Robust Global Tracking using a Seamless Structured Pattern of Dots

Lode Jorissen, Steven Maesen, Ashish Doshi, Philippe Bekaert

Hasselt University - tUL - iMinds
Expertise Centre for Digital Media - Wetenschapspark 2 - 3590 Diepenbeek, Belgium
{lode.jorissen, steven.maesen, ashish.doshi,
philippe.bekaert}@uhasselt.be

Abstract. In this paper, we present a novel optical tracking approach to accurately estimate the pose of a camera in large scene augmented reality (AR). Traditionally, larger scenes are provided with multiple markers with their own identifier and coordinate system. However, when any part of a single marker is occluded, the marker cannot be identified. Our system uses a seamless structure of dots where the world position of each dot is represented by its spatial relation to neighboring dots. By using only the dots as features, our marker can be robustly identified. We use projective invariants to estimate the global position of the features and exploit temporal coherence using optical flow. With this design, our system is more robust against occlusions. It can also give the user more freedom of movement allowing them to explore objects up close and from a distance.

Keywords: Optical Tracking, Structured Pattern, Projective Invariant, Augmented Reality

1 Introduction

Estimating the pose of a camera accurately and robustly is essential to achieve a stable augmented image. It is the basis of any Augmented Reality application, ranging from entertainment to medical visualization. In many cases, the image to be augmented is also the basis for the tracking system. These optical trackers provide an inexpensive solution to the pose estimation problem.

The majority of camera based tracking systems use markers placed in the environment. These markers need to be completely visible to the camera to identify and calculate the transformation of the camera. Most systems fail when even a tiny portion is occluded. To compensate for this, they use multiple markers to have redundant information. This poses a trade-off between the number of markers and the minimum size for the information to be read out which translates into robustness versus accuracy.

We propose to use a seamless pattern of spatial coded dots. Individual dots are identified by the spatial relation between them using projective invariant properties. As long as a minimum set of identifiable dots is visible anywhere in the image, a global pose (the absolute pose in a predefined coordinate system, e.g. defined by the marker) can be estimated. This makes our system more robust against occlusions than other marker based systems while maintaining an accurate pose estimate. Using a virtual and real test setup, we show that our tracking system reports a camera pose with an accuracy up to 1 mm and 0.06°, even with large occlusions. This results in a stable augmented image.

2 Related Work

There are different kinds of trackers, such as acoustic and mechanical trackers, that are able to estimate an object's pose. These trackers, however, are often too expensive or difficult to setup correctly. Optical trackers, on the other hand, can be made cheaply and the required cameras can be found in a wide range of devices these days.

Optical trackers can be classified into categories using different conditions. First, one can classify optical trackers as 'outside-looking-in' and 'inside-looking-out' trackers [27]. Outside-looking-in trackers use multiple calibrated cameras to determine the pose of one or more objects that move through the environment, while inside-looking-out trackers use one camera to determine its own pose with the help of the environment. Since inside-looking-out trackers are more suitable for applications such as augmented reality and virtual reality, and are more readily available than outside-looking in trackers (only one camera is needed), we will focus on this kind of optical trackers.

A second categorization is whether the tracker uses markers or natural features. Trackers can use natural features, such as corners and edges, as reference points. These trackers can be subdivided into different categories. PTAM [14], PTAMM [5] and vSLAM [12] for example build a feature map, used for tracking, at runtime. Visual Odometry [22] on the other hand only uses the information from the last few frames to determine the movement of the camera. Natural feature based trackers make it possible to determine the pose of the camera without adding markers in the environment, which means that they can work in unknown environments. These trackers, however, usually have several disadvantages. Most natural feature trackers only return a pose relative to the starting position instead of an absolute pose. This can make it difficult to align augmented reality objects with the real world. A solution to this problem could be to create a feature map first and reuse this map in subsequent executions in the same environment or to add a fiducial marker to the environment for initialization. Another disadvantage of natural feature trackers is that they often need to determine the positions of the features at run-time using only new features relative to existing features. This leads to an accumulation of errors and thus a decreased accuracy over time resulting in the estimated camera pose drifting away from the true pose. Again, this can be countered by providing a feature map of the environment, but creating an accurate map is often a time consuming and difficult task not suitable for most people. Natural feature trackers also need to detect a good amount of unique features, otherwise they will fail to determine the pose of the camera. Some control over the scene, for example adding more textured objects, can resolve this problem, but this is contrary to the purpose of natural feature tracking.

Marker-based trackers are trackers that use artificial features that are placed in the environment. These features, called markers, often contain an identification pattern, or some other method to identify them, they can serve as reference points that are used

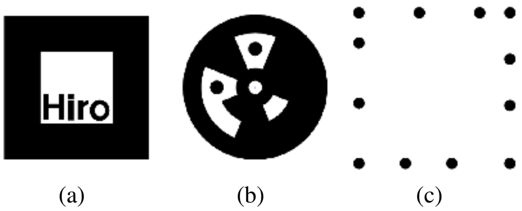


Fig. 1. A few different markers: (a) ARToolKit. (b) InterSense's marker. (c) PiTag (from [13], [19] and [2]).

to estimate the camera's extrinsic parameters. These extrinsic parameters relate world space coordinates to camera space coordinates, and thus describe the position and rotation of the camera. The markers deliver reference points or 2D-3D correspondences between the image plane and the real world that can be used to estimate the pose. Since the world position of these points is already known, the pose can often be estimated more accurate than with natural feature trackers. Markers can have different shapes, as can be seen in Figure 1, and those shapes have different advantages. Square markers, such as ARToolKit markers [13] and ARTag markers [9], usually have a thick black border, which is used to detect the marker while the inner part of the marker contains an identification tag. This tag often is an image or a binary code. Since the world-space position of the marker's four corners can easily be determined, they can be used to estimate the extrinsics of the camera. Circular markers such as TRIP [15] and InterSense's markers [19] also contain an identification tag, but the pose must be estimated using the shape of the projected circle, or with the help of multiple markers. According to Rice et al. [20] square markers can contain more data, but the pose can be calculated more accurately with the help of circular markers. Another more recent form of fiducial markers are marker fields [23]. Marker fields can be described as a grid of mutually overlapping partial markers, which allows the fiducial marker to be partially occluded.

