (more accurate) quadrilateral, we calculated the minimal enclosing horizontal rectangle to be able to compare against the provided ground truth.

The performance measures proposed by Lucas [35] are based on a matching score m_p between two text area rectangles, which is defined as the area of the intersection divided by the area of the minimum bounding box containing both rectangles. m_p is 1 for two identical rectangles and 0 for nonintersecting pairs.

[2] Libcurl is available at http://curl.haxx.se/.

Table 1.1 Average execution times on the Nokia N900 for the main steps of the processing pipeline in TranslatAR

Component	Time (ms)
Initialization upon input	
Find text bounding box	71.0
Initialize tracker	5.0
Background thread	
Text location refinement	414
Extract colors	10
OCR	630
Render translation texture	10
Per-frame operations	
Capture and preprocess frame	21.9
Tracking	8.5
Render AR overlay and display frame	7.7
Total per-frame	38.1

With the expensive steps offloaded into a background thread, the system maintains a frame rate of about 26 fps

For automatic detectors, there will not be a unique 1:1 matching between detected and ground truth areas, hence the respective best m_p for each detected and ground truth area is taken and subsequently averaged to yield precision and recall, respectively. Different values for precision and recall thus result from detecting too many or too few areas, but no distinction is made between too large and too small areas. However, due to the manual "seeding" of the algorithm, there is guaranteed to be a 1:1 matching, and therefore the ICDAR definition of precision and recall both default to the average m_p for our algorithm. For further analysis, we also calculated pixel-wise precision and recall (e.g., as used by Park and Jung [39]), i.e., the ratio of pixels correctly labeled as text versus all pixels labeled as text, and the ratio of pixels correctly labeled as text versus all text pixels.

To optimize the parameters of the algorithm, we used the training part of the ICDAR set, then evaluated the metrics on the test part. The obtained are pixel-wise precision and recall of 31 and 68 %, respectively, and an average m_p = ICDAR precision of 41 %. This falls within the middle range of values published by Lucas [35], but cannot compete with the best scoring algorithms described by Lucas [35] and Epshtein et al. [14], which achieve precision and recall values of 60–70 %. It should be noted that the described algorithm requires a single point as input, while the other algorithms are fully automatic, but also that the described algorithm runs in less than 0.5 s on a mobile device and is hence one to two orders of magnitude faster than the aforementioned algorithms (see timings in [14, 35]).

A few examples of good and bad detection are shown in Fig. 1.4. The algorithm is prone to "overshoot" all the way to the borders of the image for nonuniform backgrounds, but it rarely cuts off letters. Note that the latter error is more fatal in our application than the former (in which case the OCR still has a chance to ignore the extra parts).

Fig. 1.4 Examples of good (*top* and *mid rows*) and bad (*bottom row*) text localization on the ICDAR 2003 dataset. The *blue point* in each quadrilateral represents the (simulated) input of the user. TranslatAR's algorithm was able to very accurately detect the text at different scales and under perspective distortion. The failure cases are mostly due to very nonuniform background and/or lighting effects (first two). For very large letters, the expansion algorithm used to detect the texts bounding box can stop inside one of the letters (*bottom right*)

Table 1.2 Reasons of failure of the detection-extraction-translation process on a set of 30 video clips

Component	No. of words	% of failures	% of all
Detector failed	7	16.3	8.9
Color est. failed	6	14.0	7.6
OCR	26	60.8	32.9
Translation	4	9.3	5.1
Correct result	36 of 79	–	45.6

If one component fails, the later components are not evaluated—e.g., the OCR failed 26 times, although detector and color estimation delivered a good result

Component test. We used our own set of 30 video clips of various outdoor signs, each containing several words, to further test the system as a whole and determine which components cause failures. Here, both providing the user input as well as evaluating the result was done manually. The results are listed in Table 1.2. As emerges from the table, the OCR is the most common cause of failure, while the detector works correctly in 72 out of 79 cases.

1.2.6 Automatic Text Detection

To enhance the user experience in the mobile AR translation systems, we developed an automatic text detection algorithm (see Petter et al. [41] for full details). In order to offer a fast automatic text detector, we focused in finding a single letter. The algorithm was designed on the following premise: detecting one letter provides useful information that is processed with efficient rules to quickly find the reminder of a word. This approach allows for detecting all the contiguous text regions in an image quickly. Moreover, the algorithm presented a method that exploits the redundancy of the information contained in the video stream to remove false alarms; see Fig. 1.5 for an illustration of this automatic text detection algorithm.

The general structure of the algorithm is shown in Fig. 1.6. The algorithm works on a grayscale image and can be overall described into three main steps: (1) Localize a first potential letter (zone of interest); (2) Verify that a letter was found; (3) Find the rest of the word based on the found letter.

Step 1: Finding a Zone of Interest

The aim of this step is to find a zone of interest that may contain a letter. The approach is based on existing methods [19, 33, 51] because of their efficiency and good performance. These methods leverage the high rate of edges contained in text

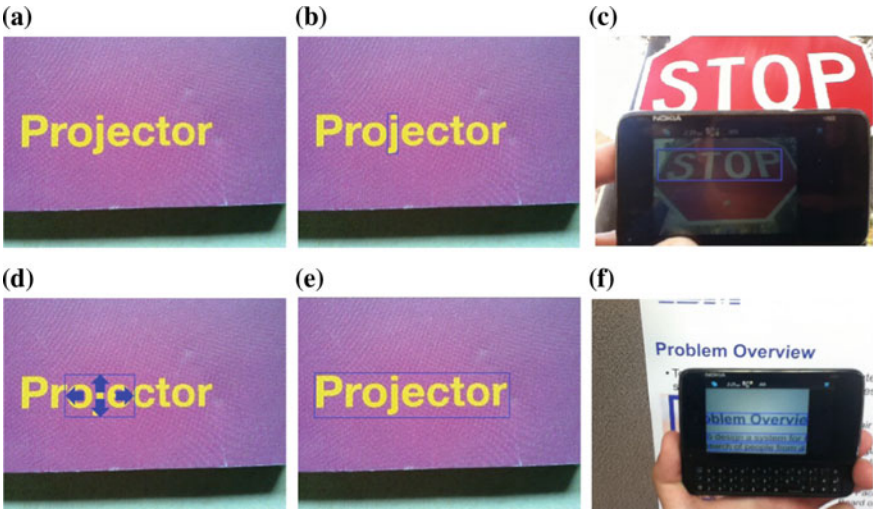


Fig. 1.5 Automatic text detection for TranslatAR. The algorithm scans the input image (**a**) until it finds a zone of interest that contains text (**b**). Subsequently, the algorithm expands the zone of interest with efficient rules (**d**), and finally, our method produces the final bounding box (**e**). Final bounding box with real examples (*right-most column*)

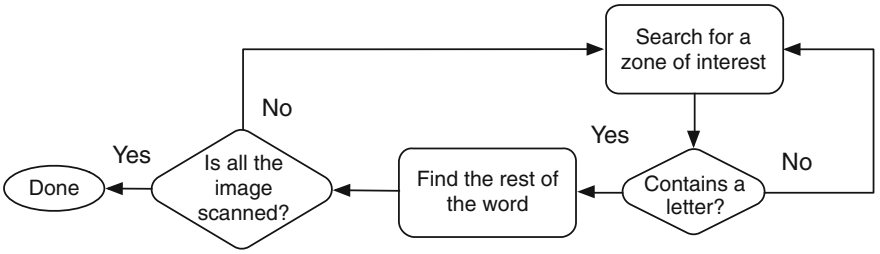


Fig. 1.6 Overview of the automatic text detection

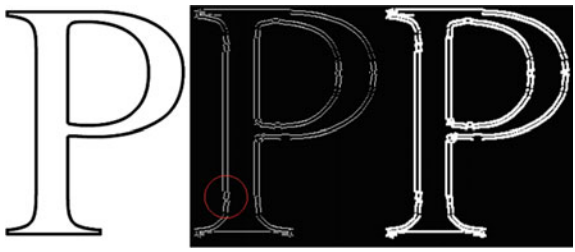


Fig. 1.7 Contour reconstruction process: Original picture (*Left*), edge map produced with Canny (*Middle*), reconstruction with dilatation operator, and a cross-shaped structuring element (*Right*)

areas. Therefore, a potential letter can be found on an edge map by building objects composed of closed contours that later can be categorized as letter or non-letter. In the following paragraphs, we explain in more detail the building blocks of this step.

Prior to detecting edges, a Gaussian smoothing filter of size 5×5 pixels is applied to reduce noise that could cause errors in further computation. The Canny edge detector [8] is used for producing a binary map indicating the presence per-pixel of every edge. This edge detector is efficient and provides accurate results which makes it suitable for our purpose.

The original image is sometimes too blurry for edges to be detected. Thus, some shapes, including letters, could be overlooked by the edge detection and not appear in the edge map (see Fig. 1.7). To ensure the continuity of the contour, a preprocessing step is necessary before starting the contour building step. Avoiding this step can produce an incorrect contour by the algorithm. For reconnecting the edge pixels together, we use dilation, a binary morphological tool. For our implementation, a cross-shaped structuring element of pixel size 3 worked the best for filling the holes in the contours.

The algorithm starts scanning from left to right and top to bottom to find an edge pixel in the binary map. When an edge pixel is found, the contour of the object containing this pixel is built with an 8-connectivity connected component algorithm. The 8-connectivity algorithm [10] is a region-based segmentation algorithm which checks the 8-pixel neighbors of a pixel and connects this pixel with its similar neighbors. Information about the bounding box containing the computed contour, such as height, width, position of the centroid, etc., is available as an outcome of this step.

Step 2: Determining if a Letter Was Found

The main intention of this step is to verify whether or not the zone of interest actually contains a character. This task is not straightforward, as the words contained on billboards, road signs, or books have different sizes or fonts which make learning precise shapes of letters a challenging task. However, since signs are typically meant to be easily readable, discriminating text regions from non-text regions with geometric information should be possible.

A Support Vector Machine (SVM) and a set of image features are adopted to accomplish the discriminating task. SVMs are widely used in the literature (e.g., [28, 52]) and are quite useful for binary categorization tasks. SVMs have a strong mathematical foundation and provide a simple geometric explanation of their classification.

