# Point-and-Shoot for Ubiquitous Tagging on Mobile Phones*

Wonwoo Lee† GIST U-VR Lab
Youngmin Park‡ GIST U-VR Lab
Vincent Lepetit§ CVLab, EPFL
Woontack Woo¶ GIST U-VR Lab

## Abstract

We propose a novel way to augment a real scene with minimalist 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 vertical or horizontal surfaces only, but this is not a restriction in practice in man-made environments, and avoids to go through any reconstruction of the 3D scene, which is still a delicate process. Our approach is inspired by recent work on perspective patch recognition [5] and we show how to modify it for better performances on mobile phones and how to exploit the phone accelerometers to relax the need for fronto-parallel views. In addition, our implementation allows to share the augmentations and the required data over peer-to-peer communication to build a shared AR space on mobile phones.

## 1 Introduction

Mobile phones have become a very popular platform for Augmented Reality (AR) applications. They are wide-spread, often equipped with hardware useful for localization and good computational capacities. Several recent great works showed that it is possible to develop Computer Vision techniques for AR on mobile phones [9, 16, 18].

In this paper, 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 going through a 3D reconstruction phase, which is still delicate to perform correctly, and using feature points, as they are not available in every scene. To do that, we combine a recent technique to recognize patches and estimate their spatial orientations [5] adapted for the mobile phone and the use of the built-in accelerometers.

In our approach, the user simply has to point the phone camera toward the location he wants to augment. Using the accelerometers, we can correct for the surface orientation, and insert the virtual objects coherently. This is possible only for horizontal or vertical surfaces only, however in man-made environments, that is not a restriction in practice. Our algorithm can guess most of the time the real surface orientation—horizontal or vertical—otherwise the user can correct it very easily. The scale factor can be defined by moving the phone toward or away from the surface, before fine-tuning. This mode of operation results in a very intuitive interaction.

Once the user defined the virtual content, its location in the real world can be recognized even under new viewpoints and tracked in 3D for consistent rendering. For detection, we adapted Gepard, a template-based approach proposed in [5] 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 viewpoint when the camera pose is slightly changed, with the texture around feature points.

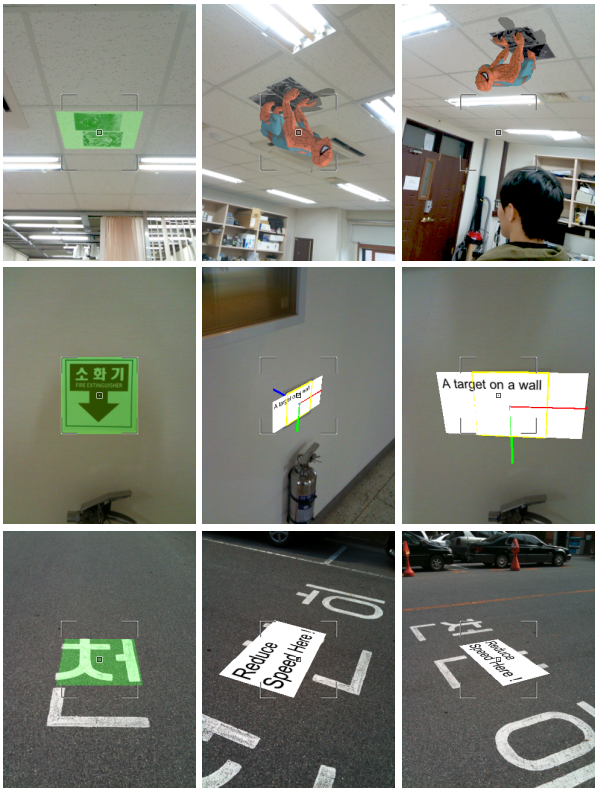



Figure 1: Overview of our approach. The user simply has to point to the desired location for the augmentation. Our method guesses the surface orientation for a coherent insertion. It can then recognize the location even under new viewpoints, and track it for real-time augmentation.

Here, we skip this the feature point detection step and use larger patches instead for better robustness. We also replaced the way the templates are computed, which consumes large amount of memory in Gepard, by rendering and blurring operations. This is more suitable to the phone architecture, and requires only a few seconds. Finally, while Gepard requires a fronto-parallel view to build the set of templates, we can exploit the phone accelerometers to rectify the captured image into a fronto-parallel view.