Another possibility is to only use dots as markers and store the identification in the relation of the positions of the dots. This is the case for marker systems such as Pi-Tag [2] and Random Dot Marker [25], which both use a projective invariant, i.e. a property that does not change under a projective transformation, to identify the points. Another example is RUNE-tag, which can be seen as a hybrid between dot based markers and circular markers [1]. These dot-based trackers have the advantage that the markers take up a relatively small area in comparison to the square and circular markers. The biggest part of the area that these dot-based markers span is empty, which means that those areas often can be occluded without too much negative effects for the pose estimation algorithm. Square and circular markers on the other hand fill the area which means that even a small occluding object can already result in a failed detection or identification.

The previously mentioned trackers have a limited tracking span. Maesen et al. [17] proposed a system that uses dots, but does not store any identification information in these features. This results in a system that can be easily extended to larger areas, but as a trade-off only returns a pose relative to the starting position and not to a fixed predefined coordinate system. Maesen et al. [18] recently proposed a new system that encodes the position of the dots using De Bruijn codes and decodes the positions with the help of cross-ratios. This allows the tracker, contrary to their earlier system, to estimate the pose in relation to a predefined coordinate system. While they can support a very large tracking area accurately, their system has little redundant information to handle large occlusions. Saito et al. [21] created a system that encodes the position in the rotation and relative position of images that are placed on a plane. The result is a system that feels less artificial than other marker-based systems but is also less accurate.

Another way of increasing the tracking space is to place multiple markers in the environment, such that at least one marker is viewable from the camera. The main disadvantage of this method is that the position of each marker needs to be determined after placing it. One could also place the markers in a grid, but still needs to find a good balance between tracking distance and robustness against occlusions. Decreasing the size of the markers also decreases the tracking distance but allows more markers to be visible. Increasing the size, decreases the chance that a complete marker is visible in the frame and therefore reduces robustness against occlusions. Libraries such as ARToolKitPlus [26] and ARUco [10] allow this kind of setup. Chen et al. [6] proposed a system where they use a map of random dot markers to track the camera over a larger area. While their system is able to handle occlusions very well, it also inherits a drawback of random dot markers. Because of the exponential increase of invariant relations that the tracker can find between detected points, the computation time can increase dramatically in cases where the detector finds a lot of points.

Our proposed system of a seamless spatial encoded pattern is designed so that any part of the pattern can be used for tracking. In contrast to the state of the art approaches, our system does not require a whole marker to be visible but only multiple distributed parts. This makes it more robust for large occlusions than most other marker designs. The design is partly based on the grid of dots that Maesen et al. [17] proposed, but introduce a robust spatial coding scheme for global pose estimation. We encode the position of each line in the grid with a unique relative placement of the dots on the lines. Thanks to the cross-ratio projective invariant [7], we are able to reconstruct this placement which allows the identification of the lines. This in turn allows us to identify the position of each visible dot, which is used to give an accurate estimate of the pose of the camera. Our system is especially designed to be robust against large occlusions as can be seen in the next sections.

3 Structured Dot Marker

Our tracking pattern consists of a spatially encoded set of dots. Compared to conventional marker systems, the dots themselves do not contain any identification information. This makes them very small, easily detectable and harder to occlude than large markers. The identification of a single dot requires the spatial relationship between its neighbors. The dots are placed on a grid structure consisting of two sets of parallel lines and are uniquely spaced to create a coded pattern. Figure 2 shows part of the structure of our marker. The dots on each line are repeatedly spaced according to three distances

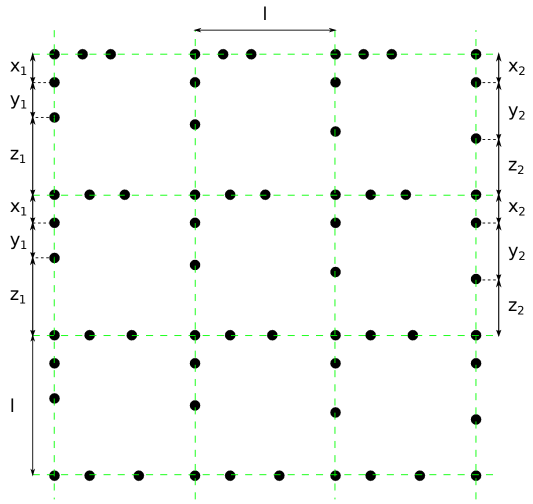


Fig. 2. The structure of our marker. Only a small part of a possible setup is shown here. Each line has a different set of distances between the points. The set (x_1, y_1, z_1) for example is not equal to the set (x_2, y_2, z_2) since both y_1 and z_1 are not equal to y_2 and z_2 respectively.

x, y and z. Each line has a unique set of distances and the sum of those distances is always equal to the line distance l, which is the distance between two subsequent parallel lines. This means that for two different lines l_1 and l_2 the corresponding sets of distances (x_1, y_1, z_1) and (x_2, y_2, z_2) are never equal to each other. Each intersection of two perpendicular lines also corresponds with a dot. By utilizing these properties it is possible to determine the world-space position of each dot, which allows us to accurately estimate the camera's true pose.

4 Tracking

As can be seen in Figure 3, the algorithm consists of four important steps: image capture, detection, identification, and pose estimation. The image capture step acquires an image from the input system. The captured image is passed on to the detection step, which consists of a dot detection step and the detection of (virtual) lines passing through the detected points. This information is then passed down to the identification step, which identifies the lines and the points lying on those lines. Using the identified points we can estimate the extrinsic parameters of the camera. In some cases where it is not possible to identify enough points, we switch to an optical flow tracking algorithm which uses information from earlier frames to track previously identified dots. The following sections describe each step in more details.

