

Visual Code Marker Detection

Kristin L. Granlund
granlund@stanford.edu

Ernesto Staroswiecki
ernestos@stanford.edu

Joëlle K. Barral
jbarral@stanford.edu

Abstract— This report presents our results for the class project for EE368 Digital Image Processing at Stanford University in the Spring of 2006.

In this project we detect and decode visual markers by first processing the image to emphasize some salient features, then identify the marker candidates by analyzing these features, and finally projecting each candidate marker to decode it and create error measures to verify it as a marker.

Our code detects and decodes every marker provided in the training set quickly and accurately.

I. INTRODUCTION

Introduced in the 1890's, punch cards were the first tool grocers used to automate the checkout process. Modern bar codes began in 1948 when Bernard Silver, a graduate student at Drexel Institute of Technology in Philadelphia, responded to the president of a local food chain asking for a system to automatically read product information during checkout. Silver and his friend Norman Woodland filed a patent application one year later, entitled "Classifying Apparatus and Method", where they described their invention as relating "to the art of article classification (...) through the medium of identifying patterns." Indeed, a bar code is defined as a machine-readable representation of information in a visual format on a surface.

Bar codes were not commonly used in industrial applications until 1981, when the United States Department of Defense adopted the use of Code 39 for marking all products sold to the United States military. Today, barcodes are widely used to implement Auto ID Data Capture (AIDC) systems that improve the speed and accuracy of computer data entry. The drive to encode ever more information in combination with the space requirements of simple barcodes led to the development of matrix codes (a type of 2-D bar code), which do not consist of bars but rather of a grid of square cells, referred to here as markers.

With wireless technology widely available and ever-improving, bar codes are an active area of research: in Japan, for example, supermarkets customers can use camera-equipped cell phones to scan a code on a food product that links to a mobile website detailing origin, soil composition, organic fertilizer content percentage, use of pesticides and herbicides and even the name of the farm on which it was grown [1].

In this class project we were asked to find one type of such markers in an image and decode its contents. We do this by first processing the image to emphasize some salient features, then identify the marker candidates by analyzing these features, and finally projecting each candidate marker to decode it and create error measures to verify it as a marker.

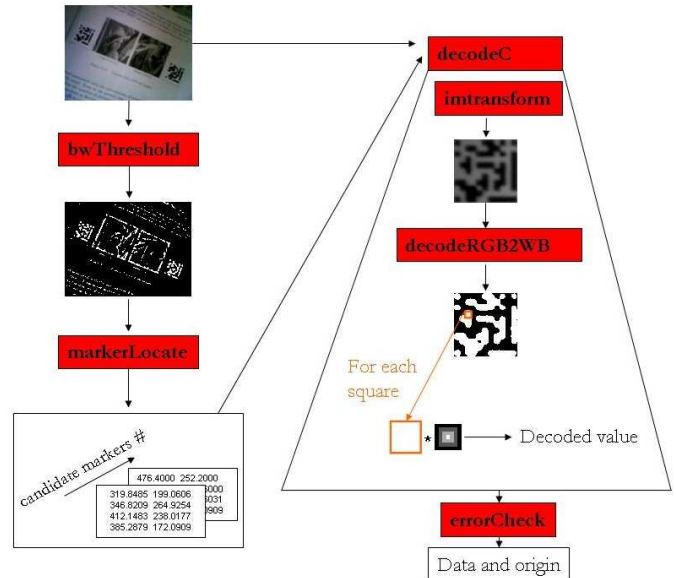


Fig. 1. General algorithm flowchart.

Our code correctly detects and decodes every marker provided in the training set.

This paper is organized as follows: section II explains our methods and algorithms used. We show our results in section III and we discuss them in section IV. We finally conclude in section V.

II. METHODS

The general algorithm used to detect and decode marker is shown in Fig. 1. The RGB image is converted into a "white and black" binary image (where the dark spots of the RGB image are now white, and the light spots are now black). On this converted image, we detect the location of the candidate markers. We then extract the data from the markers and generate error measures to reject false positives or choose among repeated markers.

