Automatic Camera Pose Initialization, using Scale, Rotation and Luminance Invariant Natural Feature Tracking

Rafael Bastos
ADETTI/ISCTE
Av. Forças Armadas
Edifício ISCTE
1500, Lisbon, Portugal
Rafael.Afonso.Bastos@gmail.com

Miguel Sales Dias
Microsoft/ISCTE
Av. Forças Armadas
Edifício ISCTE
1500, Lisbon, Portugal
Miguel.Dias@microsoft.com

ABSTRACT

The solution to the camera registration and tracking problem serves Augmented Reality, in order to provide an enhancement to the user's cognitive perception of the real world and his/her situational awareness. By analyzing the five most representative tracking and feature detection techniques, we have concluded that the Camera Pose Initialization (CPI) problem, a relevant sub-problem in the overall camera tracking problem, is still far from being solved using straightforward and non-intrusive methods. The assessed techniques often use user inputs (i.e. mouse clicking) or auxiliary artifacts (i.e. fiducial markers) to solve the CPI problem. This paper presents a novel approach to real-time scale, rotation and luminance invariant natural feature tracking, in order to solve the CPI problem using totally automatic procedures. The technique is applicable for the case of planar objects with arbitrary topologies and natural textures, and can be used in Augmented Reality. We also present a heuristic method for feature clustering, which has revealed to be efficient and reliable. The presented work uses this novel feature detection technique as a baseline for a real-time and robust planar texture tracking algorithm, which combines optical flow, backprojection and template matching techniques. The paper presents also performance and precision results of the proposed technique.

Keywords

Camera Pose Initialization, Feature Detection and Tracking, Augmented Reality, Texture Tracking, scale invariant, rotation invariant, luminance invariant.

1. INTRODUCTION

The Camera Pose Initialization (CPI) problem has been a research topic of considerable interest and constant growth in the areas of augmented reality and automatic panoramic images generation. This issue can also be defined as camera calibration problem, where the goal is to compute the intrinsic and extrinsic parameters of the real camera, aiming object registration or user tracking applications. There are a variety of different methods to accomplish this goal, with the first ones being introduced in 1992 by Caudel and Mitzell [Cau92]. We can find among these tracking methods, techniques based on circular or square fiducial markers [Art07], colored objects segmentation [Dia04] [Din04] and natural feature extraction [Kat03] [Yua06] [Che06].

Vision-based tracking systems have been using information related to the acquisition and identification of simple geometric primitives in the scenes, such as planes [Sim02] or even a combination of different techniques [Mar02]. The proliferation of vision-based tracking techniques is due to the fact that they work well in real time and are not expensive, since there is only one main cost involved: the processor's cost.

We propose a novel and automatic approach to the CPI problem, based on scale, rotation and luminance invariant natural feature extraction and tracking. This method operates without the need of any kind of extra information, like fiducial markers [Art07] [Kat03], to compute the CPI. The feature matching procedure has been optimized using a heuristic clustering algorithm, which has revealed to be efficient and reliable.

As a test case for evaluating our proposed feature extracting and matching method, we have developed

a texture tracking algorithm. Our texture tracking algorithm combines known methods such as template matching, homography computation, and texture reconstruction by back projection and optical flow computation. The algorithm is completely automated and produces real-time efficient tracking.

This paper is organized as follows. After the Introduction of section 1, we present some related work (section 2), followed by the presentation of our feature extraction technique (section 3). In section 4, we detail our heuristic feature clustering algorithm and, in Section 5, we describe our texture tracking algorithm as a test case for our proposed feature extraction and matching method. Finally, in section 6, we draw some conclusions and describe some future work.

