AFFTRACK: ROBUST TRACKING OF FEATURES IN VARIABLE-ZOOM VIDEOS

Rodrigo Minetto, Neucimar J. Leite, Jorge Stolfi

Institute of Computing
University of Campinas
Campinas, 1251 Albert Einstein, SP, Brazil
rodrigo.minetto@ic.unicamp.br, neucimar@ic.unicamp.br, stolfi@ic.unicamp.br

ABSTRACT

We describe a robust and accurate algorithm, nicknamed AffTrack, to track selected features of a rigid 3D object in a video recording, given a canonical image of each feature and its position on the object. AffTrack uses a synergistic combination of a multiscale feature finder and a flexible camera calibrator. This synergy between the two modules allows, AffTrack to recover features after occlusions of arbitrary duration. Compared to other solutions to this problem, AffTrack can handle videos with variable zoom, and variable lens distortion, does not require a complete geometric model of the object, and does not require the selection of key frames. Tests indicate that AffTrack is more robust and accurate than the popular object trackers include in H. Kato’s ARToolKit and in the OpenCV library.

Index Terms— Camera calibration, feature tracking, object tracking, augmented reality

1. INTRODUCTION

Statement of the problem. We consider here the problem of tracking a specific set of 2D features of a rigid 3D object (truck, package, etc.) in a video recording. By features we mean distinctive albedo markings on the object’s surface, intentional or incidental rigidly fixed on the object. For each feature, the user must provide a canonical image showing its appearance under uniform lighting and orthogonal projection: its position, size, and orientation in the object’s frame of reference; and the position of the feature in the first frame. The goal of the problem is to determine the set of camera parameters (including the perspective matrix \( S \), the focal length \( f \), and the radial distortion \( \kappa \)) for each frame of the video. Note that \( f \) and \( \kappa \) are variable in cameras with zoom lenses.

Our solution. We present an algorithm, nicknamed AffTrack, to solve the above problem. The algorithm (described in detail in sections 3—5) uses a synergistic combination of two main procedures, a multiscale feature finder (FF) that locates the features on each frame, and a flexible camera calibrator (CC) that computes the camera parameters from those positions.

A key aspect of AffTrack is the use of a confidence weight to express the reliability of each feature on each frame. These weights are initially computed from the quality of the match returned by the feature finder, and then iteratively adjusted by the calibrator depending on the consistency between the feature’s reported position and the position of all other features. Thus, instead of RANSAC’s binary accept/reject criterion, our algorithm uses a “fuzzy” classification of outliers.

Another key aspect of AffTrack is that the camera parameters calibrated in previous frames are used to guess the location and shape of each feature in the next video frame, as required by the feature finder.

AffTrack does not require the features to be visible or successfully identified in all frames (not even on the first one), as long as those that are visible are sufficient to determine the camera’s unknown parameters. As a result of the two-way interaction between the FF and the CC, it can usually recover a feature that was occluded or mis-identified by the finder, as soon as it becomes visible again. Increasing the number of tracked features beyond the theoretical minimum generally improves the reliability and accuracy of the calibration.

Advantages and applications. Our emphasis is on robustness, accuracy, and flexibility. The algorithm is quite efficient, because the accurate guessing of feature positions, provided by the calibrator, allow the finder to use smaller templates and windows. We hope that AffTrack will be useful in many real-world applications, such as vehicle tracking, augmented reality, etc.

Comparison to other trackers. The integration of feature tracking and camera calibration for 3D object tracking has been described in several recent articles, some of which are covered in the survey by Lepetit and Fua [1].

We mention in particular the tracker by Vacchetti et al. [2] which attempts to solve approximately the same problem as we do here. An advantage of their algorithm is that it automatically identifies trackable features and extracts their template images from the video itself. However, their algorithm requires a complete geometric model of the object. It also requires the selection of certain key frames along the video, where the features to be tracked are all visible and the object’s pose is approximately known. AffTrack only requires the features positions on the object, and does not require key-frame selection.