A. Black and White segmentation

1) *Grayscale*: The input image is first converted to gray scale to simplify the process of locating the marker since the regions of interest are only black or white. The gray levels were calculated using the average of the red and green components of the image. This takes advantage of the JPEG compression scheme, which mimics the human visual system

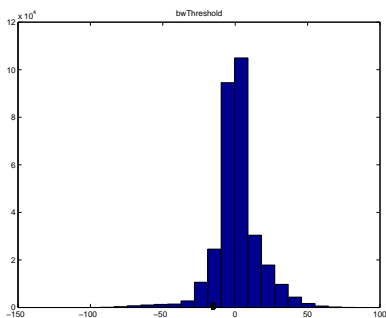


Fig. 2. Loose thresholding.

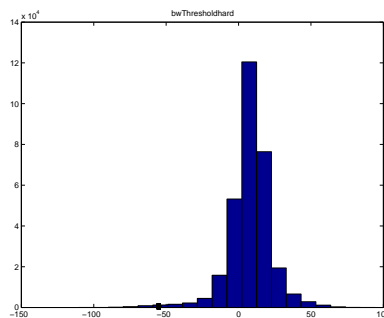


Fig. 3. Restrictive thresholding.

in that it is relatively insensitive to the information contained in the blue channel.

2) *Brightness equalization*: The gray scale image is converted to a black and white image using “adaptive” thresholding [2]; the pixel values are compared to the local mean to determine whether they are dark or light. The area over which the mean is calculated must be chosen carefully because if the area is too small, only the edges of objects will be detected and if the area is too large, there will be significant loss of detail.

We choose a kernel size depending on the size of the image. We assume that the ratio $\text{size}(\text{marker})/\text{size}(\text{image})$ do not vary drastically from one image to another. This is a strong assumption though, but it makes sense for this very application-dependent algorithm.

3) *Thresholding*: The thresholds chosen for the different black and white segmentation functions are designed to give a more or less selective segmentation. The three segmentation functions are:

- `bwThreshold` (loose) The threshold is equal to the standard deviation of the difference between the gray image and the brightness equalized image (Fig. 2).

- `bwThresholdhard` (restrictive) The threshold is equal to $-\text{abs}(\text{min})/2$ (Fig. 3).

- `decodeRGB2WB` (specific to marker-only regions) Utilizing the fact that the marker regions only contain black and white regions, the average is taken as the threshold. This assumes that the number of black and white squares in the image is roughly equal and is sensitive to the widths of the distribution but is not sensitive to outliers. Alternatively, $(\text{max}-\text{min})/2$ can also be chosen as the threshold, but is more sensitive to outliers. For our data set, we choose the less restrictive one, which is the average (Fig. 4).

4) *Marker Selection (Masked version only)*: The function `markerSelect` uses morphological techniques to identify regions which are likely to contain markers. First, the mask identifying dark regions is dilated. Regions are then labeled and discriminated against area.

This function greatly reduces the computation required by the subsequent function by significantly reducing the total area in which the corners are looked for. The reduction is greater as the number of markers in the image is smaller. Fig. 5 shows how this would be integrated with the rest of the algorithm.

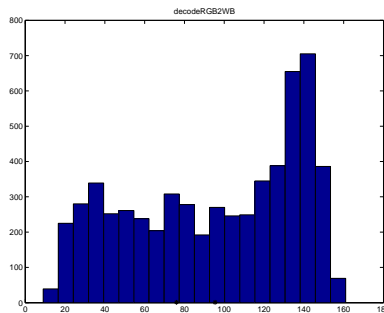


Fig. 4. Thresholding for markers only. (Left point is max-min, right point is average.)

B. Corners detection

The `markerLocate` function takes the binary image from the thresholding function as an input and labels it to identify connected regions. The properties of the regions are calculated by approximating each region with an ellipse. The centroid, angle of orientation, and the major and minor axis lengths of the ellipses are calculated and used to determine the location of the markers.

1) *'L' shape detection*: The ratio of the major and minor axis lengths is used to discriminate regions which are candidates for the guide bars of the marker. The candidates for markers are selected by locating the guide bars (the bars in the lower left corner of the marker). Long and short bars that are in the correct relative positions indicate potential markers.

2) *Corner locators detection*: Now, the corner elements are searched for along the lines of the guide bars' major axes and at distances calculated from the sizes of the bars. Then, potential markers are eliminated if the corner elements are not found. Once the fixed elements are located, the coordinates of the corners can be calculated and they are returned by the `markerLocate` function.

3) *Upper left corner locator detection*: With the guide bars and two corner elements identified we can now generate a projection and calculate the estimated position of the last corner element at the upper left corner of the marker. We select elements as the round object closest to the projected position of the corner.