2. RELATED WORK

We have assessed five representative techniques from the literature, which are based on feature tracking and that use planar object topologies as in our texture tracking test case: [Kat99], [Sim02], [Bue02], [Mai02] and [Kat03]. From this assessment, we have concluded that the majority of these systems include an offline stage to spear processing resource for online tracking. Another popular paradigm is the need of user assistance to initialize or to preprocess the tracking object. The assessed pose extraction methods (DLT [Abd71] and POSIT [Dem91]) have shown to be quite robust and efficient for real-time Augmented Reality applications. In combination with these methods, most of the presented systems apply the RANSAC [Fis81] algorithm to identify outlier features. One of the identified problems in these systems was the unsuitability of the tracking techniques for real-time purposes, since only [Sim02] and [Kat03] have proven to work in real-time (more than 25 fps). The lack of a robust and fast matching technique invariant to rotation and other affine transformations was another common identified problem. In the presence of shadows, noise or fast rotation camera/object movements the systems tend to fail tracking or to induce extreme jitter. Another common problem found was the excessive use of binary fiducial markers to accomplish calibration and tracking routines, instead of using natural elements in the real-scene or tracking object. We have concluded from this assessment that the CPI problem is still far from being solved, unless new real-time CPI methods are developed. A solution for the real-time CPI problem is the use of scale and rotation invariant features.

There are a variety of popular methods for scale and rotation invariant feature extraction, namely SURF [Bay06], SIFT [Low03], and an extension of the later, PCA-SIFT [Ke04]. Although these methods have proven to be robust and to yield good distinctive power, the lack of suitability for a real-time application is still an issue. For example, if we use an image with a resolution of 800x640 pixels, the faster method (SURF) takes 255 milliseconds to compute and extract the image features. For a real-time application, this computation time is very expensive, since we would spend ¼ of a second only for feature extraction, without taking into consideration feature matching algorithms. In this work, the challenge was to design and develop a robust scale, rotation and luminance invariant method for real-time applications.

3. FEATURE EXTRACTION

In this section we will describe our feature extraction and matching algorithm. Features are extracted using minimum eigen values [Shi94], and are made scale, rotation and brightness invariant using straightforward and real-time computer vision techniques. At the end of each section, there will be a report about performance results, using a PentiumIV 2.66GHz.

Minimum Eigen Values (MEV)

To evaluate the MEV of an image, we convert the original image to grayscale, and a block of 3x3 pixels is taken at every image position and first derivatives are computed using Sobel Operators O_x and O_y for convolution:

$$O_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad O_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{1}$$

The convolution will result in the first derivative in direction of x (D_x) and the first derivative in direction of y (D_y). We construct matrix C, where the sum is in respect to all components of the 3x3 block:

$$C = \begin{bmatrix} \sum D_x^2 & \sum D_x D_y \\ \sum D_x D_y & \sum D_y^2 \end{bmatrix} \tag{2}$$

We can solve the Eigen Values for this matrix by computing:

$$\det(C - \lambda I) = 0 \tag{3}$$

where I is the identity matrix and λ the column vector of Eigen Values. The solutions may be written as:

$$\lambda = \frac{\sum D_x^2 + \sum D_y^2 \pm \sqrt{\left(\Sigma D_x^2 + \Sigma D_y^2\right)^2 - 4\left(\Sigma D_x^2 \Sigma D_y^2 - \left(\Sigma D_x D_y\right)^2\right)}}{2} \tag{4}$$

This equation will result in two solutions: λ_1 and λ_2. The minimum λ is called the Minimum Eigen Value and must be the one to be taken in consideration according to Shi [Shi94]. For numerical stability reasons, we use Singular Value Decomposition (SVD)

[Gol93] to solve the equation. We perform feature selection by applying a threshold to the resulting MEV. For that, we have selected a threshold t value of 1% of the global maximum in the current minimum eigen values spectrum, and only features that satisfy this condition are selected. We can see an example of this technique depicted in Figure 1, and a performance summary in Table 1.

Figure 1. Prague Castle Scene (800x600, t=1%, 1243 features).

800x600	640x480	480x360	320x240
8.506 ms	5.450 ms	2.970 ms	1.469 ms

Table 1. MEV Computation time varying the input image resolution.

Scale Invariance

To make features scale invariant, we rely on a basic assumption, that is: every feature has its own intrinsic scale factor. Our challenge was to find a mechanism that could determine the intrinsic scale factor of a feature, based on simple computer vision operations, retaining the real-time requirements. If we look at an image after applying a Sobel filter (Equation 1), we can see that the edge length of the resulting derivatives is directly correlated with the zooming distance (see Figure 2).

Figure 2. Edge length of vertical and horizontal derivatives, varying zoom distance.

As the zooming distance gets larger, the thinner the derivatives will get and vice-versa. Our goal is to find the main edge length of the derivatives to compute an intrinsic scale factor. This scale factor will be used to determine the intrinsic feature patch dimension. The intrinsic feature patch will then be rescaled, in order to normalize it and make it scale invariant.

