Video-Based *In Situ* Tagging on Mobile Phones

Wonwoo Lee,Youngmin Park,Vincent Lepetit, and Woontack Woo, *Member, IEEE*

***Abstract*—We propose a novel way to augment a real-world scene with minimal user intervention on a mobile phone: the user only has to point the phone camera to the desired location of the augmentation. Our method is valid for horizontal or vertical surfaces only, but this is not a restriction in practice in manmade environments, and it avoids going through any reconstruction of the 3D scene, which is still a delicate process on a resource-limited system like a mobile phone. Our approach is inspired by recent work on perspective patch recognition, but we adapt it for better performances on mobile phones. We reduce user interaction with real scenes by exploiting the phone accelerometers to relax the need for fronto-parallel views. As a result, we can learn a planar target *in situ* from arbitrary viewpoints and augment it with virtual objects in real-time on a mobile phone.**

***Index Terms*—Mobile Phone, Augmented Reality, Camera Registration, Vanishing Point Detection.**

I. Introduction

SMARTPHONES have become a very popular platform for Augmented Reality (AR) applications. Their use is becoming widespread, and they are often equipped with hardware useful for localization and relatively good computational capacities. Several recent works have shown that it is possible to develop Computer Vision techniques for AR on mobile phones [1]–[5].

In this paper, we consider a mobile AR tagging application, where real-world objects are augmented by virtual contents through a mobile phone camera. In this scenario, users can select a target object in a database and interact with virtual contents that are overlaid on the target object. They can also add a new target object to the database and overlay virtual contents on it as well. The current target detection approaches have limitations in this scenario: Learning a new target takes time especially on mobile devices because of their limited computational power and resources; 2) the 3D structure or a fronto-parallel view image of a target is required to learn the target's appearance for recognition.

We present a novel Computer Vision-based approach that makes it easy to add augmentations to the real world, even for a non-expert user. As shown in Figure 1, we avoid the need to go through a 3D reconstruction phase, which is still delicate to perform correctly, and cumbersome as well, especially on a resource-limited system like a mobile phone. We also avoid using feature points, as they are not available in every scene. Instead, we combine a recent patch recognition technique [6] adapted to mobile phone platforms and an image rectification method that uses both Computer Vision techniques and the phone's accelerometers.

W. Lee, Y. Park, and W. Woo are with Gwnagju Institute of Science and Technology, Gwangju, 500-712, South Korea.
V. Lepetit is with EPFL CVLab, Switzerland.

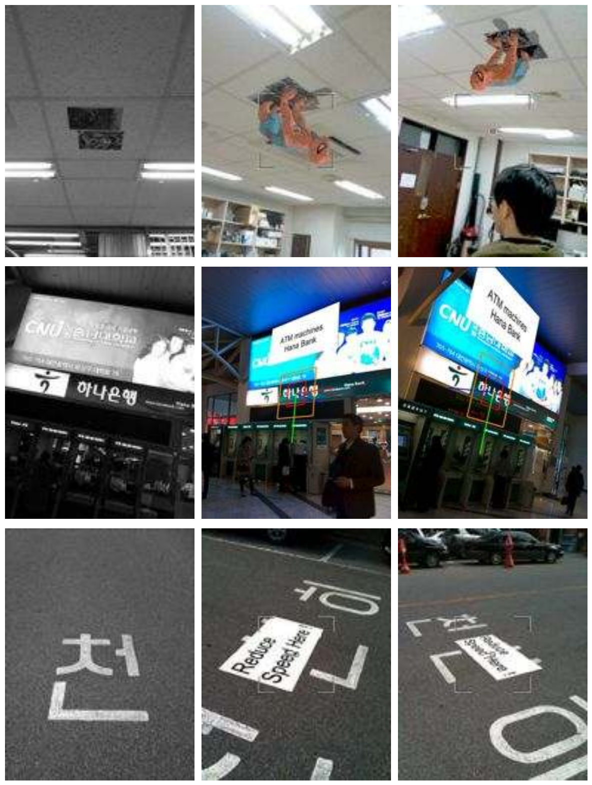


Fig. 1: Overview of our approach. The user simply has to point the mobile phone toward the desired location for the augmentation, and then take a picture like the ones on the left column. Our method guesses the surface orientation for a coherent insertion, by combining the phone's accelerometers and Computer Vision techniques. It can then recognize the location even from new viewpoints, and track it for real-time augmentation (see middle and right columns).

In our approach, the user simply has to point the phone camera toward the location he or she wants to augment. Through the phone accelerometer and line segments in the captured image, we can retrieve the surface orientation and rectify the captured image, and then insert the virtual objects in a coherent fashion. This is possible for horizontal or vertical surfaces only; however, manmade environments do not present a restriction in practice. Most of the time, our algorithm can correctly estimate the real surface orientation—either horizontal or vertical; otherwise the user can correct it quite easily. The scale factor can be defined by moving the phone toward or away from the surface, before fine-tuning it. This mode of operation results in a very intuitive interaction.

Once the user has defined the virtual content, its location in the real world can be recognized, even from new viewpoints and tracked in 3D for consistent rendering. For detection, we adapted Gepard, a template-based approach proposed in [6] to a mobile phone platform, because it works even on low-textured surfaces. The main idea behind Gepard is to compare a set of templates, each template corresponding to the average appearance of the surface from a given viewpoint when the camera pose is slightly changed, with the texture around feature points.

Here, we skip the feature point detection step and use larger patches instead for greater robustness. We also replaced the way the templates are computed, which consumes a large amount of memory in Gepard, by rendering and blurring operations. This is more suitable given the limited resources of the phone architecture, and it takes only few seconds.

Another difference with Gepard is that we can relax the need for a fronto-parallel view to build the set of templates. In [7], we showed how to exploit the phone's accelerometers to rectify the captured image into a fronto-parallel view. However, for vertical surfaces, this was possible only for rotation around the pitch axis. This paper describes an extended version of that process, which introduces a method exploiting both Computer Vision techniques and the phone's accelerometers to estimate the surface orientation under general viewing conditions.

In the remainder of the paper, we first review related work in Section II. Sections III and IV detail how the set of templates are built. Detection and tracking of a target from the templates are explained in Section V. Experimental results are given in Section VI and Section VII concludes the paper.

II. Related Work

Several recent works have demonstrated that it is possible to run Computer Vision algorithms for localization and 3D tracking on mobile phones [1]–[5]. They are all based on feature points and therefore require a fair amount of texture to work correctly. Moreover, mobile phones often have relatively low-quality cameras, which tend to blur the images under fast motion and make the feature points difficult to detect.

We therefore considered Gepard, an alternative method based on template matching, which was proven to be adaptable to poorly textured objects and blurry images [6] [1]. Given an image patch to detect, Gepard generates a set of "mean patches." Each mean patch is computed as the average of the patches seen over a limited range of viewpoints, and the ranges over all the mean patches cover all possible views. Then, by comparing an input patch to the mean patches, one can recognize it and get an estimate of the camera's viewpoint. Parallel Tracking and Mapping (PTAM) [2] relies on a related method as it compares downscaled, blurred images for camera relocalization [8], but it cannot generalize to unseen points of view.