Our algorithm is more general than H. Kato’s widely used ARToolKit [3]. Besides allowing changes in focal length, AffTrack does not require that the features have a specific and easily-recognized shape as ARToolKit does. In fact, our algorithm can reliably track and use even features that have been reduced to a single-pixel dot.

Another popular tracker is the implementation of the KLT algorithm included in the OpenCV library as the routine cvCalcOpticalFlowPyrLK. Although it is only a 2D feature tracker, its output (the locations of the feature on each frame) can be fed to any camera calibrator, such as R. Tsai’s algorithm [4], to...
create a 3D tracker. As we show in section 6, this solution suffers from unavoidable drift that frequently leads to permanent tracking failure of moving objects.

2. FEATURE PARAMETERS

The object geometry of each feature (assumed to be fixed along the entire video) consists of the position \( p_w \) of its center, the two vectors \( u_w, v_w \) that define its orientation and size (all in millimeters in the object’s coordinate system), and its canonical image \( M \). For non-rectangular features, a mask image \( W \) should also be given. See figure 1(a,b). The appearance of a feature on a video frame \( I \) is usually distorted by the perspective projection and affected by changes in lighting, lens aperture, exposure time, etc.. The geometric distortion is defined by a point \( p_t \), the center of the feature in the frame, and two vectors \( u_t, v_t \) (all in pixels, in the frame’s coordinate system) corresponding to \( u_w, v_w \). See figure 1(c). The photometric distortion can be modeled by the formula \( v \approx \alpha u + \beta \) where \( u \) is the pixel value in the canonical image \( M \) and \( v \) is the corresponding pixel value in the frame \( I \).

3. INTEGRATING THE CC AND FF ALGORITHMS

The core of AffTrack is outlined as Procedure 1 below. Its steps are described in detail in the rest of this section.

Procedure 1 AffTrack calibration for a frame \( I \)

1. Get an approximate calibration \( C' \) for this frame.
2. For each feature index \( k \):
   3. Get its estimated frame position \( p_t[k] \) and initial visibility weight \( w''[k] \).
   4. Get provisional lighting parameters \( \alpha'[k]\), \( \beta'[k] \).
   5. Obtain the template \( G'[k] \) and masks \( V'[k] \).
   6. If the feature is expected to be visible, run the feature finder to get its adjusted position \( p''_t[k] \) and its confidence weight \( w''[k] \). Otherwise set \( w''[k] \) to zero.
7. Run the camera calibrator to compute the final parameter tuple \( C \) for this frame.
8. For each feature index \( k \):
   9. Compute the final feature position \( p_t[k] \) and shape vectors \( u_t[k], v_t[k] \).
   10. Generate the definitive feature template \( G[k] \) and mask \( V[k] \).

11. Recompute \( \alpha[k], \beta[k] \).

When necessary, we will use the superscript \( (i) \) to refer to the values computed by Procedure 1 on frame number \( i \) (counting from 1). We will discuss first the general case \( i \geq 2 \). The first frame requires special handling, as described later on.

Initial camera parameters. In step 1 we obtain an initial estimate \( C' \) of the camera parameters for this frame. Except for the first frame, we set \( C'(1) \) to the calibrated parameters \( C(1-1) \) that were computed in step 7 for the previous frame.

Initial feature positions. In step 3, we obtain an initial estimate for the position \( p'_t[k] \) of each feature. Except for the first two frames, we use linear extrapolation of the feature’s motion in the two previous frames: namely, we set \( p'_t(i)[k] \) to \( 2p'_t(i-1)[k] - p'_t(i-2)[k] \); or, if \( i = 2 \), to \( p'_t(i-1)[k] \). Note that this extrapolation uses the final positions \( p''_t(i)[k] \) returned by the feature finder in step 6. In this step we set the initial confidence weight \( w''[k] \) of each feature to 1 if the feature’s guessed position \( p'_t[k] \) is inside the frame’s domain, and 0 otherwise.

