Marker Detection for Augmented Reality Applications

Technical Report · October 2008

CITATIONS
64

READS
5,833

1 author:

Martin Hirzer
Graz University of Technology
19 PUBLICATIONS **3,492** CITATIONS

SEE PROFILE

computer graphics & vision

Inffeldgasse 16/2, A-8010 Graz
www.icg.tu-graz.ac.at

TUG

Marker Detection for Augmented Reality Applications

Martin Hirzer

Inst. for Computer Graphics and Vision
Graz University of Technology, Austria

Technical Report
ICG–TR–08/05
Graz, October 27, 2008

contact: Martin Hirzer hirzer@icg.tugraz.at

Abstract

In this paper we present a fast marker detection front end for Augmented Reality (AR) applications. The proposed algorithm is inspired by the ARTag system and designed to be robust against changing illumination conditions and occlusion of markers. In order to achieve this robustness we use an edge based approach. Edge pixels found by an edge detector are linked into lines by a RANSAC-grouper. These lines in turn are grouped into quadrangles. By detecting edge pixels only on a very coarse sampling grid the runtime of our algorithm is reduced significantly, so that we attain real time performance. Several experiments have been conducted on various images and video sequences. The obtained results demonstrate that our marker detection front end is fast and robust in case of changing lighting conditions and occlusions.

Keywords: *Marker detection, Augmented Reality, Edge based line detection, Changing illumination, Occlusion*

1 Introduction

1.1 Planar Marker Systems

In [4] Fiala gives an overview of planar marker systems. There are many practical vision systems that use two-dimensional patterns to carry information. Their field of application ranges from industrial systems, where markers are designed to label parts or carry certain information, e. g. shipping data, to systems where markers are used for localization, e. g. Augmented Reality and robot navigation systems. Examples for the first case are Maxicode, used by the US Postal Service, and DataMatrix and QR (Quick Response), used in industrial settings for the purpose of part labeling. Examples for the second case are ARToolKit, ARTag and ARSTudio, three systems for Augmented Reality.

To reduce the sensitivity to lightning conditions and camera settings planar marker systems typically use bitonal markers. So there is no need to identify shades of gray, and the decision made per pixel is reduced to a threshold decision. Furthermore many marker systems use some of the marker's data bits to convey redundant information, which allows for error detection and correction. The design of the markers mainly depends on the application. Figure 1 shows some examples. DataMatrix, Maxicode and QR are applicable for encoding information under controlled environments, e. g. conveyor belts, but are not very suitable for systems that use markers for localization. The markers of these three systems are not designed for large fields of view and the perspective distortions involved. Furthermore they require a large area in the image, so that the range at which these markers can be used is very limited. And finally they do not provide enough points in the image to enable three-dimensional pose calculation.

For Augmented Reality applications on the other hand it is very important that markers can be found within a large field of view. This means that they should also be detected if they appear distorted in the image. Further on the information stored inside the marker must not be too dense in order to increase the distance at which data can be recovered from the marker. Fortunately this can easily be achieved since a marker carries less information in Augmented Reality applications, typically only an id to distinguish it from other markers. Most Augmented Reality systems also work if only one marker is visible. Hence the marker must have some distinct points, at least four, to allow for camera-marker pose calculation. Usually such markers have a quadrilateral outline, and the four corner points are used for three-dimensional pose calculation.

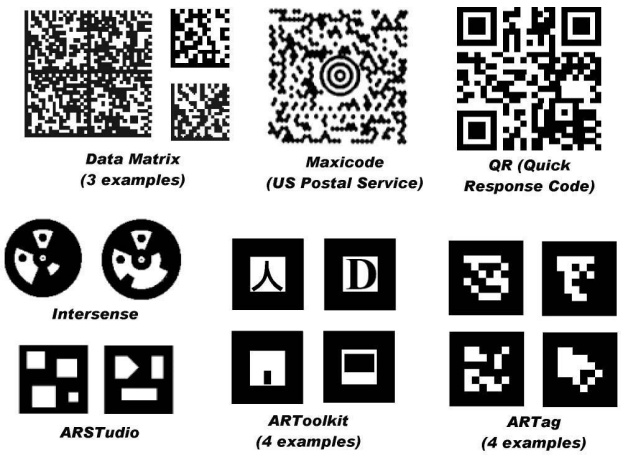


Figure 1: Different markers taken from [4]

1.1.1 ARToolKit

ARToolKit [7] is a popular planar marker system for Augmented Reality and Human Computer Interaction (HCI) systems due to its available source code. The bitonal markers consist of a square black border and a pattern in the interior. The first stage of the recognition process is finding the markers' black borders, that is finding connected groups of pixels below a certain gray value threshold. Then the contour of each group is extracted, and finally those groups surrounded by four straight lines are marked as potential markers. The four corners of every potential marker are used to calculate a homography in order to remove the perspective distortion. Once the internal pattern of a marker is brought to a canonical front view one can sample a grid of NxN (usually 16x16 or 32x32) gray values inside. These gray values form a feature vector that is compared to a library of feature vectors of known markers by correlation. The output of this template matching is a confidence factor. If this confidence factor is greater than a threshold, a marker has been found.

Although ARToolKit is useful for many applications, there are some drawbacks. First of all the detection process is threshold based. A single threshold can easily fail to detect markers under different illumination conditions, even within the same image. For example, it can happen that the white level at one marker edge is darker than the black level at the opposite edge. As a result users often have modified the code to perform local thresholding, or to run multiple detection runs for a single image, each with a different threshold (see [11] for instance). Furthermore the marker verification and identification mechanism using correlation causes high false positive and inter-marker con-

fusion rates. With increasing library size the marker uniqueness is reduced, which again increases the inter-marker confusion rate. The processing time also depends on the library size, since every feature vector must be correlated with every prototype vector in the library. And for each marker there exist several prototype vectors to cover the four possible rotations as well as different lightning and distance conditions.

1.1.2 ARTag

ARTag [4] is another planar marker system for Augmented Reality and Human Computer Interaction systems. ARTag also uses markers with a square border (black or white). In contrast to ARToolKit ARTag finds markers with an edge based approach, so one need not to deal with thresholds under different illumination conditions. Edge pixels found by an edge detector serve as basis for the marker detection process. They are linked into segments, which in turn are grouped into quadrangles. As with ARToolKit the corners of a quadrangle are used to calculate a homography so that the marker's interior can be sampled. In contrast to the patterns used in ARToolKit the interior region of an ARTag marker is filled with a 6x6 grid of black or white cells, representing 36 binary '0' or '1' symbols. This 36-bit word is then processed in the digital domain. For each of the four possible marker orientations one 36-bit sequence is obtained from the 36-bit word, with only one sequence ending up being used in the decoding process. Every 36-bit sequence encodes a 10-bit marker id, leaving 26 redundant bits for error detection, correction and uniqueness over the four possible rotations of a marker.

The edge based approach of ARTag makes the system more robust to changing illumination conditions than ARToolKit. ARTag can even cope with occlusions, broken sides and missing corners to a certain extent. This is possible because of heuristics of line segments that almost meet, so that missing segments of a marker can be estimated. Furthermore ARTag's id based markers do not require image matching with a library and therefore allow for a much faster identification than the template based markers used in ARToolKit.

1.2 Line Detection

Since the marker detection front end presented in this paper follows the ARTag approach, a brief overview of line detection methods is given at this point. Guru et al. [5] broadly classify them into four categories: statistical based, gradient based, pixel connectivity edge linking based and Hough transform based algorithms.