Moreover, in our implementation, the data associated with the augmentation and required for detection, tracking and rendering, can be shared with nearby mobile phones through peer-to-peer communication. Then, other users can share augmentations on their mobile phones even from different viewpoints. This way, the users can easily collaborate in the same AR space through their mobile phones.

In the remainder of the paper, we first review related work in

*This research is supported by Ministry of culture, Sports and Tourism (MCST) and Korea Creative Content Agency (KOCCA), under the Culture Technology(CT) Research & Development Program 2010.

†e-mail: wlee@gist.ac.kr
‡e-mail: ypark@gist.ac.kr
§e-mail: vincent.lepetit@epfl.ch
¶e-mail: wwoo@gist.ac.kr

Section 2. Section 3 details how the set of templates are built and used for detection and tracking. Experimental results are given in Section 4. We provide conclusions in Section 5.

## 2 Related Work

Several recent works showed that it is possible to run Computer Vision algorithms for localization and 3D tracking on mobile phones [9, 15, 16, 17, 18]. 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 proved to be adapted to poorly textured objects and blurry images [5]. 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 viewpoint. PTAM [9] relies on a related method as it compares downscaled, blurred images for camera relocalization [8], but cannot be generalized 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 skip the feature points detection and use comparatively much larger patches, for better robustness. Gepard also computes the mean patches as a linear combination of eigenpatches, unfortunately this requires a lot 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.

Another restriction of Gepard is to require a fronto-parallel view of the original patch to detect, or equivalently, knowledge on the 3D orientation of the patch. This could be avoided with an automated 3D reconstruction of the scene for example, but that is still difficult to perform by a non-expert user, and would require camera motion before augmenting the scene anyway. Other works have developed interactive 3D reconstruction using AR [3, 11, 12, 14], but still require some time and expertise.

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

## 3 Patch Learning and Tracking on Mobile Phones

### 3.1 Gepard [5]

Given a reference image patch $\mathbf{p}$ in a frontal view, Gepard computes a set of "mean patches". The original expression of a mean patch $\overline{\mathbf{p}_h}$ is:

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

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

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

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

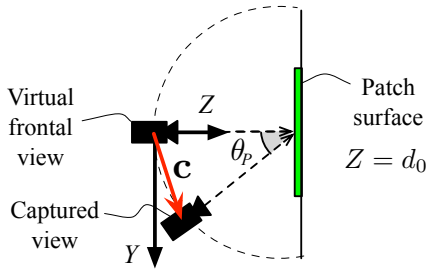



Figure 2: Defining the camera pose in the case of a vertical surface. Without loss of generality the pose of the frontal view is defined as $[\mathbf{I}|\mathbf{0}]$. Then in the case of a vertical surface, the coordinates of the camera center $\mathbf{c}$ are $[0, d_0 \sin\theta_p, d_0(1-\cos\theta_p)]^\top$.

In practice, Eq. (1) is not used directly as this would be very costly. Instead, Gepard uses an approximation that can be summarized as:

$$\mathbf{a} = \mathbf{P}_{\text{PCA}}\mathbf{p}\,, \text{ and} \tag{3}$$

$$\forall h\ \overline{\mathbf{p}_h} = \mathbf{V}_h\mathbf{a}\,, \tag{4}$$

where $\mathbf{P}_{\text{PCA}}$ and the $\mathbf{V}_h$ are matrices that can be precomputed. $\mathbf{P}_{\text{PCA}}$ first projects the patch into an eigenspace to obtain a relatively short vector and speed up the product with the $\mathbf{V}_h$ matrices, which columns are the corresponding eigenvectors after the transformation described by Eq. (1). These matrices are very large, and cannot easily be stored in the memory of a mobile phone. We give below another approximation that does not use these memory consuming precomputations.

### 3.2 Guessing the Camera Pose

As shown in the first column of Figure 1, we can relax the need for a fronto-parallel view by estimating the orientation of the surface and its distance to the camera.

Since we are looking for an orientation in the 3D space, we have three degrees-of-freedom to estimate. We first assume that the patch lies on a surface that is horizontal or vertical. This is very often the case in man-made environments, where floors, ceilings, walls, tables, ... compose most of the geometry. Using the phone accelerometers, we can estimate the angle the phone makes with the gravity force, which provides another degree of freedom. Finally, to predict the surface orientation, we assume the phone can only rotates in pitch, setting the remaining angle to estimate to 0. This angle can be modified by the user later on.