We start by normalizing the results (giving N_x and N_y) of the Sobel operators O_x and O_y (Equation 1) convolution:

$$N_x(x,y) = \frac{D_x(x,y) - \min(D)}{\max(D) - \min(D)} \qquad (5)$$
$$N_y(x,y) = \frac{D_y(x,y) - \min(D)}{\max(D) - \min(D)}$$

Where, the *min* function determines the minimum value for the specified patch. These normalized results (N_x and N_y) are threshold using a value of 0.5, resulting in 2 binary images (B_x and B_y). For each row (in the case of B_y) and for each column (B_x), we find the number of consecutive connected components by accumulating the number of occurrences of a determined connection value on a 1D edge length histogram vector $T(B)$. For B_x we will accumulate column connection values in $T(B)$, and for B_y we will accumulate row connection values. The value $max(T)$, at vector position v will be the global edge length maximum. Instead of using $T[v]$ directly as the intrinsic scale factor s, we can smooth the result by applying a parabolic interpolation, since s will be the local maximum:

$$a = \frac{T[v-1] + T[v+1] - 2T[v]}{2} \qquad (6)$$
$$b = \frac{(T[v-1] - T[v]) / a - 1}{2}$$
$$c = T[v] - b^2 \times a$$
$$s = c + b$$

Since other derivatives may appear inside when zooming in or zooming out, we only apply this procedure to a surrounding area of $7x7$ pixels of the feature center.

Figure 3. Scale invariant algorithm. Green dashes square: expanded area by s=2.2 (55x55). Pink crosses square: original feature patch (n=25, 25x25). White square (center): processed area (7x7). The right image represents the rescaled final patch.

Finally, assuming these computations are applied to a feature F, centered at (x_c, y_c), with a square size of nxn, where n is the starting feature size; instead of using this feature area, we will expand it to $(n.s)$x$(n.s)$ around (x_c, y_c), and rescale it again to nxn. We exemplify this procedure in Figure 3. Performance tests show that this operation (scale factor computation

and rescaling), takes about 0.012 milliseconds per feature, since derivatives were already pre-computed in the previous step.

Rotation Invariance

Assuming the feature's data is the $n \times n$ grayscale image patch (g_i) centered at (c_x, c_y), which is already scale invariant, the feature's information is extracted in a rotation invariant manner. For this purpose we have designed a function $\theta(g_i)$ which finds the main orientation angle of the feature g_i, in the form:

$$\theta(g_i) = b \max\left(\mathrm{H}(g_i)\right) \qquad (7)$$

In this equation, *max* corresponds to the function which determines the vector index of $H(g_i)$ which contains the highest value of the orientation of g_i, that is, the main orientation of feature g_i. The $H(g_i)$ function computes the orientation histogram (a vector) of a given grayscale feature g_i. This histogram vector is composed by b elements (b is the total number of histogram bins), where each element corresponds to a $360º/b$ degrees interval. We can define an indexing function $\kappa(g_i, x, y)$ for the histogram vector $H(g_i)$ as:

$$\kappa(g_i, x, y) = \arctan\left(\frac{\partial(g_i(x,y))/\partial y}{\partial(g_i(x,y))/\partial x}\right) \cdot \frac{b}{360º} \qquad (8)$$

The $H(g_i)$ histogram vector at index $\kappa(g_i, x, y)$ accumulates in the following manner:

$$\mathrm{H}(g_i)[\kappa(g_i,x,y)] \; += \sqrt{\left(\frac{\partial(g_i(x,y))}{\partial x}\right)^2 + \left(\frac{\partial(g_i(x,y))}{\partial y}\right)^2} \qquad (9)$$

After finding $\theta(g_i)$ – the grayscale patch main orientation – we create the final rotation invariant feature ($\boldsymbol{g_r}$), which can be found by performing a simple off-centered (c_x, c_y) 2D rotation of $\theta(g_i)$ degrees to the $\boldsymbol{g_i}$ grayscale patch.