An example of a statistical based line detector is the hypothesize-and-test algorithm proposed by Mansouri et al. [9]. It hypothesizes the existence of line segments of specified lengths by the use of local information. To verify a hypothesis the statistical properties of a model of an ideal segment are explored. Another example is line detection based on small eigenvalue analysis, as proposed by Guru et al. themselves. Their algorithm scans the input edge image with a moving mask. At every position the small eigenvalue of the covariance matrix of the edge pixels within the mask and connected to the center pixel of the mask is evaluated. If this small eigenvalue is less than a predefined threshold, the corresponding connected pixels are considered to be linear edge pixels.

In contrast to this gradient based algorithms use gradient magnitude and orientation properties to detect lines. The detection is based on pixels with high gradient magnitude and support regions, derived from the gradient orientation of the line (see [2] for instance).

The third line detector category mentioned by Guru et al. encompasses algorithms that find local edge pixels, link them into contours based on proximity and orientation, and finally combine these contours into longer, relatively straight line pieces (see [10] and [3] for example). The fact that the connectivity among all identified linear edge pixels is very much ensured allows pixel connectivity edge linking algorithms to outperform other line detection methods in many cases.

The last category are Hough transform based detectors. The well known Hough transform [6] detects geometrical figures using their parametrical representation. For lines the polar representation is usually chosen ($r = x \cdot \cos(\theta) + y \cdot \sin(\theta)$, r: distance between the line and the origin, θ: angle between the line's normal and the x-axis). The input is a binary edge image where all edge pixels found in the image are set. The Hough transform requires a so called accumulator array. This array has one counter for every possible parameter combination (r, θ). Every line that can be built by connecting two of the edge pixels is considered, and the associated parameters r and θ determine the accumulator array value that has to be incremented. After all possible lines have been processed, high array values represent lines that are very likely. Although the Hough transform is very robust to noise and discontinuities in an image, there are limiting drawbacks. As one can imagine from the description of the method, the Hough transform is incapable of finding the end points of a line. Furthermore short lines in the image result in only low peaks in the accumulator array and therefore are likely to be missed. Finally the Hough transform has a high computation time and requires a lot of memory, because there is one counter for every possible parameter combination. To overcome these limitations several versions have been derived

from the standard Hough transform. The probabilistic Hough transform [8], for example, reduces the computation time by processing only a randomly selected subset of edge pixels. Of course this also lowers the robustness and precision of the result, and finding a trade-off between computational complexity and solution quality can be a hard task. Another improved version of the Hough transform is presented in [1]. The authors propose an algorithm that finds complete line segments using the Hough transform. This means that their algorithm does not just determine a line's parameters r and θ like the standard Hough transform, but also its end points.

2 Algorithm

As already mentioned, our marker detection front end follows the ARTag approach. Edge pixels found by an edge detector are linked into segments, which in turn are grouped into quadrangles. Our algorithm consists of three main steps. First line segments are found by detecting edgels (short for edge pixels) on a coarse sampling grid and linking them together. Then the line segments are merged in order to obtain longer lines. In the next step all detected lines are extended based on gradient information, so that we receive lines of full length. Finally these are grouped into quadrangles. In the following the three main steps, line detection, line extension and line grouping are described in detail.

2.1 Line Detection

2.1.1 RANSAC Based Line Segment Detector

The line detector used in our application must find marker edges accurately, so that their intersections, the marker's corner points, can be located. Moreover the proposed marker detection front end should allow an Augmented Reality application to run in real time, so a very fast line detector is required. Since we need an efficient detector, and also want to be able to easily add problem-specific heuristics when detecting markers, Hough based methods are inappropriate for our application. Among the line detection approaches stated in Section 1.2 pixel connectivity edge linking algorithms seem to be best suited, because they are said to outperform other approaches. A very efficient line detector out of this category is described in [3]. It consists of two steps: a sparse sampling of the image data and a RANSAC-grouper, which follows the hypothesize-and-test scheme.

In the first step the algorithm tries to find candidate points for lines, so called edgels. To avoid processing of all pixels in an image this is done on a

rather coarse sampling grid, making the algorithm very fast. The sampling grid is usually rectangular and consists of widely spaced horizontal and vertical scanlines, but other directions for the scanlines are also possible. Each of these scanlines is convolved with an one-dimensional derivative of Gaussian kernel to calculate the component of the intensity gradient along the scanline. Local maxima of the intensity gradient that are greater than a certain threshold are considered edgels, and the orientation of each edgel is calculated ($\theta = \arctan(g_y/g_x)$, g_y: y-component of the gradient, g_x: x-component of the gradient). Note that the orientation of edgels, and with it the orientation of line segments and finally lines, can take on values from 0° to 360°, depending on the image's intensity values. This means that edgels that are located at a black-white edge and edgels that are located at a white-black edge of same direction differ about 180° in their orientation. From the above description it is evident that the detector can be tuned to a certain line orientation by aligning the scanlines perpendicular to this orientation.

In the second stage a RANSAC-grouper is used to find straight line segments on the basis of the edgels found in the first stage. Therefore the image is divided into small regions which are processed consecutively by the grouper. To hypothesize a line the RANSAC-grouper uses two randomly chosen edgels, whose orientations are compatible with the line connecting them. Now the number of edgels that support the hypothesis is determined. To count as a supporting edgel for a line hypothesis the edgel must lie close to the considered line and have an orientation compatible with it (see Figure 2 for clarification). This process of hypothesizing lines and counting their supporting edgels is repeated several times in order to find the line that receives the most support. If this dominant line has enough support, it is deemed present in the image, and all of its supporting edgels are removed from the edgel set. To detect all such dominant lines within a region the entire process is repeated until either most of the edgels have been removed or an upper limit of iterations has been reached. After processing all regions in this way the detection algorithm is finished.

The proposed algorithm is extremely fast and tunable for both the scale and orientation of the desired line segments. A disadvantage is the slightly anisotropic detection behavior if a rectangular sampling grid is used. The reason therefore is the discrimination against diagonal line segments caused by the rectangular grid.

To improve the performance of the proposed line detector in our application we have made a little adaptation, because we only want to detect black marker borders on bright backgrounds. So if we have a color image we can prevent the line detector from finding unimportant edgels. One of the image's three color channels is processed by the line detector as described

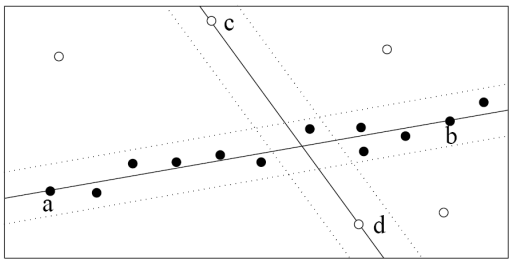


Figure 2: The RANSAC-grouper in presence of statistical noise and gross outliers. Here two line hypotheses are tested. Line ab has 12 supporting edgels (edgels within the dotted boundaries), whereas line cd has only 3. Hence line ab is deemed present in the image, and line cd is ruled out. The outliers are represented by open circles. The figure is taken from [3].

above. The only difference is that if an edgel is found in this color channel we also have to ensure that there is an approximately equally strong edgel at the same position in each of the remaining two channels. Strong edgels that have almost the same intensity gradient value in all three color channels correspond to black-white edges, whereas edgels with different values correspond to color edges. Removing such superfluous color edgels decreases the algorithm run time, as less edgels have to be considered in the line finding step, and increases robustness at the same time, since a lot of irrelevant lines are not even detected. Of course if the input image is a gray value image, and therefore consists of only one single channel, we cannot distinguish between important and superfluous edgels. In this case all strong edgels must be processed. Figure 3 shows an example image overlaid with scanlines, edgels and line segments. When visualizing the further steps we will use the same image, so that one can observe how our algorithm detects markers, starting with edgels and ending up with complete quadrangles.