In order to select the best features to address this discrimination task, experiments were conducted for evaluating several combinations of image features. The most effective features found were the First-Order Moments (FOM)

$$\mathrm{FOM}_1 = \sum_x \sum_y xI(x, y) \tag{1.2}$$

$$\mathrm{FOM}_2 = \sum_x \sum_y yI(x, y), \tag{1.3}$$

and Second-Order Moments (SOM) normalized with the number of pixels on the contour (NB),

$$\mathrm{SOM}_1 = \frac{\sum_x \sum_y x^2 I(x, y)}{\mathrm{NB}} \tag{1.4}$$

$$\mathrm{SOM}_2 = \frac{\sum_x \sum_y y^2 I(x, y)}{\mathrm{NB}}, \tag{1.5}$$

where x, y are the coordinates of the pixel in the clipped zone of interest and $I(x, y)$ denotes the intensity of the pixel.

Step 3: Finding the Rest of the Word

In order to robustly find the rest of the word given the position of the first letter, we combined two features that provide information about the surrounding characters: image intensities and the edge map. These features determine when to stop scanning in the surrounding areas, and therefore, to determine the spatial extent of the bounding box.

Given the first letter of the word or phrase to be detected, the background and foreground intensities in the grayscale domain can be extracted. We can safely assume in most cases that each word is contained in a homogeneous colored background and the letters have approximately the same intensity; we can then infer the intensity and

find the remaining letters. The K-Means algorithm with k = 2 is used to find the two intensities. In order to know which intensity corresponds to the letter, we create a second bounding box with the same center as the first bounding box of the zone of interest. The width and height of the new bounding box are computed as follows: $\text{width}_2 = \text{width} + \delta w$ and $\text{height}_2 = \text{height} + \delta h$, where $\delta h = \delta w = 2$ pixels (see Fig. 1.8). The pixels on the perimeter of that new box are likely to be background elements, and therefore, the closest intensity to the mean of those perimeter pixels is chosen to be the intensity of the background. Consequently, the remaining intensity is attributed to the letter.

Edge pixels around the found letter are likely to be part of the rest of the word because text regions present a high edge density. Useful information to estimate the position of the remaining letters is extracted from the adjacent edge pixels of the zone of interest.

In order to speed up the full word bounding box computation, the algorithm scans the image horizontally with three line segments. A single line segment is positioned on top, middle, and bottom of the found characters bounding box. Each segment is then scanned on the left and right side of the zone of interest considering a gap of size s on every side. The algorithm looks for pixels with intensities similar to the letters intensity along the segment. Edge pixels that are present in the analyzed gap are considered simultaneously. In this manner we guarantee that in fact we are likely to see pixels representing letters on the image. The size of the gap used in our algorithm is calculated as follows: $s = 1.1 \times H$, where H is the height of the found letter. The size as a function of the height allows us to consider the breach that exists between two adjacent characters in a word. However, when such breach is less than $1.1 \times H$, the algorithm considers both adjacent words as a single word. The procedure is applied until no edge pixels are detected or no similar intensity is found in the analyzed gap of every line segment. As an outcome of this procedure we obtain the width of the bounding box.

To find the height boundaries, the algorithm scans pixels along horizontal line segments with lengths equals to the computed bounding box widths described earlier (see Fig. 1.9a). The algorithm scans these lines following the same pixel criteria of intensities and edges used earlier. The algorithm moves the lines up and down until this criteria is fulfilled.

The combination of these two procedures computes a rectangular bounding box that encloses the letters of a certain text in the analyzed image (see Fig. 1.9b). Scanning with three horizontal parallel line segments tolerates a certain perspective

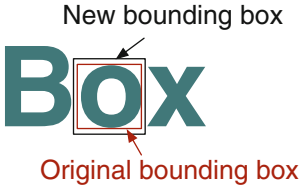


Fig. 1.8 Method to find the intensity of the background. A second box is created and the mean of the intensity of the pixels on the perimeter of the new box is associated to the background

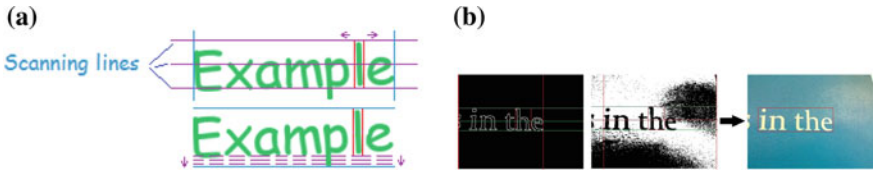


Fig. 1.9 *Left* Horizontal and vertical scanning to find letter pixels (intensity or edge pixels). Gaps of size s are analyzed between the letters during the procedure. *Right* Considering only information from edges (*left*) or intensity (*middle*) can determine an incorrect bounding box. Combining both features produces a better bounding box (*right*). **a** Scanning lines. **b** Bounding box computation

distortion of the letters that compose the word. However, the produced bounding box computed with these procedures may be slightly larger or smaller than the minimum bounding box due to noise present in the image.

Once a word is found, the search for additional words in the image continues until every pixel of the image not part of a word bounding box has been scanned.

Filtering False Alarms by Leveraging Temporal Information from Video Stream

An additional step is applied when the algorithm is used on a video stream, which is the case in an augmented reality translation system. In order to keep track of stable text regions and remove false alarms as much as possible, the algorithm leverages the temporal information that we can obtain from the video stream. We are interested in tracking these stable text regions. Since the scene does not change much from frame to frame, assuming that the frames on the video stream are generated at a high frame rate, the stable regions repeat and the position and area of the true positives detected bounding boxes does not vary much; therefore, false alarms will behave more unstably in this sense. The stability of these correct bounding boxes allows the algorithm to remove a fair amount of false positives.

The algorithm retains the center position and the area of the detected bounding boxes on the first frame. On subsequent frames, the system redetects the bounding boxes and matches them with the previously seen boxes based on areas and centroids. For every retained bounding box we increment a counter c if the bounding box matches a previously seen region, and decremented if it is not seen. A bounding box is considered to be stable if $c > 1$.

There are three different cases for matching that occur when comparing two bounding boxes (see Fig. 1.10):

Fig. 1.10 Considered cases when comparing bounding boxes: Inclusion (*left*), Intersection (*middle*), and Disjunction (*right*)

1. One of the bounding boxes contains the other one.
2. The two bounding boxes intersect.
3. The two bounding boxes do not intersect and neither of them contains the other one.

Cases 1 and 2 are situations where the bounding boxes in question can represent the same word. In order to know which case correspond to two bounding boxes, the positions of their upper left and lower right corners are compared. Once two bounding boxes are considered to be potentially the same word, further aspects are analyzed in order to determine a match.

To determine a match for the first case 1 the algorithm evaluates the ratio r between the smallest area and the biggest area. A match is determined if $r > 0.7$. For the second case, the algorithm evaluates the absolute value of the displacement of the centers $c_1 = (x_1, y_1)$ and $c_2 = (x_2, y_2)$, i.e., $\delta x = |x_1 - x_2|$ and $\delta y = |y_1 - y_2|$, as well as the ratio of the areas used in the first case. The method declares a match considering the following criteria: $\delta x < \varepsilon_x$, $\delta y < \varepsilon_y$, and $r > 0.7$, where $\varepsilon_x = 0.35 \times$ width, $\varepsilon_y = 0.35 \times$ height (the height and width correspond to the smallest bounding box). Subsequently, the algorithm averages the centroids and areas of the matching bounding boxes in order to keep track of the box on the remaining frames.

Evaluation

We carried out a series of experiments in order to thoroughly evaluate the text detection algorithm and the integration of this method with TranslatAR.

We created our own dataset to test the algorithm in a more realistic context, i.e., low-resolution, mobile camera, and others. This dataset comprises 400 images, each containing a single word from natural scene which follow the assumptions made for this project (see Sect. 1.2.6), and 400 non-text images.

In order to evaluate the performance of the proposed method, we evaluated every outcome manually, and the outcome was labeled as successful if all the letters of the word were contained in the bounding box. The results of this experiment are reported in Table 1.3.

Table 1.3 Accuracy of the automatic text detector

True positives (%)	False positives (%)	Precision (%)	Recall (%)	f-score (%)
87	41	68	87	76

Table 1.4 Distribution of failure for the missed words

1st step (%)	2nd step (%)	3rd step (%)
4.57	81.04	14.39

Fig. 1.11 Words correctly detected (*left*) and failures (*right*)

The algorithm found the majority of the words; however, it is also susceptible to a substantial rate of false alarms. By analyzing the failures, both missing words and false alarms, we concluded that the main problem occurs when verifying the zone of interest (i.e., the second step, see Sect. "Step 2: Determining if a Letter Was Found"). The SVM was the most common source of failure for the case of false negatives, failing to detect words (see Table 1.4).

It was also observed that false alarms arise in images with high edge densities, as the first step declares those regions as zones of interest and therefore the second step declares them as text regions. Moreover, another observation was that the SVM with SOM/FOM as features tends to declare any symmetrical non-text region in an image as text (see Fig. 1.11).

1.2.7 Discussion and Future Work

Recently, new text detection [24, 37, 38] and extraction [5, 29] in natural scenes algorithms and new powerful mobile devices have become available. Thus, these algorithms can potentially improve the performance of the text detection and extraction significantly as long as they run efficiently on a mobile device. As shown in this section, the OCR (or text extraction system) and the text detection components are the most challenging and important pieces in this application, as they enable the computation of a good translation.

1.3 Indoor Localization

The computational capability of mobile phones has been rapidly increasing to the point where augmented reality has become feasible on such devices. In this section, we describe an approach to indoor localization and pose estimation in order to support augmented reality applications in an indoor environment and on a mobile phone platform.

Estimating an accurate camera pose is crucial for delivering a high-quality augmented reality experience, because the application needs to understand how the camera is oriented and located with respect to the scene in order to augment virtual information accurately. In this section, we describe a system [40] that localizes the device in a familiar environment and determines its position and orientation using the camera and sensors in the mobile device. Once the 6 degrees-of-freedom (DOF) pose is determined, 3D virtual objects from a database can be projected onto the image and displayed for the mobile user.