Lighting parameters. In step 4 we obtain estimates \( \alpha'[k], \beta'[k] \) for the lighting coefficients of each feature. Except for the first frame, we use the coefficients \( \alpha'[1-1][k], \beta'[1-1][k] \) that were computed for the previous frame in step 11.

Synthetic templates. In step 5 we obtain the template \( G'[k] \) for each feature, deformed according to the estimated camera parameters \( C' \), and the corresponding mask image \( V[k] \). Except for the first frame, here too we use the templates and masks \( G(1-1) \) and \( V(1-1) \) computed for the previous frame in step 10.

Feature location. In step 6, the feature finder (described in section 4) is called for each feature. The finder is given the template \( \alpha'[k] G'[k] + \beta'[k] \), the corresponding mask \( V'[k] \), and the guessed position \( p'_t[k] \). It returns a found position \( p''_t[k] \), and a weight \( w''[k] \) that quantifies the reality of \( p''_t[k] \).

Weighted calibration. In step 7, the camera calibrator (section 5) is called to compute the final camera parameters \( C \) for the current frame, from the known object positions \( p_w[k] \) of all features and their frame positions \( p'_t[k] \), returned by the finder in step 6. The calibrator also uses the confidence weights \( w''[k] \) provided by the finder.

Final feature positions. In step 9, the parameter tuple \( C \) calibrated in step 7 is used to compute the final position \( p''_t[k] \) of each feature in the current frame, from its object position \( p_w[k] \). In this step we also re-compute the frame geometry vectors \( u_t[k], v_t[k] \) for each feature, from the object shape vectors \( u_w[k], v_w[k] \), by numerical differentiation of the object-to-sensor map.

Recomputing the synthetic templates. In step 10, we recompute the template \( G[k] \) of each feature and its mask \( V[k] \) from the canonical template \( M[k] \) and its mask \( W[k] \), using the shape vectors \( u_t[k], v_t[k] \) obtained in steps 9. See figure 2.

Adaptive color correction. Finally, in step 11, we re-estimate the lighting coefficients \( \alpha[k], \beta[k] \) of each feature, by linear regression of the deformed template \( G[k] \), translated to the position \( p''_t[k] \), against the frame image \( I \).

Processing of frame 1. For the first frame of the video \( i = 1 \), the user must explicitly provide approximate positions \( p'_t[k] \) and the corresponding confidence weights \( w''[k] \) for a sufficiently large subset of the features. In that case, we first skip...
steps 1–6, and set $p''[k]$ to the user given positions. Next we run steps 7–11 to get a calibrated camera parameter record $C$ for that frame, as well as official positions $p_i[k]$, template images $G[k]$, $V[k]$, and parameters $\alpha[k]$ and $\beta[k]$ for all features that are expected to be visible in the frame. Then we run the whole Procedure 1 again on the first frame, using those computed values for $C'$ in step 1, for $G'[k]$ and $V'[k]$ in step 5, for $p'_i[k]$ in step 3, and for $\alpha'[k], \beta'[k]$ in step 4.

4. THE FEATURE FINDING ALGORITHM

Our feature finder uses a variant of the Kanade-Lucas-Tomasi (KLT) algorithm, applied independently to each feature. It tries to find the point $p''[k]$ such that the template $G$ best matches the sub-image $H(p''[k])$ of $I$ centered at $p$. The quality of the match is measured by the mean quadratic discrepancy function

$$E(p) = \frac{1}{\int \int V(q)(G(q) - I(p + q))^2 dq}{\int \int V(q) dq}$$

(1)

where the integrals on $q$ span the domain of $G$. Our implementation uses Newton’s method to solve the non-linear equation $\nabla E(p) = (0, 0)$. Using symbolic differentiation of formula (1) to get the gradient $\nabla E(p)$ and the Hessian $\nabla^2 E(p)$.

The finder also returns its estimate $w''[k]$ of the probability that $p''[k]$ is a meaningful result — that is, not an outlier. This estimate is obtained by comparing the value of $E(p'')$ with the value of $E$ that is expected for a correct match, considering the amount of noise present in the frame image.