To decrease the run time of our marker detection algorithm in videos we do not scan all frames completely. Instead we track markers. Previous marker positions are used to guide the line detection process. While lines are detected in the whole first frame, the following frames are only processed partially to save time. Line detection is only performed in image regions that contained a marker in the previous frame. After a certain number of frames one frame is processed completely again, so that new markers that have come into the field of view can be detected. In Figure 4 marker tracking is visualized.

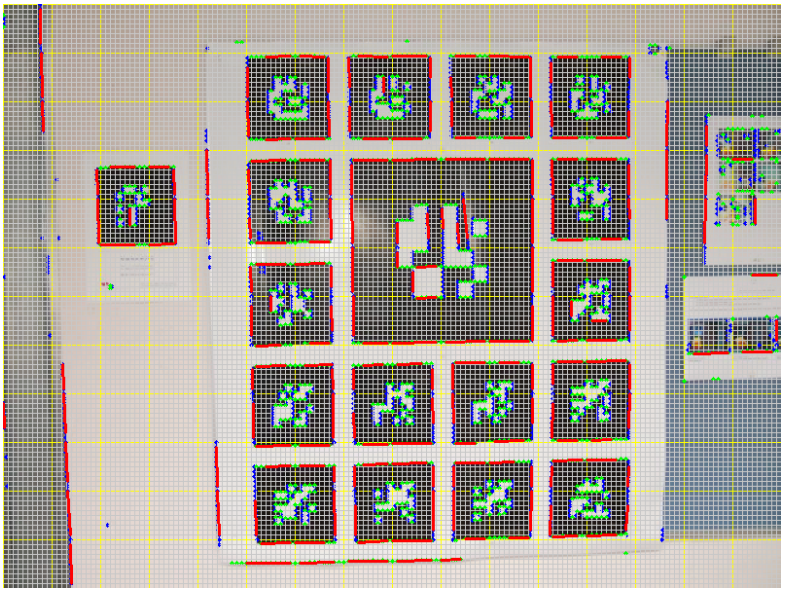
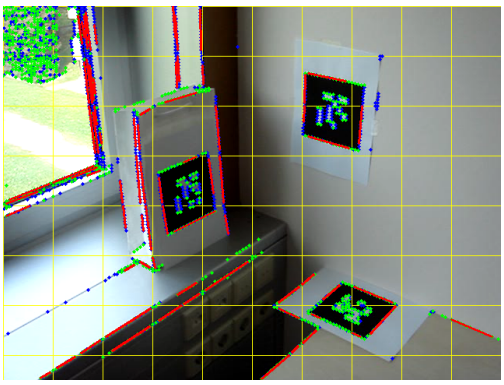


Figure 3: Image overlaid with scanlines (gray). The yellow ones represent region borders. Line segments are shown in red, edgels are marked blue (x-direction) and green (y-direction).

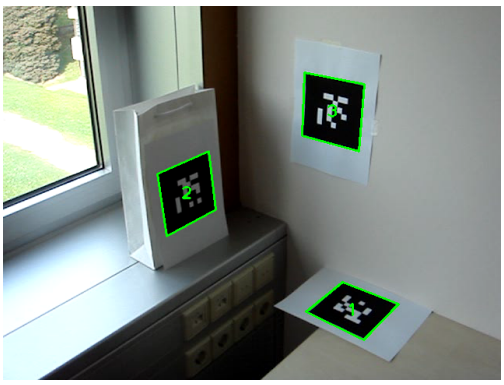
2.1.2 Merging of Line Segments

After the first step we only have rather short line segments. In order to obtain longer lines we must merge corresponding line segments. We do so by testing all merging possibilities. Two line segments are merged if they meet two criteria. First their orientations must be compatible. This is a pretty obvious criterion since we want to find straight lines. We define a maximum allowed orientation difference, and only line segment pairs that have an orientation difference smaller than this threshold are further examined. But just checking the orientations of two line segments is of course not sufficient, because this would lead to the situation that parallel line segments that do not lie on the same line are merged too.

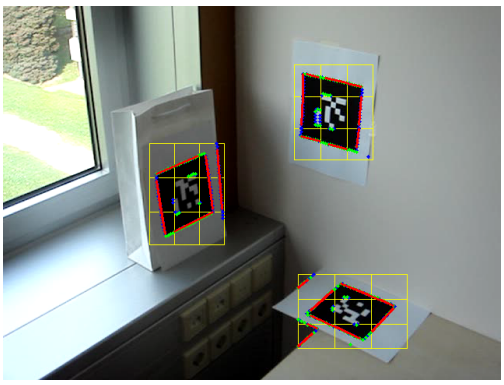
Therefore the second criterion relates to the connection line of the two line segments. The orientation of the connection line must also be compatible with the orientations of the two line segments. Since the connection line now must have an orientation similar to the line segments, parallel line segments that do not lie on the same line are no longer merged. But there is still one case left in which non-corresponding line segments are merged. Imagine, for example, that several markers are visible in an image, and that these markers are aligned on a regular grid (like in Figure 3). With the merging tests defined so far it would be possible that line segments of neighboring markers



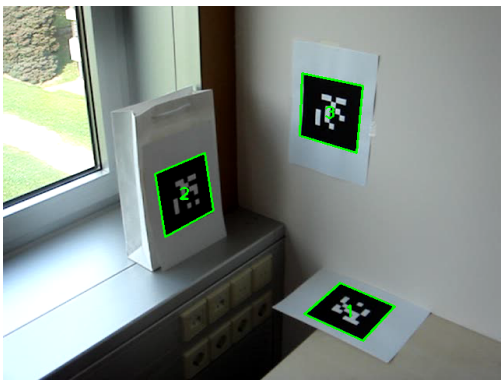
(a) Scan whole image.



(b) Detected markers



(c) Scan only image regions according to detected markers.



(d) Detected markers

Figure 4: Marker tracking. First markers are searched in the whole image (Figure 4a). Then the detected markers (Figure 4b) are used to concentrate the search effort on image areas that contained a marker in the previous frame (Figure 4c). Usually all markers are found again (Figure 4d). Note that the individual image areas in Figure 4c are subdivided into nine subregions, and that the central subregion is not processed, because it mainly covers the marker's internal pattern.

link together (they as well as their connection line have similar orientations). This would result in long lines bridging over any other image regions, like the white background that surrounds the markers. To avoid this we have to check the connection line point by point. The gradient orientation at all points of the connection line must be compatible with its orientation. The calculation of the gradient orientations is the same as for edgels. Finally line segments that also pass this last test are merged.

Up to now nothing has been said about the distance between line segments in the merging process. Of course we want to link line segments that are close to each other, because such line segments are likely to belong to the same line. Hence in an early version of the merging algorithm we used a threshold value for the maximum allowed distance. The problem was that on the one hand if the value was too small not all corresponding line segments were merged, so that the result contained gaps. On the other hand choosing a too great value could lead to the situation that exterior line segments of a line were merged first. As a result it could happen that interior line segments were not merged with the line anymore, because the distance to the line's end points was too long now. And the remaining line segments could easily cause problems in later stages. To overcome this problem we ordered the line segment merging process. Now it starts with the two compatible line segments with the shortest connection line, then come the two compatible line segments with the second shortest connection line, and so on. In this way we ensure that closer line segments are merged before line segments that are farther apart.

To decrease the run time of our algorithm two merging steps are carried out. As already mentioned in Section 2.1.1, the image is partitioned into small regions in order to obtain line segments. The first merging step is carried out per region and only takes line segments of the current region into account. The second merging step is then applied to the whole image. With the two step approach we avoid having to check all line segment combinations within the whole image, because the local merging step reduces the number of line segments significantly for the global run.