We use a heuristics to determine if the surface is either horizontal or vertical. Let denote $\theta_p$ the angle the phone makes with a horizontal plane as provided by the accelerometers, so that $\theta_p = 0$ when the camera points toward the horizon. Then, the following predictions are often true in practice:

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

In case our guess is wrong, the user can correct it by directly choosing a surface model, either horizontal or vertical.

This gives us the camera orientation with respect to the surface. To define a full pose, we still need its translation. Unfortunately, it is not possible to get the distance between the camera and the surface from a single image. Instead we use an arbitrary value $d_0$. Thus, the user can define the scale of the augmentation by simply moving the phone toward or away from the surface: As shown in

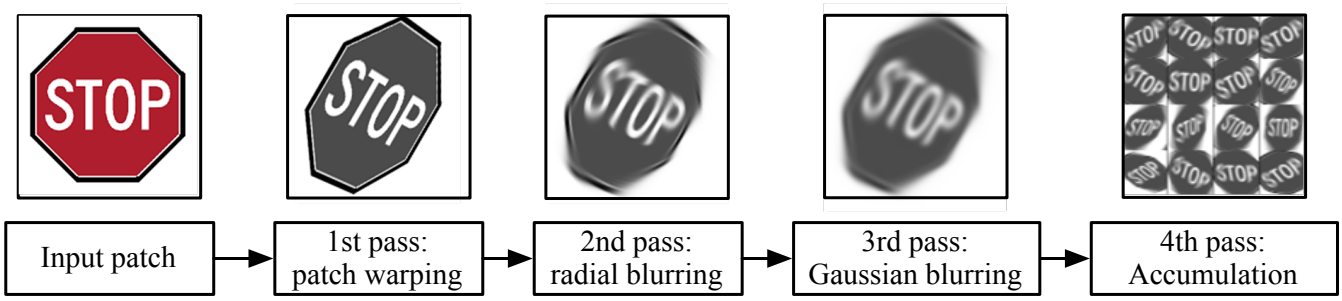



Figure 3: Creation of the mean patches by blurring. To compute each mean patch, the reference patch is first warped, and radial blur and Gaussian blur are then applied. The resulting patches are accumulated into a texture on the GPU to send them to the CPU in one single read.

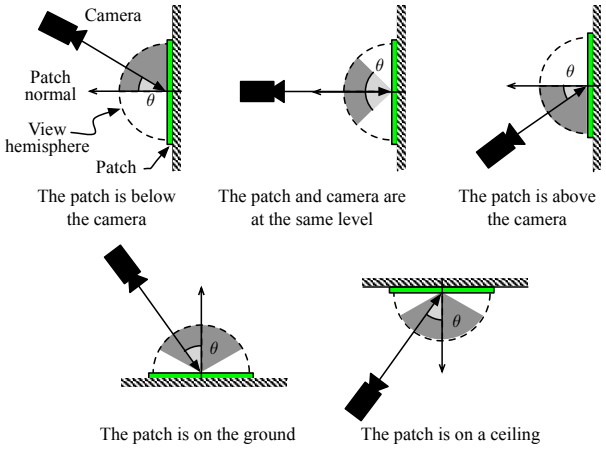



Figure 4: Sampling the likely camera poses only. The regions marked in dark grey represent the range of poses sampled by the $P_h$s. These regions vary depending on the relative positions of the phone and patch as determined by the accelerometers.

the third row of Figure 11, the augmentation will correspond to a large object if the camera is far away from the surface; conversely, as shown in the fourth row, the augmentation will correspond to a small object if the camera is close to the surface. This is very intuitive but limited to some range of scale within which the user can move the phone, and the interface lets the user adjust the scale if needed.

As illustrated in Figure 2, we can finally describe how we generate a virtual fronto-parallel view of the target from the input image. Without loss of generality we 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 $\mathbf{t} = -\mathbf{Rc}$. From [4], the expression of the homography $\mathbf{H}_{f\leftarrow c}$ that warps the captured image to the virtual frontal view is then:

$$\mathbf{H}_{f\leftarrow c} = \mathbf{K}\left(\mathbf{R} - \frac{\mathbf{t}\mathbf{n}^\top}{d_0}\right)^{-1}\mathbf{K}^{-1}, \qquad (5)$$

where $\mathbf{K}$ is the camera calibration matrix and $\mathbf{n}$ the vector $[0,0,-1]^\top$.