5. THE CAMERA CALIBRATOR

AffTrack’s calibrator takes a list $D$ of $m$ data pairs as input, where each pair consists of the object coordinates of a point on the scene, its apparent coordinates $p''[k]$ in the frame image, and a confidence weight $w''[k]$, for $k = 1, \ldots, M$. It outputs the parameter tuple $C$ that provides the best fit to the data.

The calibration begins by computing the matrix $S$ and focal length $f$ for a camera that is located very far from the scene and which best matches the given data. For such a camera, the mapping of object coordinates $p_o$ to undistorted projected coordinates $p_u$ is an affine map, that can be found by standard linear least-squares fitting followed by orthogonalization of the map’s $3 \times 3$ linear sub-matrix.

Like R.Willson [5], we then use the algorithm lmdif, from the Netlib/MINPACK package, to find the tuple $C$ that minimizes the quadratic image-space mean error metric

$$Q(C) = \sum_{k=1}^{n} w''[k] (p_i[k] - p'_i[k])^2$$

(2)

where $p''[k]$ are the frame coordinates of feature $k$ from the input data $D$, and $p'_i[k]$ are the frame coordinates computed from $p_o[k]$ and $C$.

<table>
<thead>
<tr>
<th>Video</th>
<th>n</th>
<th>m</th>
<th>Obs.</th>
<th>AffTrack</th>
<th>ARToolkit</th>
<th>OpenCV</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>338</td>
<td>24</td>
<td></td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>r2</td>
<td>270</td>
<td>8</td>
<td></td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>r3</td>
<td>301</td>
<td>8</td>
<td></td>
<td>100</td>
<td>50</td>
<td>44</td>
</tr>
<tr>
<td>r4</td>
<td>244</td>
<td>8</td>
<td></td>
<td>100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>r5</td>
<td>130</td>
<td>24</td>
<td></td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r6</td>
<td>233</td>
<td>28</td>
<td></td>
<td>100</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>p1</td>
<td>501</td>
<td>24</td>
<td></td>
<td>100</td>
<td>92</td>
<td>2.36</td>
</tr>
<tr>
<td>p2</td>
<td>501</td>
<td>8</td>
<td></td>
<td>100</td>
<td>45</td>
<td>2.49</td>
</tr>
<tr>
<td>p3</td>
<td>501</td>
<td>8</td>
<td></td>
<td>100</td>
<td>10</td>
<td>3.56</td>
</tr>
<tr>
<td>p4</td>
<td>251</td>
<td>8</td>
<td></td>
<td>100</td>
<td>99</td>
<td>3.43</td>
</tr>
<tr>
<td>p5</td>
<td>251</td>
<td>8</td>
<td></td>
<td>100</td>
<td>80</td>
<td>2.02</td>
</tr>
<tr>
<td>p6</td>
<td>251</td>
<td>24</td>
<td></td>
<td>100</td>
<td>26</td>
<td>3.71</td>
</tr>
<tr>
<td>p7</td>
<td>501</td>
<td>8</td>
<td></td>
<td>100</td>
<td>85.93</td>
<td>99</td>
</tr>
<tr>
<td>p8</td>
<td>501</td>
<td>26</td>
<td></td>
<td>100</td>
<td>26</td>
<td>4.23</td>
</tr>
</tbody>
</table>

6. TESTS

We tested our algorithm on six real and eight computer-generated videos [6], summarized in table 1.

The real videos $r1$–$r6$ were shot with a hand-held consumer-grade DVR (Canon Optura 40) with variable zoom. They were recorded in MPEG format at $320 \times 240$ resolution, 14.98 frames per second. The object to be tracked was a stage from various 3D models (including a virtual version of a 3D model of the stage, above, and a realistic office model of the stage, above, and a realistic office model of the stage, above, and a realistic office model of the stage), with variable zoom.