However, Gepard is not directly adapted to mobile phone applications. It considers only patches centered on feature points. Since we wanted to avoid feature point detection, we skipped the feature points detection and use comparatively much larger patches to achieve greater robustness. Gepard also computes the mean patches as a linear combination of eigenpatches; unfortunately, this requires a great deal of memory to store all the precomputed data. We therefore propose a way to simulate the computation of the mean patches that does not require precomputed data.

[1]The Gepard algorithm is referred to as ALGO2 in [6].

Another restriction of Gepard is that it requires a fronto-parallel view of the original patch to detect, or equivalently, knowledge of its 3D orientation. This could be avoided, for example, by using an automated 3D reconstruction of the scene, but a non-expert user would still find that difficult to perform, and it would require camera motion before augmenting the scene anyway. Other works have developed interactive 3D reconstruction using AR [9]–[12], but they still require some time and expertise.

In this work, we take a much more drastic approach, but one that appears to be very convenient in actual practice. We assume that the real surface is planar and either horizontal or vertical, and we try to guess its relative orientation with the phone by using the phone accelerometers. This results in a very intuitive and quick process, which is very desirable on a mobile phone.

III. Estimating the Surface Orientation

In the scenario of our application, the user points the phone toward the surface that is to be augmented and captures an image. As we want to augment the surface in a convincing 3D fashion, even from novel viewpoints, we need to know the 3D orientation of the surface in the captured image. From this orientation, we can generate views of the surface from other viewpoints that will be used for recognition. This process will be detailed in the next section. This section focuses on the surface orientation estimation.

We first use the accelerometer values to guess whether the surface targeted by the phone user is either horizontal or vertical. Let us denote by θ_h the angle the phone makes with a horizontal plane as provided by the accelerometer, so that $\theta_h = 0$ when the camera points toward the horizon. Then, the following assumptions are often true in practice:

- If $-\frac{\pi}{4} < \theta_h < +\frac{\pi}{4}$ then the surface is vertical,
- otherwise the surface is horizontal.

If our guess should be wrong, the user can correct it by directly choosing a surface model, either horizontal or vertical.

If the surface is horizontal, it becomes relatively easy to estimate the surface orientation because the accelerometers can provide the phone orientation with respect to the gravity force that is collinear with the normal of a horizontal surface. However, doing this with a vertical surface is more difficult: The surface may be slanted, as depicted in Figure 1, and the orientation of this type of surface cannot be estimated from the accelerometers only. We therefore provide a method that combines the accelerometers output with Computer Vision techniques. We consider, in turn, these two cases, estimating the orientation of horizontal and vertical surfaces, for the remainder of this section.

A. Horizontal Surface

In the case of a horizontal target, it is possible to compute the surface orientation from the accelerometers, as previously noted. However, we still need the surface translation to define

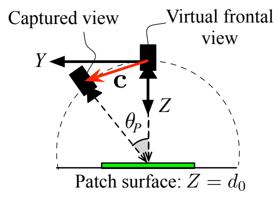


Fig. 2: Relationship between the captured view and the fronto-parallel view.

a full pose. Unfortunately, it is not possible to calculate the distance between the camera and the surface from a single image. Instead we use an arbitrary value d_0. In practice, d_0 is chosen to set the augmentation scale to a value so that the augmentation more or less fills in the captured image. Thus, the user can define the scale of the augmentation by simply moving the phone toward or away from the surface. As shown in most of the figures in this paper, the augmentation will correspond to a large object if the camera is far away from the surface; conversely, the augmentation will correspond to a small object if the camera is close to the surface. This is very intuitive, but it is limited to a certain range of scale within which the user can move the phone, and the interface lets the user adjust the scale if necessary.

The relationship between the captured view and fronto-parallel view is illustrated in Figure 2. Without loss of generality, we can set the pose of the virtual camera in the fronto-parallel location as $[\mathbf{I}|\mathbf{0}]$. The orientation obtained as explained above gives us the rotation matrix $\mathbf{R}$ for the captured image, which is a rotation around the X-axis in this coordinate system. It is easy to see that the coordinates of the camera center, $\mathbf{c}$, are $[0, d_0 \sin\theta_p, d_0(1-\cos\theta_p)]^\top$, and the translation vector for the captured image is $\mathbf{t} = -\mathbf{R}\mathbf{c}$.

From [13], the expression of the homography $\mathbf{H}_{f\leftarrow c}$ that warps the captured image to the virtual frontal view is then:

$$\mathbf{H}_{f\leftarrow c} = \mathbf{K}\left(\mathbf{R} - \frac{\mathbf{t}\mathbf{n}^\top}{d_0}\right)^{-1}\mathbf{K}^{-1}, \tag{1}$$

where $\mathbf{K}$ is the camera calibration matrix and $\mathbf{n}$ the vector $[0,0,-1]^\top$. This will be used in the next section to generate the data required to detect the target surface and estimate its orientation from novel views.

B. Vertical Surface

The assumption of the sole existence of pitch rotation, which is used in the horizontal target case, is not generally applicable to a vertical target: The orientation between the camera and the target surface changes, depending not only on the camera's movement but also on the target's rotations in the vertical axis. We therefore use the image itself, in addition to the accelerometers to estimate the orientation of the vertical surface.

The orientation of a planar surface can be estimated from the vanishing points of its projection. Vanishing point estimation from a single image has been extensively studied [13], and most of the algorithms, including recent ones, rely on straight line extraction and clustering [14], [15]. However, vanishing point estimation is still a burden on mobile phones due to their limited computational power.

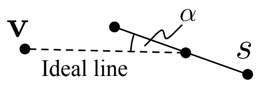


Fig. 3: Distance $\alpha(\mathbf{s},\mathbf{v})$ between a point, $\mathbf{v}$, and a line segment, $\mathbf{s}$. It is defined as the angle between $\mathbf{s}$ and the straight line that passes through $\mathbf{v}_v$ and the mid-point of $\mathbf{s}$.

We therefore propose a method that also exploits the accelerometers: By predicting the vanishing point of vertical lines from the accelerometer values, we can speed up and make this estimation more reliable.

The phone accelerometer provides the direction of gravity, $\mathbf{g}$, in the phone's local coordinate system. Since $\mathbf{g}$ is vertical, the vanishing point, $\mathbf{v}_v$, of vertical lines in the captured image can be obtained by simply projecting $\mathbf{g}$ onto the image plane:

$$\mathbf{v}_v = \mathbf{K}\mathbf{g}\,. \tag{2}$$

However, we cannot rely only on the accelerometer values because the phone accelerometer values are unstable due to sensor noise and small hand movement. Hence, we only use Eq. (2) to predict $\mathbf{v}_v$ and refine it further using information from the image.

We first extract straight line segments using the fast algorithm proposed in [16]. We ignore segments shorter than a threshold l_{th} because their directions are not reliable and could affect the vanishing point estimation ($l_{th} = 15$ pixels works well in practice). We then consider the segments that are likely to be vertical lines by measuring their distances to the initial estimate of $\mathbf{v}_v$. We adopt the distance function illustrated in Figure 3 and proposed in [17]. It is defined as the angle between the segment and the line that passes through $\mathbf{v}_v$ and the mid-point of the segment.

The segments for which this distance is smaller than a threshold are considered to be projections of vertical 3D lines. From these segments, we can estimate $\mathbf{v}_v$ by using *Random Sample Consensus (RANSAC)* [18]. The point that minimizes the mean distance from inliers is finally kept as the vanishing point, $\mathbf{v}_v$, of vertical lines.

