TranslatAR: A Mobile Augmented Reality Translator

Victor Fragoso Steffen Gauglitz Shane Zamora Jim Kleban Matthew Turk
University of California, Santa Barbara
{vfragoso@cs, sgauglitz@cs, char42@cs, kleban@ece, mturk@cs}.ucsb.edu

Abstract

We present a mobile augmented reality (AR) translation system, using a smartphone's camera and touchscreen, that requires the user to simply tap on the word of interest once in order to produce a translation, presented as an AR overlay. The translation seamlessly replaces the original text in the live camera stream, matching background and foreground colors estimated from the source images. For this purpose, we developed an efficient algorithm for accurately detecting the location and orientation of the text in a live camera stream that is robust to perspective distortion, and we combine it with OCR and a text-to-text translation engine. Our experimental results, using the ICDAR 2003 dataset and our own set of video sequences, quantify the accuracy of our detection and analyze the sources of failure among the system's components. With the OCR and translation running in a background thread, the system runs at 26 fps on a current generation smartphone (Nokia N900) and offers a particularly easy-to-use and simple method for translation, especially in situations in which typing or correct pronunciation (for systems with speech input) is cumbersome or impossible.

1. Introduction

Written text is one of the most common methods for conveying information in our daily lives. However, when written text is encountered in a language unfamiliar to an individual, the information cannot be conveyed. To alleviate this problem, many aides for translation have been devised, from simple dictionaries to electronic devices that simplify the translation process in one way or another. These devices can be classified by the way in which the text is entered (*e.g.*, manually searching through an alphabetic index, typed in via keyboard, speech input) as well as how the translation is presented (*e.g.*, as text in a book, text on a screen, voice output).

Using a smartphone with touchscreen and camera as the physical device, we present a system for automatic translation of visual text that has an efficient and easy-to-use input

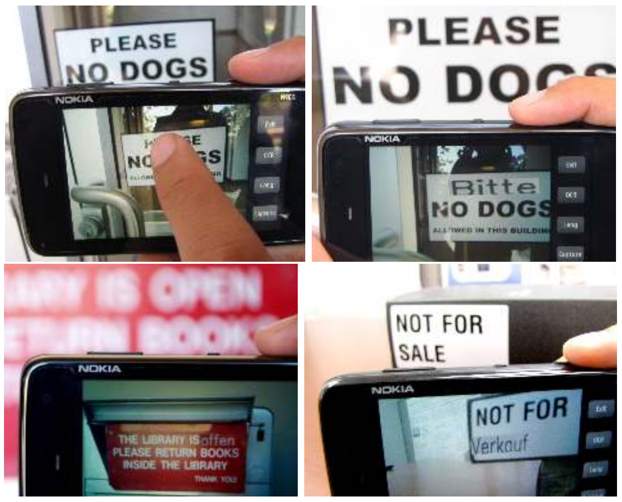

Figure 1. TranslatAR in operation: the user detects a word he/she wishes to translate and taps on it (top left). The system automatically detects the extent of the word, extracts the letters via OCR, and translates the text. The translation is then presented as live AR overlay (top right). Bottom row: TranslatAR used in two other situations.

and a natural and compelling form of presentation. The use of our system, dubbed TranslatAR, is shown in Fig. 1: the user simply taps on a word he/she wishes to translate, the system automatically detects, extracts, tracks and translates the text, and finally presents the translation as a live Augmented Reality (AR) overlay.

In contrast to previously proposed systems, almost all of the processing is done on the mobile device itself, providing immediate feedback. Only the text-to-text translation uses an online translation service in the current implementation, but it is conceptually straightforward to integrate an appropriate dictionary on the device itself.

The remainder of this paper is structured as follows: Section 2 gives an overview of relevant related work; Section 3 provides details of the algorithms used in our system; Section 4 gives implementation details; Section 5 presents experimental results; finally, we conclude in Section 6.

2. Related Work

Computer vision-based translation. Systems for semi-automatic and automatic vision-based translation have previously been proposed, *e.g.*, [7, 13, 17, 18]. However, all of them are based on a client-server architecture to offload expensive operations (detection, extraction and translation) and hence cannot operate without network connection or provide immediate feedback. The translation is provided either in the form of speech synthesis [18] or a simple text display on the screen[1].

Similar systems are now available as commercial applications, such as Google Goggles[2] (albeit with manual text detection) and ABBYY Fototranslate[3].