The application has two main phases: an offline data acquisition and an online pose estimation. The offline data acquisition phase consists of building a database by acquiring images at different locations in the environment, while the online pose estimation computes the position and orientation of the device by matching features

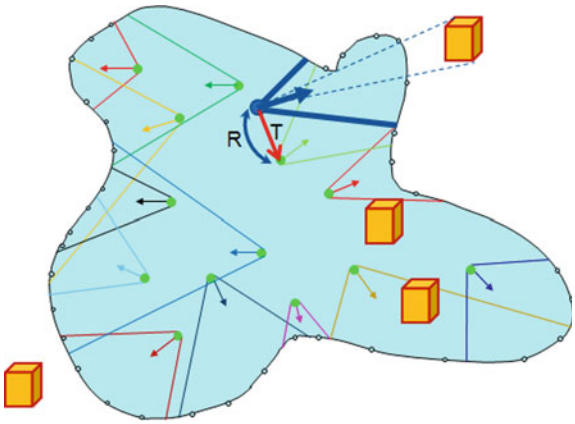


Fig. 1.12 Indoor localization application overview. The database contains several images taken at different locations (the *green dots*) in the indoor environment (*blue blob*). The arrows are the optical axes and the angles represent fields of view. The pose (translation T and rotation R) between the cell phone image (*bold blue lines*) and a database image is computed by image matching. Then the 3D virtual objects (represented by *yellow cubes*) are projected onto the mobile device image. Here, only one virtual object is seen by the mobile device

between the device image and an image from the database; the pose estimation also uses information from the sensors (accelerometer and magnetometer) for the computation as we discuss later in this section. In Fig. 1.12 we show an overview of the application.

The application enables the user both to visualize virtual objects in the camera image and to localize the user in a familiar environment. We describe in detail the process of building the database and the pose estimation algorithm used on the mobile phone. We discuss the performance evaluation of the proposed algorithm as well as its accuracy in terms of re-projection error of the 3D virtual objects onto the cell phone image.

1.3.1 Building the Environment Database

The first step in the application is to acquire data in advance about the environment; e.g., to go through a museum and create an image database that will be used subsequently by AR applications in the building. This involves carrying a camera rig through the space and preprocessing this data on a host computer. For each image, the acquisition process stores the pose of the camera (its absolute rotation and position) and its intrinsic parameters. Then the process extracts SURF features [2] in each of the images and stores their positions in the image as well as their descriptors. The goal of the mobile application is to localize the user metrically. Therefore, the process uses a stereo camera for building the image database and to estimate the depth for

every pixel of each captured image. As a result, the process stores the 3D position of the features in each image. For a reason to be explained below, the 3D position of the center of the images are stored as well.

Among all the detected SURF features, the system chooses the most robust ones; i.e., those that are likely to be detected by mobile camera device and be close to the new feature in terms of descriptor similarity. In order to do that, the system tracks each feature over several frames and keeps only the ones that were successfully tracked over all the frames. The criterion to keep a feature from one frame to the next one is the following: the feature position must remain close to its previous position and their descriptor distances are small enough. In practice, the system tracks the SURF features, while the stereo camera remains still.

1.3.2 Computing a Rotation Matrix from Sensors

Once the database has been collected and processed, the indoor environment is ready to support location-based AR on the mobile phone. As it is described later, having a coarse estimate of the pose makes the image retrieval step easier and accelerates the pose estimation algorithm. For this work, we used the N97 phone from Nokia. The device has several sensors, such as GPS, an accelerometer, a magnetometer, and a rotation sensor, that can help the application in estimating the pose.

As the first part of the pose estimation step, a "world" coordinate system is defined. As a right-handed Cartesian coordinate system, the application used the system (**E**, **g**, **N**), where **E** is the unit vector representing east, **g** is the unit vector representing the gravity force, and **N** is the unit vector representing north. In this section, we describe how the system obtains the rotation matrix of the camera pose from sensor measurements.

The accelerometer senses the second derivative of the position. Assuming the measurements are noise free, it is thus theoretically feasible to obtain the position by double integrating the accelerometer data. However, experiments showed that the data produced by this sensor is too noisy to get a reliable estimation of the position. Figure 1.13 shows the results of an experiment comparing the ground truth 2D trajectory and the trajectory estimated with the accelerometer data, while the user walked holding the phone upright. The graph shows a bird's-eye view, with an equal number of points in both curves. An accurate trajectory estimate would overlap the rectangular ground truth; in contrast, the accelerometer-based position estimate was wildly off.

Another solution to estimate the position is to use the GPS data which gives the location of the user with a few meters of error. Depending on the application that can be adequate. However, if the system is used indoors there is usually no GPS signal available, so the position cannot be estimated with the cell phone's GPS sensor. Therefore, if there is no GPS signal available, the system uses the last computed

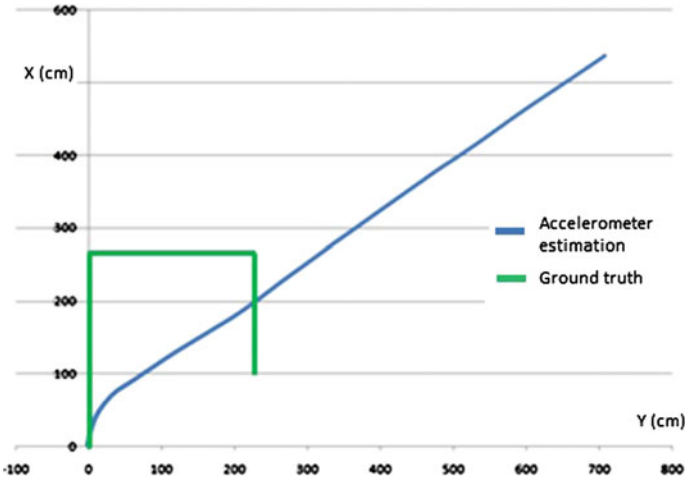


Fig. 1.13 Accelerometer accuracy for trajectory estimation. The 2D position (*top-down view*) of a walking user holding the cell phone is estimated using the accelerometer data (*blue line*) and compared to ground truth (*green line*). The accelerator-only estimate is not useful

user location (i.e., the position computed for the previous frame). Moreover, the application assumes that the user does not walk more than two meters between two frames, so that the previous estimated position can be reused to estimate a new one.

The accelerometer sensor outputs three channels that represent the acceleration along the three cell phone axes. Besides measuring acceleration due to the user, the accelerometer measures the gravity acceleration. Thus if the user is not moving, the three gravity components projected in the cell phone reference system can be measured by the sensor. This enabled the system to obtain the tilt of the phone, that is, two parameters out of three of the cell phone rotation matrix. The same information can also be obtained from the rotation sensor. This sensor gives three angles that represent the amount of rotation around the three cell phone axes. These angles can be used to retrieve the three components of the gravity vector in the cell phone reference frame. The advantage of using this rotation sensor rather than the accelerometer is that the outputs of this sensor are not sensitive to user movements. However, the sensor is less precise than the accelerometer because the angles are quantized to 15°.

The last parameter to compute is the amount of rotation around a vertical axis. To fully determine the rotation, the system needs additional information from the magnetometer. The 2D magnetometer (or digital compass) outputs the angle from the projection of the North vector onto the cell phone plane and one cell phone axis. This angle gives the system one parameter, which is enough to compute the full orientation of the mobile device because the system already has two parameters given by the accelerometer/rotation sensor. By expressing the components of all three cell phone axes from the gravity components and the magnetometer angle, the application can

obtain the full cell phone rotation matrix. In the subsequent paragraphs, we will describe how the system estimates the camera pose leveraging this rotation matrix computation.

1.3.3 Localizing the Mobile Device

Once the augmented reality application captures an image, it searches the database for the most relevant stored image to fully estimate the device's pose. The database can contain an arbitrarily large number of images and searching among all the images is not feasible, since the feature matching process is a time-expensive step on the mobile device. Fortunately, most of the database images do not need to be searched because the portion of the environment they represent is not even visible to the user. We describe how the camera pose can be obtained from the mobile device's sensors and use it so that the system can discard images that are not likely to being seen by the user; for example, images that represent a scene behind the user.

The system uses two criteria to select only the relevant images. First, the system checks that the database image centers are visible by the mobile phone camera. This is computed using the stored 3D points representing the database image centers. For this criterion, the system assumes two premises: (1) the overlapping region (if there is some) between two images (the database image and the mobile camera image) is not large enough for estimating the pose accurately when a center is not inside the cameras field of view; and (2) the user moves smoothly so that user's location is continuous. Due to the uncertainty on the estimated phone orientation from sensors, the system extends the field of view to search for 3D points. The system increases the number of images to be searched as a function of the uncertainty, i.e., the more uncertainty the more images to search. Second, the system discards images whose orientation differ significantly from the camera's orientation to prevent bad image matching configurations. Thanks to these assumptions, the image search is restricted and the search process is more efficient.

For every database image, the system loads in memory its 3D center point and the absolute pose of the stereo camera that captured the image; this information is about 36 bytes for each image. The system loads feature descriptors on demand for each image as well.

Among the images that have been selected to be searched, the system selects the best matching candidate by using a classic nearest-neighbor feature matching approach. The SURF features are detected in the cell phone image and the descriptors are computed at each of these points. Then for every image of the database found as a result of the search process aforementioned, the features from the cell phone image are matched to the ones from the database image. For each feature of the cell phone image, the nearest neighbor in the database image features set is searched. Subsequently, the system keeps only good matches, that is, matches that have a low enough distance between each other, and also the ones for which the ratio between the second best distance and the best distance is high enough; note that this is the

inverse of the Lowe's ratio [34]. As an outcome of this method, the system obtains sets of good matches between the cell phone image and each of the possible database images. Subsequently, the system selects the image from the database that has the highest number of good matches with the cell phone image, as it is used to compute the pose.

The nearest-neighbor (NN) search of descriptors for feature matching is performed using a KD-Tree structure. The system computes and stores a KD-Tree for every image during the preprocessing step. The NN search is implemented using the best-bin-first technique [3], a technique which is about 90 % as accurate as linear search but 100 times faster.

Once the selected database image has been matched with the cell phone image, the system obtains a set of candidate matches, from which incorrect matches need to be removed. The system estimates first a homography via the least median of square algorithm [43], and keeps only points that roughly satisfy the planar criterion checked via the estimated homography. The system uses a high threshold so that depth changes are allowed. Subsequently, a fundamental matrix is computed exploiting the matches supporting the previously computed homography within a RANSAC [15] scheme using the 8-point algorithm [23].