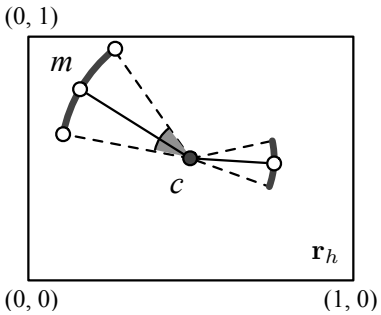



Figure 5: Applying radial blur to a warped image. The pixels intensities are replaced by the average intensity along an arc of a circle. The length $l$ of the arc varies linearly with the distance to the center of the patch.

### 3.3 Generating the Mean Patches with Blurring

As can be seen in [5], the mean patches look like the reference patch after some non-uniform blur. This is related to Geometric Blur [2], 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 more efficient. As shown in Figure 3, we use a combination of radial blur and Gaussian blur, performed on the mobile phone's GPU, to compute the mean patches.

**Warping** In order to approximate a mean patch $\overline{\mathbf{p}_h}$, the reference image—corresponding to a frontal view—is first rendered into a new patch we denote by $\mathbf{p}_h$ to correspond to pose $P_h$. Rendering a $320 \times 426$ image on the phone's GPU takes only about 0.3 *ms*, whereas CPU-based patch warping took about 100 *ms* in our experiments.

Using many poses $P_h$ would slow down the computation times during learning and detection, and would be a waste of memory usage. On the other hand, too sparse poses would result in detection failures. Therefore, we sample a limited range of the poses, avoiding unlikely poses where the surface would be too slanted. Figure 4 summarizes the different configurations.

To generate the poses, we regularly sample every 20 degrees the rotations around the three axes over the range given by Figure 4. We also use 3 scale factors (0.5, 1, 2) we apply to $d_0$ before computing the translation as in Section 3.2 in order to detect the surface from a range of distances. We then directly use OpenGL to render $\mathbf{p}_h$.

**Radial blur** In the second pass, radial blur is applied to $\mathbf{p}_h$ to get a new patch $\mathbf{r}_h$. As depicted by Figure 5, 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

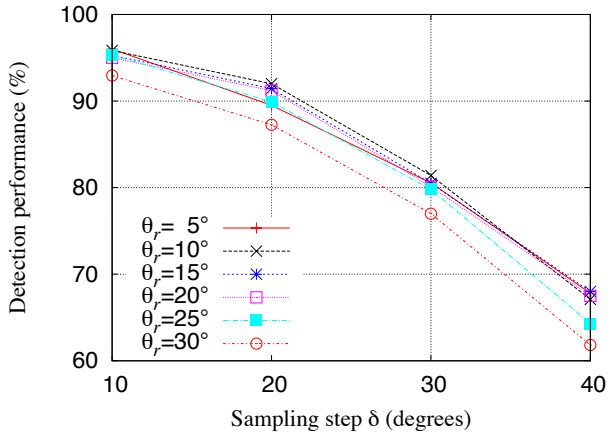



Figure 6: Detection performance with varying pose sampling steps and maximum blur ranges.

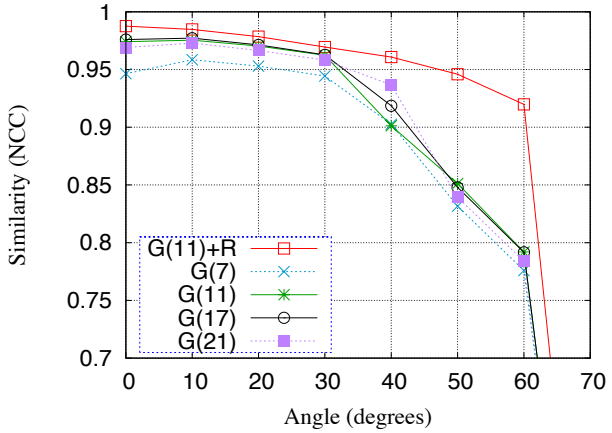



Figure 7: Effectiveness of radial blur. Combining the radial blur and the Gaussian blur outperforms simple Gaussian blurring.

$m$. The length $l$ of the arc varies linearly with the distance between $c$ and $m$:

$$l = \theta_r \|c - m\| . \quad (6)$$

In this formula, $\theta_r$ is a parameter expressed in radians and we use the value $\theta_r = 0.17$ in practice, about 10 degrees. The pose sampling step and the maximum radial blur range are experimentally determined to guarantee reasonable detection performance. According to results shown in Figure 6, our algorithm achieves good performance with the selected parameters.