Figure 4. Rotation invariant algorithm (Left: $\boldsymbol{g_i}$ patch ($n=25$, $\theta(g_i)=288º$); Right: $\boldsymbol{g_r}$ patch)

Irrespective to the orientation of feature $\boldsymbol{g_i}$, the feature $\boldsymbol{g_r}$, is the version of the original always oriented towards the patch main direction (see Figure 4). Performance tests show that this computation (patch rotation), takes about 0.019 milliseconds per feature, with $n = 15$ (see Table 2).

n=15	*n=25*	*n=35*	*n=45*
0.019 ms	0.034 ms	0.061 ms	0.094 ms

Table 2. Rotated patch computation time, varying the patch size.

Luminance Invariance

Given two scale and rotation invariant features, feature matching is accomplished using a template matching technique which is luminance invariant [Bas05] and uses the invariant image grayscale templates. This technique uses the image average and standard deviation to obtain a normalized cross correlation (NCC) value between features. For two feature patches (I and P), we compute their mean value (μ_I and μ_P) and their standard deviation (σ_I and σ_P), allowing us to find the correlation factor ρ using the following equations:

$$\mu_I = \frac{1}{xy}\sum_x\sum_y I(x,y) \quad \sigma_I = \left(\sum_x\sum_y (I(x,y)-\mu_I)^2\right)^{\frac{1}{2}} \qquad (10)$$

$$\mu_P = \frac{1}{xy}\sum_x\sum_y P(x,y) \quad \sigma_P = \left(\sum_x\sum_y (P(x,y)-\mu_P)^2\right)^{\frac{1}{2}}$$

$$\rho = \frac{\sum_x\sum_y (I(x,y)-\mu_I)(P(x,y)-\mu_P)}{\sigma_I \sigma_P} \qquad (11)$$

A value above 0.7 (70%) is a satisfying correlation factor. We use a circular feature mask to improve feature correlation matching, since pixels near to the centre tend to be more similar than the farther ones. This template matching procedure is less sensitive to small variations of scaling and rotation. Performance tests have show that each template match operation time, varying the patch size (n), consumes the following processor times: 0.002 ms ($n=15$), 0.003 ms ($n=25$), 0.005 ms ($n=35$) and 0.008 ms ($n=45$).

4. FEATURE CLUSTERING

To enable efficient feature matching, the features database is organized in clusters, each one aggregating the corresponding possible features. Our heuristic method states that these clusters have a binary identification value (a kind of simple and efficient feature signature), that is obtained by evaluating certain regions of the feature patch in relation to its average. By dividing the feature patch into 8 different regions (left, right, top, bottom, top-right diagonal, down-left diagonal, top-left diagonal and bottom-right diagonal), and by comparing these areas' average pixel value with the feature patch global average value, we obtain an 8 digit binary result. For each one of these areas we obtain a 0 value if the region average is smaller than the global average, otherwise we obtain a value of 1. For the sake of clarity, we exemplify this procedure in Figure 5.

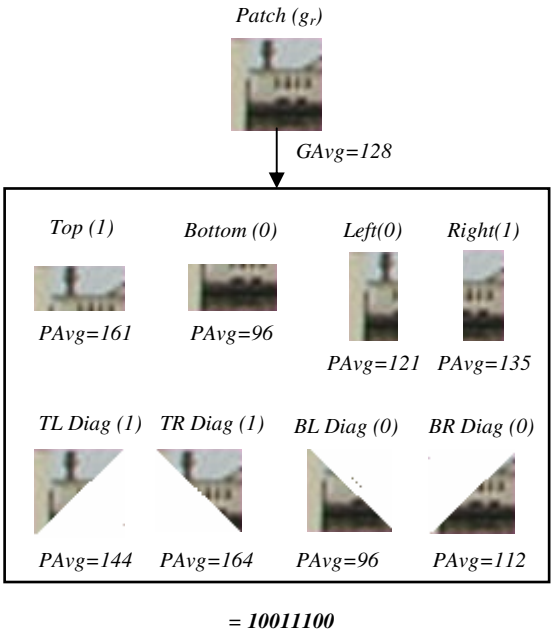

Figure 5. Binary identifier creation example.