At this point the system has a set of putative correct matches between the cell phone and the database images, which enable the application to compute the pose. From these matches, the system then computes a set of 2D-3D matches by associating the 2D feature detected on the device with the 3D point corresponding to the 2D feature on the database image. Then, given these 2D-3D matches, the system solves for the translation and rotation of the device with respect to the world. To explain this in detail, let c_i be 2D point in the cell phone image, X_i be a 3D point in the database coordinate system, and K_c be the calibration matrix of the cell phone. The algorithm minimizes the reprojection error over the mobile device's extrinsic parameters (rotation matrix R and translation vector T), i.e.,

$$\underset{R,T}{\text{minimize}} \quad \sum_i \left\| \frac{K_c(RX_i + T)}{\alpha} - \begin{bmatrix} c_i \\ 1 \end{bmatrix} \right\|_2^2, \tag{1.6}$$

where $\alpha = (K_c(RX_i + T))_3$ is the third vector entry. This measure is minimized as the system's goal is to align the virtual with the real world as accurately as possible. To this end, the reprojection error (Eq. 1.6) is in the form of a least-squares problem which can be solved via the Levenberg–Marquardt [31] (LM) method. However, to use the LM solver an initial solution must be computed first. We describe a method to initialize this solver in the following paragraphs.

The initialization method assumes that the rotation matrix $\hat{R}$ can be estimated via sensor measurements. Then the method focuses on finding a good initial translation vector T. To this end, the method obtains the translation vector by solving the following problem:

$$\underset{T}{\text{minimize}} \quad \sum_i \left\| K_c(\hat{R}X_i + T) - \alpha \begin{bmatrix} c_i \\ 1 \end{bmatrix} \right\|_2^2. \tag{1.7}$$

Equation (1.7) uses the estimated rotation matrix $\hat{R}$ from the sensor measurements and the problem ends up being a linear unconstrained least-squares problem, which can be solved for T efficiently using linear algebra methods.

After solving the problem described in Eq. (1.6) over the rotation matrix R and a translation vector T, which are the parameters describing how the camera is oriented and positioned with respect to the scene, the system is now able to display AR augmentations onto the device's image.

1.3.4 Evaluation

To evaluate the approach, we built a database of an environment and used planar objects in the scene for visual assessment. The evaluation consisted in estimating the camera pose and drawing a quadrangle enclosing planar objects that were recognized and depicted by the mobile device. In Fig. 1.14 we show an augmentation of planar objects in the scene and confirm that minimizing the reprojection error finds a good camera pose estimate that can be useful for an AR application. In Fig. 1.15a, we show quadrangle augmentations of three different recognized planar objects in the scene.

It is possible to use the estimates in order to localize the user within an environment; the translation vector T is the parameter that reveals the location of the user with respect to the scene. A visualization of these localizations are shown in Fig. 1.15b. From this experiment, the observed estimated user's location error was about 10–15 cm.

The implementation was done in Symbian C++ and Open C/C++ on a Nokia N97 cell phone. It is equipped with a single-core ARM 434 MHz processor with

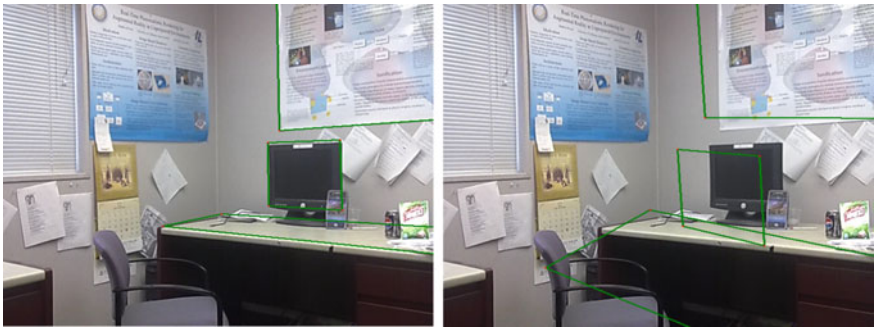


Fig. 1.14 Augmented images with (*left*) and without (*right*) the reprojection error minimization. The *green quadrangle* is the augmented virtual information onto the real scene

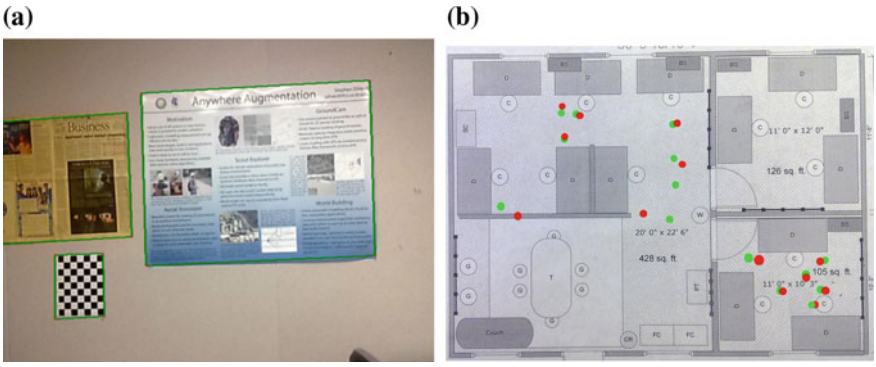


Fig. 1.15 *Left* Quadrangle augmentation of planar objects after estimating the mobile camera pose. *Right* Localization of the user within an environment by using the camera pose estimates. *Green points* are ground truth positions and *red points* are the estimated ones. One can notice a *single green point* alone in the middle of the floor plan. The matching process failed here because the cell phone image quality was inadequate. **a** Augmentation of 3 objects. **b** Estimated users' position

128Mb of RAM. The most expensive step in this application is the SURF detection an description algorithm, which takes more than 8 s to run on a 640 × 480 image. This is mostly due to the fact that Symbian only emulates the floating point precision because it does not natively support it; the used SURF implementation uses floating point numbers. This could be reduced by using a fixed-precision version of the SURF algorithm (or using a mobile platform with floating point computation, which are now common). For comparison, the algorithm runs at 10 fps for 640 × 480 images on a 1.73 GHz computer. The second most expensive computation is feature matching, which took about 1.1 s. The pose estimation took about one third of a second; the pose refinement algorithms used double precision numbers, which increased execution times.

1.3.5 Discussion and Future Work

The approach presented in this section was tailored for mobile phones that did not have powerful computational resources. However, this has changed recently and now we can find powerful mobile devices containing floating point units, multi-core and fast processors, and more RAM. Fortunately, algorithms solving for the camera pose from 2D to 3D correspondences, also known as the perspective-n-point (PnP) problem, have become more efficient and mobile device friendly, e.g., [27, 30, 46]. As potential future directions, these PnP can be used to directly estimate camera poses. Moreover, inertial measurements can be leveraged and used in combination with the aforementioned algorithms to quickly and accurately compute a camera pose estimate.

1.4 Keypoint Correspondences and Robust Model Estimation

In many augmented reality (AR) applications, the accuracy of the camera pose is a critical component to ensure a high-quality augmentation. This is because it is necessary to understand how the camera is positioned and oriented with respect to the world in order to accurately augment virtual objects onto images; see for instance the indoor localization application described in Sect. 1.3.

A common approach to estimate camera poses is by understanding the relative motion of the cameras depicting a scene; this process is a crucial part in structure from motion (SfM) [13]. To get an understanding of all the relative camera motions given a collection of images, a set of keypoint correspondences between image pairs must be computed first. Subsequently, different models, such as homographies, essential matrices, and fundamental matrices, are computed from these correspondences and are used later to extract valuable camera pose information.

In general, we wish to compute these models as quickly as possible. This is very important in particular to mobile augmented reality applications because they need to perform the augmentations as fast as possible. Nevertheless, several nuisances make this estimation process nontrivial; for instance, a critical nuisance is the presence of incorrect keypoint correspondences between image pairs. These incorrect correspondences, the "outliers," have to be filtered in order to compute accurate models.

In this section, we present two approaches that speed up the process of robustly estimating models from contaminated keypoint correspondences with outliers. We describe two different methods to estimate the correctness of the correspondences leveraging information from the matching distances using the statistical theory of extreme values [9, 12].

1.4.1 Computing Keypoint Correspondences

To compute keypoint correspondences between a reference image and a query image, we first need to detect features or keypoints on both images. Subsequently, for every keypoint a descriptor is computed, e.g., SIFT [34] or SURF [2]. These descriptors, which are a representation for every detected keypoint, are used to establish the keypoint correspondences following the nearest-neighbor (NN) rule: the jth query keypoint is assigned to the $i^\star$th reference keypoint such that

$$i^\star = \arg\min_i \left\{ \|\mathbf{q}_j - \mathbf{r}_i\| \right\}_{i=1}^{n}, \tag{1.8}$$

where $\mathbf{r}_i$ and $\mathbf{q}_j$ are the reference and query descriptors, respectively. In other words, the NN rule computes the least dissimilar reference keypoint given a query keypoint.

1.4.2 Predicting Correctness for Keypoint Correspondences

Incorrect correspondences, or outliers, occur when the truth reference keypoint is not the least dissimilar, or when the truth reference keypoint was not detected in the query image. To detect these outliers produced by the NN rule (see Eq. 1.8), we proposed a predictor based on the statistical theory of extreme values [16]. The predictor, which is called MRRayleigh, computes a correctness probability which later is used to label the correspondences as correct or incorrect and to speed up a robust estimation process.

The main premise of the predictor, which was inspired by the work of Scheirer et al. [44], is that computing a statistical model for the minimum distances generated from the incorrect correspondences is possible by exploiting the statistical theory of extreme values. Thus, checking if a minimum distance used in the NN rule is likely to be a sample generated from this model or not allows us to estimate the correctness of the correspondence.