For the vanishing point, $\mathbf{v}_h$, of horizontal lines, we use the J-Linkage algorithm [14], based on line clustering. J-Linkage first builds the m hypotheses, the candidates for $\mathbf{v}_h$. The hypotheses are computed as the intersections of two lines randomly selected from the set of extracted straight line segments from which we removed the vertical lines used to estimate $\mathbf{v}_v$.

To retain only the promising hypotheses only, we apply the orthogonality constraint, which is that two vanishing points should satisfy:

$$(\mathbf{K}^{-1}\mathbf{v}_v)^\top \cdot (\mathbf{K}^{-1}\mathbf{v}_h) = 0\,. \tag{3}$$

We therefore keep the intersection $\mathbf{p}$ of two lines as a hypothesis only if it satisfies:

$$(\mathbf{K}^{-1}\mathbf{v}_v)^\top \cdot (\mathbf{K}^{-1}\mathbf{h}) \leq \tau_h\,, \tag{4}$$

where $\mathbf{h}$ is the intersection point of the two lines. τ_h is a fixed threshold. We compute m hypotheses $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_m$. Five hundred hypotheses are typically used in [14], but in practice considering $m = 100$ hypotheses is sufficient, thanks to the orthogonality constraint.

J-Linkage then determines which hypotheses can correspond to the vanishing point by clustering the lines using an agglomerative approach. Initially each line is considered as a cluster, and the similarity between two clusters is measured by the Jaccard distance [19]:

$$d(C_1, C_2) = \frac{|\mathrm{pref}(C_1) \cup \mathrm{pref}(C_2)| - |\mathrm{pref}(C_1) \cap \mathrm{pref}(C_2)|}{|\mathrm{pref}(C_1) \cup \mathrm{pref}(C_2)|}, \tag{5}$$

where $\mathrm{pref}(C_1)$ and $\mathrm{pref}(C_2)$ are the "preference sets" of the two clusters. If a cluster contains only one line, its preference set is defined as the set of hypotheses that are close enough to the line; that is, if the distance function $\alpha(,)$ of Figure 3 is smaller than a threshold. If a cluster contains more than one line, its preference set is the intersection of the preference sets of its elements.

At each iteration, the clusters are compared against each other, and the two clusters having the minimal distance are merged. This process is iterated until the minimal distance between clusters becomes 1, i.e., there is no overlap between the preference sets of any two clusters. Each final cluster therefore provides one point. The point minimizing the orthogonality test in Eq. (3) is chosen as the vanishing point, $\mathbf{v}_h$, of horizontal lines.

Once both vanishing points have been obtained, we can compute the surface orientation. The rotation, $\mathbf{R}$, between the frontal view and the captured view is obtained as detailed in the appendix. The translation vector, $\mathbf{t}$, can then simply be computed as

$$\mathbf{t} = \mathbf{R}\mathbf{d} - \mathbf{d}, \tag{6}$$

where $\mathbf{d} = (0, 0, d_0)^\top$. From $\mathbf{R}$ and $\mathbf{t}$, the warping homography $\mathbf{H}_{f \leftarrow c}$ is computed as in Eq. (1).

Figure 4 shows the results of vanishing points estimation and image rectification. The vertical and horizontal lines are robustly extracted from the input image and vanishing points are estimated from them. As we can see the initial vertical vanishing point is close to the correct solution but is still inaccurate. Our method refines it with line segments, and the resulting horizontal and vertical vanishing points are then quite accurate. It can also estimate vanishing points in cluttered scenes as shown in rows 2 and 3 of Figure 4.

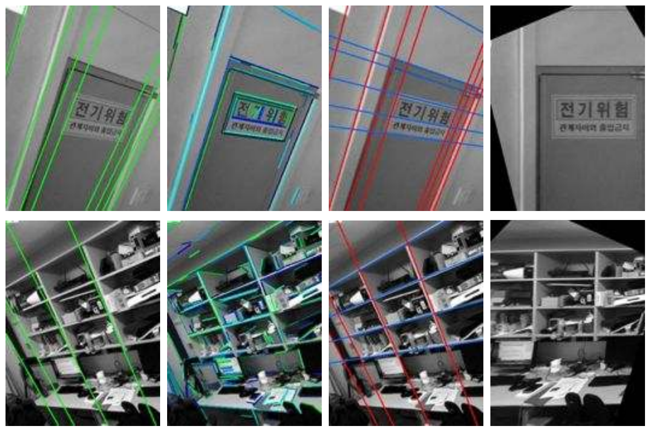


Fig. 4: Vanishing points detection and image rectification. First column: lines passing through the initial solution of the vertical vanishing point $\mathbf{v}_v$; second column: detected line segments; third column: lines passing through the estimated vanishing points; fourth column: fronto-parallel view image. The red and blue lines pass through $\mathbf{v}_v$ after refinement and the horizontal vanishing point $\mathbf{v}_h$.

IV. Generating Data for Patch Recognition

A. Review of Gepard [6]

Given a reference image patch, $\mathbf{p}$, in a frontal view, Gepard computes a set of "mean patches." Then it recognizes a surface visible in the captured image and estimates its orientation by matching it against the mean patches. The original expression of a mean patch $\overline{\mathbf{p}_h}$ is:

$$\overline{\mathbf{p}_h} = \frac{1}{|\mathscr{P}_h|} \sum_{P \in \mathscr{P}_h} \mathbf{w}(\mathbf{p}, P), \tag{7}$$

where P represents a camera pose, and $\mathbf{w}(\mathbf{p}, P)$ is a patch $\mathbf{p}$ seen under pose P. Each set $\mathscr{P}_h$ is made of poses around a pose that we will denote by P_h. The poses P_h are regularly sampled, and together all the $\mathscr{P}_h$'s span the set of all possible poses.

In Gepard, learning a patch simply means computing the corresponding $\overline{\mathbf{p}_h}$. Then, for an input patch, $\mathbf{q}$, one can compute:

$$e = \min_h \|\mathbf{q} - \overline{\mathbf{p}_h}\|^2, \text{ and } \hat{h} = \operatorname*{argmin}_h \|\mathbf{q} - \overline{\mathbf{p}_h}\|^2. \tag{8}$$

If the patch difference, e, is small, $\mathbf{q}$ is the same patch as $\mathbf{p}$ but is viewed from a pose close to $P_{\hat{h}}$. That gives a good estimate of the patch spatial orientation, which is further refined through template matching techniques.

In practice, Eq. (7) is not used directly, as this would be very costly. Instead, Gepard uses an approximation that exploits the fact that image warping is a linear transformation. A patch, $\mathbf{p}$, is decomposed into its mean and principal components as

$$\mathbf{p} \propto \overline{\mathbf{v}} + \sum_{l=1}^{L} \alpha_l \mathbf{v}_l \tag{9}$$

where $\overline{\mathbf{v}}$ and $\mathbf{v}_i$ are the mean and principal components of a large set of image patches. L is the dimension of principal components. Then, Eq. (7) becomes

$$\overline{\mathbf{p}_h} \propto \frac{1}{|\mathscr{P}_h|} \sum_{P \in \mathscr{P}_h} \mathbf{w}\left(\overline{\mathbf{v}} + \sum_{l=1}^{L} \alpha_l \mathbf{v}_l, P\right). \tag{10}$$

In Gepard, both $\overline{\mathbf{v}}$ and $\mathbf{v}_i$ are computed offline. See [6] for more details.