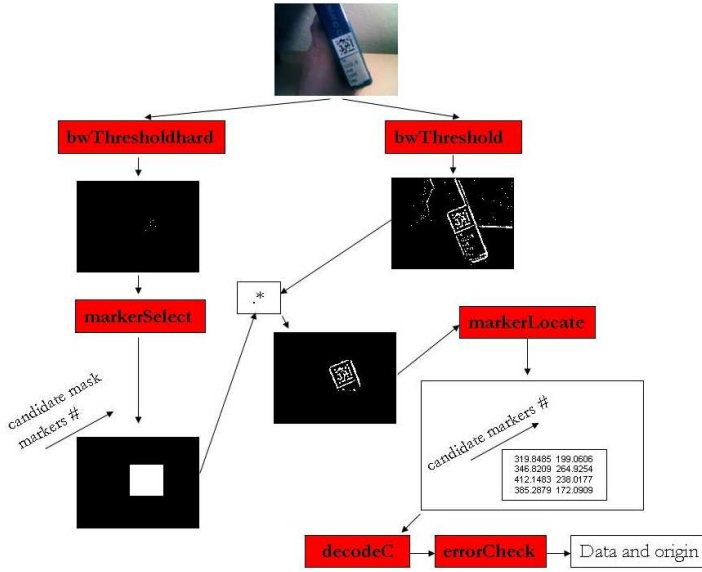


Fig. 5. Algorithm with masks flowchart.

Cropped Image with Single Marker



Fig. 6. Cropped marker.

C. Data extraction

With the information of the candidate markers locations we now re-process the original image to find the information encoded in the marker.

1) *Markers transformation*: The decode function receives the coordinates of the corners of each candidate marker. These coordinates are used to find the transform that best describes the projection of the marker relative to the plane of the camera sensors. The portion of the image designated as the marker is transformed using the inverse of the projective transform. Fig. 6 shows a cropped marker from an image. Fig. 7 shows this marker after it has been transformed to a square.

2) *Information retrieval*: The image is then converted to gray scale by exploiting the fact that the marker is predominantly black and white. The image is again converted into a binary image before extracting the pixel values. Fig. 8 shows the binary image for a marker. The values of each pixel are now calculated with a weighting function that favors the center of the pixel over the edges. A reconstructed image is shown in Fig. 9.

3) *Error checking*: The error measure is calculated by comparing the measured values and known values as shown in the equation below, where wm is the weighting matrix, p_i

Transformed Image



Fig. 7. A marker transformed to a square.

Binary Image



Fig. 8. Binary image of a marker.

is the pixel value, and kv_i is the known value of the pixel:

$$Error_measure = \sqrt{\frac{1}{38} \sum_{i=1}^{38} (wm. * p_i - kv_i)^2}$$

Similarly, the confidence measure is calculated by comparing the measured values to the estimated values, where \hat{p}_i is the estimated value:

$$Confidence_measure = 1 - \sqrt{\frac{1}{83} \sum_{i=1}^{83} (wm. * p_i - \hat{p}_i)^2}$$

III. RESULTS

A. Error measures

When looking at the error measures for our markers, we note that the maximum error for a correctly identified marker is around 0.014, while the minimum error for a false positive is around 0.1 (for a marker with one corner miss-identified, when the coordinates do not correspond to a marker at all the minimum error is on the order of 0.39). This means that non-markers can be rejected by comparing to an error threshold.

Re-built Marker



Fig. 9. Reconstructed marker (inverted).

B. Computational efficiency

We measure that our code runs fairly quickly on the training images. The total runtime on a group member's computer (Win XP Pro, Matlab 7.1, 1GB RAM, Intel Core Duo 2GHz) is about 10.5 seconds (the slowest image takes around 1.4 seconds to run).

We also calculated the run times for both the regular algorithm and the algorithm with masks on the SCIEN machines:

Image	Time(regular)(s)	Time(masked)(s)
training_1.jpg	1.65	3.35
training_2.jpg	1.15	2.57
training_3.jpg	1.07	4.87
training_4.jpg	1.20	2.19
training_5.jpg	1.35	5.77
training_6.jpg	0.92	2.62
training_7.jpg	1.04	2.31
training_8.jpg	1.18	3.05
training_9.jpg	1.50	2.67
training_10.jpg	2.24	2.37
training_11.jpg	1.23	3.68
training_12.jpg	1.44	3.22

C. Results on the training data set

The training set consists of 12 images, containing 23 markers, with 1909 bits of data. Our code correctly identifies all 1909 bits without any false positives nor repeats.