More formally, we consider the descriptor distance $d_{ij} = \|\mathbf{q}_j - \mathbf{r}_i\|$ to be a continuous random variable following a distribution F. We know that some distances correspond to correct correspondences and others to incorrect ones. Thus, there are two underlying random processes generating distances for correct and incorrect correspondences, which we call F_c and $F_{\bar{c}}$, respectively. The NN matching process then takes a decision by observing several distances samples from these distributions. Because we are matching 2D features corresponding to actual 3D points, there must be a single correct answer and thus a single distance corresponding to a correct match. However, we can also have the case that there is no correct answer at all because a reference keypoint was not detected in the query image. Therefore, we can expect that there is at most a distance corresponding to a correct match among all the distances computed when using the NN matcher for a query $\mathbf{q}_j$. In other words, we have at most a single sample drawn from F_c and many samples from $F_{\bar{c}}$.

In order to compute a model that explains the behavior of the minimum distance that the $F_{\bar{c}}$ process can generate, we use the distributions suggested from the statistical theory of extreme values. In Fig. 1.16 we provide an illustration of the densities involved for this processing. Next, we review the main theorem used in our approach.

Review of the Fisher–Tippet–Gnedenko Theorem

The Fisher–Tippet–Gnedenko Theorem, also known as the block maxima theorem, provides a family of distributions to model the maximum or minimum values that a random process can generate:

Theorem 1 *Let X_i be a sequence of i.i.d. random variables and let*

$$M_n = \max\{X_1, \ldots, X_n\}$$

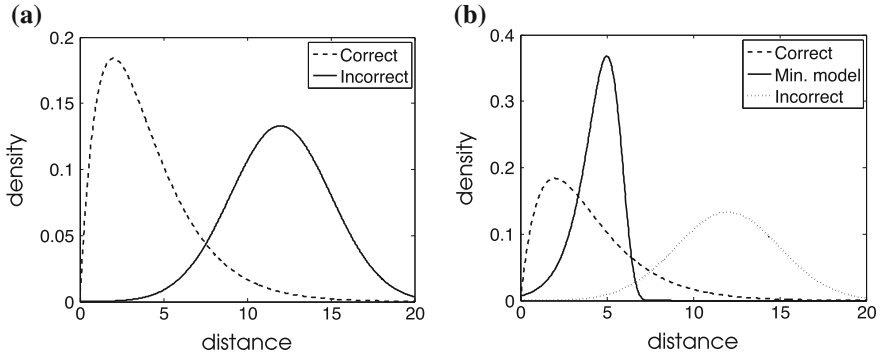


Fig. 1.16 *Left* The two underlying random processes' densities involved in the generation of the distances, correct match distances density (*dashed curve*) and incorrect match distances density (*continuous curve*). *Right* The minimum distance that the random process for incorrect matches can generate is a random variable, thus a density describing it can be obtained (*continuous curve*); the underlying two random processes: correct (*dashed curve*) and incorrect (*dotted curve*) processes. **a** Random processes. **b** The minimum model

denote the maximum. If there exist sequences of normalizing constants $a_n > 0$, $b_n \in \mathbb{R}$, *and a nondegenerate probability distribution function G, such that*

$$\mathbb{P}(a_n^{-1}(M_n - b_n) \leq z) \rightarrow G(z) \;\; as \;\; n \rightarrow \infty \tag{1.9}$$

then $G(z)$ *is of the same type as one of the three extremal-type distributions: Gumbel, Fréchet, and Weibull.*

In other words, the block maxima theorem states that the rescaled sample maximum $\left(a_n^{-1}(M_n - b_n)\right)$ converges in distribution to a variable having an extremal-type distribution. We refer the reader to [9, 12] for the proof of this theorem. Although Theorem 1 considers maximum values, we can still use it to model sampled minima using one of the three extremal-type distributions. To do so, we must first apply a simple transformation: $X' = -X \Rightarrow \max\left\{X'\right\} = -\min\{X\}$.

To determine exactly which of the three extremal-type distribution to use for modeling the maxima/minima, we need the domain of attraction tests [9, 12]. However, the generalized extreme value distribution (GEV),

$$G(z; \mu, \sigma, \xi) = \begin{cases} \exp\left\{-\left[1 + \xi\left(\frac{z-\mu}{\sigma}\right)\right]^{-\frac{1}{\xi}}\right\} & \text{if} \quad \xi \neq 0 \\ \exp\left\{-\exp\left[-\frac{z-\mu}{\sigma}\right]\right\} & \text{if} \quad \xi = 0 \end{cases}, \tag{1.10}$$

subsumes the three extremal-type distributions. Thus, we can use the GEV to model maxima or minima from a random process, avoiding the domain of attraction tests. The GEV distribution has three parameters: location μ, scale σ, and shape ξ.

MRRayleigh: The Predictor

To estimate the correctness of a correspondence, we calculate a confidence or belief given their distances used in the NN matching process. To do so, we proposed the MRRayleigh predictor [16], shown in Algorithm 1.

Algorithm 1 MRRayleigh

Require: $\{d_{1:n}\}$, k, and $\delta \in (0, 1)$
Ensure: $v \in \{1, 0\}$ and p
1: $D_k \leftarrow$ Get the smallest k samples from $\{d_{1:n}\}$
2: $d^\star \leftarrow \min D_k$
3: $\sigma \leftarrow$ Fit Rayleigh distribution to $D_k \setminus d^\star$
4: $p \leftarrow \mathbb{P}(C = \text{correct} | d^\star, D_k) = 1 - \text{RayleighCDF}(d^\star; \sigma)$
5: **if** $p > \delta$ **then**
6: Predict correspondence as correct: $v = 1$
7: **else**
8: Predict correspondence as incorrect: $v = 0$
9: **end if**

This predictor requires the distances $\{d_{1:n}\}$ obtained by comparing a given query descriptor $\mathbf{q}$ with the set of reference descriptors $\{\mathbf{r}_i\}_{i=1}^n$; k, a number of samples that define the left tail of $F_{\bar{c}}$; and a threshold δ, which is used to decide if a correspondence is correct or incorrect. The predictor computes a correctness confidence or belief p and returns $v = 1$ when the correspondence is likely to be correct, and $v = 0$ otherwise.

The idea of the predictor is to compute a model for the minimum distance that the process $F_{\bar{c}}$ can generate using the distributions stated in Theorem 1, and use it to verify if the minimum sample used in the NN matcher is a sample that is likely to be generated from the computed model. To compute this model, the algorithm selects the k lowest distances, which are samples from the tail of $F_{\bar{c}}$ (Step 1). Subsequently, the algorithm fits a Rayleigh distribution, which is a special case of the Weibull distribution, to the k samples discarding the minimum (Steps 2–3). Next, the algorithm computes the confidence by evaluating the Rayleigh's inverse cumulative distribution function (cdf) at the minimum distance (Step 4). Finally, the algorithm decides given this confidence and a threshold if the correspondence is likely to be correct or incorrect (Steps 5–9).

In Fig. 1.17, we present the prediction performance using two different descriptors, SIFT and SURF, for computing correspondences on the publicly available affine covariant features dataset [36]. This dataset contains eight sub-datasets, each with systematic variations of a single imaging condition: viewpoint, scale, image blur, illumination, or jpeg compression. Every sub-dataset contains six images: a reference image and five query images of the same scene varying a single imaging condition. In addition, every sub-dataset provides five homographies that relate the reference image with each of the query images in the sub-dataset. These homographies were

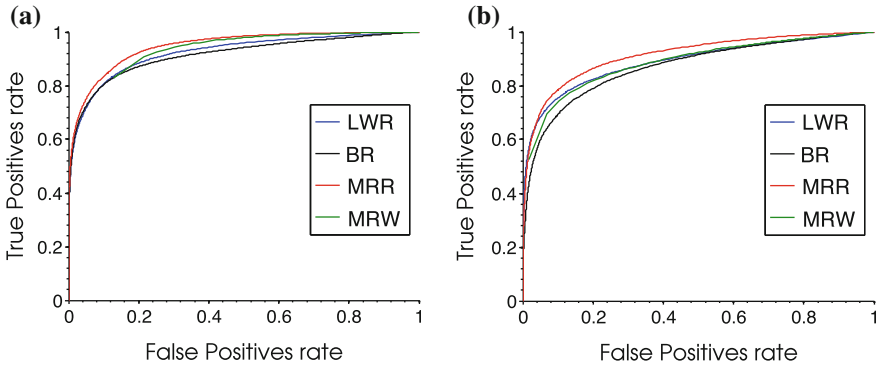


Fig. 1.17 ROC curves for correctness correspondence prediction. The experiment compares Lowe's ratio [34] (LWR), Brown's ratio [7] (BR), Meta-Recognition [44] (MRW), and MRRayleigh [16] (MRR) on the Oxford dataset [36]. MRRayleigh outperforms the other predictors. *Left* Prediction performance for SIFT matches. *Right* Prediction performance for SURF matches. **a** SIFT matches. **b** SURF matches

used to compute the ground truth for correct correspondences for SIFT and SURF matches. To obtain the receiver operating characteristic (ROC) curves, the threshold δ was varied from 0 to 1, and $k = 0.5\,\%n$, where n is the number of reference features. We can see in Fig. 1.17 that the proposed MRRayleigh (MRR) outperforms the other methods, Lowe's ratio [34] (LWR), Brown's ratio [7] (BR), and Meta-Recognition [44] which uses Weibull distribution for prediction, regardless of the descriptor.

1.4.3 Nonuniform Sampling Strategies for Robust Model Estimation

Because the MRRayleigh algorithm provides a confidence on the correctness of a NN matching decision, it is possible to create a nonuniform sampling strategy for robustly estimating a model leveraging the computed confidences. The classical method to estimate these models robustly in the presence of outliers is RANSAC [15]. This method samples the data uniformly to generate hypotheses or models, which later are checked against all the data to assess their quality. The hypothesis that explains most of the data is the solution that this method returns. To speed up the convergence of this method, nonuniform sampling strategies can be devised (e.g., [11, 16, 18, 42]).

SWIGS: A Nonuniform Sampling Strategy from MRRayleigh Predictions

We describe two approaches leveraging the benefits of extreme value theory (EVT). The first method, SWIGS [16], uses the confidence computed by MRRayleigh to compute a nonuniform sampling strategy. The strategy is obtained by computing a weight for every correspondence, i.e.,

$$w_i = \frac{p_i}{\sum_{i=1}^{n} p_i}, \tag{1.11}$$

where p_i is the confidence for the ith correspondence. These weights form a discrete probability mass function over the correspondences, which is used as the nonuniform sampling strategy.