This approach requires a large amount of memory for storing the precomputed data, and is therefore not suitable for a mobile phone. We give below another approach that does not rely these memory-consuming precomputations.

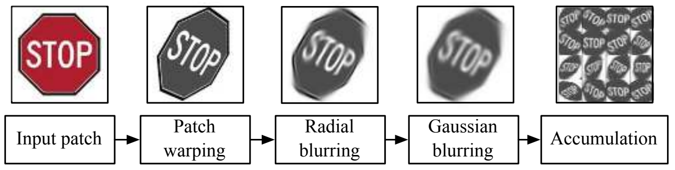


Fig. 5: Creation of the mean patches by blurring. To compute each mean patch, the reference patch is first warped, and radial blur and Gaussian blur are then applied. The resulting patches are accumulated into a texture on the GPU to send them to the CPU in one single read.

B. Patch Learning on a Mobile Phone

As can be seen in [6], the mean patches look like the reference patch after some non-uniform blur. This is related to Geometric Blur [20], which also blurs images for recognition and matching; however Geometric Blur relies on Gaussian smoothing with a spatially varying standard deviation, which is slow. We propose here an alternative to generate the mean patches, which is also based on blurring, but is a more efficient method. As shown in Figure 5, we use a combination of radial blur and Gaussian blur, performed on the mobile phone's graphics processing unit (GPU), to compute the mean patches.

a) Warping: In order to approximate a mean patch, $\overline{\mathbf{p}_h}$, the central patch in the surface frontal view is first rendered into a new patch we denote by $\mathbf{p}_h$ to correspond to pose P_h. Rendering on the phone's GPU takes only about 0.3 *ms*, whereas CPU-based patch warping took about 100 *ms* in our experiments.

To generate the poses, we regularly sample the rotations every 20 degrees around the three axes. We also use 3 scale factors (0.5, 1, 2) that we apply to d_0 before computing the translation in order to detect the surface from a range of distances. We then directly use OpenGL ES to render $\mathbf{p}_h$.

b) Radial blur: In the second pass, radial blur is applied to $\mathbf{p}_h$ to get a new patch $\mathbf{r}_h$. The intensity of each pixel, m, of $\mathbf{r}_h$ is computed as the average of the pixel intensities over an arc of a circle centered on the patch center c and going through m. The length of the arc, l, varies linearly with the distance between c and m:

$$l = \theta_r \|c - m\| \,. \tag{11}$$

In this formula, θ_r is a parameter expressed in radians and we use the value $\theta_r = 0.17$, which, in practice, is about 10 degrees. The pose sampling step and the maximum radial blur range are experimentally determined to guarantee reasonable detection performance. According to the results shown in Figure 6, our algorithm achieves good performance with the selected parameters [2]. Although the performance is a little better with the 5 degrees sampling step, the 10 degrees sampling step was selected because it gives almost the same performance with a fewer number of views, which makes the learning stage faster.

To confirm the effectiveness of the radial blur, we measured the patch detection performance against viewpoint changes

[2]The performance is measured with a set of images that are used for the experiments shown in Figure 13.

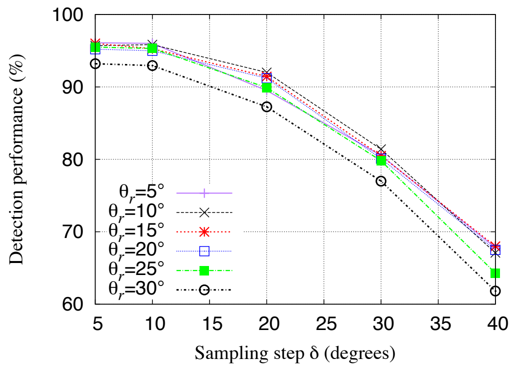


Fig. 6: Detection performance with varying pose sampling steps and maximum blur ranges.

on the Graffiti image set [3] with different blurring schemes. In the experiment, we changed the viewpoint by rotating the reference image by 0 to 70 degrees and measured the similarity to the reference patch through *Normalized Cross Correlation* (*NCC*). Patch detection is considered successful if the similarity exceeds 0.9. Figure 7 shows the results, where R and $G(m)$ represent the radial blur and the Gaussian blur with a $m \times m$ kernel, respectively: The mean patches computed by combining the radial blur with the Gaussian blur outperform those generated through Gaussian blur only.

c) Gaussian blur: Gaussian blur is then applied to $\mathbf{r}_h$, and the resulting patch approximates a mean patch $\overline{\mathbf{p}_h}$ as given by Eq. (7). The Gaussian filter is separable, and therefore is implemented with two 1D filters for efficiency. In practice, we use $\sigma = 11$ for the Gaussian kernel standard deviation.

d) Downsampling and Accumulation: As in Gepard, $\overline{\mathbf{p}_h}$ is downsampled from 128×128 to 32×32 and normalized to be robust to light changes. It is finally stored in a texture buffer of the GPU with the other generated mean patches. The accumulation of multiple patches in a texture reduces the number of readbacks from the GPU.

V. Detection and Tracking

Once a patch has been learned, it can be detected in new images to initialize a tracking algorithm based on template matching. If a known patch is visible in the captured image, we first extract the patch in the image center to detect the patch and to estimate its orientation. By contrast with Gepard, we apply to this patch the transformation detailed in Section IV; that is, we apply the same radial and Gaussian blurs and downscaling. This gives us more tolerance to translation and rotation, and the small additional computational burden is possible as it is done on only one patch for each captured image.

This gives us an input patch $\mathbf{q}$, and we can proceed as in Eq. (8): If e is small enough, we assume the input patch is one of the learned patches, and we use the corresponding pose $P_{\hat{h}}$, to initialize a tracking algorithm. We chose to use the ESM-Blur algorithm [21], as it is robust to motion blur, which is

[3]Available at http://www.robots.ox.ac.uk/~vgg/research/affine

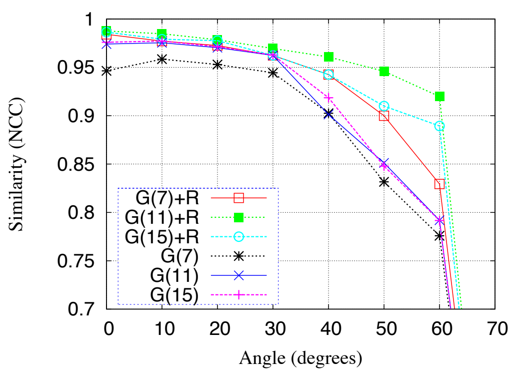


Fig. 7: Effectiveness of radial blur. Combining the radial blur and the Gaussian blur outperforms simple Gaussian blurring.