D. Results with masked version

The function `markerSelect` uses morphological techniques to identify regions which are likely to contain markers. First, the mask identifying dark regions is dilated. Regions are then labeled and discriminated against area.

This function greatly reduces the computation required by the subsequent function by significantly reducing the total area in which the corners are looked for. The reduction is greater as the number of markers in the image is smaller.

In the masked version of the algorithm, with the training data set, out of 23 markers, 3 markers require using a 'double_mask': in two of them (Fig. 10 and Fig. 11), it is indeed a false positive, and checking in a double sized box does not return new markers (as expected). In one of the three cases (Fig. 12) the detected region covered only half of the marker, hence the failure to detect its four corners; doubling the mask is an effective solution to locate the marker. Doubling the area is reasonable because in the marker selection function, we discriminate on areas that are above half the expected mean area.

IV. DISCUSSION

A. Advantages and Drawbacks

The main limitations of our code are sensitivity to lighting conditions, perspective, respective size of the image and the marker, and overall image quality.

Since the algorithm uses adaptive thresholding, it is sensitive to both lighting and size. Though the width of the averaging filter is related to the size of the input image, it is still constant

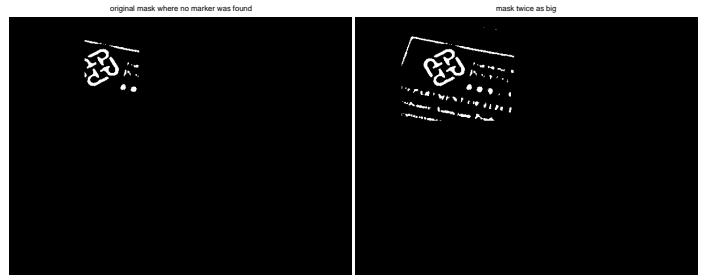


Fig. 10. False positive (masked version).

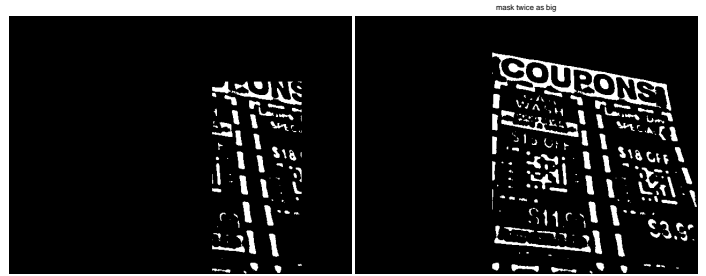


Fig. 11. False positive (masked version).

regardless of the size of the markers in the image, so it assumes that the picture was taken at a reasonable distance and under standard lighting. If the marker is too large or too small, it will be distorted or overlooked. If the lighting causes narrow bright spots or dark spots, the marker can be distorted. Worse, if the bright spots or dark spots saturate a region of the image containing a marker, the marker will be overlooked. If the picture was taken at a severe angle (i.e. the camera was not held roughly normal to the surface on which the marker lies), the algorithm may fail because the relative dimensions of the regions in the marker may be so skewed that the fixed elements are falsely categorized. For example, the shorter guide bar may appear longer than the longer one due to the perspective of the image. Then the algorithm will incorrectly identify the orientation angle of the marker. It should be noted that this problem is only an issue if the marker is significantly skewed. Our algorithm can identify markers for which the shorter guide bar appears longer than the longer guide bar; the issue only occurs when the longer guide bar is below the threshold for qualifying as a long bar.

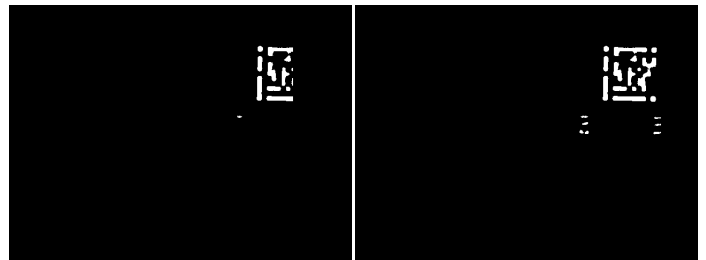


Fig. 12. True positive (masked version).