While these systems are similar to ours, our system is able to execute all of the image processing tasks on the mobile device while tracking the text in real-time, and it offers a particularly easy-to-use (single click) and compelling (AR overlay of the result) user interface.

Text detection in natural scenes. Localizing and recognizing text in video streams has been researched extensively for tasks such as information retrieval and license plate identification (*e.g.*, [8]).

In the ICDAR 2005 competition for automatic text detection [12], the algorithm by Becker (unpublished) performed best, but it is rather expensive and hence not suitable for live translation on a mobile device. Chen and Yuille [4]'s algorithm is significantly faster and performed almost as good. Very recently, Epshtein *et al.* [6] presented a detector based on the "stroke width transform" and showed very promising results on the ICDAR dataset.

Several systems have been devised for mobile platforms in particular, but they expect the word to be in the center of the image [14] and/or do not handle perspective distortion [5, 11].

Tracking & AR on mobile phones. Visual tracking and AR without markers in real-time have successfully been demonstrated on mobile phones, both with known natural feature targets [16] and without [9]. While the above systems detect and then track keypoints, tracking may also be done via image alignment-based techniques [1, 3].

3. Overview of the System

The system's architecture, shown in Fig. 2, was designed such that all expensive operations run in a background thread, while the system maintains interactive framerates for tracking and augmentation. The individual components are described in the following sections.

[1]Note that [7] mentions live AR feedback as future work.
[2]http://www.google.com/mobile/goggles
[3]http://www.abbyy.com/fototranslate/

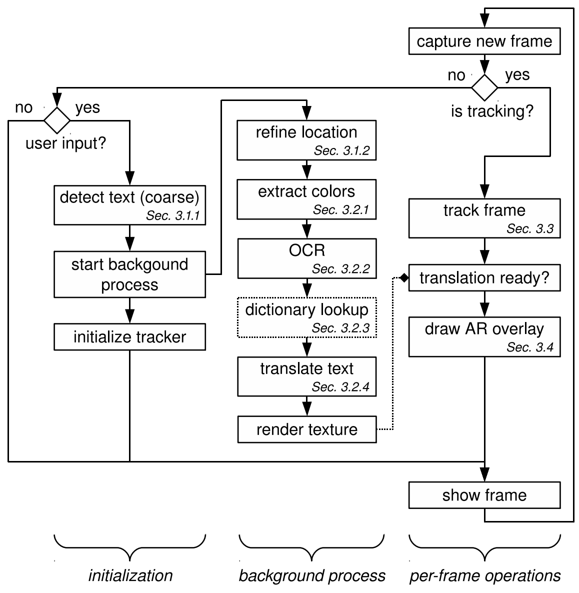

Figure 2. Architecture of TranslatAR. Initialization and per-frame operations run in the main thread, while the rest runs in the background.

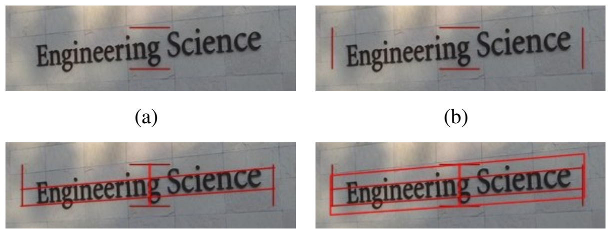

Figure 3. Text detection in operation. First, the vertical extent of the text is determined (a), then – using the assumed text height – the horizontal extent (b). A constrained and modified Hough Transform is used to determine the exact baseline and orientation (c), and finally, the area is expanded to account for ascenders and descenders (d).

3.1. Text detection

Given a point c onto which the user tapped, the system first finds the bounding box around the word, then the exact location and orientation of the text within.

3.1.1 Bounding box

To find approximate upper and lower boundaries of the text, 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 upwards and downwards, respectively, until the following criterion is met (for δy consecutive scanlines):

$$\max_{(x,y)\in s_h} |I_x(x,y)| < \varepsilon \tag{1}$$

that is, until s_h does not cross any vertical edges. The example in Fig. 3(a) shows the final upper and lower location of s_h.

The same idea is applied to compute the left and right boundaries, sweeping a vertical line segment s_v over I_y. We make use of the 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 assumed text height). Here, the required width of the "gap" δx is set slightly larger so that the algorithm does not stop between letters. The result is shown in Fig. 3(b). Values for ε, δx, δy and the lengths of s_h and s_v were obtained experimentally (cf. Section 5).