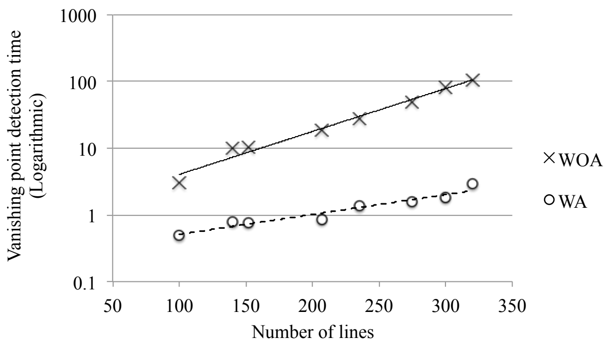


Fig. 9: Logarithmic plot of vanishing point detection speed depending on the number of lines. By exploiting the accelerometers, our approach (*WA*) is more than 10 times faster than the method in [14] (*WOA*).

a frequent problem with the low-quality cameras of mobile phones. ESM-Blur is used, first, to refine the pose provided by the detection, and then to track the patch in the subsequent captured images. When tracking fails, we re-run the detection procedure. Tracking with ESM-Blur is accelerated by NEON, a set of SIMD (Single Instruction Multiple Data) instructions of ARM CPUs for faster speed.

VI. Experimental Results

We implemented our method on both a mobile phone and a PC. We used Apple's iPhone 3GS and iPhone 4 as our mobile phone platforms and a PC with a 2.4 GHz CPU and a GeForce 8800GTX GPU. Cameras capture videos in 480×360 on the mobile phones and 640×480 on the PC. Note that the speed of our approach is independent of the size of the input images. In both platforms, cameras are calibrated in advance. We assume that the phone camera's focal length is fixed, although it has an auto-focus function. We always set the focus of the camera at the center of image, where we take a patch to learn it.

We set the size of an input patch to 128×128 and the number of views for patch learning to 225, which provides good detection performance with reasonable speed. Currently, the algorithm is implemented for single target detection. The amount of memory required for the data learned from a target depends on the sampled patch size and the number of views to learn. In our experiments, each mean patch is downsampled to 32×32, and it requires 4 kilobytes for pixel intensities and 36 bytes for a 3×3 homography pose matrix. For 225 views, about 907 kilobytes are required in total to store the data learned from a target.

A. Fronto-parallel Image Generation

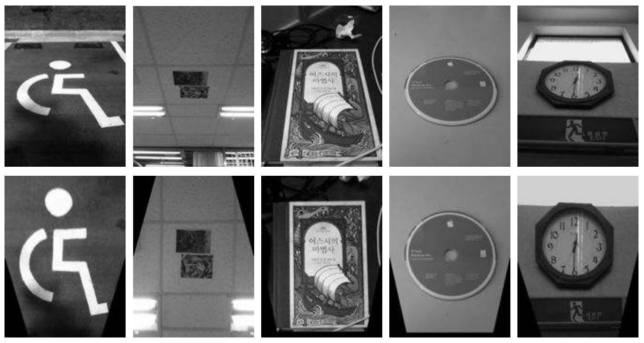
Figure 8 shows the captured images of planar targets and the corresponding fronto-parallel view images. Horizontal targets are rectified based on the phone accelerometer values and our approach successfully retrieves the corresponding images as they would be seen from frontal views. Rectification results of vertical targets are shown in Figure 8(b). Using vanishing points allows users to capture images of the targets from arbitrary viewpoints and the target images are correctly rectified even when the textures of the targets are complex. A vertical target can also be rectified through the accelerometer if there are not enough lines for vanishing point detection in its image, as shown in the last column of Figure 8(a). However, in this case, the camera's motion relative to the target surface is limited to the pitch rotation only as we assume for the horizontal target.

Rectification for horizontal targets is accomplished swiftly because no additional complex computations are required for rectification. Warping a 320×480 image takes about 110-130 *ms*. On the other hand, to do the same thing with vertical targets takes a few seconds, due to the vanishing point detection step. The speed of vanishing point detection depends on how many lines exist in the captured image. Typically, 100 to 250 lines are extracted from a real scene and vanishing point detection takes less than 2 seconds. We compared the vanishing point detection speed of our approach (denoted by *WA*) with the method in [14] (denoted by *WOA*), which retrieves vanishing points only from lines. In our implementation of [14], we set the maximum number of hypotheses to 500 and skipped the refinement step through Expectation-Maximization because we also did not conduct the refinement step. As shown in Figure 9, adopting the phone accelerometer drastically reduces vanishing point detection time.

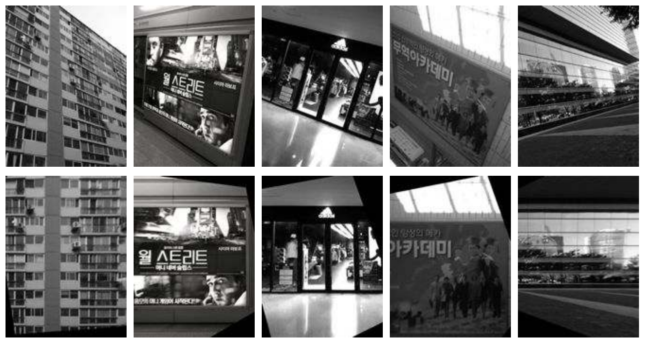
To evaluate the accuracy of detected vanishing points, the reference vanishing points are obtained by manual operations. Ten people, who were non-experts in Computer Vision and image processing, were asked to find horizontal and vertical lines as accurately as possible from 10 images captured from ordinary real environments. The error between the reference vanishing point and the one computed by our method is measured as the angle between two 3D rays, i.e., back-projections of them. The average error is 1.02 degrees for the horizontal vanishing point and 0.86 degrees for the vertical vanishing point. For our purpose of frontal view generation, the accuracy is in an acceptable range, considering the rectification results in Figure 8. With a small loss of accuracy, we can get significant speed ups on mobile phones.

B. Learning Speed

Figure 10 compares the time required for learning on both PC and mobile phone platforms. While the four main steps—



(a)



(b)

Fig. 8: Examples of estimated fronto-parallel views (bottom row) from different input images (top row). (a) Only the phone accelerometers are needed for horizontal surfaces (columns 1 to 4). It is also true when a vertical surface is rotated in pitch only, as shown in column 4. (b) However, for slanted vertical surfaces, we need to combine the accelerometer's output with vanishing points detection.

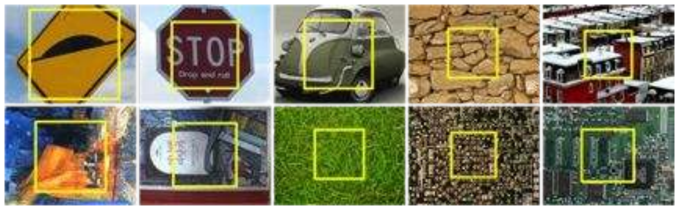


Fig. 11: Planar targets used for evaluation. From the top left: Sign-1, Sign-2, Car, Wall, City, Cafe, Book, Grass, Macmini, and Board. The patches delimited by the yellow squares are used as a reference patch.