4.1 Image Capture

The first step in the algorithm acquires an image from the input device. Input can be taken from a camera, a video or a virtual scene. Because we assume that we are working

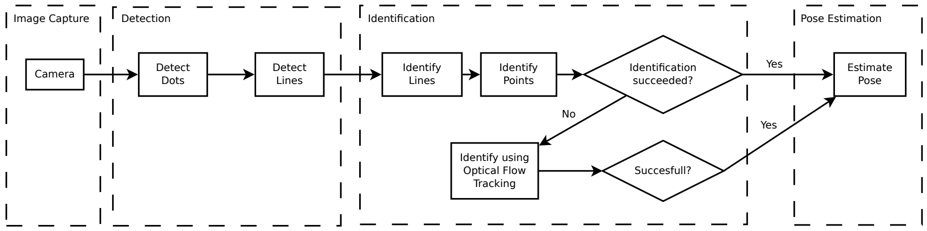


Fig. 3. Overview of our tracking algorithm. Our system consists of 4 steps, more specifically image capture, detection, identification, and pose estimation.

with a standard pinhole camera model, we need to undo the lens distortion on the image. It is more efficient to only undistort the positions of the detected points right after the dot detection step. The first option will take more computing time than the latter, but it can be more suitable for certain applications such as augmented reality, where one needs to display the input image in the background.

Both lens distortion and intrinsic camera parameters (focal length, principal point, etc.) can be calculated in a preprocessing step as they do not change when using a fixed focus camera. We do this by using a checkerboard pattern provided by the OpenCV framework [4].

4.2 Dot Detection

After acquiring the image we need to detect the dots that are visible in the image. This step consists of a thresholding algorithm which creates a binary representation of the image, and a connected component labeling algorithm which groups the pixels belonging to the same object together. We used the adaptive thresholding algorithm described by Bradley and Roth [3] to create a detection algorithm that can handle to varying lighting conditions. This makes it possible to detect dots that are hulled in shadows, or to use the tracker in most environments without the need to manually change the thresholding parameters.

As soon as we have the binary representation of our input image, we use the two-pass connected component labeling algorithm described by Di Stefano and Bulgarelli [8] to detect the pixels belonging to the same object. Objects are filtered using their size, the number of pixels, and the ratio between the number of pixels and the area of the bounding box, so that only the objects remain that are most likely a dot. The centers of gravity of the remaining objects are assumed to be the image positions of the dots. These are the points that we will try to identify. In the following step they will be used to calculate the lines passing through the points.

4.3 Line Detection

In this step, we determine the lines passing through the points. This is done by iterating over each point that is not yet part of a line, and selecting its two nearest neighbors. We then generate the two lines that pass through the selected point and each of its neighbors.

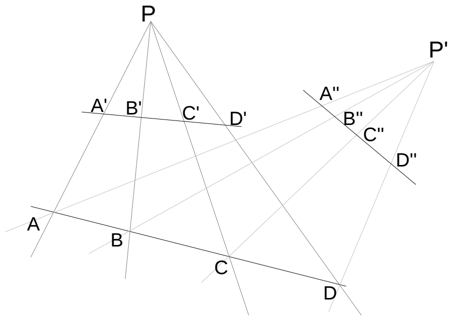


Fig. 4. Illustration of the projective invariance of the cross-ratio τ. In this example, according to the property of the cross-ratio, $\tau(A,B,C,D) = \tau(A',B',C',D') = \tau(A'',B'',C'',D'')$.

For each line we determine the detected points that lie on it: the distance from the point to the line should be smaller than a certain threshold. We update the parameters of the line for each point that lies on it to make sure that we find most of the points on the line. A line will only be passed to the line identification step when there are enough points associated with it. The line identification step will filter out falsely detected lines.

4.4 Line Identification

After detecting the lines we must identify them. We use the projective invariant cross-ratio, which, combined with the fact that the distances between the points on the same line are repeated, allows us to identify the lines.

Cross-Ratio The cross-ratio of four collinear points can be calculated as follows [7]:

$$\tau(A,B,C,D) = \frac{d(A,C)}{d(B,C)} / \frac{d(A,D)}{d(B,D)} = \frac{d(A,C)}{d(B,C)} \cdot \frac{d(B,D)}{d(A,D)} \tag{1}$$

where $d(A,B)$ is the signed distance from A to B such that $d(A,B) = -d(B,A)$. The cross-ratio is projective invariant, i.e. if we look at the projection of four collinear points on the image plane, the cross-ratio of those projections is always the same and thus it is independent of the camera's pose. This means that in Figure 4, $\tau(A,B,C,D) = \tau(A',B',C',D') = \tau(A'',B'',C'',D'')$. Calculating the cross-ratio using a different order for the points can result in a different value. In fact there are six different cross-ratio values for each set of four collinear points, but these values are all dependent on each other. If we use the same order of points or if we reverse the order, the resulting value will always be the same.

Identification It would be easy to identify the lines if we knew the true distance between the points, but it is well known that under a projective transformation the distance

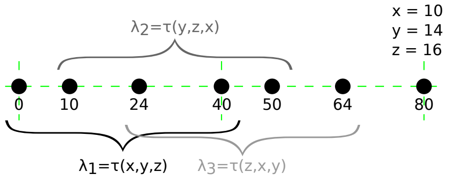


Fig. 5. For six subsequent points there are three independent cross-ratios (λ_1, λ_2, λ_3) as shown in this example.

between two points is not preserved. Since the cross-ratio of four collinear points is dependent on three variables, namely the three distances between the points, one needs to have three cross-ratios that are dependent on these distances to reconstruct the original distances. For the repeated dot pattern on a line one can find three cross-ratios for six subsequent points, as can be seen in Figure 5. These cross-ratios can be written as:

$$\lambda_1 = \tau(x,y,z) = \frac{(x+y)}{y} / \frac{x+y+z}{y+z} = \frac{(x+y)\cdot(y+z)}{y\cdot(x+y+z)} \tag{2}$$

$$\lambda_2 = \tau(y,z,x) = \frac{(y+z)}{z} / \frac{x+y+z}{z+x} = \frac{(y+z)\cdot(z+x)}{z\cdot(x+y+z)} \tag{3}$$

$$\lambda_3 = \tau(z,x,y) = \frac{(z+x)}{x} / \frac{x+y+z}{x+y} = \frac{(z+x)\cdot(x+y)}{x\cdot(x+y+z)} \tag{4}$$