In order to improve the numerical stability and reduce the number of variables in the optimization, we replace the param- eters $T_x, T_y, T_z$ by the camera’s object coordinates $V_x, V_y, V_z$ and the rotation submatrix $R$ by the equivalent Euler angles $R_0, R_1, R_2$. We also replace the focal length $f$ by $\log(f/\hat{z}_c)$ where $\hat{z}_c$ is the mean $z$ coordinate of the features in the camera’s system.

After finding the parameters $C$ that minimize the error function (2), we recompute the confidence weights $w[k]$ by Bayesian classification. Specifically, we assume that the frame position errors $e_t[k] = p_t[k] - p'_t[k]$ are two-dimensional Gaussian variables, with different variances $\sigma_i$ for inliers and $\sigma_o$ for outliers. The input weights $w''[k]$ are taken to be the a priori probabilities of the points being inliers. We use Bayes’s formula to compute $w[k]$ as the posterior probability of $p'_t[k]$ being an inliner.

In some of the videos, the frame images generated by the POVRay were modified by radial distortion, mixture with of 5% noise, Gaussian blurring, and conversion to and from JPEG format at 85% quality, to simulate the lower image quality of real videos.
The videos were processed with (1) our integrated AffTrack algorithm (2) the H.Kato’s ARToolKit tracker [3] (3) and with the recursive tracker included in the OpenCV library [7] followed by R. Willson’s implementation of Tsai’s camera calibrator [4, 5]. ARToolKit could be used only on those videos which included its special patterns and were shot with fixed focal length. The outer corners of each ARToolKit pattern were treated as four separate features in the other two trackers. For AffTrack and OpenCV, the frame coordinates of each feature on the first frame (only) were picked by hand.

**Evaluation.** The results of the tests are summarized in table 1. For each program, we classified each frame as ‘success’ or ‘failure’ according to whether the calibration produced minimally usable parameters or not. For the synthetic videos, for which the true camera parameters \( \hat{C}(i) \) were known, we considered frame \( i \) to be a success if the root mean square error

\[
q^{(i)} = \left[ \frac{1}{m} \sum_{k=1}^{m} w[k] |p_k[i][k] - \hat{p}_k[i][k]|^2 \right]^{1/2}
\]

was less than 8 pixels, where \( \hat{p}_k[i][k] \) is the position of feature \( k \) implied by \( p_k[i][k] \) and \( \hat{C}(i) \). For the real videos, we visually classified every frame for which the image positions computed by any two of the programs disagreed by more than 8 pixels in the RMS sense, and counted all other frames as successes.

**Discussion.** These and other tests confirmed that AffTrack is free from long-term drift, even though it carries information from one frame to the next. AffTrack also has consistently less jitter than the other two trackers. See figure 3. ARToolkit too is free from long-term drift, although its RMS frame position error is about twice as large as AffTrack’s. On the other hand the OpenCV+Tsai combination often exhibits runaway drift in fast-moving videos.

As shown in table 1, our algorithm successfully calibrated 100% of the frames in all videos, whereas the other two trackers had significant failure rates, sometimes up to 100%. Besides being prone to runaway drift, the OpenCV tracker also tends to fails permanently whenever any feature becomes occluded. The ARToolKit tracker too is sensitive to occlusion, but usually recovers once the features become visible again. On the other hand, ARToolKit locates each target independently so it often matches the wrong pattern and/or the correct pattern in the wrong orientation. See figure 4. Such pattern orientation errors are the reason for ARToolKit’s large \( q \) on video p7.

![Fig. 4](image-url) Frame 07 of video p5 showing failure of the ARToolKit’s finder due to feature confusion

![Fig. 5](image-url) Frames 140, 145 and 150 from video r4, showing AffTrack’s resistance to occlusion and feature confusion.

**7. CONCLUSIONS**

Tests have shown that AffTrack is significantly more robust than other available 3D trackers. By passing information in both directions between the feature finder and camera calibrator our algorithm can adapt to changes in feature size, shape, illumination.

**8. REFERENCES**