(At the top we have the patch that is being clustered; at the bottom we have all 8 regions that form the binary identifier)

When a feature patch is processed and created, this evaluation is performed, and this feature is inserted in the corresponding cluster using the obtained binary identification. When matching a feature, we also compute the binary identification of the candidate feature, which allow us to only match with potential candidates instead of matching with all features in the database. Performance tests have shown that this algorithm can reduce to ~10% the number of possible matching operations. In the Prague Castle Scene (Figure 1), in some clusters, the number of matches per feature is reduced from 1243 to 73. The average consumed time per match was also reduced to 12.8%.

Accuracy Results

In this section we present some accuracy results in what concerns the variation of luminance, scaling and rotation. For these testes we have used a determined image as a basis for (see Figure 6). The luminance test consists in changing the global image luminance by a determined percentage value (see Chart 1). The scaling test relies on an isotropic rescaling of the original image, also by a percentage value (see Chart 2). Finally, the rotation test consists on applying a rotation transformation to the original image, using steps of 30º (see Chart 3). In each test, the "full matches" group indicates the percentage of successfully matched features at a given instance. The "outliers" group indicates the percentage of false matches in the given "full matches" group. We can see an example depicted in Figure 6.

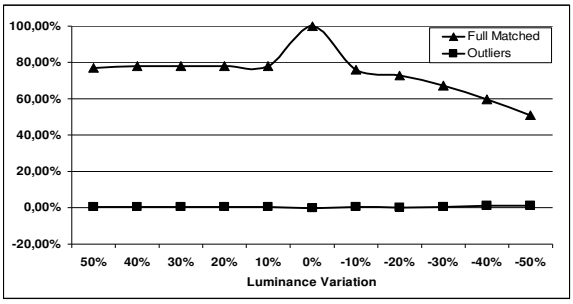

Chart 1. Luminance Test.

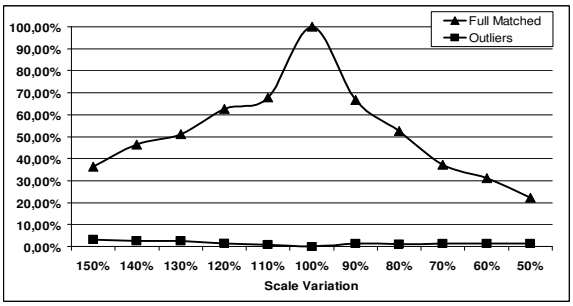

Chart 2. Scaling Test.

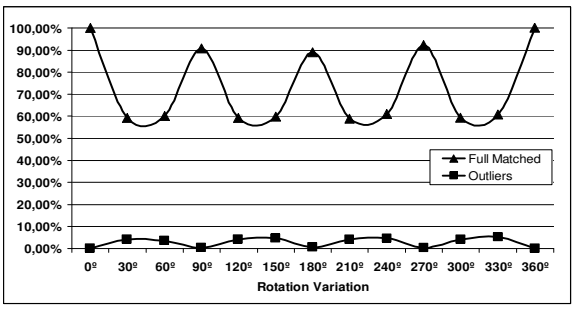

Chart 3. Rotation Test.

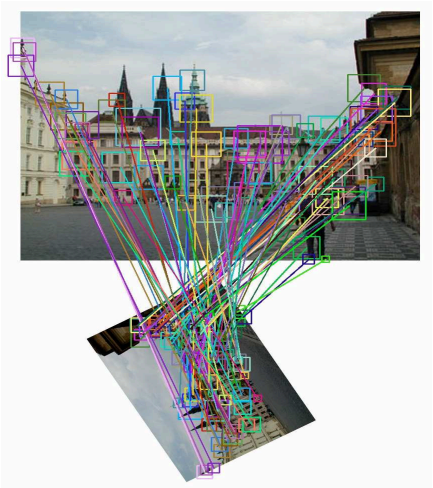

Figure 6. Accuracy test scenario

In Figure 6 the top image is the Prague Castle Scene (800x600, *n=15*, *t=1%,* 1243 features), and the bottom image is a version of the first (scale=*25%*, luminance=*-30%*, rotation = *240º*, 740 features). The results for this test were: 25.33% of full matches (187 features) and 7.72% of outliers (14 features).

Performance Results

Some performance tests were made using the Prague Castle Scene, varying the resolution size. These tests consist in extracting features and matching them against each other, using *n=15*, *t=1%* and assuming a clustering matching reduction of *~10%* (see Table 3).