This coarse region of interest, namely, the bounding box, is sufficient to initialize the tracker (cf. Section 3.3). A separate background thread is started in which all remaining operations for text extraction and translation are executed while the foreground system can capture, track and display live frames.

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 non-uniform backgrounds (cf. evaluations in Section 5).

3.1.2 Location & orientation refinement

To detect the exact location and orientation of the upper and lower "baselines" of the text, we apply a constrained and modified Hough Transform as follows: Firstly, 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 (we think reasonable) assumption/limitation that the user will hold the phone roughly parallel to the text.

Secondly, we optimize 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 may be seen in Fig. 3(c). Finally, lines are moved vertically until no edge intersections are detected to account for ascenders and descenders (Fig. 3(d)).

The resulting quadrilateral region of interest is warped into a rectangle, correcting any perspective distortion and showing the text as if seen orthogonally.

3.2. Text extraction, Recognition and Translation

The warped image produced as described above is used to extract background and foreground color, as well as to "read" the word via OCR.

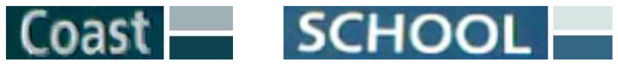

Figure 4. Two detected and automatically rectified text areas with estimated foreground and background color on their right side.

3.2.1 Foreground and background color estimation

We assume that the letters have a single, constant color with reasonable 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 [2] with $k = 2$. To differentiate between foreground and background, we retrieve a few labeled samples along the left and right borders and assume that the background color is the one with the majority of the collected labels (this is justified as our detection algorithm automatically includes a small margin, cf. Section 3.1.2).

This approach estimates both colors very accurately and fast when the assumptions are met. It will fail for very non-uniform background 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.

3.2.2 OCR

With the rectified image of the word, we rely on a standard OCR system for extraction and recognition of the letters. We used Tesseract [15], 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), we calculate the ratio of alphanumeric characters to all characters in the string as a rough indicator of successful extraction.

3.2.3 Dictionary lookup

The following (optional) step was motivated by preliminary tests with the OCR which showed that single letters were frequently misrecognized (cf. Fig. 6 in the evaluation).

With a string returned from the OCR, we search through a dictionary of valid words to identify the nearest neighbor with respect to the Levenshtein distance [10]. 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. Its benefit will be evaluated in Section 5.

3.2.4 Translation

With the extracted string, we use Google Translate[4], 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.

3.3. Visual Tracking

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 it is in the general case: (1) we may assume 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) we are only interested in tracking over short periods of time (as long as it takes the system to obtain the translation + the user to read it), (4) we can assume a "cooperative" user who will not move the phone jerkily.

We implemented our tracking system based on ESM [3], in which an image region is tracked by iteratively minimizing the difference between a reference frame (the template) and 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.

3.4. AR Overlay

Based on the transformation computed by the tracker, a graphical augmentation is rendered onto the live video screen; first a placeholder ("please wait...") is displayed while the text is being translated, and then, as soon it becomes available, the translation itself.

4. Implementation Details

We implemented our system on the Nokia N900, 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. Our code was developed in C++, using OpenCV and libCVD for computer vision tasks (processing frames of size 320x240), 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 frame's homography as initial estimate for the 8 degree-of-freedom alignment.

[4]`http://www.google.com/webmasters/igoogle/translate.html`

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 & preprocess frame	21.9
tracking	8.5
render AR overlay & display frame	7.7
total per-frame	38.1

Table 1. Average execution times on the Nokia N900 for the main steps of the processing pipeline. With the expensive steps offloaded into a background thread, the system maintains a frame rate of about **26 fps**.

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.

5. Evaluation

Runtime. Table 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.

Text detection accuracy. We used the ICDAR 2003 detection dataset[5] to evaluate our text localization method. 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 our algorithm was designed to work with video frames of a fixed size, we resized the images to 320x240 pixels. To conduct automated evaluation, we simulate the required user input: as starting point c, we take the center of the rectangle provided by the dataset and we adjust it properly to the new dimensions. As the dataset only provides an enclosing horizontal rectangle, and since our algorithm computes the (more accurate) quadrilateral, we then calculate the minimal enclosing horizontal rectangle to be able to

[5]`http://algoval.essex.ac.uk/icdar/Datasets.html`

compare against the provided ground truth.

The performance measures proposed in [12] 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 non-intersecting pairs.