B. Tradeoffs

In developing code for this project, we must choose from several options with different tradeoffs. One particular tradeoff is between computational simplicity and flexibility of application. In the case where computational complexity is not an issue, we could generate code to correctly decode a larger set of images (with markers at more slanted perspectives, with more extreme illuminations, with larger or smaller markers, etc.). A more general algorithm could include several learning iterations to determine the brightness, distribution and size of bright spots, the possible sizes of the markers, and then adjust the parameters of the algorithm used to find and decode the markers. It is likely that this approach would positively detect more markers than the algorithm presented.

However, the actual constraints of this project make such an approach impractical and unnecessary. We are working to produce code that should run on portable devices such as cell phones or PDAs. This constrains the computing power, memory, and runtime demands for our program. We also consider that most cell phone cameras produce extremely noisy images due to hardware, software, and practical limitations of the crude cameras (e.g., one of the authors is the proud owner of a Motorola V600 cell phone with a camera that he carries in his pocket, and as a result the camera has more lint than the total lint of half the Packard building students' navels, combined), which further limits the effectiveness of various efforts to decode images.

Taking this into consideration, we decided that adaptive (iterative) determination of the various thresholds and parameters (e.g. the thresholds for generating the binary image, the size of smoothing filters) would be too computationally expensive for this application and the *ad hoc* determination of these values based on the algorithm's performance on the training set provided is justified. The masked version of the algorithm is in this spirit: it is faster, but places more constraints on the range of images to which it can be successfully applied. It does a perfect job on the given data set, but is less reliable with respect to an unknown data set.

C. Other methods considered, Future work

1) *Segmentation*: To gain insight into how well the black and white segmentation is expected to work, we looked at the statistics of the training data set. The RGB values of the pixels decoded as black and white, respectively, can be used to optimize the thresholds *a fortiori*. Three modes are defined to evaluate the statistics:

- mode 1: the average value and standard deviation of all the pixels eventually labeled black and eventually labeled white are calculated.
- mode 2: the weighted averages of the pixels labeled black and the pixels labeled white are calculated using the same kernel that was used in the decoding function.
- mode 3: the average values of the center pixels of each square are calculated for the pixels labeled black and the pixels labeled white.

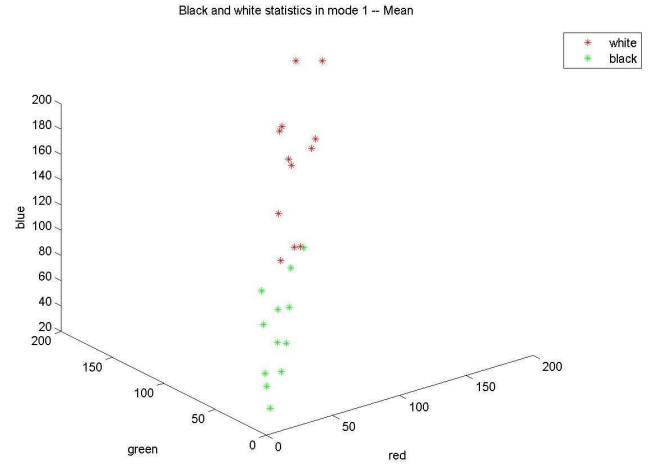


Fig. 13. The average value of all the pixels eventually labeled black and eventually labeled white.

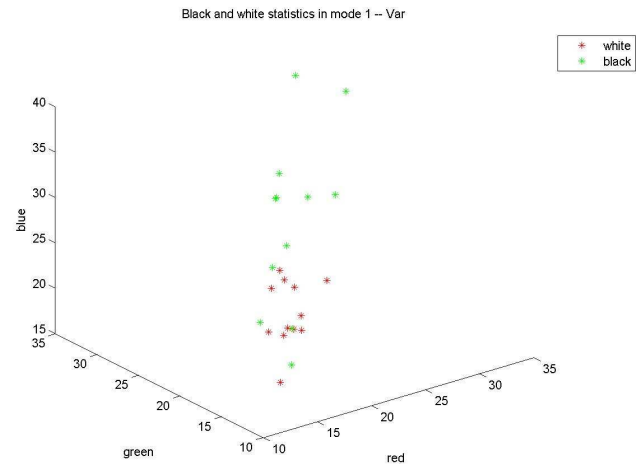


Fig. 14. The standard deviation of all the pixels eventually labeled black and eventually labeled white.