2.2 Line Extension

So far we have detected line segments and merged them to obtain longer lines. But we still cannot be sure that these lines represent the corresponding image edges entirely, because the length of the detected lines depends on the positions of the line segments. Usually there is a short piece missing at both ends of a line. Hence we try to extend all lines straightly at both of their ends to receive lines of full length. We again us the gradient orientation for

this task. After selecting one of the two end points the line is elongated one pixel there. Now the gradient orientation of this new line point is checked. If it is compatible with the line's orientation, the new point is added to the line. Then the next extension point is examined. If it fits the line, it is also added. This process continues until we find an extension point with a different gradient orientation. At this point the true image edge seems to stop, hence it becomes the new line end point. Now that we have extended the line at one end the other end is processed in the same way.

After a line has been extended, a final line end point test is carried out. Since we are searching black markers on bright backgrounds, the gray value of a test point lying on a further extension of the line beyond the line end point is examined. If this test point is bright enough, the corresponding line end point is valid for corner detection (see Section 2.3.1). Otherwise the line is likely to be a line inside the marker's pattern (white cells on black background), or the according marker edge stops at a dark, occluding object. Anyway, the corresponding line end point is tagged as invalid for corner detection. If both of a line's end points are unsuitable for corner detection, the line is removed. Figure 5 visualizes the extension algorithm.

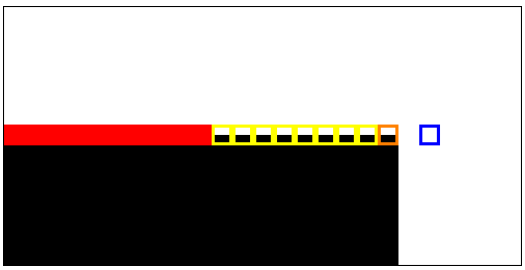
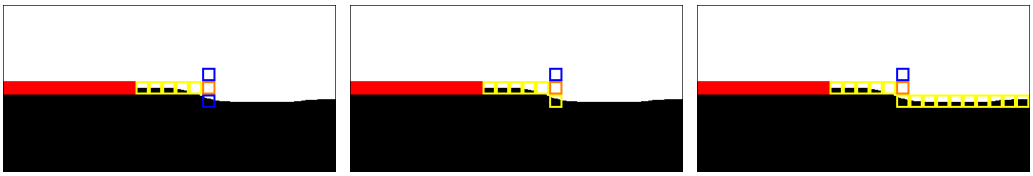


Figure 5: A line (red) is extended by adding compatible line points (yellow). This goes on until an incompatible extension point (orange) is found. Finally a test point (blue) is examined to determine whether the newly found line end is suitable for corner detection.

However, there are also other reasons that can cause the extension process to stop. One reason is the image border. If a line runs out of the image, the extension process of course stops at the image border. In such a case the corresponding line is removed, because it is unsuitable for reliable marker corner detection. Furthermore the extension process can be hindered by slight deviations of a line from the true image edge. A line that is not perfectly aligned with the corresponding image edge will depart from it when being extended. Further on sometimes, due to distortions induced by the camera, the edges in an image are not absolutely straight. As a result a purely straight line extension will fail in such cases. To overcome these

problems the extension process was modified, so that it now does not only take points that lie in line with the corresponding line into account, but also points lying perpendicular to the current line growing direction. This allows for compensating slight deviations and curvatures and hence makes the extension algorithm more robust. Figure 6 shows how the extensions algorithm adapts to a slightly distorted line. In Figure 7 our example image overlaid with the detected lines and their extensions is depicted.



(a) **(b)** **(c)**

Figure 6: Figure 6a shows how the line extension process stops due to a distorted section of the line. Again compatible extension points are marked yellow, incompatible ones are marked orange. Now the two points lying perpendicular to the current line growing direction, marked blue, are examined. The lower one has a similar orientation to the line (Figure 6b), so the extension process continues there (Figure 6c).

2.3 Quadrangle Detection

2.3.1 Corner Detection

The last step is the detection of quadrangles based on the set of extended lines. To obtain quadrangles we search for corner points by intersecting lines. The algorithm picks one line out of the set and tries to find a corner by intersecting the chosen line with the right one among all other lines of the set. To find a suitable second line several tests are carried out. First of all the two lines must not be nearly parallel, because we want to find quadrangular markers. By doing so our algorithm misses extremely distorted markers where neighboring sides are approximately parallel. But this is not a big issue since the interior of such markers cannot be sampled reliably anyway. The next test checks the smallest distance between the two lines' end points. There are four possible end point combinations for two lines ab and cd: ac, ad, bc and bd. If the minimum among these distances is smaller than a certain threshold, the two lines are further examined. The two corresponding end points mark the line ends where the intersection, if passing the next test, will

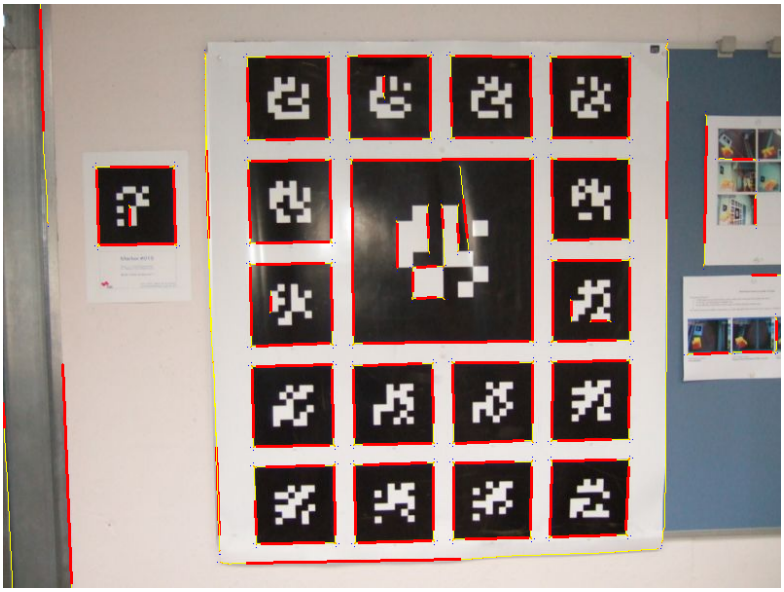
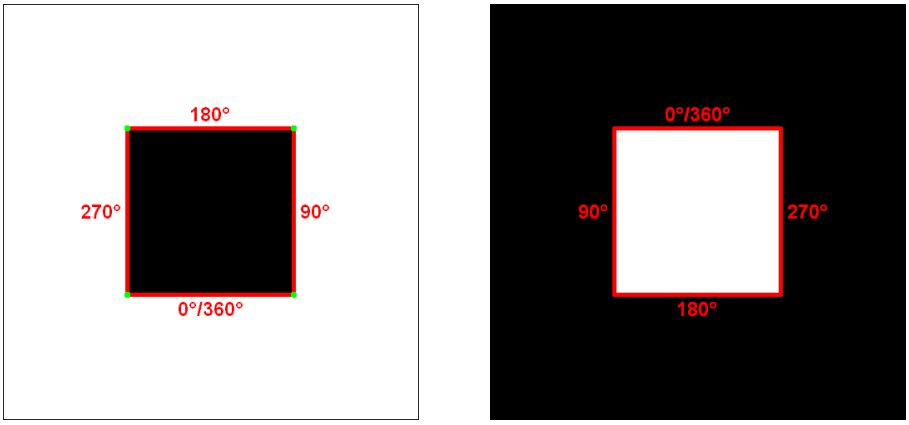