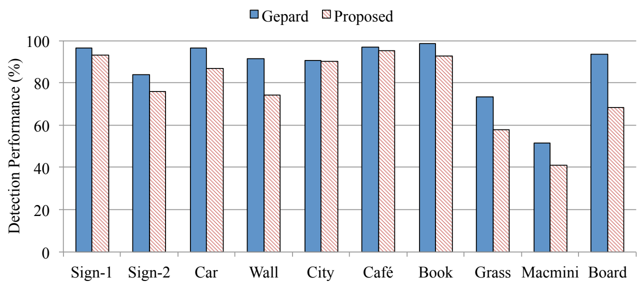


Fig. 12: Performance comparison with Gepard. Our approach performs slightly worse in terms of recognition rates, but it is better adapted to mobile phones.

warping, radial blur, Gaussian blur, and accumulation—take approximately the same time on a PC, on the mobile phone, radial blur clearly becomes the bottleneck. This can be explained by the fact that only horizontal and vertical memory accesses are needed for Gaussian blurring, whereas radial blur needs more complex accesses, and the phone's GPU is not adapted yet to this type of access.

Given the improvements in terms of recognition rates that radial blur provides, it is worth using the radial blur despite its relatively heavy computational burden. For 225 mean patches, the mean patch computation time is about 5 seconds, which is a good trade-off between the time required for learning and the recognition performance.

C. Patch Detection and Tracking

Figure 13 shows the patch detection results against viewpoint changes and image noise. We used 10 planar targets with different textures in this experiment.[4] Apart from the viewpoint changes, we added zero-mean Gaussian noises with a standard deviation σ ranging from 0 to 30 to pixel intensities.

The first two targets (Sign-1 and Sign-2) are low-textured objects, which are common in the real world. The proposed algorithm successfully detects those two targets under viewpoint changes up to 60 degrees, regardless of the amount of noise. The next 5 targets are more textured. Our detection method remains robust to viewpoint changes up to 60 degrees and noises up to $\sigma = 30$. The last three targets (Grass, MacMini, and Board) have rich but repetitive textures, with thin structures. This case was the worst one for our approach, and the *NCC* score dropped more quickly.

The comparison between our algorithm and Gepard [6] is shown in Figure 12. In all data sets, Gepard outperformed our method, but the performance loss was not large if the targets have rich textures. Gepard also reveals some weakness in repeated textures, although it is still better than ours. We expect this is because Gepard exploits the two-step pose optimization based on [23], [24], while we use only one [21].

Compared to Gepard, the advantage of our approach is in memory usage, which is crucial on mobile phone platforms. Typically, Gepard requires a load of about 30-90MB of precomputed data, depending on its parameters [5], which is a large amount of data for a mobile phone platform to store in memory. In contrast, there is no need for precomputed data in our method. We have obtained the capability of online learning on mobile phones by reducing the memory usage, while sacrificing detection performance only a little.

The speed of patch detection is shown in Table I. Patch

[4]Some image data is available at http://www.metaio.com/research. See [22] for details.

[5]We used the source code available at http://campar.in.tum.de/Main/StefanHinterstoisser.

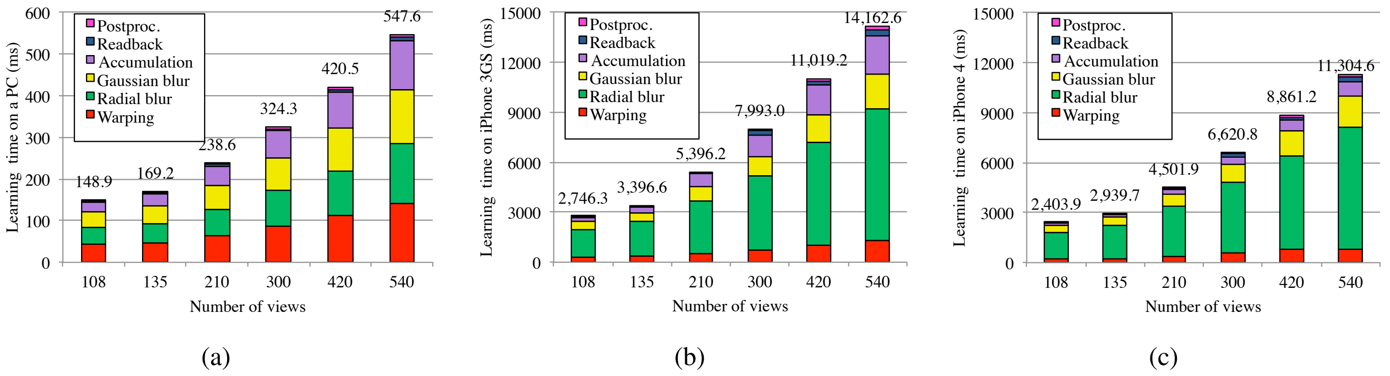


Fig. 10: Computation times for learning the mean patches. The overall time increases on (a) the PC , (b) iPhone 3GS, and (c) iPhone 4 as the number of views increases.

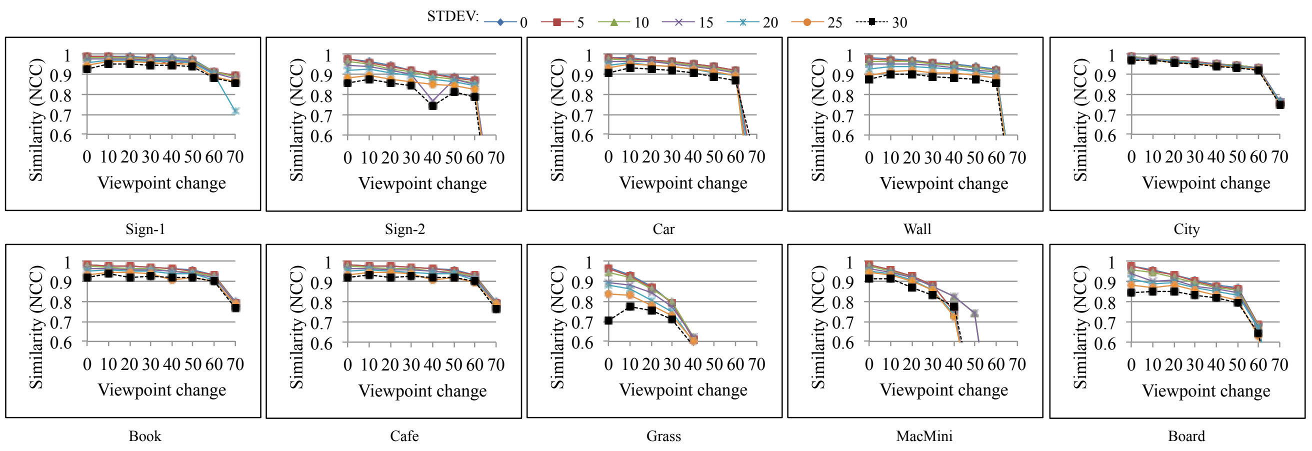


Fig. 13: Evaluation of the accuracy of the retrieved pose for different types of images. We applied our method to retrieve the pose and plotted the Normalised Cross-Correlation between the original patch and the test images rectified by the retrieved pose. Each curve corresponds to a different amount of noise. For the first patches, we obtain very good results up to 60 degrees. Patches with high frequencies like Grass, MacMini, and Board yield lower performances.