To confirm the effectiveness of the radial blur, we measured the patch detection performance against viewpoint changes on the Graffiti image set [1] 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 *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 Gaussian blur outperform those generated with Gaussian blur only.

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. (1). 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.

Downsampling and Accumulation As in Gepard, $\overline{\mathbf{p}_h}$ is downsampled from $128 \times 128$ to $32 \times 32$ 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.

### 3.4 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.

To detect the patch if a known patch is visible in the captured image and estimate its orientation, we first extract the patch in the image center. By contrast with Gepard, we apply to this patch the transformation detailed in Section 3.3, 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. (2): 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 [13], as it is robust to motion blur, which is frequent with mobile phones' low-quality cameras. ESM-Blur is used to first refine the pose provided by the detection, and then track the patch in the subsequent captured images. When tracking fails, we re-run the detection procedure.

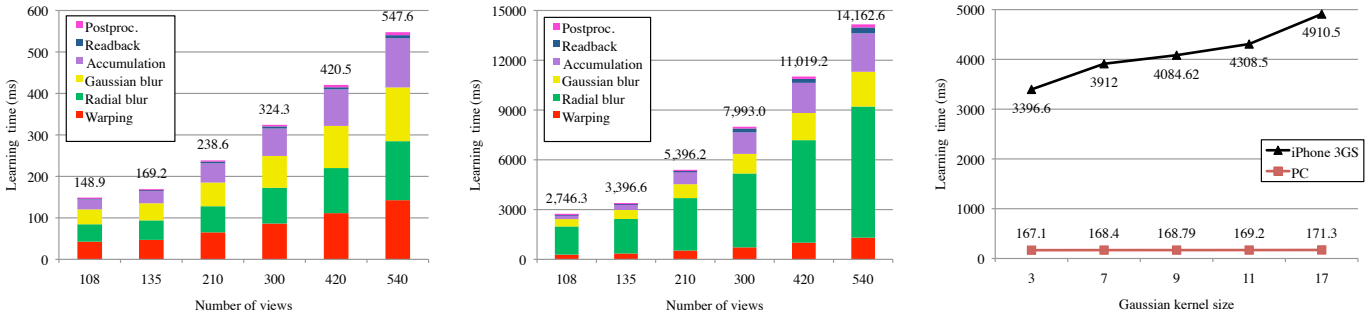
An obstacle to template matching-based tracking methods on mobile phones is the computations of large matrices, which takes most of the time required for tracking. Since it is relatively slow on mobile phones, we use NEON, a set of SIMD (Single Instruction Multiple Data) instructions of ARM CPUs, for faster computations. The matrix computation code written in NEON runs about 3-5 times faster than standard C code.

## 4 Experimental Results

We implemented our method on both a mobile phone and a PC. We used the Apple iPhone 3GS, which has a 600 MHz CPU and a PowerVR SGX GPU, and a PC with a 2.4 GHz CPU and a GeForce 8800GTX GPU. Cameras capture videos in $640 \times 480$ on the PC and $480 \times 360$ on the iPhone 3GS. Note that the speed of our approach is independent on the size of the input images. In both platforms, cameras are calibrated in advance. We assume 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.

We set the size of an input patch to $128 \times 128$ 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 $32 \times 32$ and it requires 4 kilobytes for pixel intensities and 36 bytes for a $3 \times 3$ pose matrix. For overall data with 225 views, about 900 kilobytes are required to store the data learned from a target.

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




(a) (b) (c)

Figure 8: Learning times. The overall time increases on both (a) the PC and (b) the mobile phone as the number of views increases. (c) On the PC, the size of the Gaussian kernel does not have much effect, while larger kernels result in slower computations on the mobile phone.

### 4.1 Learning Speed

Figure 8 compares the time required for learning on both PC and mobile phone platforms. While the four main steps—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, while radial blur needs more complex accesses and the phone's GPU is not adapted yet to this type of access. Another difference is that increasing the size of the Gaussian blur kernel hardly affects the computation time on the PC, while it increases proportionally on the mobile phone.

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

### 4.2 Patch Detection Performance

Figure 9 shows the patch detection results against viewpoint changes and image noise. We used 10 planar targets with different textures in this experiment[2]. Apart from the viewpoint changes, we add zero-mean Gaussian noises with a standard deviation $\sigma$ 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 is robust to viewpoint changes up to 60 degrees and noises up to $\sigma = 30$. The last 3 targets (Grass, MacMini, and Board) have rich but repetitive textures, with thin structures. This case is the worse for our approach and the *NCC* score drops more quickly.