Figure 7: By merging line segments we obtain longer lines (red). Afterwards these longer lines are further extended (yellow).

be. This last test once more checks the orientations of the two lines. This time the algorithm takes into account that we only want to detect markers, i.e. black regions surrounded by white background. And at this point it is known at which end of each line the intersection point, and with it the corner, will be located. We can use this information to verify that the two lines are enclosing a dark region by checking their orientations. Remember that the orientation of a line depends on the intensity values, and that lines of same direction can differ about 180° in their orientations. For example, imagine an image of a black, squared marker on white background. Let us further assume that the marker is aligned with the image so that the lower edge has an orientation of 0°/360°, the right edge has an orientation of 90°, the upper edge has an orientation of 180°, and the left edge has an orientation of 270°. Suppose that four lines, one at each of the marker's edges, were found. We now consider the lower horizontal line, which separates the black region sitting above it from the white region below. We want to intersect this line with the right vertical line in order to obtain the lower right corner point. But before actually intersecting these lines we must check that the orientation of the right line lies in the range of 0° to 180°. If so we are sure that these two lines enclose the black marker region. In contrast to this the valid orientation range is different when searching for the lower left corner point. The left vertical line must have an orientation lying in the range of

180° to 360° to be qualified for intersecting it with the lower line. See Figure 8 for clarification. Note that we need not to wonder about border cases here because, as stated above, neighboring lines cannot be nearly parallel. Lastly line pairs that have passed this final test are intersected, and the intersection point is stored as one of the marker's corer points.



(a) Compatible line orientations

(b) Incompatible line orientations

Figure 8: The left figure shows a black square surrounded by a white background. The four lines' orientations are compatible, and hence four corners (green) are detected. The right figure shows the opposite case, a white square surrounded by a black background. Here the orientations of the lines are incompatible, so no corners are found.

But just finding corner points by simply intersecting suitable line pairs is not sufficient to identify markers. We must find four corners belonging to the same marker. Also their sequence should be ordered to avoid cross-sequences, which means that if we follow the corners according to their sequence we should not cross the marker's interior (e.g. marker $ABCD$, possible cross-sequence $ACBD$). Hence the corner detection algorithm is recursive. It starts with a first line and tries to find a corner by intersecting this line with a second one, just like described above. If a corner point has been found, the algorithm continues with the second line. It now tries to intersect this line with any of the remaining lines in order to find a second corner point. If successful the algorithm moves on to the newly attached line and searches for a third corner point. This procedure is repeated until either four corners have been found or no corner has been found in the current detection run. In the latter case the corner detection process continues at the second, yet unprocessed end point of the first line of the recursion. Again the procedure is

repeated until either the total number of corners equals four or no corner has been found in the current detection run. In this way the algorithm detects corners that are connected by a line chain. Ideally this chain is closed and consists of four lines and four corners. Figure 9 visualizes the corner detection process, and Figure 10 shows the example image overlaid with the extended lines and detected corners.

2.3.2 Quadrangle Construction

If four corners have been detected in the previous step, we have found a complete quadrangle, and thus we are finished. But if less than four corners have been detected, for instance, due to occlusion like in the example shown in Figure 9, we must complete the quadrangle. In case of three detected corners the fourth corner can be estimated by simply intersecting the two lines that are adjacent to the gap. In most cases this estimation is quite accurate. The next scenario is that only two corners have been detected. In such a case we can just try to complete the quadrangle by connecting the two open ends of the line chain. However, whether or not the obtained quadrangle matches the corresponding marker accurately depends on the precision of the two lines that have been connected. If they represent the corresponding marker edges well, the estimated quadrangle will be correct. But if at least one of the two lines is too short the estimated quadrangle will be inappropriate. The last possible case is that only one corner has been detected. In this situation the quadrangle cannot be completed anymore, so it is rejected. The different cases are depicted in Figure 11. Figure 12 shows our example image overlaid with detected markers.

3 Results

To evaluate our algorithm we compared it to the open source marker tracking library ARToolKitPlus [11], an improved version of ARToolKit that is especially targeted at mobile devices. The improvements include automatic thresholding, vignetting compensation and the use of binary marker patterns (like those in ARTag) among others. Automatic thresholding is a technique to adapt the threshold that is used for marker detection to changing lighting conditions. After having one or more markers detected in a video, the median of all extracted marker pixels is calculated and used as threshold for the detection process in the next video frame. If no marker has been found, a randomized threshold is used until a new marker is detected. Calculating the marker detection threshold in this way is clearly an advantage over the