TABLE I: Patch detection speed (unit: ms)

	iPhone 3GS		iPhone 4	
	Mean	Stdev	Mean	Stdev
Mean patches comparison	3.06	0.25	2.5	0.19
Pose estimation / tracking	64.0	30.9	51.7	22.6

detection consists of two main parts, mean patches comparison and pose estimation. The mean patches comparison takes about 3 *ms* with 225 views. The speed of pose estimation and tracking with ESM-Blur can vary greatly as the number of iterations required for pose optimization changes depending on the accuracy of the initial pose provided by patch detection. In practice, we set the maximum number of iterations to 50, which results in a reasonable speed (10 to 20 frames per second) and accurate registration.

D. Real World Examples

Figure 14 shows the result of the patch detection on outdoor objects. The first column shows the input images. In all the examples, the fronto-parallel views of the objects are unavailable in the captured images, but the method we proposed estimated correctly the right camera pose. After learning from the input images, the objects are successfully detected and tracked by our algorithm on mobile phones. Note that our algorithm works well even with poor textures. In the examples, we assume that the origin of the world coordinate system is at the center of the detected patch, and virtual objects are synthesized on it. The orientations of virtual contents are pre-defined for specific target types, i.e., horizontal and vertical.

The target's surface should be planar, or it should be far enough to be seen as planar from the camera's viewpoint. If our method is applied to a non-planar scene, detecting the target may not be possible because the target's image will be distorted in fronto-parallel view generation step.

E. Discussions

1) Accelerometer and Gyroscope Sensors: The accelerometer measured on mobile phones are noisy and this may affect the estimation of the frontal view. However, the noises are not large, and there is only negligible distortion in the warped fronto-parallel images. In the case of vertical targets, the accelerometer values are used only for initialization and thus, the noises does not affect to the vanishing point estimation

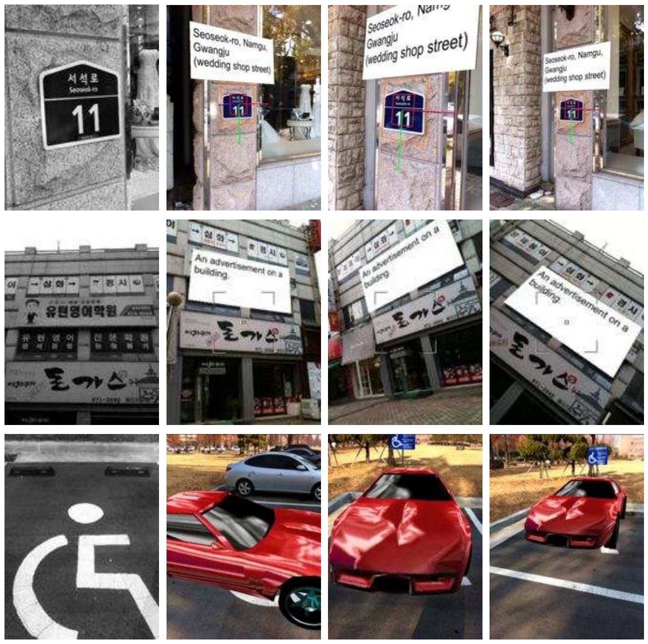

서석로
Seosok-ro
11
Seoseok-ro, Namgu, Gwangju (wedding shop street)
An advertisement on a building.

Fig. 14: Results on different types of surfaces. For all these examples, the user simply had to shoot the image on the left and select the computer model he or she wanted to add. Note that our method works well with low-textured objects.

results. The gyroscope sensor, which has been adopted by recent mobile phones, can provides the device's orientation relative to a reference. If the reference is set to gravity, then the gyroscope sensor could replace the accelerometer in our approach and the assumption of pitch rotation can be removed if a target is horizontal. If the target is vertical, however, we still need to estimate vanishing points for warping to the fronto-parallel view.

2) Limitations: Figure 15 shows some cases of failure in fronto-parallel view generation and target recognition. The computed frontal view image becomes unreliable if the target surface is slanted too much, because the target is imaged as a small number of pixels and hence the warped image becomes too blurred. With a vertical target, fronto-parallel view generation fails when there are no horizontal or vertical lines in a scene. However, real-world scenes usually contain horizontal and vertical lines, and thus this is not a significant limitation. Recognition failures typically happen on surfaces with repetitive textures that make it difficult to identify a specific target or on glossy objects whose textures changes depending on the viewpoints.

VII. Conclusion

We proposed an approach to Augmented Reality on mobile phones that is very intuitive to use by combining recent Computer Vision techniques and the use of the phone sensors. It can be adapted to the possibilities of mobile phones as it lets users add augmentations even in outdoor environments with very limited need for user intervention, and it requires only a limited amount of computational power.

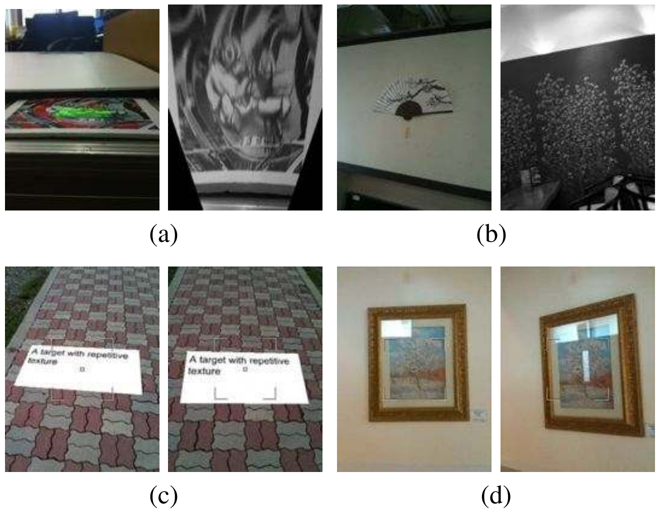


Fig. 15: Cases of failure. (a) The generated fronto-parallel view image becomes unreliable when a surface is slanted too much; (b) Vanishing point estimation fails when there are no horizontal or vertical lines; (c) and (d) Recognition failures typically happen on surfaces with repetitive textures such as brick patterns or on glossy objects.

Appendix
Rotation Matrix from Two Vanishing Points

This appendix describes how to compute the rotation matrix between two views given the vanishing points of two sets of orthogonal lines in the two views. Back-projecting the vanishing points $\mathbf{v}_i$ and $\mathbf{v}'_i$ of one set gives the 3D directions $\mathbf{d}_i$ and $\mathbf{d}'_i$ of the lines in each view's local coordinate frame:

$$\mathbf{d}_i = \frac{\mathbf{K}^{-1}\mathbf{v}_i}{\|\mathbf{K}^{-1}\mathbf{v}_i\|} \qquad \mathbf{d}'_i = \frac{\mathbf{K}^{-1}\mathbf{v}'_i}{\|\mathbf{K}^{-1}\mathbf{v}'_i\|} \tag{12}$$

Since the vanishing points are affected by rotation only, $\mathbf{d}_i$ and $\mathbf{d}'_i$ are related by $\mathbf{R}\mathbf{d}'_i = \mathbf{d}_i$, and we have:

$$\mathbf{R}\begin{bmatrix} | & | & | \\ \mathbf{d}'_1 & \mathbf{d}'_2 & \mathbf{d}'_3 \\ | & | & | \end{bmatrix} = \begin{bmatrix} | & | & | \\ \mathbf{d}_1 & \mathbf{d}_2 & \mathbf{d}_3 \\ | & | & | \end{bmatrix}. \tag{13}$$