The comparison between our algorithm and Gepard [5] is shown in Figure 10. In all data sets, Gepard outperforms our method but the performance loss is not large with the targets having rich textures. Gepard also reveals weakness to repeated textures although it is still better than ours. We expect that it is because Gepard exploits the two-step pose optimization based on [1, 7], while we use only one [13].

### 4.3 Patch Detection Speed

The speed of patch detection is shown in Table 1. Patch 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 with ESM-Blur can vary a lot 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 (10-15 fps) and accurate registration.

[2]Some image data is available at http://www.metaio.com/research. See [10] for details.

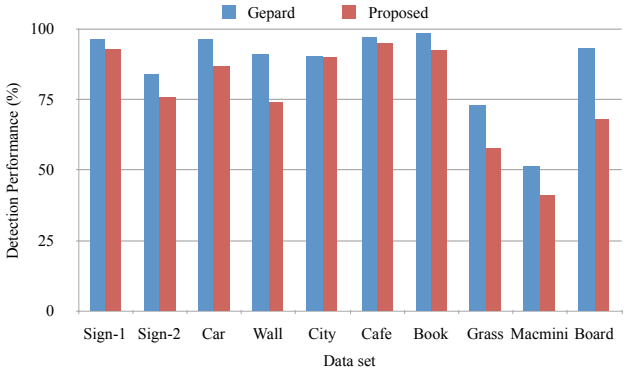



Figure 10: Performance comparison with Gepard.

Table 1: Patch detection speed on a mobile phone in milliseconds

| | Mean | StDev | Min | Max |
|---|---|---|---|---|
| Mean patches comparison | 3.06 | 0.25 | 2.76 | 3.4 |
| Pose estimation | 64.0 | 30.9 | 11.1 | 98.3 |

### 4.4 Real World Examples

Figure 11 shows the result of the patch detection with target objects from the real world. The first column shows the input images, with green guide squares that represent the regions to learn. For the first two examples, the fronto-parallel views of the objects are available. After learning from the input images, the objects are successfully detected and tracked by our algorithm on mobile phones. In contrast, the frontal views of the targets are unavailable for the last three examples. It is possible to estimate the right pose of the camera through the proposed fronto-parallel view generation method. Note that our algorithm works well even with poor textures. In the examples, we assume 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.

Figure 12 shows some failure cases. Failures typically happen on surfaces with repetitive textures such as brick patterns or on glossy objects.

### 4.5 Sharing Augmentations

The data learned from the target object and related contents can be shared with nearby users through wireless communication, which is one of common capabilities of mobile phones. When a connection between two mobile phones is established, the learned mean patches and the virtual contents are transmitted from one to another, in a compressed file over a peer-to-peer Bluetooth channel. Thus, users can build an AR space *in situ* and collaborate with virtual contents in the same AR space.

### 4.6 Discussion

As we showed in previous sections, the proposed algorithm works well with real world objects having horizontal and vertical surfaces. Although a target surface is neither horizontal nor vertical, the target can still be learned and detected by our method, but the augmentations will have wrong registrations with respect to the geometry of the target surface.

Our point-and-shoot approach requires the user to point the target with a mobile phone's camera manually. It has both negative and positive aspects in target detection. The negative aspect is no detection occurs when the target patch is out of the image center. On the other hand, the manual pointing to a target allows learning and detection of the target that does not have a feature point at its center.

## 5 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. Our approach actually fits closely the requirements of Anywhere Augmentation as defined in [6] since it supports learning and detection of a target object on the fly and lets users add augmentations even in low-textured environments with very limited need for user intervention.


Figure 12: Failure cases. Failures typically happen on surfaces with repetitive textures such as brick patterns or on glossy objects.

## References


[1] S. Benhimane and E. Malis. Homography-Based 2d Visual Tracking and Servoing. *International Journal of Robotics Research*, 26(7):661–676, 2007.

[2] A. Berg and J. Malik. Geometric blur for template matching. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2002.

[3] 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.

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

[5] S. Hinterstoisser, V. Lepetit, S. Benhimane, P. Fua, and N. Navab. Learning Real-Time Perspective Patch Rectification. *International Journal of Computer Vision*, 2010. In Press.