Resolution	Features	Extraction (ms)	Matching (ms)	Total
320x240	285	10.30	16.25	26.55 (38 fps)
480x360	523	19.18	54.71	73.89 (14 fps)
640x480	818	30.81	133.82	164.63 (6 fps)
800x600	1243	47.04	309.01	356.05 (3 fps)

Table 3. Performance Test Results (*t=1%, n=15*, clustering matching reduction of ~10%).

In order to maintain real-time performance (25 fps) for all the presented resolutions, one must adapt the threshold extraction factor *t*, reducing the number of features (see Chart 4).

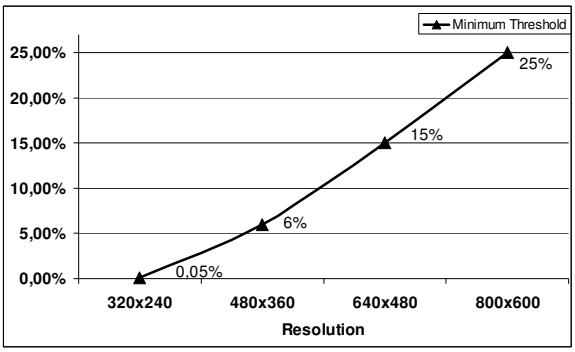

Chart 4. Minimum Threshold (*t*) required for each resolution to work in real-time (25 fps).

5. TEST CASE - TEXTURE TRACKING

As a test case for our novel natural feature detection technique, we have chosen the texture tracking paradigm, integrated in a real-time augmented reality (AR) application. We've chosen this test case since we have been developing new AR tracking methods since 2003, and already have a stable AR texture tracking system [Bas05]. The only constraint in the previous system was the need of black contours surrounding the texture to track, in order to compute the CPI. This test case is an advance of our previous work, since with this novel technique there is no need for the use of black contours. Our hardware setup is straightforward: a Webcam (320x240) connected to a PentiumIV (2.66GHz). We have knowledge about the camera intrinsic parameters, since it was previously calibrated using popular methods [Zha99]. The system process flow starts at an offline stage, where the planar texture image is preprocessed so that all natural features can be extracted using our proposed method. The algorithmic process is divided in two stages: Camera Pose Initialization and Feature Tracking. In the second stage, to increase performance, we use the previous texture pose to derive the current pose, based on optical flow and back projection techniques.

Stage 1 - Camera Pose Initialization

The CPI main goal is to find the first texture's pose, so that subsequent poses can be derived using the method proposed on the next stage. We apply our feature extraction and matching techniques at each camera frame, using the preprocessed texture image as baseline for comparison (Figure 8 – Left). We have chosen the RANSAC [Fis81] algorithm to identify outliers and the Direct Linear Transform (DLT) [Abd71] method to compute the planar object pose (6DOF – rotation & translation). Subsequently, we minimize the reprojection error, to refine the resulting pose, using a Gradient Descent (GD) technique [Brak04]. It is assumed that at the end of this stage, we have a camera pose.

Stage 2 - Feature Tracking

The feature tracking stage's main goal is to derive the current texture pose, using previous information, namely the number of previous features detected and tracked, and the previous texture pose. This stage relies on the assumption that the previous pose is a "good pose", and that we can back project the texture so that more features can be found and matched. The main problem of this assumption is that when large camera displacements are performed, the previous pose is not a "good pose" for applying the back projection technique, since the resulting image will be completely displaced.

Optical Flow

To overcome the displacement problem, we use an optical flow technique [Bou99], and apply it to the previous tracked features. The optical flow computes the current feature position based on the previous one, using the current and previous tracking images (see Figure 7). The feature matches must be refined to discard outliers. We use the RANSAC algorithm to identify these false matches, and compute the correct pose for the current image (DLT/GD). With the use

of the RANSAC algorithm a new problem arises: we run out of features. We must collect new features for the next pose computation – feature tracking stage.

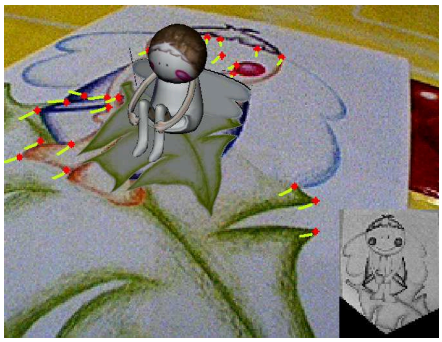

Figure 7. Virtual object registration with optical flow overlay. The bottom right image corresponds to the current back projected image.