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 our 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 calculate *pixel-wise* precision and recall (*e.g.*, as used by [14]), *i.e.*, the ratio of pixels correctly labeled as text vs. all pixels labeled as text, and the ratio of pixels correctly labeled as text vs. all text pixels.

We first optimized the values of our parameters (cf. Section 3.1) using the training part of the ICDAR set, then evaluated the metrics on the test part.

We obtained a 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 in [12], but cannot compete with the best scoring algorithms in [12] and [6], which achieve precision and recall values of 60 – 70%. It should again be noted that our algorithm requires a single point as input, while the other algorithms are fully automatic, but also that our 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 [6, 12]).

A few examples of good and bad detection are shown in Fig. 5. The algorithm is prone to "overshoot" all the way to the borders of the image for non-uniform backgrounds, but 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).

OCR with and without dictionary. To test the OCR, we used the ICDAR 2003 recognition dataset, which consists of 1110 images, each containing a single word, as well as the corresponding ground truth strings. We then used the Levenshtein distance [10] to measure the similarity between the obtained and ground truth strings. Fig. 6 compares the results of the raw OCR with OCR plus the dictionary module and shows that the dictionary has the potential of increasing the percentage of correct words significantly; hence it is promising to investigate appropriate, well-scaling solutions.

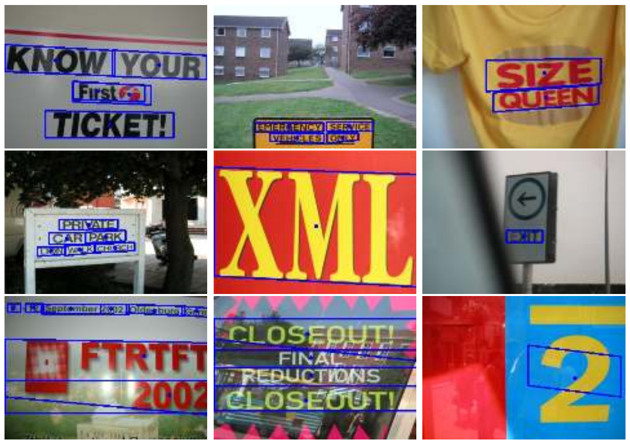

Figure 5. Examples of good (top and mid rows) and bad (bottom row) text localization on the ICDAR 2003 dataset. The blue point in each quadrilateral represents the (simulated) input of the user. Our algorithm was able to very accurately detect the text at different scales and under perspective distortion. The failure cases are mostly due to very non-uniform background and/or lighting effects (first two). For very large letters, the expansion algorithm used to detect the text's bounding box (cf. Section 3.1.1) can stop *inside* one of the letters (bottom right).

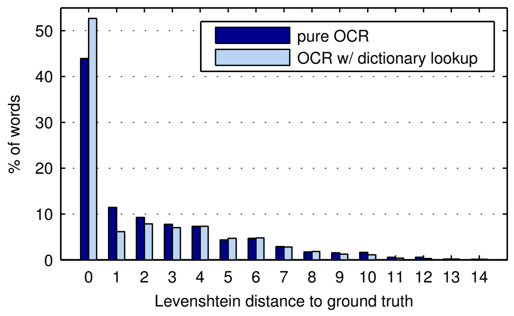

Figure 6. Error (Levenshtein distance) measured between the extracted strings and the ground truth. Results are shown with and without dictionary lookup.

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 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.

6. Conclusions

In this work, we presented a prototype for a real-time, mobile, visual translation system, which requires only a single tap on the word of interest and presents its result as a live AR rendering.

Component	# of words	% of all failures	% of all
detector failed	7	16.3	8.9
color est. failed	6	14.0	7.6
OCR failed	26	60.8	32.9
translation failed	4	9.3	5.1
correct result	36 of 79	–	45.6

Table 2. Reasons of failure of the detection-extraction-translation process on a set of 30 video clips. 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.

We showed results quantifying the running time, the accuracy of the semi-automatic text detection, the potential of using a dictionary to improve the OCR, and the frequency of failure of the different components.

Though the accuracy is not as high as with some text detector systems, our algorithm achieved respectable precision and recall values on the difficult ICDAR dataset given the constraints of real-time performance. Nevertheless, an important aspect for future work would be to increase the accuracy and robustness. A promising approach would be to integrate the recent work by Epshtein *et al*. [6] and to try to decrease the runtime requirements by, for example, leveraging the seed point provided by the user.