Since scaling the distances with a constant gives the same cross-ratio (thus $\tau(x,y,z) = \tau(kx,ky,kz)$), we need to make sure that we find the correct solution. This can be done using the line distance, since

$$l = x+y+z \tag{5}$$

We can now rewrite equations (2), (3) and (4) as follows:

$$y = \frac{x\cdot z}{l\cdot(\lambda_1 - 1)} \tag{6}$$

$$x = \frac{y\cdot z}{l\cdot(\lambda_3 - 1)} \tag{7}$$

$$z = \frac{x\cdot y}{l\cdot(\lambda_2 - 1)} \tag{8}$$

If we replace y in (7) by (6) and z by (8) we respectively find:

$$z = l\cdot\sqrt{(\lambda_1 - 1)(\lambda_3 - 1)} \tag{9}$$

$$y = l\cdot\sqrt{(\lambda_2 - 1)(\lambda_3 - 1)} \tag{10}$$

which can be substituted in equation (7) to find x. We can now use these equations to calculate the original distances between the points. Because of measurement errors and noise, the reconstructed distances will also contain noise. We are able to cope with this by limiting the number of possible distances, i.e. rounding the calculated distances (x,y,z) towards the nearest available correct distance values $(\tilde{x},\tilde{y},\tilde{z})$.

It is a good idea to limit the number of positions on which a point can be placed by dividing the length l in intervals, and placing the dots only at those intervals. Assuming that the line distance l is set at 80mm, and every point is placed at one of the 80 different intervals: the maximum allowed error for the calculated distances is 0.5 mm. If we would only allow 40 intervals of 2 mm, the maximum allowed error would be 1 mm. This means that we can increase the robustness against noise by decreasing the number of positions on which a point can be placed at the cost of a smaller set of lines that can be used by the marker. We can achieve this by replacing l by the number of intervals between two subsequent lines and setting the distances x, y and z equal to the number of intervals between the points.

The calculated distances are used to construct a unique hash-value to identify the lines:

$$h = d_1 + d_2 \cdot l + d_3 \cdot l^2 \tag{11}$$

where d_1, d_2 and d_3 are the corresponding distances $\tilde{x}$, $\tilde{y}$ and $\tilde{z}$ sorted in ascending order:

$$(d_1, d_2, d_3) = \text{sorted}(\tilde{x},\tilde{y},\tilde{z}) \tag{12}$$

The resulting value can be used to lookup information about the line, like whether the line is a horizontal or a vertical line and the position of the line along the axis. This hash-value also adds the constraint during the generation of the setup that the sum of the distances has to be unique for each line. One should always check that the sum of the distances is equal to l before checking the hash as this will filter out most outliers.

The set of distances where $d_3 - d_1 \leq 1$ should be avoided to encode a line in the setup. The reason for this is that these distances result in cross-ratios that are almost equal to those of lines with evenly spaced dots, which turns out to be very common in our setup (for example diagonal lines passing through the intersection of perpendicular lines).

Each group of six subsequent points that is identified is stored for that line, and will be reused in the point identification stage. We ignore a line if there are conflicting identifications for it.

It might be possible that, at the end of this step, there still are some incorrectly identified lines. This is something that we have to take into consideration in the point identification step.

4.5 Point Identification

It is possible to identify the points if we have at least three correctly identified lines, where one line is perpendicular to the other two. We are able to determine at least one coordinate of the points on these lines since we know the horizontal or vertical location of the lines. This means that we only have to find the position of the point along the line.

We used a RANSAC approach to determine the positions of the detected points. First, we randomly select three identified lines, where two (E_1 and E_2) of them are parallel to each other and the other one L is perpendicular to them. This means there are two intersection points, whose positions, on the marker plane and on the image plane, we know: the intersections of E_1 and E_2 with L. We call these points the anchor points, as they help us determine the absolute world positions of the detected points. Using this information it is possible to determine the directions of the axes of the coordinate system. For example, if we know the position of E_1 along the x-axis is smaller than the position of E_2, then we know that x-axis of the coordinate system is parallel to L and that it increments in the direction of E_1 to E_2. Because of this we also know the direction of the y-axis. After determining the directions, the points on the lines are ordered in ascending order.

For each line L, E_1 and E_2 we can now determine the positions of the points by looking at each group $\{G\}$ of six subsequent points. Each group $\{G\}$ was found during the line identification step. An example of this calculation is given in Figure 6:

1. Determine the relative positions of the points in $\{G\}$ such that one point, which lies on the intersection of two lines (not necessarily the anchor points), is placed at the origin of this relative system (its position along the line is zero).
2. Use these relative world positions of the points in $\{G\}$ and their positions along the projected line to determine the homographic transformation H which maps points on the projected line to the original line. We describe a method to determine H later on.
3. Map the image position p of one of the anchor points to this relative world space using H. Call this position p'.
4. The displacement from the relative positions to the true positions is given by $a - p'$ where a is the true position of the anchor point.
5. The true displacement δ can be found by rounding of the calculated displacement to the nearest multiple of l.
6. Add δ to the relative positions of the points in $\{G\}$ to find the true world position.

We need to do these steps for each $\{G\}$ because H is too inaccurate to determine the location of the points on the lines outside of group $\{G\}$. This is also the reason why we place the relative coordinate system at an intersection: since the distance between this intersection and the selected anchor point always is a multiple of l it is possible to remove the error in the displacement $a - p'$ by rounding it to the nearest multiple of l.

The transformation H can be calculated in a similar way as the Direct Linear Transformation (DLT) for obtaining the 2D homography [11]. The matrix H is given as:

$$H = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix}$$

The linear system to be solved is:

$$Ah = 0$$

with

$$A = \begin{bmatrix} -e'_1 & -1 & e'_1 \cdot e_1 & e_1 \\ \vdots & \vdots & \vdots & \vdots \\ -e'_n & -1 & e'_n \cdot e_n & e_n \end{bmatrix}$$

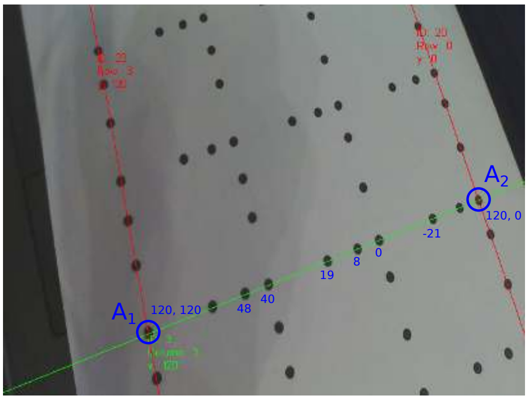


Fig. 6. The world positions of the two intersection points are determined using the lines' information. Note that this position is not yet coupled to the underlying dot. The relative positions of the points in a group is determined, and this information is used to determine the offset. Considering anchor-point A_1 the displacement equals $120-80=40$ where 120 is the true position along the line and 80 is the relative position determined using H.