[6] T. Höllerer, J. Wither, and S. DiVerdi. "Anywhere Augmentation": Towards Mobile Augmented Reality in Unprepared Environments. In G. Gartner, W. E. Cartwright, and M. P. Peterson, editors, *Location Based Services and TeleCartography*, pages 393–416. Springer, 2007.

[7] F. Jurie and M. Dhome. Hyperplane Approximation for Template Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 996–1000, 2002.

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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*, pages 163–166, 2009.

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

[15] 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*, pages 2937–2944, 2009.

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

[17] 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.

[18] 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*, pages 57–64, 2009.

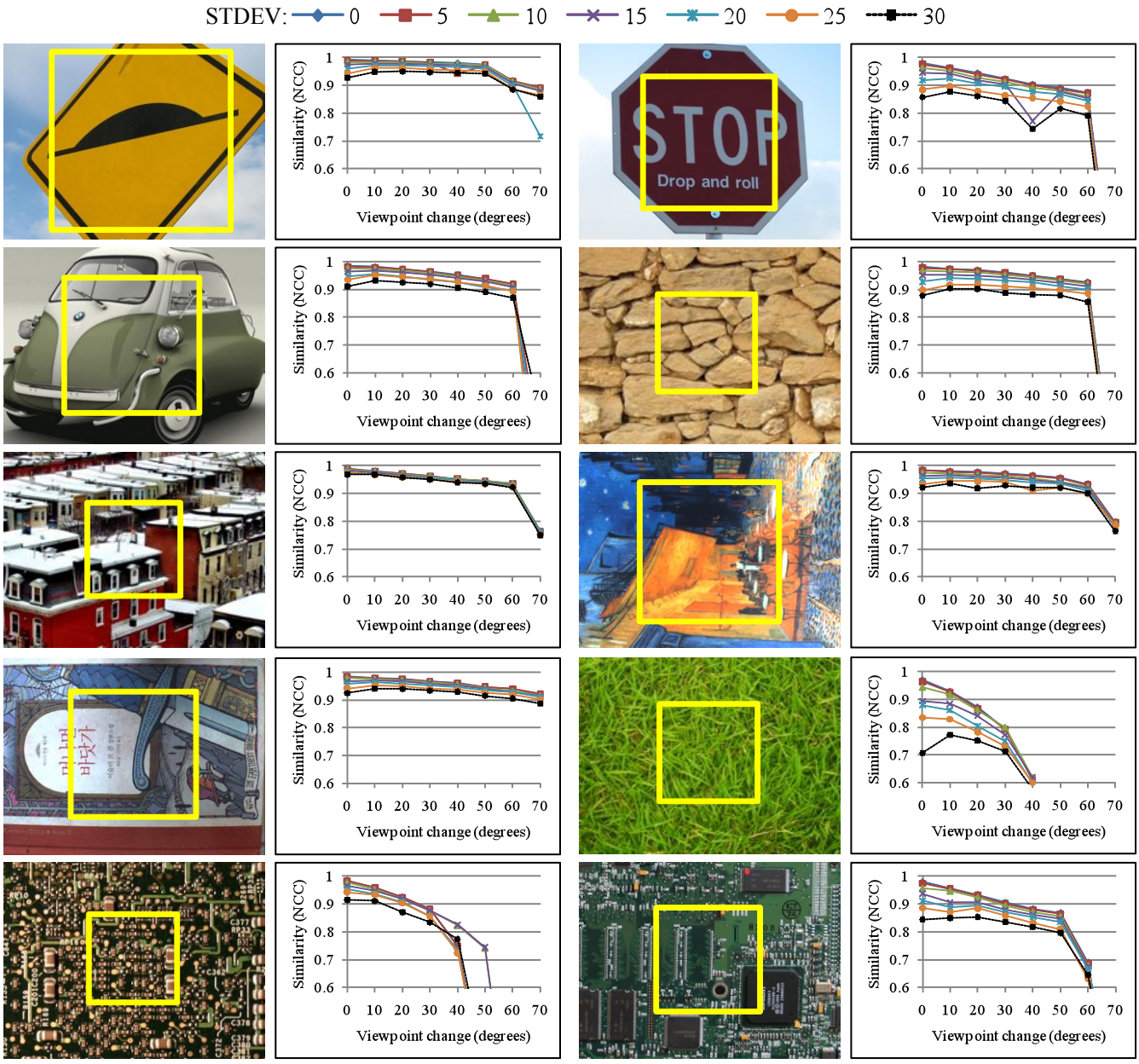



Figure 9: Evaluation of the accuracy of the retrieved pose for different types of images. From the top left: Sign-1, Sign-2, Car, Wall, City, Cafe, Book, Grass, Macmini, and Board. For each of the patches delimited by the yellow squares in the first and third columns, we generated test images by rendering the patch under different angles and adding different amounts of image noise. We then 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.