To evaluate the performance of SWIGS (the nonuniform sampling strategy), we presented an experiment on homography estimation in a dense matching scenario [16] using the affine covariant features dataset [36]. The nonuniform sampling strategy was combined with MLESAC [48], a variant of RANSAC whose purpose is to calculate a hypothesis or model that maximizes a likelihood function instead of maximizing the support of the model.

The experiment compared SWIGS with other nonuniform sampling methods combined with MLESAC: BEEM [22]; a Guided-Sampling [47] with a general distribution considering all the imaging conditions (GEN); a Guided-Sampling [47] that considers only the distribution for a specific imaging condition (SPEC); BLOGS [6] where $m_l = d_1^{-1}$, and $m_{lr} = m_{lc} = d_2^{-1}$ as our approach considers a different matching procedure; and a classical random sampling (uniform distribution) for a baseline.

The results of this experiment are shown in Fig. 1.18, where the first two rows show the results obtained for SIFT, and the rest for SURF matches. The percentage of correct matches or correspondences are presented in the first and third rows, while the iterations are in the second and fourth rows. The x-axis indicates the index of the images contained in the considered sub-datasets (omitting the reference image, which is index 1); an increasing index represents a larger variation with respect to the reference image. Each column presents the results for a different sub-dataset: bikes, boat, graf, trees and wall, from left to right.

We can observe that SWIGS tends to require in general fewer iterations than the other methods (second and fourth rows) to find models that consider a comparable or higher percentage of correct matches within the allowed number of iterations (first and third row). We note that SWIGS, SPEC, and BEEM tend to find models that consider approximately the same number of matches. The GEN method struggles more to find models that consider a high percentage of correct matches in scenes with repetitive textures, e.g., wall, and trees sub-datasets; repetitive textures can cause a considerable overlap between correct and incorrect matching scores distributions. BLOGS and a random sampling (Uniform) method perform similarly in finding models that consider a high portion of the correct matches.

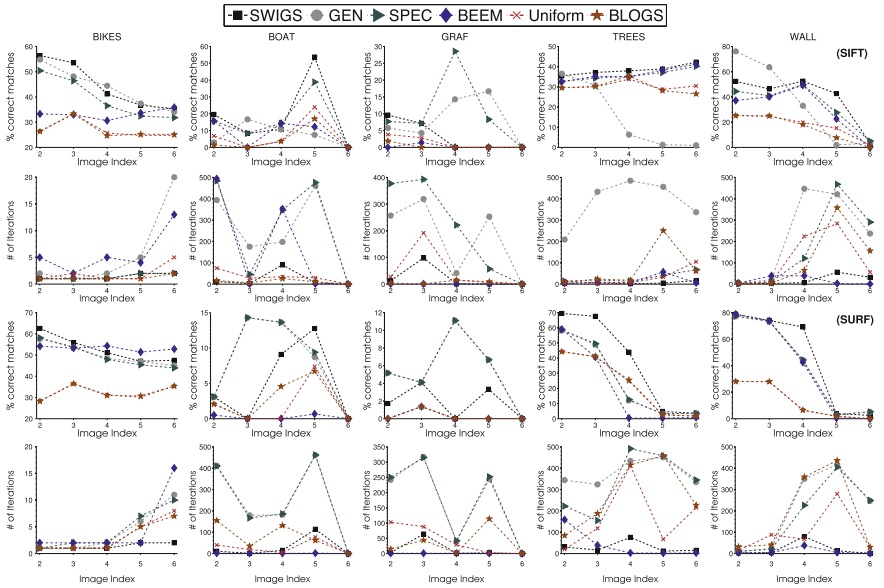


Fig. 1.18 Performance evaluation across several sub-datasets (bikes, boat, graf, trees, wall from *left* to *right*). Of all the 5000 repetitions of the experiment, the *first* and *third rows* present the median of the percentage of correct matches found by the best computed models within the allowed number of iterations, while the *second* and *fourth rows* present the median number of iterations at which the best model was found. The *first* and *second rows* present the results for SIFT, and the *third* and *fourth* for SURF

The experimental results presented in this section demonstrate that SWIGS can perform similarly or better in finding models that consider a good portion of correct matches in a dense matching scenario. The experiments also show that SWIGS tends to require fewer iterations than the other guiding sampling methods without sacrificing the number of correct matches found. Moreover, this confirms that MRRayleigh confidences tend to identify good matches, and these confidences yield an efficient and accurate nonuniform sampling strategy.

EVSAC: A Nonuniform Sampling Strategy for Low Inlier Ratio Cases

The second method that leverages extreme value theory (EVT), EVSAC [18], estimates the correctness belief p_i differently. The main premise of EVSAC is that there is a single pair of distributions F_c and $F_{\bar{c}}$ when matching two images. In contrast, MRRayleigh assumes there exist a pair of distributions F_c and $F_{\bar{c}}$ for every query feature, i.e., for every NN search for the query feature. Given this new assumption, the task is to find the parameters for F_c and $F_{\bar{c}}$ as well as the mixture

parameter ε to compute a model for the minimum distances used by the NN matcher. These minimum distances can be considered as samples drawn from a distribution $F = \varepsilon F_c + (1 - \varepsilon)F_{\bar{c}}$.

Algorithm 2 EVSAC

Require: $\left\{d_{i,1:k}\right\}_{i=1}^{n}$
Ensure: $\{w_i\}_{i=1}^{n}$ and $\{p_i\}_{i=1}^{n}$
1: $\mathbf{v} \leftarrow \text{Predict}\left(\left\{d_{i,1:k}\right\}_{i=1}^{n}\right)$
2: $(\alpha, \beta) \leftarrow \text{FitGamma}\left(\left\{d_{i,(1)} \text{ such that } v_i = 1\right\}\right)$
3: $(\mu, \sigma, \xi) \leftarrow \text{FitGEV}\left(\left\{d_{i,(2)}\right\}\right)$
4: Calculate the empirical cdf using $d_{i,j^\star}$
5: Find ε by solving (1.12)
6: Calculate posterior weights p_i using Eq. (1.13)
7: Calculate weights w_i using Eq. (1.14)
8: Use the weights w_i for generating hypotheses

EVSAC's algorithm (shown in Algorithm 2) computes these parameters as well as the new nonuniform sampling strategy. EVSAC requires the k smallest distances $\left\{d_{i,1:k}\right\}_{i=1}^{n}$ for every ith correspondence, and computes the weights w_i as well as the correctness confidence p_i. The first step in this algorithm is to label each correspondence as correct or incorrect (Step 1); for this step, EVSAC uses the MRRayleigh predictor algorithm. Subsequently, the algorithm fits a gamma distribution to the distances of those correspondences labeled as correct, i.e., $v_i = 1$, in step 2. Then, the algorithm fits a generalized extreme value distribution (GEV) to the second smallest distances in step 3.

EVSAC uses the GEV distribution to model the underlying distribution that the minimum distances from the incorrect correspondences follow; this is because now we have several minimum distances sampled from a single distribution $F_{\bar{c}}$. This implies that the mixture model explaining the minimum distances in the matching process becomes $F = \varepsilon F_c + (1 - \varepsilon)G_{\bar{c}}$, where $G_{\bar{c}}$ is the GEV distribution. Theorem 1 applies only for modeling the minimum distances sampled from $F_{\bar{c}}$ because we have several samples from this distribution, i.e., we have more incorrect correspondences assuming that there is at most a single correct correspondence. On the other hand, this Theorem does not apply to F_c because we sample a single sample at most when we match a query feature; recall that Theorem 1 requires a sufficiently large number of samples taken from the underlying distribution.

After estimating the parameters of the distributions, EVSAC estimates the mixture model parameter ε in steps 4 and 5. To do so, EVSAC solves the following constrained least-squares problem:

$$\begin{aligned} \underset{\mathbf{y}}{\text{minimize}} \quad & \frac{1}{2}\|A\mathbf{y} - \mathbf{b}\|_2^2 \\ \text{subject to} \quad & \mathbf{1}^{\mathrm{T}}\mathbf{y} = 1 \\ & \mathbf{0} \preceq \mathbf{y} \preceq \mathbf{u}, \end{aligned} \tag{1.12}$$

where the symbol $\preceq$ indicates entrywise comparison, and

$$A = \begin{bmatrix} F_c(d_1) & G_{\bar{c}}(d_1) \\ \vdots & \vdots \\ F_c(d_n) & G_{\bar{c}}(d_n) \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} F(d_1) \\ \vdots \\ F(d_n) \end{bmatrix}, \mathbf{y} = \begin{bmatrix} \varepsilon \\ \varepsilon' \end{bmatrix}, \text{ and } \mathbf{u} = \begin{bmatrix} \tau \\ 1 \end{bmatrix}.$$

The matrix $A \in \mathbb{R}^{n \times 2}$ is formed by evaluating all the n minimum distances on the cumulative distributions functions. The vector $\mathbf{b} \in \mathbb{R}^n$ is formed by evaluating the empirical cumulative distribution function on all the minimum distances. EVSAC imposes the constraint $\varepsilon \leq \tau$, as this improves the quality of the estimation of the mixing parameter. τ is computed as the ratio of the number of correspondences labeled as correct and n.

EVSAC computes in step 6 the correctness believes $p_i = \mathbb{P}(C = \text{correct}|d)$ for every correspondence using the Bayes' rule:

$$\mathbb{P}(C = \text{correct}|d) = \frac{\varepsilon f_c(d)}{\varepsilon f_c(d) + (1 - \varepsilon) g_{\bar{c}}(d)}, \tag{1.13}$$

where C is a discrete random variable indicating correctness, d is a minimum distance, and f_c and $g_{\bar{c}}$ are the probability density functions for F_c and $G_{\bar{c}}$, respectively. Subsequently, EVSAC calculates the weights w_i for every correspondence as follows:

$$w_i = \frac{p_i v_i}{\sum_{i=1}^{n} p_i v_i}, \tag{1.14}$$

where v_i is the binary value returned by the predictor in step 1. These weights w_i again describe a probability mass function over the correspondences which is used as the nonuniform sampling strategy.