where e_i are the signed distances to a reference point along the projected line, e_i' are the signed distances to this reference point along the line in world space and

$$h = \begin{bmatrix} h_{11} \; h_{12} \; h_{21} \; h_{22} \end{bmatrix}^{\mathsf{T}}$$

This equation can be solved using the Singular Value Decomposition (SVD). A position on the projected line can be mapped to the world line as follows:

$$\lambda \begin{bmatrix} e' \\ 1 \end{bmatrix} = H \begin{bmatrix} e \\ 1 \end{bmatrix}$$

After calculating the positions of the points on the three selected lines we can calculate the positions of the other points in the image by using a 2D homography that maps the points in the image to the marker plane. If an unprojected point is close enough to another point on the plane the unprojected point is assumed to be that point and the image point is assigned that position.

Because of the fact that some lines may be falsely identified it is necessary to select multiple sets of three lines and only keep the results of the best set. We can use these image-space and world-space correspondences to determine the pose of the camera as in section 4.7.

4.6 Optical Flow Tracking

Even if not enough points are detected to establish full identification, we still want to identify those points. This situation happens for example when the camera moves too close to the marker plane, in which case we are unable to find enough points to detect and identify lines. Another situation where this can happen is when the camera view is occluded, leaving only very few markers visible. By exploiting the temporal coherence between frames, we can still identify these detected dots using optical flow tracking. By keeping track of the pose of the camera in the last few frames, it is possible to estimate the new image-space position for the current frame using extrapolation. This new pose can be used to estimate the new position of the points that we identified in the previous frame. Next, we map the points detected in the new frame to the estimated locations which gives us new 2D-3D correspondences, which are used to estimate the real pose. The mapping is done by finding the nearest detected point in the new frame for each estimated position. It is possible that during the mapping step some point are falsely identified. These outliers can be detected by reprojecting the position of the dots on the image. Assuming that we have more inliers than outliers, the outliers will have a larger reprojection error than the inliers. The inliers can then be used to determine a homography between the image plane and the marker plane, so that the newly detected points can be identified.

4.7 Pose Estimation

Given the 2D-3D correspondences, we minimize the reprojection error with the help of the projection equation that projects a given 3D point (X,Y,Z) on the image plane using the estimated extrinsic parameters R and t. The reprojection error can be calculated using the sum of squared differences (SSD):

$$e = \sum_{i=1}^{N} \left((x_i - x'_i)^2 + (y_i - y'_i)^2 \right) \tag{13}$$

where N is the number of points, (x_i, y_i) is the true projection of the i$^{\text{th}}$ point and (x'_i, y'_i) is the projection of the point which is calculated from the projection equation and the estimated extrinsics:

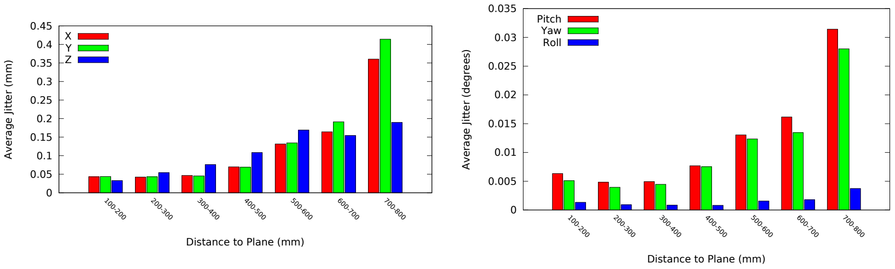
$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = K[R|t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{14}$$

where (X,Y,Z) is the 3D world point, K are the intrinsic and R and t are the extrinsic parameters of the camera. These extrinsic parameters relate world coordinates to a camera coordinate system where the camera is placed at the origin.

To solve this problem, we first estimate the homography between the image plane and the marker plane [11]. This method gives us a good initial estimate for the camera's pose. We can use this to initialize the Levenbergh-Marquardt algorithm which refines the pose iteratively by minimizing Equation (13).

5 Results

Our tracking system is implemented using the OpenCV framework [4]. We used the Levenbergh-Marquardt implementation of Lourakis [16] the estimate the pose. Both a virtual and real world setup are constructed to test the accuracy of the algorithm as well



(a) (b)

Fig. 7. Both the accuracy of the position (a) and the rotation (b) decrease if the distance to the marker increases.

as the influence of the different parameters of the setup on the accuracy. We also looked at the processing time and the robustness of the tracker.

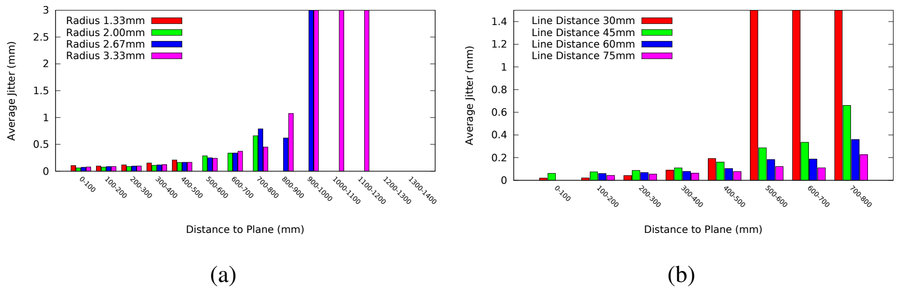
5.1 Virtual Setup

The virtual setup consisted of an OpenGL scene that contained our marker. This allowed us to compare the calculated position and orientation against the ground truth. We used a reference setup to compare the influence of the different parameters. This reference setup had a line distance of 45 mm, and the dots had a radius of 2 mm. The setup consisted of 10 horizontal and 10 vertical lines. The segments between two adjacent lines were split up into 40 intervals. Dots were placed on the lines with a minimum distance of 8 intervals between each other to prevent overlapping dots. The camera had a resolution of 640×480 pixels and a horizontal field of view of $75°$.