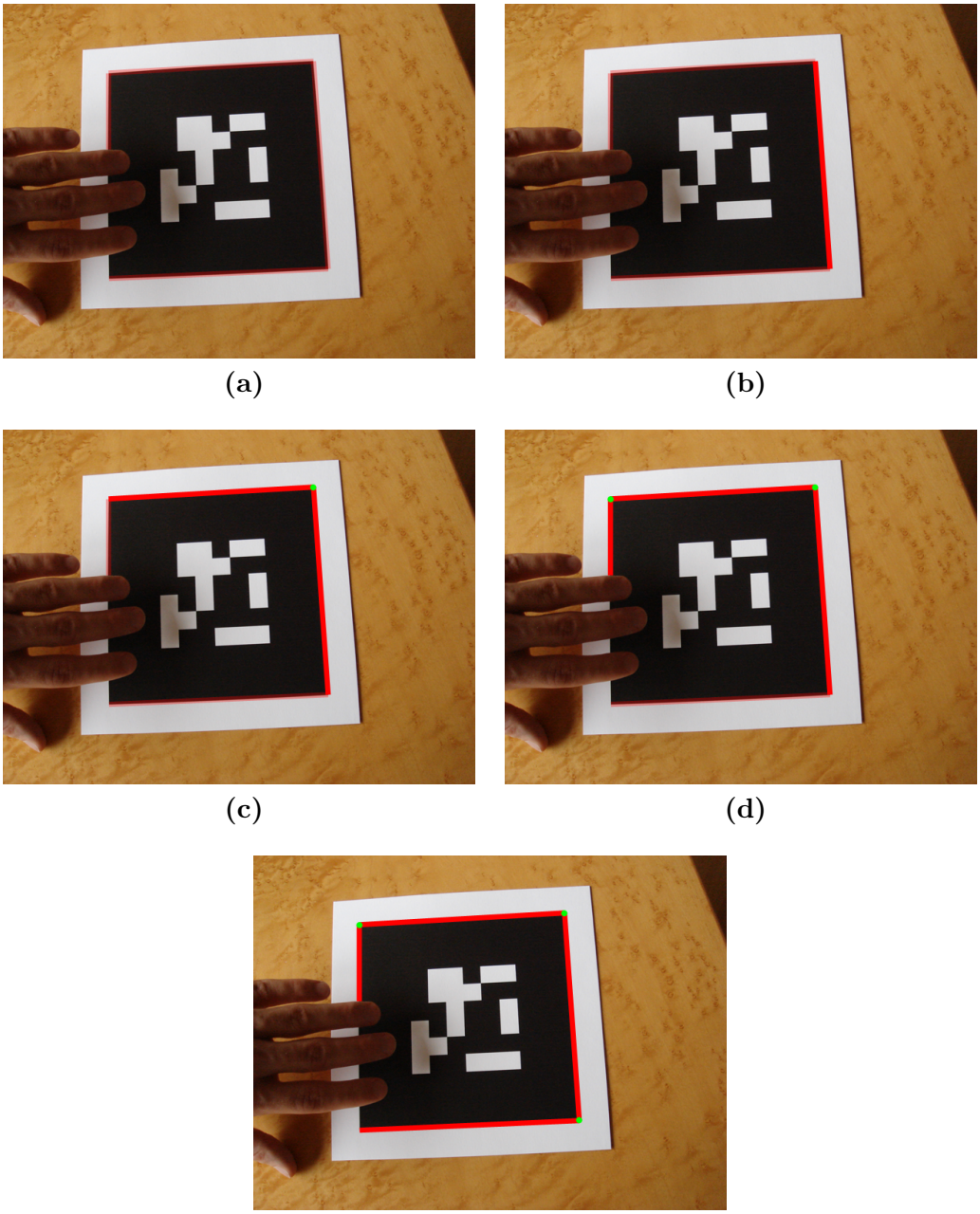


Figure 9: Figure 9a shows a marker and the detected marker edges (reddishly transparent). The left edge is not detected entirely because of occlusion. In Figure 9b the randomly chosen first line where the recursive corner detection algorithm starts is marked red. The algorithm finds a corner (green) by intersecting the first line with the upper line (Figure 9c), then continues with this second line, and finally finds a second corner (Figure 9d). Due to the occlusion the recursion cannot be continued at this end of the line chain. So the algorithm examines the second, yet unprocessed end point of the first line of the recursion and finds a third corner there (Figure 9e).

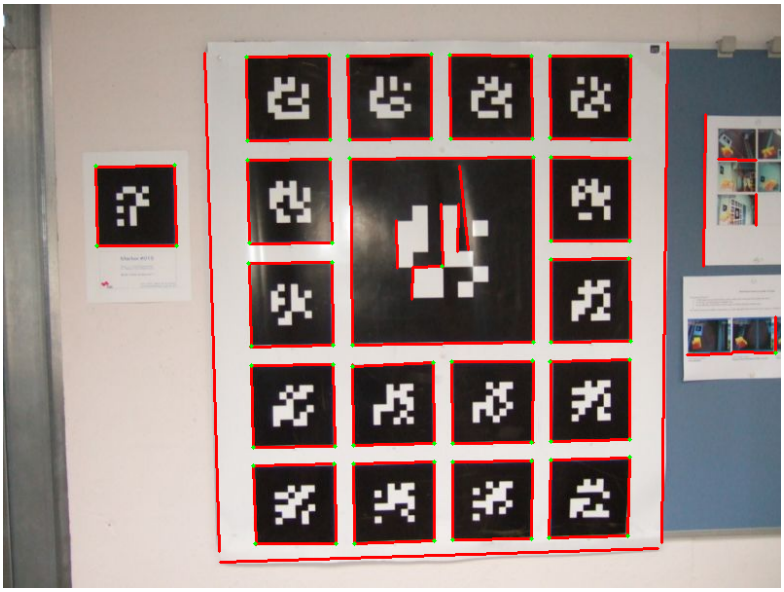


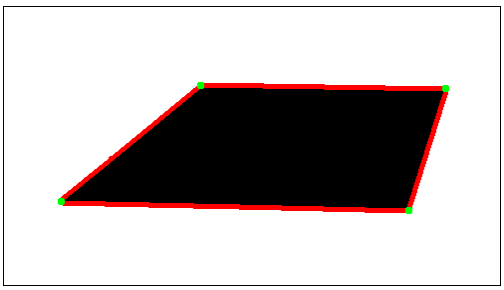
Figure 10: Extended lines (red) are intersected in order to obtain corners (green).

original ARToolKit approach, which uses a fixed threshold and thus is likely to fail in case of changing illumination conditions. To compare our detection algorithm to ARToolKitPlus we measured the marker detection and identification performance of both methods. In order to do so we included the marker identification mechanism of ARToolKitPlus, which is similar to the one of ARTag described in Section 1.1.2, in our algorithm.

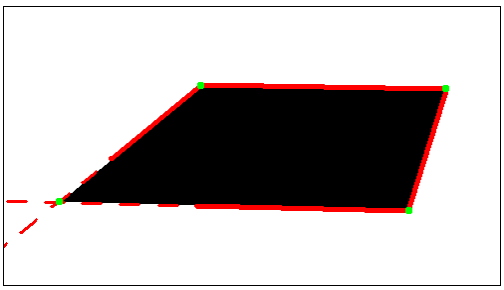
We applied our marker detection algorithm as well as ARToolKitPlus to the example image that we have used so far and two short video sequences, both having a duration of approximately half a minute. The first video shows three markers on a typical desktop, with two of them occluded. The second video also contains three markers. This time the markers are located near a window in such a way that they are illuminated differently. In each of the two videos all three markers are completely visible in all frames. The resolution of the example image is 800x600, whereas the two videos have a resolution of 640x480.

Figure 13 shows the results for the example image. As one can see, our algorithm is able to detect all markers present in the image, whereas ARToolKitPlus misses marker #124 due to the reflection caused by the camera flashlight. This is a direct consequence of the threshold based detection mechanism used in ARToolKitPlus.

In Figure 14 three representative frames of the desktop video, once over-

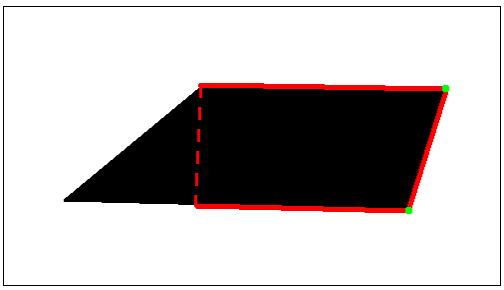


(a) 4 corners detected, quadrangle found.

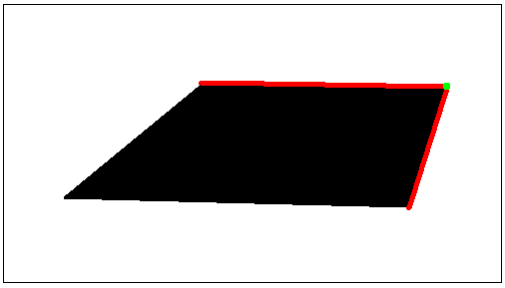


(b) 3 corners detected, fourth corner estimated.

(c) 2 corners detected, estimated quadrangle correct.



(d) 2 corners detected, estimated quadrangle incorrect.



(e) 1 corner detected, no quadrangle can be built.

Figure 11: Different starting situations for the quadrangle construction process

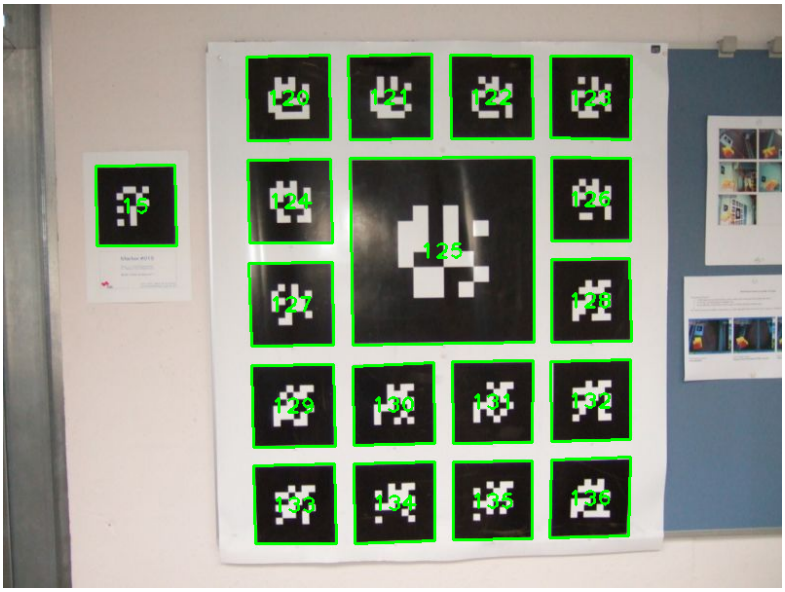
Figure 12: Detected markers

laid with the markers detected by our algorithm, and once overlaid with the markers found by ARToolKitPlus, are depicted. One can see clearly that our algorithm detected and identified all three markers correctly, whereas ARToolKitPlus failed in detecting the occluded markers. Table 1 summarizes the results for each algorithm.