We now present an evaluation of the performance of EVSAC to find the parameters of our probabilistic framework: ε, and the distribution parameters using the MRRayleigh predictor [16] only as the predictor in step 1. We compared the estimated parameters against the parameters obtained assuming that we had a perfect correct match detector.

We first examine the accuracy of the estimation of ε in Table 1.5. The estimate of ε using the upper bound in vector $\mathbf{u}$ used in (1.12), $\hat{\varepsilon}$, tends to be closer to the real value, while the estimate without the upper bound ($\tilde{\varepsilon}$) can overshoot sometimes.

Next, we examine the quality of the estimation of the different probability densities and the posterior used to compute the weights w_i. In the first column of Fig. 1.19, we can observe that EVSAC (continuous curves) is able to approximate with a good accuracy the mixture of densities obtained with the ground truth data (dashed curves). In the second column, we present the posterior probabilities computed from the estimated model (continuous curves) and the posterior obtained from the ground truth (dashed curves). This means that EVSAC estimates an accurate posterior that essentially maximizes the information in the matching distances when computing a confidence value.

Table 1.5 Estimation of ε comparison: $\hat{\varepsilon}$ is the estimation with τ set as an upper bound (see Eq. 1.12), and $\tilde{\varepsilon}$ is without

Image pairs	ε	$\hat{\varepsilon}$	$\tilde{\varepsilon}$
Oxford-Bark (1–4 SURF)	0.0131	0.0141	0.1870
Oxford-Boat (1–6 SURF)	0.0257	0.0270	0.1429
Oxford-Bark (1–3 SIFT)	0.0479	0.0438	0.1291
Oxford-Trees (1–6 SIFT)	0.1028	0.1119	0.2467
Strecha-Brussel (2–3 SIFT)	0.1855	0.2067	0.2263
Strecha-Brussel (1–2 SURF)	0.2964	0.3115	0.3632

The upper bounded estimate tends to provide more accurate estimations

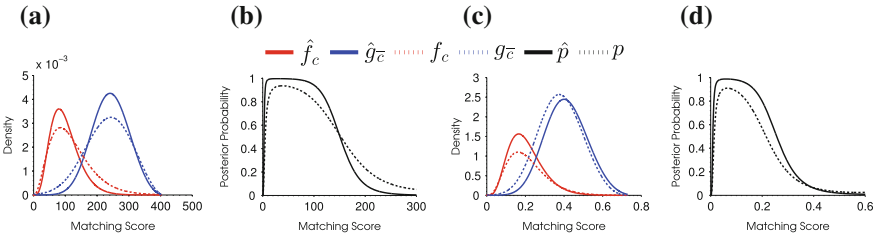
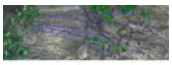
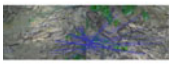
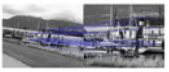
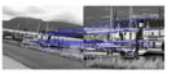


Fig. 1.19 Comparison of the mixture of densities and posterior probability computed using EVSAC against the ground truth for a pair of images with SIFT matches (**a**–**b**) and SURF matches (**c**–**d**). In both experiments the matching score metric is the Euclidean distance. The density estimations $\hat{f}_c$ and $\hat{g}_{\bar{c}}$ are close to the densities f_c and $g_{\bar{c}}$ computed with an oracle. In the second column, we compare the estimated posterior probability $\hat{p}$ with the posterior p computed with the oracle

To evaluate the nonuniform sampling strategy that EVSAC computes, a homography experiment is presented. EVSAC is compared against the following nonuniform sampling algorithms: Guided-MLESAC [47], BEEM's prior estimation step [22], BLOGS' global search mechanism [6], and PROSAC [11]. All these sampling algorithms were included in a classical hypothesis-test loop, where the support was always being maximized, and a solution was considered "good" if it satisfied the maximality constraint, i.e., the constraint that a good hypothesis was generated within a certain number of iterations (see [11] for more details on this constraint). The homography was computed using the OpenCV findHomography() function without the RANSAC option. An inlier (or correct correspondence) was considered if the reprojection error of the homography was less than 5 pixels. The algorithms were allowed to run until a maximum number of iterations (hypothesis test loops) calculated adaptively is reached, and the algorithm converged when 90 % of the inliers (correct matches) were detected. The found hypothesis was refined afterwards using a nonlinear method.

The results of this experiment are summarized in Table 1.6. The affine covariant features dataset [36] used for the experiment presented very challenging scenarios, where the inlier ratios ε ranged from 1–10 % for SIFT and SURF matches. The experiments were run 300 times. We present the average number of inliers detected (I); the average RMS reprojection error (E) in pixels w.r.t.to the error achieved by

Table 1.6 Homography estimation results for SIFT and SURF matches

		BEEM	BLOGS	PROSAC	GMLESAC	EVSAC
A: SURF	I	14 ± 0	14 ± 0	14 ± 0	14 ± 0	14 ± 0
	E	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
	M	1443	2524	4	1521	11
	T	563.1	1008.3	2	511	4.2
	F	0	0	0	0	0
$\varepsilon = 0.01, n = 992$	R	100%	96%	0.33%	0.33%	100%
B: SIFT	I	12 ± 3	12 ± 2	10 ± 2	11 ± 3	12 ± 2
	E	0.1 ± 0.02	0.1 ± 0.04	0.37 ± 0.03	0.24 ± 0.04	0.16 ± 0.03
	M	41	17	2752900	10044	10
	T	13.3	5.3	375482	1446.4	3.3
	F	6.6	10.7	136.7	37.8	12.1
$\varepsilon = 0.02, n = 992$	R	100%	100%	100%	100%	100%
C: SURF	I	24 ± 2	22 ± 4	27 ± 4	29 ± 2	24 ± 2
	E	0.1 ± 0.01	0.1 ± 0.01	0.1 ± 0.02	0.03 ± 0.05	0.2 ± 0.1
	M	3741	4072	1458250	209963	1965
	T	1172.3	1509.5	284838	28092.1	572.3
	F	2	9	3.2	2.9	4.2
$\varepsilon = 0.035, n = 992$	R	100%	99%	100%	100%	100%
D: SURF	I	39 ± 2	39 ± 2	38 ± 6	38 ± 9	39 ± 2
	E	0.04 ± 0.02	0.01 ± 0.001	0.02 ± 0.1	0.01 ± 0.2	0.02 ± 0.01
	M	39	82	655073	4151	4
	T	10.9	21.5	145207	838.4	1.3
	F	0	0.1	1.3	3.9	0.1
$\varepsilon = 0.04, n = 981$	R	100%	100%	98%	99%	100%
E: SURF	I	40 ± 2	40 ± 2	38 ± 6	33 ± 9	40 ± 2
	E	0.04 ± 0.03	0.002 ± 0.001	0.45 ± 0.53	0.02 ± 0.13	0.02 ± 0.01
	M	149	355	321952	4713	92
	T	42.9	98.5	56176.5	2111.9	26.3
	F	0.3	0.2	0.6	35.7	0.8
$\varepsilon = 0.05, n = 807$	R	100%	100%	100%	100%	100%
F: SIFT	I	41 ± 3	41 ± 4	36 ± 6	41 ± 3	41 ± 3
	E	0.08 ± 0.02	0.002 ± 0.01	0.17 ± 0.04	0.05 ± 0.02	0.04 ± 0.02
	M	71	14	218811	341	22
	T	15.7	3.5	40750.7	67.5	5.4
	F	1.7	0.7	8.4	1.1	0.1
$\varepsilon = 0.05, n = 807$	R	100%	100%	100%	100%	100%
G: SURF	I	81 ± 8	81 ± 9	60 ± 23	81 ± 8	82 ± 7
	E	0.02 ± 0.01	0.03 ± 0.009	0.02 ± 0.06	0.02 ± 0.006	0.03 ± 0.004
	M	177	193	4507	918	73
	T	49.6	53.6	1616.2	226.5	21.9
	F	0.8	0.7	13.3	0.6	0.4
$\varepsilon = 0.10, n = 992$	R	100%	100%	100%	100%	100%
H: SIFT	I	82 ± 9	82 ± 9	69 ± 16	80 ± 8	82 ± 8
	E	0.02 ± 0.016	0.05 ± 0.008	0.09 ± 0.04	0.03 ± 0.003	0.03 ± 0.003
	M	72	26	3649	773	8
	T	13.9	5.5	834	120.1	4.1
	F	0.7	1.1	4.5	1	0.8
$\varepsilon = 0.103, n = 992$	R	100%	100%	100%	100%	100%

The results are sorted by inlier ratio (ε) in ascending order. EVSAC performed well when the inlier ratio is low, and performed equivalently when the inlier ratio increased

the ground truth data; the average number of models/hypotheses generated (M); the average time in milliseconds (T); the average Frobenius norm of the error between estimated homography and the computed homography with the ground truth (F); and the percentage of "good" runs where each algorithm converged (R). The results are sorted in ascending order by the inlier ratio. We can observe that EVSAC tends to perform overall faster when the inlier ratio is very low (see rows **A**, **B**, **C**, **D**, and **E**), and performs equivalent or faster than BEEM and BLOGS as soon as the inlier ratio increased (see rows **F**, **G**, **H**). PROSAC and GMLESAC struggled to converge fast when the inlier ratio was very low ($\varepsilon < 11\,\%$).

1.4.4 Discussion and Future Work

We have presented two different nonuniform sampling strategies that can help in estimating models, such as homographies, essential matrices, and fundamental matrices, robustly in the presence of outliers. The two methods leverage the correctness confidences that the statistical theory of extreme values allows us to compute. These nonuniform sampling methods can help in speeding up various processes, such as structure-from-motion and feature-based tracking, for use in mobile applications. A natural extension of these nonuniform sampling algorithms is to modify them so that they can work for estimations of camera poses from 2D to 3D correspondences, which is an important step for augmented reality applications as shown in Sect. 1.3.

1.5 Summary

In this chapter, we have provided insight into some of the problems, constraints, and opportunities that arise in the domain of computer vision for mobile augmented reality applications. Mobile AR requires robust, real-time computer vision methods for tracking, modeling, localization, and other tasks, executing on a mobile device with a foreground process that may require significant resources, and with additional sensors that may aid the visual processing. As these devices become even more ubiquitous, powerful, and integrated into people's daily lives, the opportunities for mobile computer vision will continue to grow rapidly.