We first tested the accuracy of our tracker in the given setup. As we can see in Figures 7(a) and 7(b), the accuracy decreases if the distance to the plane increases. The maximum distance to the marker was approximately 70cm. Beyond 70cm, failure rate increases due to the inability to detect the dots. During the tests with this setup the average deviation was about 0.1 mm for the position and $0.01°$ for the rotation.

5.2 Real World Setup

We also tested the tracker in real world conditions. We used a low-cost camera with a resolution of 640×480 pixels and a horizontal field of view of $53°$. The camera was placed on a tripod to determine the jitter. We used different poses for the camera and did this calculation for different setups. From this information we found that increasing the dot radius decreases the jitter for poses closer to the marker plane, but increases the jitter for poses further away from the plane. This can be explained by the fact that each dot now fills a larger area on the image plane, which reduces the influence of the noise in the image, and thus result in a more stable pose when the camera is closer to the marker. This also means that the distance between two points becomes smaller,



(a) (b)

Fig. 8. Influence of (a) the radius of the dots and (b) the line distance on the tracking distance and the accuracy.

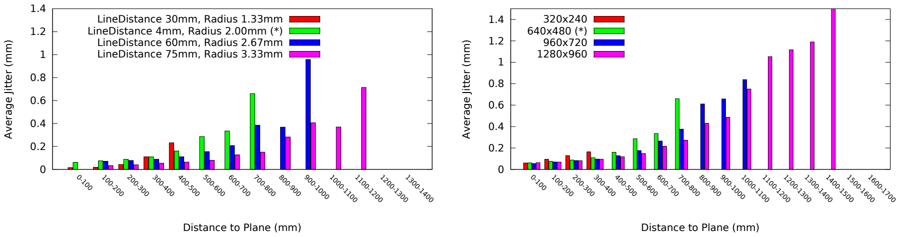
which will lead to points blurring into each other when the camera is situated at a greater distance to the plane, which results in an increased jitter. Increasing the number of intervals or decreasing the line distance also increased the jitter since the maximum allowed deviation for the calculated distances in the line identification step decreased. The average jitter for a typical desktop setup was about 1 mm for the position, and 0.06° for the rotation. From these results we can conclude that both noise and blur effects influence the accuracy of the tracker.

5.3 Increasing Tracking Distance

It turns out that there are two main possibilities to increase the tracking distance: increase both the line distance and the radius of the dots simultaneously, or increase the camera resolution. As we can see in Figure 8(a), simply increasing the radius already increases the tracking distance, but the accuracy decrements dramatically at a certain point. Increasing only the line distance doesn't increase the tracking distance, but it does increase the accuracy as one can see in Figure 8(b). This can be explained by the fact that, at a certain distance, the distance between the projected points becomes too small to give an accurate result. Increasing both the line distance and the radius of the dots will thus increase the tracking distance while maintaining a decent accuracy, as can be seen in Figure 9(a). Increasing the camera resolution results in a similar behavior as can be seen in Figure 9(b). Another method to increase the tracking distance is to decrease the field of view, but this might be impractical in the real world.

5.4 Robustness

Next, we examined the influence of the optical flow tracking algorithm on the number of frames for which a pose could be determined. On a prerecorded path existing of 1758 frames the number of failed identifications dropped from 81 frames to six frames when we enabled the optical flow tracking algorithm. Figure 10(a) visualizes the decrease of failures when optical flow tracking is enabled. Similar results were obtained for other paths, which clearly shows that the optical flow tracking algorithm increases the robustness of the tracker.



(a) (b)

Fig. 9. (a) Influence of both the line distance and the radius on the tracking distance and the accuracy. (b) Increasing the camera resolution increases both the accuracy and the tracking distance.

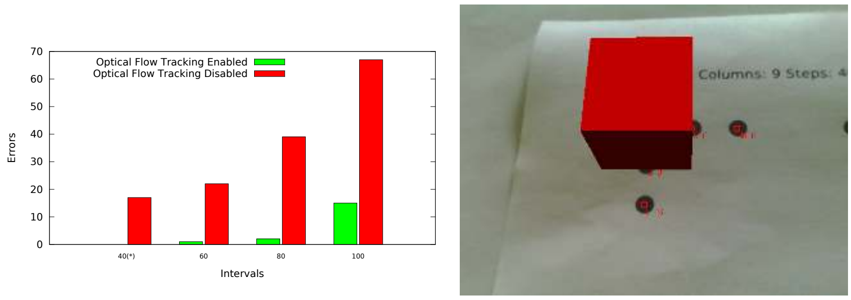
Using optical flow tracking, it is possible to estimate the pose with at least four non-collinear points, which makes it possible to track the pose in more unusual cases. This allows us, for example, to move the camera close to the marker plane, as can be seen in Figure 10(b).

The small area of the points compared to the area of the plane on which they are placed and the redundancy of the encoded positions already allows us to estimate the pose in cases where big parts of the setup are occluded. It can happen that the occluding object makes it impossible to identify at least three lines, in which case the tracker can again rely on the optical flow tracking step. This means that both marker design and the optical flow tracking algorithm helped to create a tracking system that is more robust against occlusions than most other marker based trackers. Examples of this are given in Figure 11.

Since we use an adaptive thresholding algorithm to create the binary image in the dot detection step, the tracker is able to detect dots under varying lighting conditions. This allows the usage of the tracker in an environment where the lighting is not controlled.

Intervals	Offset (intervals)	Number of Lines
20	1	31
20	4	8
40	1	132
40	4	79
40	8	29
60	8	126
60	12	60
80	1	531
80	8	288
80	16	100

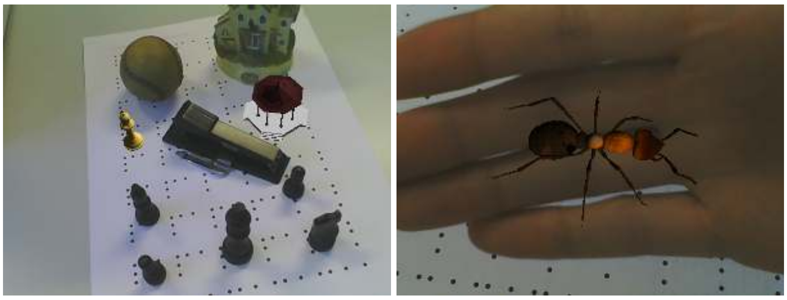
Table 1. The number of available lines is dependent on the number of intervals in which the line distance is split and the minimum offset between the points.