Texture Backprojection

The texture backprojection [Bas05] consists in unrolling the texture's perspective distortion at a given frame, resulting in an untransformed and similar image to the one that was preprocessed in the offline stage. For that we use the correct pose information for the current image, apply the inverse transformation of common projective geometry concepts (see Figure 8). Since the back projected image is placed in a similar form as the base texture image, we can template match all the remaining valid features in a 2D untransformed space. By projecting the found feature positions using the correct pose information and the intrinsic camera parameters, we will obtain the 2D position for each feature match in the camera image subspace. The positive matches will be refined again (RANSAC) and a new refined pose will be computed (DLT/GD). More information can be found at [Bas05].

Figure 8. Left: base texture image (Copyright Solutions by Heart); Right: back projected image.

Feature Matching

Here we introduce a novel concept for feature matching, assuming we have two similar images that may differ in small pixel displacements (base texture image and back projected image), and using our previous proposed template matching metric. Our proposed method consists in finding a local maximum, in a determined sub-region. Having a key template T_k centred at (x_k, y_k) covering a *15x15* area of the original template image, and a search region also centred at (x_k, y_k) covering a *25x25* area of the back projected image, we can define the search algorithm for each feature in the following steps:

1. Define (x_s, y_s) as the centre position of the search template T_s extracted from the backprojected image, starting with the values of (x_k,y_k).

2. Define θ_x and θ_y as offsets of the current search template centre, starting each one with a *0* value.

3. Template match T_k (x_k, y_k) with T_s $(x_s+\theta_x, y_+\theta_y)$, varying θ_x and θ_y from -1 to 1, giving 9 possible centre positions (e.g. (x_s-1, y_s-1), (x_s, y_s-1), (x_s+1, y_s-1) … (x_s+1, y_s+1)).

4. Find the θ_x and θ_y that maximizes the matching function for the 9 possible centre positions.

5. If the found θ_x and θ_y are both different from *0*, then we update the current (x_s, y_s) with the θ_x and θ_y which maximize the matching function. We now have $x_s=x_s+\theta_x$ and $y_s=y_s+\theta_y$. The algorithm starts back from point 3, using the correct updated values, unless x_s or y_s have invalid values, since they are restricted to the limits constrained by the search region. In the latter case, the result will be the current x_s and y_s before the update process.

6. If the found x_s and y_s are both equal to *0*, then the T_s which maximizes the matching function is centred at the current (x_s, y_s). We have found the possible 2D corresponding position of (x_k, y_k) in the backprojected image. We illustrate this procedure in Figure 7.

Figure 9. Various steps of the search algorithm and the final result (pink/brighter square).

Accuracy and Performance Results

We have developed an original technique to generate synthetic video evaluation sets. In the general case, these evaluation test sets were based on the textured 3D planar object pose simulation, subject to translation and rotation DOF, much like the ones that occur when using the system with a HMD. This stream is then used to feed our texture tracking system, so that the obtained camera poses can be mathematically compared with the known simulated poses of the texture plane. The accuracy tests have shown that our algorithm has an average error of: 1.45 mm for translation, 0.76 degrees for rotation and 2.64 pixels for reprojection. In what concerns performance, the system operates at ~35 fps at the CPI stage and at ~60 fps at the feature tracking stage.

6. CONCLUSION AND FUTURE WORK

We have proposed a novel approach to real-time scale, rotation and luminance invariant natural feature tracking, in order to solve the CPI problem using totally automatic and real-time procedures. We have also proposed a heuristic method for feature clustering, which can reduce the number of feature matching operation to ~10%. We presented a real-time augmented reality texture tracking algorithm which uses this novel feature detection technique as a baseline and a new approach to feature matching by local maximum. This algorithm has millimetric and sub-degree precision, as has been stated by our accuracy tests. However, our tests have shown that our technique is still very sensitive to features at different scales and some degrees of rotation. We don't find this fact preoccupying since we are aware that most of the error has its origin on the bilinear interpolation filter we've used when creating the tests images, which have altered the strength of the image's derivatives. As future work we intend to compare our technique with other "*de facto*" algorithms (SIFT, SURF, PCA-SIFT) and to enable general 3D object tracking using 3D reconstruction and model based tracking techniques, using our technique as the main core.