In our problem, the first view is the virtual frontal view and $\mathbf{d}_1$, $\mathbf{d}_2$, and $\mathbf{d}_3$ become $(1,0,0)^\top$, $(0,1,0)^\top$, and $(0,0,1)^\top$, respectively. The matrix on the right hand side of Eq. (13) is therefore the Identity matrix. For the second view, only two directions, $\mathbf{d}'_1$ and $\mathbf{d}'_2$ are available from two vanishing points. The third direction can be obtained as the vector product of these two directions: $\mathbf{d}'_3 = \mathbf{d}'_1 \times \mathbf{d}'_2$. $\mathbf{R}$ can then be computed as:

$$\mathbf{R} = \begin{bmatrix} | & | & | \\ \mathbf{d}'_1 & \mathbf{d}'_2 & \mathbf{d}'_3 \\ | & | & | \end{bmatrix}^\top. \tag{14}$$

Acknowledgment

This research is supported in part by the Global Frontier R&D Program on <Human-centered Interaction for Coexistence> funded by the National Research Foundation of Korea grant funded by the Korean Government(MEST)

(NRF-M1AXA003-20100029751), and in part by Ministry of culture, Sports and Tourism (MCST) and Korea Creative Content Agency (KOCCA), under the Culture Technology(CT) Research & Development Program 2009.

References

[1] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Pose Tracking from Natural Features on Mobile Phones," in *Proceedings of the International Symposium on Mixed and Augmented Reality*, September 2008.

[2] G. Klein and D. Murray, "Parallel Tracking and Mapping on a Camera Phone," in *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2009, pp. 83–86.

[3] D.-N. Ta, W.-C. Chen, N. Gelfand, and K. Pulli, "Surftrac: Efficient Tracking and Continuous Object Recognition Using Local Feature Descriptors," in *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2009, pp. 2937–2944.

[4] S. Taylor and T. Drummond, "Multiple Target Localisation At Over 100 Fps," in *Proceedings of the British Machine Vision Conference*, September 2009, pp. 7–10.

[5] D. Wagner, D. Schmalstieg, and H. Bischof, "Multiple Target Detection and Tracking With Guaranteed Framerates on Mobile Phones," in *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2009, pp. 57–64.

[6] S. Hinterstoisser, V. Lepetit, S. Benhimane, P. Fua, and N. Navab, "Learning real-time perspective patch rectification," *International Journal of Computer Vision*, vol. 91, pp. 107–130, January 2011.

[7] W. Lee, Y. Park, W. Woo, and V. Lepetit, "Point-and-Shoot for Ubiquitous Tagging on Mobile Phones," in *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2010.

[8] G. Klein and D. Murray, "Improving the Agility of Keyframe-Based SLAM," in *Proceedings of the European Conference on Computer Vision*, October 2008, pp. 802–815.

[9] P. Bunnun and W. Mayol-Cuevas, "OutlinAR: An Assisted Interactive Model Building System with Reduced Computational Effort," in *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2008.

[10] J. Neubert, J. Pretlove, and T. Drummond, "Semi-Autonomous Generation of Appearance-based Edge Models from Image Sequences," in *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2007.

[11] Q. Pan, G. Reitmayr, and T. Drummond, "ProFORMA: Probabilistic Feature-based On-line Rapid Model Acquisition," in *Proceedings of the British Machine Vision Conference*, 2009.

[12] G. Simon, "Immersive Image-Based Modeling of Polyhedral Scenes," in *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2009.

[13] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[14] J.-P. Tardif, "Non-Iterative Approach for Fast and Accurate Vanishing Point Detection," in *Proceedings of the International Conference on Computer Vision*, Sep. 2009, pp. 1250–1257.

[15] H. Kong, J.-Y. Audibert, and J. Ponce, "Vanishing Point Detection for Road Detection," in *Proceedings of the Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 96 –103.

[16] R. G. von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall, "LSD: A Fast Line Segment Detector with a False Detection Control," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 4, pp. 722–732, April 2010.

[17] C. Rother, "A new approach to Vanishing Point Detection in Architectural Environments," *Image and Vision Computing*, no. 9-10, pp. 647–655, 2002.

[18] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[19] R. Toldo and A. Fusiello, "Robust Multiple Structures Estimation with J-Linkage," in *Proceedings of the European Conference on Computer Vision*, 2008, pp. 537–547.

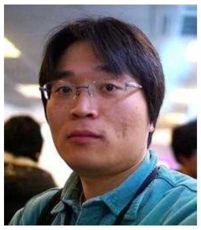
[20] A. Berg and J. Malik, "Geometric Blur for Template Matching," in *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2002.

[21] Y. Park, V. Lepetit, and W. Woo, "ESM-Blur: Handling & Rendering Blur in 3D Tracking and Augmentation," in *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2009, pp. 163–166.

[22] S. Lieberknecht, S. Benhimane, P. Meier, and N. Navab, "A Dataset and Evaluation Methodology for Template-Based Tracking Algorithms," in *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2009.

[23] S. Benhimane and E. Malis, "Homography-Based 2d Visual Tracking and Servoing," *International Journal of Robotics Research*, vol. 26, no. 7, pp. 661–676, 2007.

[24] F. Jurie and M. Dhome, "Hyperplane Approximation for Template Matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 996 –1000, Jul. 2002.



Wonwoo Lee received his B.S. in Dept. of Mechanical Engineering from Hanyang University, Seoul, S. Korea in 2003 and M.S. in the Dept. of Information and communication from Gwangju Institute of Science and Technology (GIST), Gwangju, S. Korea, in 2004. Currently, he is a Ph.D. student in DIC, GIST. His research interests include multi-view segmentation, real-time object detection, augmented reality, and GPU computing.

Youngmin Park received the BS degree in computer engineering from the Kangwon National University (KNU), Gangwon-do, Korea, in 2004, and the MS degree from the Department of Information and Communications (DIC), Gwangju Institute of Science and Technology (GIST), Gwangju, Korea, in 2006, where he is currently working toward the PhD degree. His research interests include Augmented Reality, 3D vision-based tracking, object detection, and mobile computing.

Vincent Lepetit is a Senior Researcher at the Computer Vision Laboratory, EPFL. He received the engineering and master degrees in Computer Science from the ESIAL in 1996. He received the PhD degree in Computer Vision in 2001 from the University of Nancy, France, after working in the ISA INRIA team. He then joined the Virtual Reality Lab at EPFL (Swiss Federal Institute of Technology) as a post-doctoral fellow and became a founding member of the Computer Vision Laboratory. His research interests include vision-based Augmented Reality, 3D camera tracking, object recognition and 3D reconstruction.

Woontack Woo received his B.S. in Electronics Engineering from Kyungpook National University in 1989 and his M.S. in Electronics and Electrical Engineering from POSTECH in 1991. In 1998, he received his Ph.D. in Electrical Engineering Systems from University of Southern California (USC). In 1999, as an invited researcher, he joined Advanced Telecommunications Research (ATR), Kyoto, Japan. Since Feb. 2001, he has been with the Gwangju Institute of Science and Technology (GIST), where he is a Professor in the Department of Information and Communications and Director of Culture Technology Institute. His research interests include 3D computer vision and its applications including attentive AR and mediated reality, HCI, affective sensing and context-aware for ubiquitous computing, etc.