(a)

(b)

Fig. 10. (a) Optical flow tracking can help to determine the pose for an image when insufficient dots are visible for full identification. This histogram shows the number of frames where the pose could not be detected in function of the number of intervals, while the camera was moving away from the marker. The path consisted of 661 frames. (b)



(a)

(b)

Fig. 11. Even when big parts of the marker are occluded, it is still possible to determine the camera's pose.

5.5 Number of Lines

One of the factors that determines the area that can be tracked is the number of available lines, since they define the height and the width of the marker's plane. The number of lines is dependent on the number of intervals we divide the line distance l in, and the minimum number of intervals between two adjacent points. Table 1 gives the number of available lines for a given number of intervals and a minimum distance. One should be careful when choosing these values, since an increase in the number of intervals may lead to a decreased number of successful identifications as can be seen in Figure 10(a). These failures can be limited with the help of optical flow tracking. A decrease of the minimum distance between the points may result in overlapping points or may increase the negative influence that motion blur has on the detection step: if the distance is too small two nearby dots may blur into each other, and may thus be detected as just one point. This can of course be mitigated by decreasing the radius of the dots, but this will also lead to a decreased tracking distance.

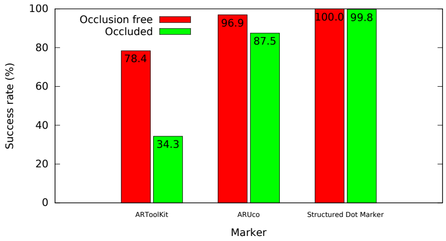


Fig. 12. A comparison of the success rate of marker identification and tracking with and without occlusions.

5.6 Processing Time

On our test system [1] the processing time for each frame, with a resolution of 640x480 pixels, was on average 13ms or 77 frames per second. When a very large part of the pattern becomes visible, the processing time could go up to 24ms. In our current implementation the dot detection step was approximately 6 ms, while the line detection and the line identification step was 0.3 ms and 0.6 ms respectively. Identifying the individual dots took on average 6 ms, including the optical flow algorithm when not enough lines are identified. The processing time of the dot detection is highly dependent on the resolution of the images, which means that it will increase if the resolution increases. The dot identification step on the other hand seems to be dependent on the number of selected groups of three lines.

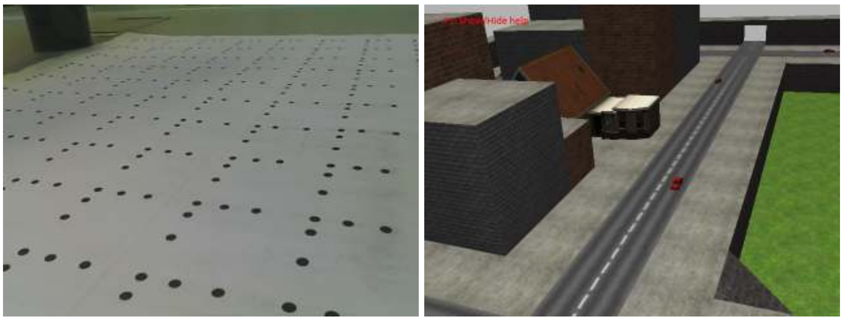
5.7 Comparative Analysis

In this section, performance comparisons were made between Structured Dot Markers (SDM) and two well known trackers. The first tracker was ARToolkit [13] with its default multimarker setup and 65mm wide markers. The other tested tracker was ARUco [10]. Its setup was a 6×8 marker grid and 35mm wide markers. The SDM grid setup consisted of 9 rows and 6 columns, with a line spacing of 45mm. Each dot had a radius of 2mm.

The first experiment relates to performance analysis without occlusions. For each tracker, 1000 frames were captured whilst the camera was moved along a set path. The experimental results are presented (red columns) in Figure 12. The green columns relate to the second experiment of performance analysis with occlusions.

Both ARToolKit and ARUco had reasonable success rates of 78.4% and 96.9% respectively, for the first experiment and lowered successes for the second experiment, with ARToolKit only recording a 34.3% success rate. In comparison, SDM had a 100% success rate for marker identification without occlusion and a fairly high success rate of 99% with occlusion. Though the performance results were surprising, it was not entirely unexpected that SDM would outperform ARToolKit and ARUco. As outlined

[1] Desktop workstation with 2.80GHz Intel Core i7 CPU



(a) (b)

Fig. 13. (a) The image taken by the camera. (b) A screenshot of a VRPN client using the calculated pose.

earlier, this is because our marker design is organized in a structured grid, with easily recognized features (dots) and the ability to identify individual lines. However, our tracking system is susceptible to abrupt movements that cause the optical flow algorithm to fail, thereby needing to reinitialize or when the camera is hovering too close over the marker. It is also susceptible to large motion blurs, as with any other tracker in the field.

6 Applications

We implemented both an augmented reality and a virtual reality application to show how well it can perform in a real world situation. For example, in Figure 11(a) we can see that our tracking solution can still produce a stable augmented image, even with most of the dots occluded by the hand of the user. When the camera moves closer to the pattern, the optical flow algorithm makes sure that tracking can still be continued, as shown in Figure 10(b). Our system can also cope with objects being placed on top of our pattern. Figure 11(b) shows that it has little effect on the tracking stability and accuracy. We would like to refer to our accompanying video for more results demonstrating the use of AR with our tracking system.

We added the ability to start a Virtual-Reality Peripheral Network (VRPN) server [24] to our system. This allows (already existing) virtual reality applications that implement the VRPN protocol to connect to our tracker. This means that the tracker can easily be integrated into other applications. An example of this is shown in Figure 13.

The supplementary video[2] shows these applications in action.

7 Conclusion

We proposed a novel tracking approach using a seamless structured pattern of dots that can estimate the global position and orientation of a camera accurately and robustly.