The TranslatAR sign translation system described in Sect. 1.2 provides an example of a full application using computer vision and augmented reality in a mobile environment. The indoor localization capability of Sect. 1.3 gives insight into a vision-based resource that may be used by AR or other kinds of mobile applications that require spatial information (i.e., camera pose). Advances in fast, robust keypoint correspondences and model estimation (Sect. 1.4) indicate how efficient low-level choices, informed by theory, can provide tracking and modeling that is well suited to the mobile domain with its limited resources.

With recent advances in all areas of mobile AR, there is now great enthusiasm for real-world applications on mobile devices—an increasingly important domain for the computer vision field.

Acknowledgments We wish to acknowledge our colleagues who were involved in various aspects of the research reported on in this chapter: Steffen Gauglitz, Shane Zamora, Jim Kleban, Marc Petter, Charles Baur, Pradeep Sen, Sergio Rodriguez. This work was partially supported by UC MEXUS-CONACYT (Fellowship 212913) and NSF award 1219261. Parts of this chapter present research originally published in references [16–18, 40, 41].

References

1. Arthur, D., Vassilvitskii, S.: K-means++: the advantages of careful seeding. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'07, pp. 1027–1035. Society for Industrial and Applied Mathematics, New Orleans, Louisiana (2007)
2. Bay, H., Ess, A., Tuytelaars, T., Van Gool, L.: Speeded-up robust features (SURF). Comput. Vis. Image Underst. **110**(3), 346–359 (2008)
3. Beis, J.S., Lowe, D.G.: Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (1997)
4. Benhimane, S., Malis, E.: Real-time image-based tracking of planes using efficient second-order minimization. Proc. IEEE Int. Conf. Intell. Robot. Syst. (IROS 2004) **1**, 943–948 (2004)
5. Bissacco, A., Cummins, M., Netzer, Y., Neven, H.: Photoocr: reading text in uncontrolled conditions. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV) (2013)
6. Brahmachari, A.S., Sarkar, S.: Blogs: balanced local and global search for non-degenerate two view epipolar geometry. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1685–1692 (2009)
7. Brown, M., Winder, S., Szeliski, R.: In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2005)
8. Canny, J.: A computational approach to edge detection. IEEE Trans. Pattern Anal. Mach. Intell. **8**(6), 679–698 (1986)
9. Castillo, E., Hadi, A.S., Balakrishnan, N., Sarabia, J.M.: Extreme Value and Related Models with Applications in Engineering and Science. Wiley, Hoboken (2005)
10. Cheng, C.-C., Peng, G.-J., Hwang, W.-L.: Subband weighting with pixel connectivity for 3-d wavelet coding. IEEE Trans. Image Process. **18**(1), 52–62 (2009)
11. Chum, O., Matas, J.: Matching with prosac—progressive sample consensus. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2005)
12. Coles, S.: An Introduction to Statistical Modeling of Extreme Values. Springer, Berlin (2001)
13. Crandall, D., Owens, A., Snavely, N., Huttenlocher, D.: SfM with MRFs: discrete-continuous optimization for large-scale reconstruction. IEEE Trans. Pattern Anal. Mach. Intell. **35**(12), 12 (2013)
14. Epshtein, B., Ofek, E., Wexler, Y.: Detecting text in natural scenes with stroke width transform. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2010)
15. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM **24**(6), 381–395 (1981)
16. Fragoso, V., Turk, M.: SWIGS: a swift guided sampling method. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2013)

17. Fragoso, V., Gauglitz, S., Zamora, S., Kleban, J., Turk, M.: TranslatAR: a mobile augmented reality translator. In: Proceedings of the IEEE Workshop on Applications of Computer Vision (WACV'11) (2011)
18. Fragoso, V., Sen, P., Rodriguez, S., Turk, M.: EVSAC: accelerating hypotheses generation by modeling matching scores with extreme value theory. In: Proceedings of IEEE International Conference on Computer Vision (ICCV) (2013)
19. Gao, J., Yang, J.: An adaptive algorithm for text detection from natural scenes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2001)
20. Gauglitz, S., Höllerer, T., Turk, M.: Evaluation of interest point detectors and feature descriptors for visual tracking. Int. J. Comput. Vis. **94**(3), 335–360 (2011)
21. Gauglitz, S., Sweeney, C., Ventura, J., Turk, M., Höllerer, T.: Live tracking and mapping from both general and rotation-only camera motion. In: Proceedings of the 11th IEEE International Symposium on Mixed and Augmented Reality (ISMAR'12), pp. 13–22. Atlanta, Georgia (2012)
22. Goshen, L., Shimshoni, I.: Balanced exploration and exploitation model search for efficient epipolar geometry estimation. IEEE Trans. Pattern Anal. Mach. Intell. **30**(7), 1230–1242 (2008)
23. Hartley, R.I., Zisserman, A.: Multiple View Geometry in Computer Vision. Cambridge University Press, Cambridge (2000). ISBN 0521623049
24. Jaderberg, M., Vedaldi, A., Zisserman, A.: Deep features for text spotting. In: Computer Vision ECCV 2014. Lecture Notes in Computer Science, vol. 8692, pp. 512–528. Springer International Publishing, Berlin (2014)
25. Kato, H., Billinghurst, M.: Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In: Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality, 1999 (IWAR'99), pp. 85–94 (1999)
26. Klein, G., Murray, D.: Parallel tracking and mapping for small AR workspaces. In: Proceedings of the Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07), Nara, Japan (2007)
27. Kneip, L., Li, H., Seo, Y.: UPnP: an optimal O(n) solution to the absolute pose problem with universal applicability. In: Computer Vision ECCV 2014. Lecture Notes in Computer Science, vol. 8689, pp. 127–142. Springer International Publishing, Berlin (2014)
28. Lee, C.W., Jung, K., Kim, H.J.: Automatic text detection and removal in video sequences. Pattern Recognit. Lett. **24**(15), 2607–2623 (2003)
29. Lee, C.-Y., Bhardwaj, A., Di, W., Jagadeesh, V., Piramuthu, R.: Region-based discriminative feature pooling for scene text recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2014)
30. Lepetit, V., Moreno-Noguer, F., Fua, P.: EPnP: an accurate O(n) solution to the PnP problem. Int. J. Comput. Vis. **81**(2), 155–166 (2009)
31. Levenberg, K.: A method for the solution of certain non-linear problems in least squares. Q. J. Appl. Math. **II**(2), 164–168 (1944)
32. Levenshtein, I.V.: Binary codes capable of correcting deletions, insertions, and reversals. Cybern. Control Theory **10**(8), 707–710 (1966)
33. Liu, Y., Goto, S., Ikenaga, T.: A contour-based robust algorithm for text detection in color images. IEICE—Trans. Inf. Syst. **E89–D**(3), 1221–1230 (2006)
34. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vis. **60**(2), 91–110 (2004)
35. Lucas, S.M.: LCDAR 2005 text locating competition results. Proc. IEEE Conf. Doc. Anal. Recognit. **1**, 80–84 (2005)
36. Mikolajczyk, K., Schmid, C.: A performance evaluation of local descriptors. IEEE Trans. Pattern Anal. Mach. Intell. **27**(10), 1615–1630 (2005)
37. Neumann, L., Matas, J.: Real-time scene text localization and recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2012)
38. Neumann, L., Matas, J.: Scene text localization and recognition with oriented stroke detection. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV) (2013)

39. Park, A., Jung, K.: Automatic word detection system for document image using mobile devices. In: Human-Computer Interaction. Interaction Platforms and Techniques. Lecture Notes in Computer Science, vol. 4551, pp. 438–444. Springer, Berlin (2007)
40. Paucher, P., Turk, M.: Location-based augmented reality on mobile phones. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops) (2010)
41. Petter, M., Fragoso, V., Turk, M., Baur, C.: Automatic text detection for mobile augmented reality translation. In: Proceedings of IEEE International Conference on Computer Vision Workshops (ICCV Workshops) (2011)
42. Raguram, R., Frahm, J.-M., Pollefeys, M.: A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus. In: Computer Vision ECCV 2008. Springer, Berlin (2008)
43. Rousseeuw, P.J.: Least median of squares regression. J. Am. Stat. Assoc. **79**(388), 871–880 (1984)
44. Scheirer, W.J., Rocha, A., Micheals, R.J., Boult, T.E.: Meta-eecognition: the theory and practice of recognition score analysis. IEEE Trans. Pattern Anal. Mach. Intell. **33**(8), 1689–1695 (2011)
45. Smith, R.: An overview of the tesseract ocr engine. In: Proceedings of the Ninth International Conference on Document Analysis and Recognition, ICDAR'07, vol. 02, pp. 629–633. IEEE Computer Society (2007)
46. Sweeney, C., Fragoso, V., Hllerer, T., Turk, M.: gDLS: a scalable solution to the generalized pose and scale problem. In: Computer Vision ECCV 2014. Lecture Notes in Computer Science, vol. 8692, pp. 16–31. Springer International Publishing, Berlin (2014)
47. Tordoff, B.J., Murray, D.W.: Guided-MLESAC: faster image transform estimation by using matching priors. IEEE Trans. Pattern Anal. Mach. Intell. **27**(10), 1523–1535 (2005)
48. Torr, P.H.S., Zisserman, A.: MLESAC: a new robust estimator with application to estimating image geometry. Comput. Vis. Image Underst. **78**(1), 138–156 (2000)
49. Wagner, D., Schmalstieg, D.: Artoolkitplus for pose tracking on mobile devices. In: Proceedings of the 12th Computer Vision Winter Workshop (CVWW'07), pp. 139–146 (2007)
50. Wagner, D., Mulloni, A., Langlotz, T., Schmalstieg, D.: Real-time panoramic mapping and tracking on mobile phones. In: IEEE Virtual Reality Conference (VR). IEEE, pp. 211–218 (2010)
51. Ye, Q., Gao, W., Wang, W., Zeng, W.: A robust text detection algorithm in images and video frames. Proc. IEEE Int. Conf. Inf. Commun. Signal Process. **2**, 802–806 (2003)
52. Ye, Q., Huang, Q., Gao, W., Zhao, D.: Fast and robust text detection in images and video frames. Image Vis. Comput. **23**(6), 565–576 (2005)