7. REFERENCES

[Abd71] Abdel-Aziz, Y.I. and Karara, H.M., "Direct Linear Transformation into Object Space Coordinates in Close-Range Photogrammetry", in Procedures of Symposium of Close-Range Photogrammetry, 1971.

[Art07] http://www.hitl.washington.edu/artoolkit/

[Bas05] Bastos, R., Dias, J.M.S., "Fully Automated Texture Tracking Based on Natural Features Extraction and Template Matching", in ACE Technology, 2005.

[Bay06] Bay, H., Tuytelaars, T., Gool, L.V., "SURF: Speeded Up Robust Features", Proceedings of the ninth ECCV, 2006.

[Bou99] Bouguet, J.-Y., "Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the Algorithm", Intel Corporation, Microprocessor Research Labs., 1999.

[Brak04] Brakke, K. A., "Surface Evolver Manual", in Mathematics Department, Susquehanna University, Selinsgrove, 2004.

[Bue02] Buenaposada, J. M., Baumela, L., Real-Time Tracking and Estimation of Plane Pose", in ICPR 02, 2002.

[Cau92] Caudell, T.P. and Mizell, D.W., "Augmented Reality: An Application of Head-Up Display Technology to Manual Manufac-turing Processes," in Proceedings IEEE Hawaii International Conference on Systems Sciences, 1992.

[Che06] Chen, J. et al., "An Improved Real-Time Natural Feature Tracking Algorithm for AR Application", pp. 119-124, ICAT'06, 2006.

[Dem91] DeMenthon, D., Davis, L. S., "Model-based object pose in 25 lines of code", in European Conference on Computer Vision, 1991.

[Dia04] Dias, J.M.S., Silva, P., Nádia, J., Bastos, R., "ARTIC: Augmented Reality Tangible Interface by Color Evaluation", in Interacção, Lisbon, 2004,.

[Din04] Diniz, N., "AN APPROACH TO 3D DIGITAL DESIGN Free Hand Form Generation", in DCC´04 MIT 19-21 2004.

[Fis81] Fischler, M. A., Bolles, R. C., "Random Sample Consensus: A paradigm for model fitting with applications to image analysis and automated cartography", IN Communications of the Assoc. Comp. Mach., 1981.

[Gol93] Golub, G. H., and Van Loan, C. F., "Matrix Computations", in 2nd ed. Johns Hopkins University Press, 1993.

[Kat03] Kato, Hirokazu , Tachibana, K., Billinghurst, M., Grafe, M., "A Registration Method based on Texture Tracking using AR-ToolKit", in The Second IEEE International ART Workshop, Tokyo, Japan, 2003.

[Kat99] Kato, H., Billinghurst, M., "Maker Tracking and HDM calibration for a video-based augmented reality conferencing system", in International Workshop on Augmented Reality (IWAR), USA, 1999.

[Ke04] Ke Y., Sukthankar R., "PCA-SIFT: A More Distinctive Representation for Local Image Descriptors", CVPR, 2004.

[Low03]Lowe, D. G. "Distinctive Image Features from Scale-Invariant Keypoints", in IJCV, 60, 2, pp 91-110, 2003.

[Mai02] Malik, S., Roth, G., McDonald, C., "Robust 2D Tracking for Real-time Augmented Reality", in Proceedings of Vision Interface (VI), pp 399-406, Calgary, Alberta, Canada, 2002.

[Mar02] E. Marchand and F. Chaumette. Virtual visual servoing: a framework for real-time augmented reality. In EUROGRAPHICS' 02 Conference Proceeding, 2002.

[Shi94] Shi, J., Tomasi, C., "Good Features to Track", in IEEE Conference on CVPR, 1994.

[Sim02] Simon, G., Berger, M-O., "Reconstructing while registering: a novel approach for markerless augmented reality", in ISMAR' 02 Conference Proceeding, 2002.

[Yua06] Yuan, C. "Markerless Pose Tracking for Augmented Reality", in ISVC 2006.

[Zha99] Zhang, Z., "Flexible Camera Calibration By Viewing a Plane From Unknown orientations", ICCV'99, Greece, 1999.