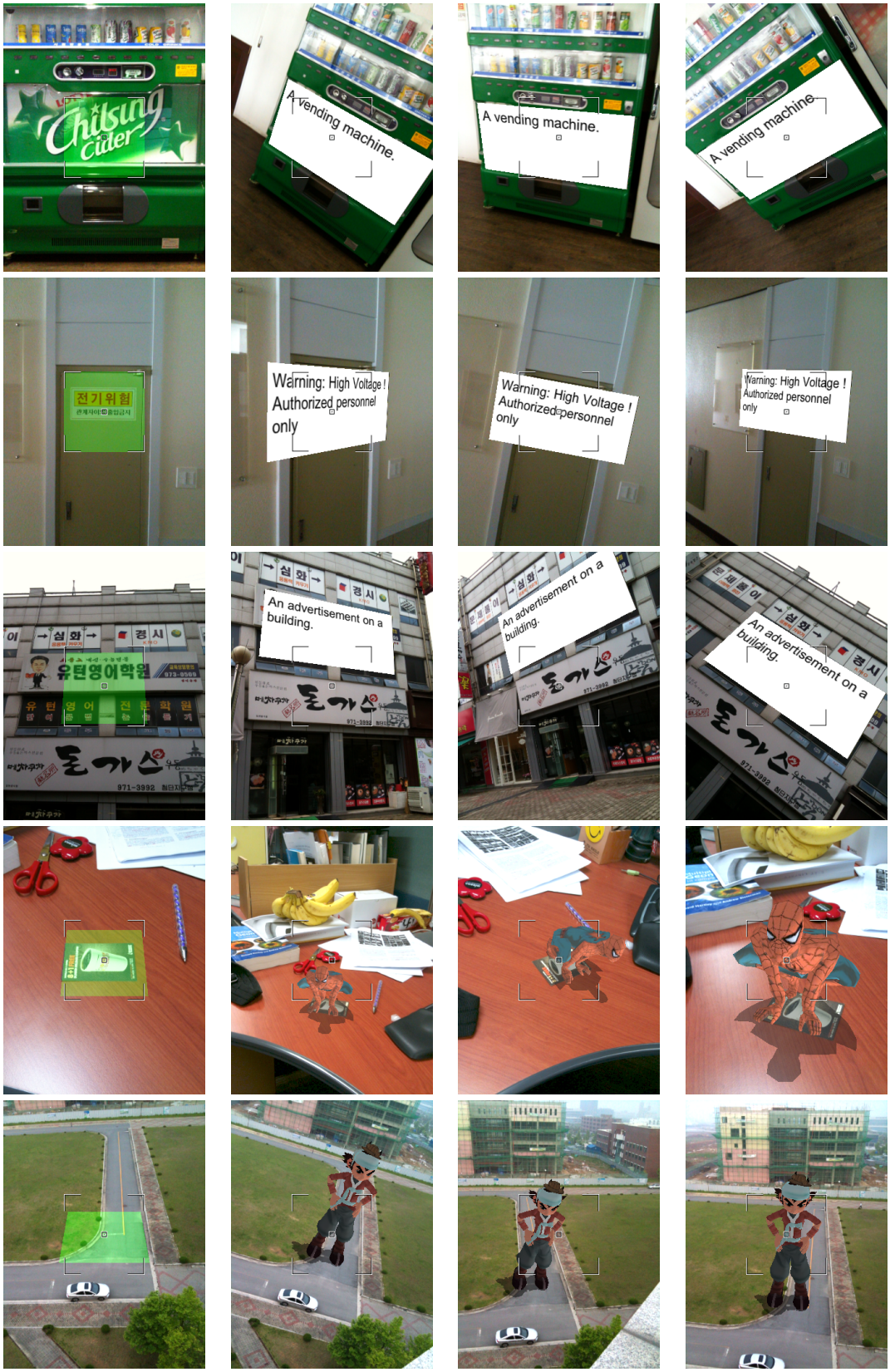



Figure 11: Results on different types of surfaces. For all these examples, the user simply had to shoot the image on the left and select the computer model he wanted to add. Note that our method works well with low-textured objects.