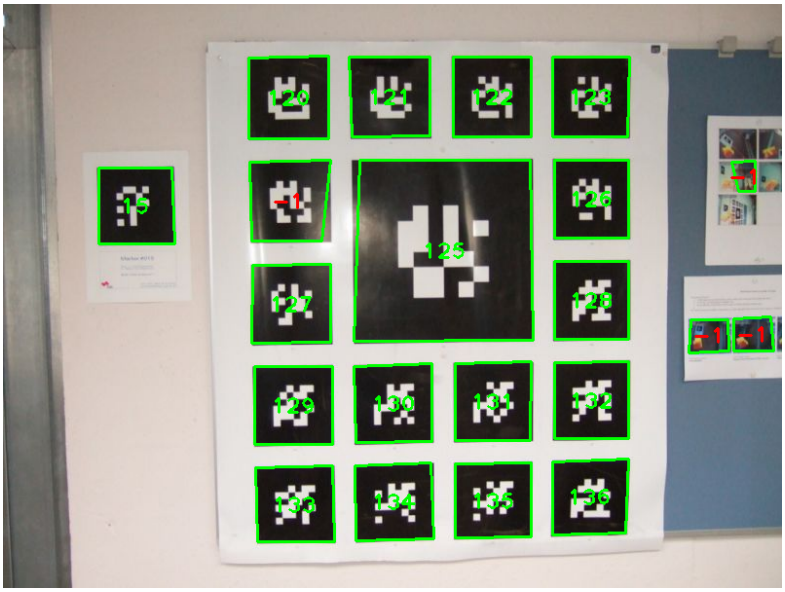
Marker Id	Our Algorithm	ARToolKitPlus
0	99.90%	100.00%
1	98.43%	0.00%
2	99.41%	0.00%

Table 1: Percentage of frames of the desktop scene video in which the individual markers are detected and identified correctly

In the second video the focus is on changing lighting conditions and differently illuminated markers (Figure 15). At the beginning all markers are very dark due to backlight. As the camera moves, the markers become brighter, especially markers #1 and #3. Now both algorithms start detecting the markers, although the results are quite unstable. Our algorithm identifies markers #1 and #3 correctly, but still misses marker #2 because it is too dark. ARToolKitPlus on the other hand is now able to identify all three markers, although marker #2 is only sporadically detected. At this point something interesting happens. As the camera moves further, all three mark-



(a) Our algorithm



(b) ARToolKitPlus

Figure 13: Detected and identified markers in the example image

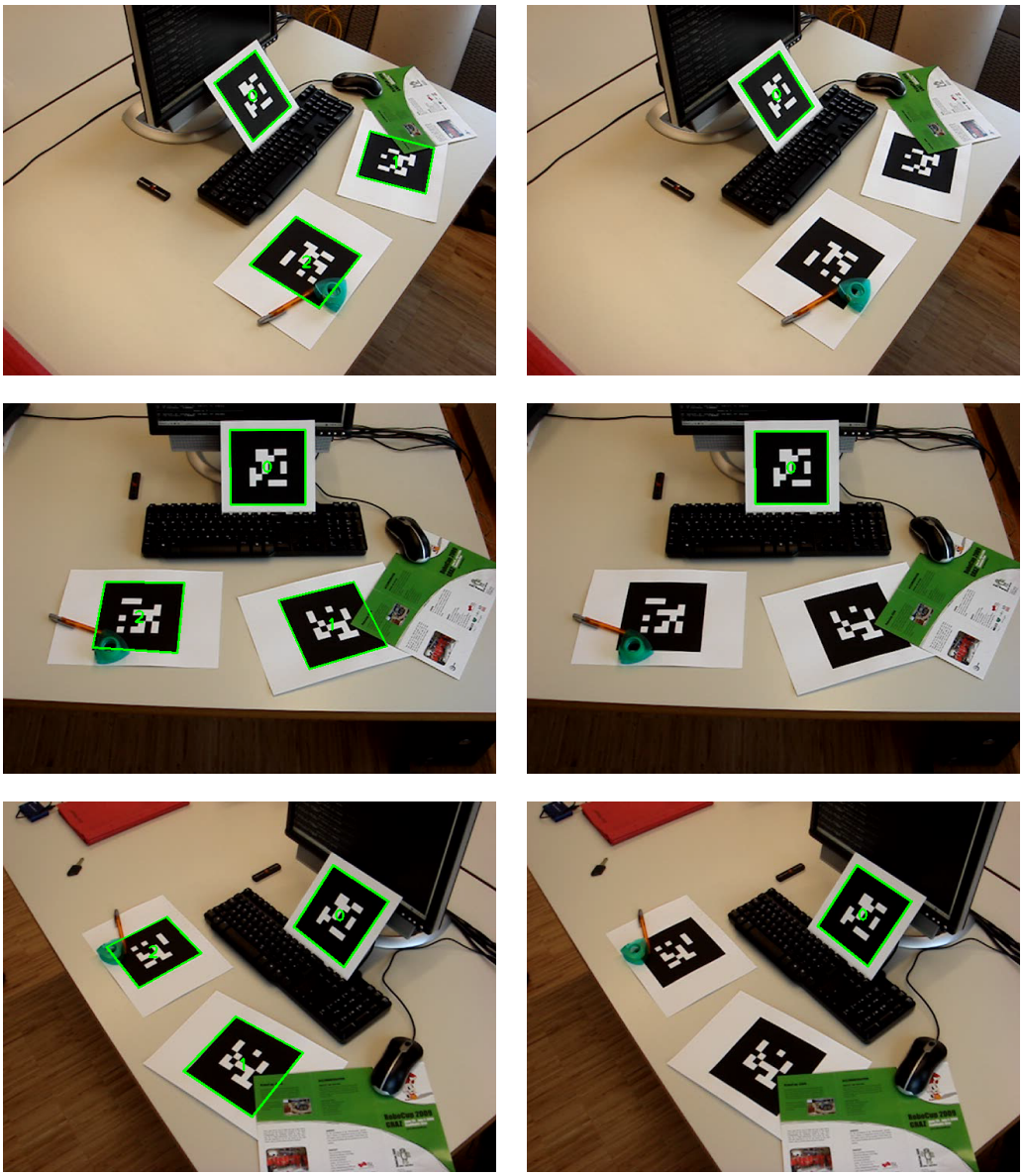


Figure 14: Three frames taken from the video of the desktop scene. The rows are arranged chronologically. The left column shows the markers detected by our algorithm, the right column depicts the results of ARToolKitPlus.

ers become brighter again, with marker #2 still being the darkest of them. For our algorithm all markers are bright enough now. Hence they are identified correctly. But ARToolKitPlus now fails to detect marker #2. Only markers #1 and #3 are found anymore. The reason for this seems to be the mechanism that is used to calculate the detection threshold. It is defined as the median of all extracted marker pixels. Since markers #1 and #3 are well illuminated now, the threshold is set to a value that is suitable for such well lit markers. Consequently marker #2, which is still significantly darker than the other two markers, is missed by ARToolKitPlus. The results for the second video are summarized in Table 2.