[2] `http://research.edm.uhasselt.be/\%7Esmaesen/pubs/AVR14/StructuredDotMarker.mp4`

The tracker works over relatively large areas without relying on individual identifiable markers as used in related systems. Instead our dots are encoded using the spatial relationship with its neighbors which can be identified using projective invariants. We can also exploit the temporal coherence between frames to establish a robust tracking result in cases where other systems fail. We have shown that our system is able to handle large occlusions, either by objects placed on top or by dynamic occlusions of users. This makes it very useful in Mixed or Augmented Reality applications where occlusions are frequent.

We have conducted extensive tests in a virtual scene to identify the major parameters that contribute to the accuracy of our system. Using the ground truth of the tracking data in a virtual scene, we established that we could achieve a global accuracy of 0.1 mm and 0.01°. In our real world setup, we used a standard low-cost camera with a resolution of 640x480 to test the accuracy in a typical desktop AR scene. These cameras suffer from measurement noise, motion blur and lens distortion. Under these conditions, the average jitter of our system was measured to be 1 mm for the position and 0.06° for the rotation of the estimated camera pose. This gives a stable augmented image, even under large occlusions, as shown in our results. Compared with other marker based tracking systems, we showed that our system outperformed ARToolkit and ARUco at handling occlusions. Our tracking system can process camera images at a rate of 77 frames per second (average of 13 ms per frame).

Further work is planned to include hierarchical based structured dots, optimization for mobile devices and expanding our system for more diverse applications.

References

1. F. Bergamasco, A. Albarelli, E. Rodola, and A. Torsello. Rune-tag: A high accuracy fiducial marker with strong occlusion resilience. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 113–120, June 2011.
2. F. Bergamasco, A. Albarelli, and A. Torsello. Pi-Tag: a fast image-space marker design based on projective invariants. *Machine Vision and Applications*, 24(6):1295–1310, 2013.
3. D. Bradley and G. Roth. Adaptive Thresholding using the Integral Image. *Journal of Graphics, GPU, and Game Tools*, 12(2):13–21, 2007.
4. G. Bradski. *Dr. Dobb's Journal of Software Tools*.
5. R. O. Castle, G. Klein, and D. W. Murray. Video-rate Localization in Multiple Maps for Wearable Augmented Reality. In *Proc 12th IEEE Int Symp on Wearable Computers, Pittsburgh PA, Sept 28 - Oct 1, 2008*, pages 15–22, 2008.
6. L. Chen, H. Fu, W. Ho, A. Li, and C. lan Tai. Scalable Maps of Random Dots for Middle-scale Locative Mobile Games, 2013.
7. R. Courant and H. Robbins. *What is Mathematics?* Oxford University Press, 1996.
8. L. Di Stefano and A. Bulgarelli. A simple and efficient connected components labeling algorithm. In *Proceedings of the International Conference on Image Analysis and Processing.*, pages 322–327, 1999.
9. M. Fiala. ARTag revision 1, a fiducial marker system using digital techniques. Technical Report NRC 47419/ERB-1117, Nov. 2004.
10. S. Garrido-Jurado, R. Muoz-Salinas, F. Madrid-Cuevas, and M. Marn-Jimnez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280 – 2292, 2014.

11. R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004.
12. N. Karlsson, E. Di Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. Munich. The vSLAM Algorithm for Robust Localization and Mapping. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA 2005).*, pages 24–29, 2005.
13. D. H. Kato. ARToolKit. `http://www.hitl.washington.edu/artoolkit/`, 1999.
14. G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.
15. D. López de Ipiña, P. R. S. Mendonça, and A. Hopper. TRIP: A Low-Cost Vision-Based Location System for Ubiquitous Computing. *Personal Ubiquitous Comput.*, 6(3):206–219, Jan. 2002.
16. M. Lourakis. levmar: Levenberg-Marquardt nonlinear least squares algorithms in C/C++. `http://www.ics.forth.gr/~lourakis/levmar/`, Jul. 2004. [Accessed on 31 Jan. 2005.].
17. S. Maesen and P. Bekaert. Low-Cost, Wide-Area Tracking for Virtual Environments. In *IEEE VR 2007 Workshop "Trends and Issues in Tracking for Virtual Environments"*, pages 16–21, Charlotte, NC, USA, March 2007. IEEE, Shaker Verlag, Aachen, Germany.
18. S. Maesen, P. Goorts, and P. Bekaert. Scalable optical tracking for navigating large virtual environments using spatially encoded markers. In *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology*, VRST '13, pages 101–110, New York, NY, USA, 2013. ACM.
19. L. Naimark and E. Foxlin. Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker. In *Proceedings of the 1st International Symposium on Mixed and Augmented Reality*, ISMAR '02, pages 27–, Washington, DC, USA, 2002. IEEE Computer Society.
20. A. C. Rice, R. K. Harle, and A. R. Beresford. Analysing fundamental properties of marker-based vision system designs, 2006.
21. S. Saito, A. Hiyama, T. Tanikawa, and M. Hirose. Indoor Marker-based Localization Using Coded Seamless Pattern for Interior Decoration. In *Virtual Reality Conference, 2007. VR '07. IEEE*, pages 67–74, 2007.
22. D. Scaramuzza and F. Fraundorfer. Visual odometry [tutorial]. *Robotics Automation Magazine, IEEE*, 18(4):80–92, 2011.
23. I. Szentandrási, M. Zachariáš, J. Havel, A. Herout, M. Dubská, and R. Kajan. Uniform Marker Fields: Camera localization by orientable De Bruijn tori. In *Proceedings of ISMAR*, 2012.
24. R. M. Taylor, II, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser. VRPN: a device-independent, network-transparent VR peripheral system. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '01, pages 55–61, New York, NY, USA, 2001. ACM.
25. H. Uchiyama and H. Saito. Random dot markers. In *Proceedings of the 2011 IEEE Virtual Reality Conference*, VR '11, pages 271–272, Washington, DC, USA, 2011. IEEE Computer Society.
26. D. Wagner and D. Schmalstieg. ARToolKitPlus for Pose Tracking on Mobile Devices, 2007.
27. G. Welch, G. Bishop, L. Vicci, R. Vicci, S. Brumback, K. Keller, and M. Place. High-Performance Wide-Area Optical Tracking: The HiBall Tracking System. pages 1–21. MIT Press, 2001.