The results (Fig. 13, Fig. 14, Fig. 15, Fig. 16) confirm that a simple threshold is enough, since both clusters are well separated.

2) *Corner identification*: Since only four points are required to describe a projective transformation, the `cp2tform` function can be used to find all three corner elements. Knowing the centroids and orientation angles of the two guide bars, the end points of each bar can be calculated and used to find the transformation of the bars. The inverse transformation can be used to estimate the centroids of the nearest two corner elements. The corner elements would be identified as round objects whose centroids are located in the vicinity of the expected centroids. An estimate of the ground truth location can be similarly estimated from the centroids of the two guide bars and neighboring corner elements (as described in the method section). This two-stage approach will reduce the estimation error because the four end points of the guide

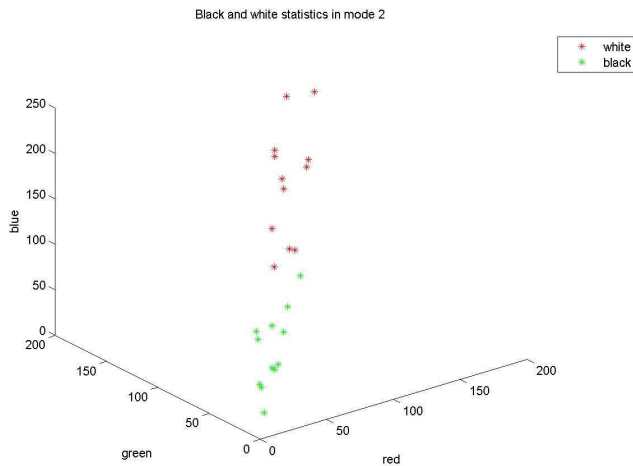


Fig. 15. The weighted averages of the pixels labeled black and the pixels labeled white.

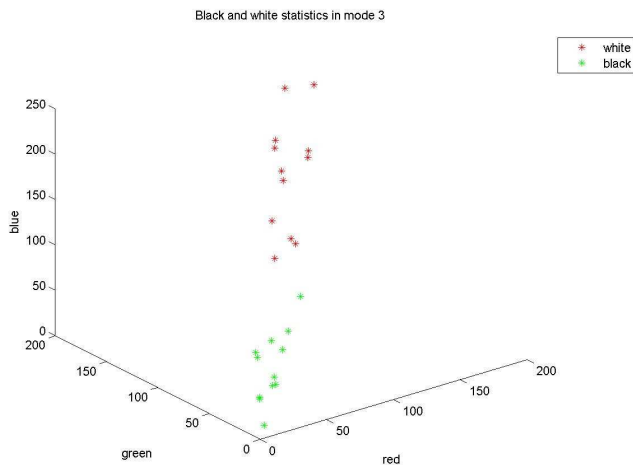


Fig. 16. The average values of the center pixels of each square labeled black and labeled white.

bars are not a very robust set of points for determining the transformation. This modification should reduce the number of computations necessary to locate the fixed elements, reducing the computational load.

V. CONCLUSION

This algorithm is designed to work on reasonable images as would be taken by a camera phone. It is understood that the quality of the images is limited by the cell phone's hardware and software. To accommodate for this, the algorithm is somewhat forgiving. At the same time, it is assumed that the user makes an effort to take a useful picture of the marker. That is to say, the picture is taken at a reasonable distance (not too close, not too far away) and at a reasonable angle (i.e. roughly normal to the marker surface). This is a reasonable assumption because the user is interested in garnering information from the picture, so there is no benefit of taking an obtuse picture. These

assumptions were used to simplify the algorithm. Since the application is for cell phones, where the processor resources are limited, the algorithm should be simple (in terms of computational demands) and quick, to provide the user with information in a timely manner. This algorithm works well with the training set provided: it correctly decodes every marker without any false positives. We are confident that this algorithm will work on similar images.

REFERENCES

- [1] 2005. [Online]. Available: <http://www.wirelesswatch.jp/>
- [2] M. Rohs, "Real-world interaction with camera-phones," *2nd International Symposium on Ubiquitous Computing Systems (UCS 2004)*, pp. 39–48, 2004.