Marker Id	Our Algorithm	ARToolKitPlus
1	86.05%	79.20%
2	63.18%	12.53%
3	78.94%	81.78%

Table 2: Percentage of frames of the window scene video in which the individual markers are detected and identified correctly

The runtime of our algorithm of course depends on the size and content of an image. The more black-white edges are present in the image the more edgels, line segments and finally lines have to be examined. On a typical desktop PC (Pentium 4 3.0 GHz) the detection algorithm (without marker id calculation) takes around 70 ms to process the example image (Figure 13). For snapshots of the two videos (Figures 14 and 15) the processing time is approximately 40 ms. In videos we gain additional speed by tracking markers. For the desktop video we achieve a frame rate of about 40 fps, for the video of the window scene approximately 55 fps.

4 Conclusion

We have presented a fast and robust marker detection front end inspired by the ARTag system. By using an edge based approach we gain robustness against changing lighting conditions and occlusions. We compared our algorithm to ARToolKitPlus, a threshold based marker detection system. The experiments revealed the advantages of our algorithm over threshold based systems in cases of changing illumination and marker occlusion.

However, there is still room for future research. For example, if a marker is occluded by a very dark object this object might form a black-white edge with the bright background surrounding the marker. Thus our algorithm cannot distinguish the object's edge from the marker's edges, and so the

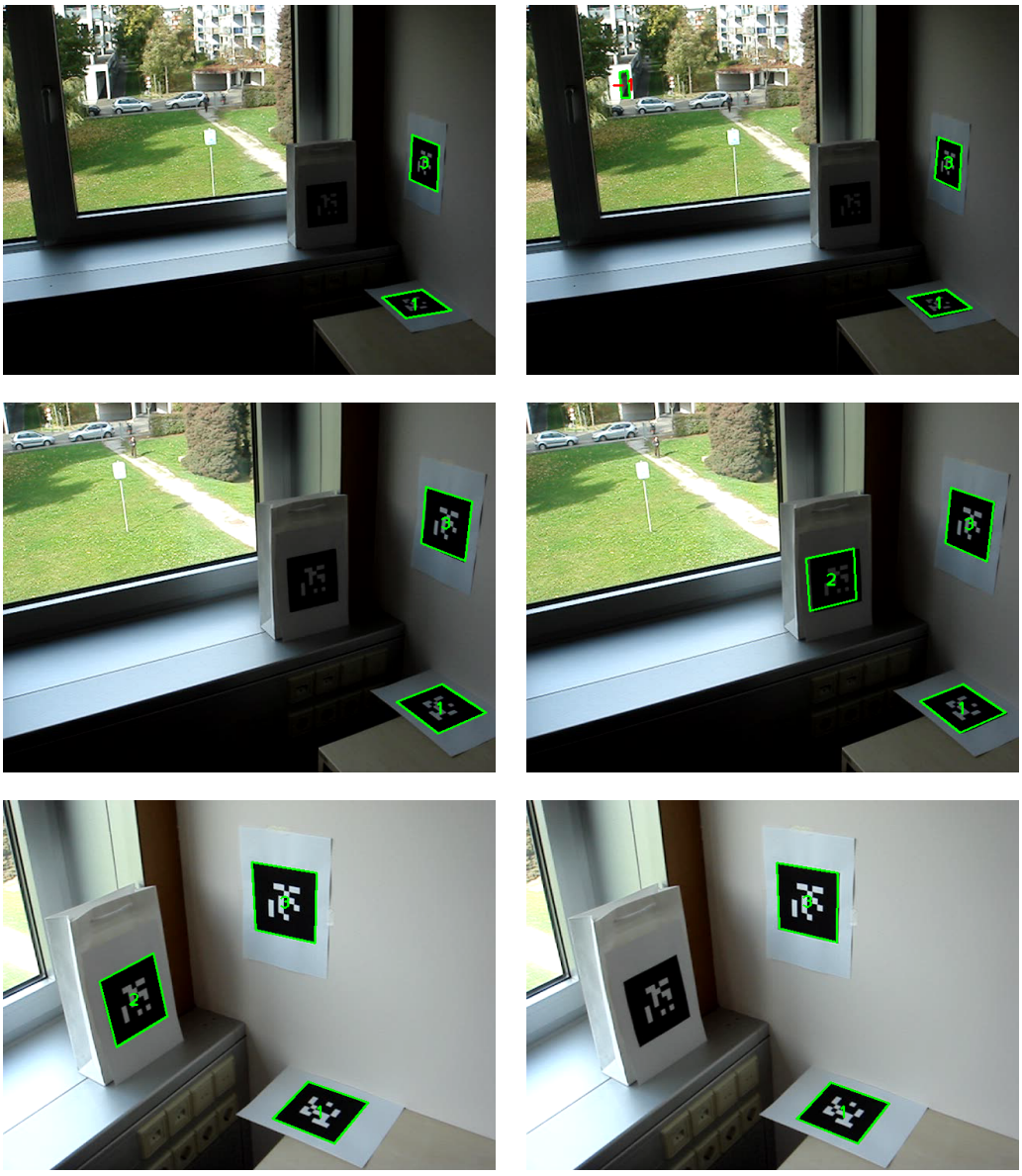


Figure 15: Three frames taken from the video of the window scene. The rows are arranged chronologically. The left column shows the markers detected by our algorithm, the right column depicts the results of ARToolKitPlus.

detection process, if it was not restricted, could find more than four corners for a single marker. But just restricting the maximum allowed number of corners per marker to four does not ensure that the four right corners are chosen. One has to examine all detected corners and choose that sequence of four corners that represents the marker best. This might be achieved by using heuristics (e.g. searched quadrangles must be convex) or the tracking history.

References

[1] M. Atiquzzaman and M. Akhtar. Complete line segment description using the hough transform. *Image and Vision Computing*, 12(5):267–273, 1994. 5

[2] J. Burns, A. Hanson, and E. Riseman. Extracting straight lines. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 8(4):425–455, 1986. 4

[3] J. Clarke, S. Carlsson, and A. Zisserman. Detecting and tracking linear features efficiently. In *Proc. British Machine Vision Conf.*, 1996. 4, 5, 7

[4] M. Fiala. Artag, a fiducial marker system using digital techniques. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2005. 1, 2, 3

[5] D. Guru, B. Shekar, and P. Nagabhushan. A simple and robust line detection algorithm based on small eigenvalue analysis. *Pattern Recognition Letters*, 25(1):1–13, 2004. 3

[6] P. Hough. Method and means for recognizing complex patterns, U.S. Patent No. 3069654, 1962. 4

[7] I. Kato and M. Billinghurst. *ARToolkit User Manual, Version 2.33.* Human Interface Technology Lab, University of Washington, 2000. 2

[8] N. Kiryati, Y. Eldar, and A. Bruckstein. A probabilistic hough transform. *Pattern Recognition*, 24(4):303–316, 1991. 5

[9] A.-R. Mansouri, A. Malowany, and M. Levine. Line detection in digital pictures: A hypothesis prediction/verification paradigm. *Computer Vision, Graphics, and Image Processing*, 40(1):95–114, 1987. 4

[10] R. Nevatia and K. Babu. Linear feature extraction and description. *Computer Vision, Graphics, and Image Processing*, 13(3):257–269, 1980. 4

[11] D. Wagner and D. Schmalstieg. Artoolkitplus for pose tracking on mobile devices. In *Proc. Computer Vision Winter Workshop*, 2007. 2, 15