The component-wise analysis revealed that the OCR engine is the most likely cause of failure. Apart from waiting for improved OCR and hoping that improved detection also improves the OCR result, one could also leverage the outcome of the tracker to fuse multiple images over time in order to increase the image quality. Additionally, our evaluations suggest that the recognition result may be improved by integrating a spell checker.

To our knowledge, we have developed the first fully operational proof-of-concept system to enable users to perceive text in the environment and in the language of choice via the "magic lens" of AR with a single click.

Acknowledgments: This research was supported in part by Nokia Corporation. We would like to thank Daniel Vaquero, Saiph Savage, Julia Vraspir, and Panuakdet Suwannatat for their help. V. Fragoso would like to thank UC MEXUS-CONACYT (Fellowship 212913) for the funding.

References

[1] A. Adams, N. Gelfand, and K. Pulli. Viewfinder alignment. *Computer Graphics Forum (Proc. Eurographics*, pages 597–606, 2008.

[2] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *SODA '07: Proc. 18th annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[3] S. Benhimane and E. Malis. Real-time image-based tracking of planes using efficient second-order minimization. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, pages 943–948, 2004.

[4] X. Chen and A. L. Yuille. Detecting and reading text in natural scenes. In *Proc. IEEE CVPR 2004*, March 2004.

[5] T. N. Dinh, J. Park, and G. Lee. Low-complexity text extraction in korean signboards for mobile applications. In *Computer and Information Technology, 2008. CIT 2008. 8th IEEE Intl. Conf. on*, pages 333 –337, 8-11 2008.

[6] B. Epshtein, E. Ofek, and Y. Wexler. Detecting text in natural scenes with stroke width transform. In *Proc. IEEE CVPR 2010*, July 2010.

[7] I. Haritaoglu. Scene text extraction and translation for handheld devices. In *Proc. IEEE CVPR 2001*, volume 2, pages II–408–II–413 vol.2, 2001.

[8] A. Jain and B. Yu. Automatic text location in images and video frames. In *Proc. 14th Intl. Conf. on Pattern Recognition*, volume 2, pages 1497–1499, Aug 1998.

[9] G. Klein and D. Murray. Parallel tracking and mapping on a camera phone. In *Proc. 8th IEEE and ACM Intl. Symposium on Mixed and Augmented Reality (ISMAR 2009)*, pages 83–86, Oct. 2009.

[10] V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 10(8):707–710, 1966. Original in *Doklady Akademii Nauk SSSR* 163(4): 845–848 (1965).

[11] X. Liu and J. Samarabandu. An edge-based text region extraction algorithm for indoor mobile robot navigation. In *Proc. IEEE Intl. Conf. Mechatronics and Automation*, volume 2, pages 701–706 Vol. 2, July-1 Aug. 2005.

[12] S. M. Lucas. ICDAR 2005 text locating competition results. In *Document Analysis and Recognition, 2005. Proc.. 8th Intl. Conf. on*, pages 80 – 84 Vol. 1, 29 2005.

[13] H. Nakajima, Y. Matsuo, M. Nagata, and K. Saito. Portable translator capable of recognizing characters on signboard and menu captured by built-in camera. In *ACL '05: Proc. ACL 2005 on Interactive poster and demonstration sessions*, pages 61–64, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

[14] A. Park and K. Jung. Automatic word detection system for document image using mobile devices. In *HCI (2)*, pages 438–444, 2007.

[15] R. Smith. An overview of the tesseract ocr engine. In *Proc. 9th Intl. Conf. on Document Analysis and Recognition (ICDAR'07)*, pages 629–633, Washington, DC, USA, 2007. IEEE Computer Society.

[16] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Pose tracking from natural features on mobile phones. In *Proc. 7th IEEE and ACM Intl. Symposium on Mixed and Augmented Reality (ISMAR'08)*, Cambridge, UK, Sept. 15–18 2008.

[17] Y. Watanabe, K. Sono, K. Yokomizo, and Y. Okada. Translation camera on mobile phone. In *Multimedia and Expo, 2003. ICME '03. Proc.. 2003 Intl. Conf. on*, volume 2, pages II–177–80 vol.2, July 2003.

[18] J. Yang, J. Gao, Y. Zhang, and A. Waibel. Towards automatic sign translation. In *Proc. 1st Intl. Conf. on Human language technology research (HLT'01))*, pages 1–6, Morristown, NJ, USA, 2001. Association for Computational Linguistics.