Synchronizing Video Sequences

Tinne Tuytelaars¹ and Luc Van Gool¹,²

¹. ESAT - PSI - Visics, University of Leuven, Belgium  ². BIWI, ETH, Zürich, Switzerland
{tinne.tuytelaars, luc.vangool}@esat.kuleuven.ac.be

Abstract

We present a novel method for automatically synchronizing two video sequences of the same event. Unlike previously proposed methods, we do not put any restrictive constraints on the scene nor on the camera motions: our method can deal with independently moving cameras, wide baseline conditions, and general 3D scenes. It starts from five point correspondences throughout the video sequences, that are provided using wide baseline matching and tracking techniques. It is efficient, in that it can be implemented in a non-combinatorial way. The feasibility of the method is demonstrated by preliminary experimental results.

1 Introduction

Digital video cameras are becoming more and more common in our everyday life. As a result, chances that the same event is recorded by multiple cameras increase. Clearly, having two such recordings brings a lot of valuable additional information. Indeed, it allows to recover the unknown depth parameter and hence makes it possible to perform accurate measurements in the real world. For instance, from two overlapping surveillance cameras, one could retrieve the height of an intruder or his exact location, without the limitations of one-view forensic measurement techniques (e.g. [3]) such as the need for vanishing points and lines or planar structures in the scene. In a similar vain, several recordings of the same event can be exploited to definitely settle controversies in football games, as was done by [9].

Of course, the same goal can also be achieved using dedicated hardware to ensure the two videos are recorded in a synchronized way. However, this is only possible for professional applications where more expensive equipment can be used. Moreover, videos are not always taken for the purpose of extracting 3D information from them, even though it might turn out to be useful at a later stage.

Another alternative is to manually perform the synchronization. Although this can be a viable solution for some applications (e.g. [9]), it is labour-intensive so again expensive. Most importantly, it does not always yield sufficiently accurate results. Look for instance at the example movies shown in our experimental results section. If it weren’t for the groundtruth information we had available for one of the two sequences, it is very hard to judge the accuracy of the obtained results. In general cases, it is very hard for a human observer to synchronize video sequences even up to a few frames.

Finally, for some applications it is not so much the final registration that matters, but rather the very fact that they can be synchronized (or not). This allows e.g. to retrieve videos of the same event from a database or to classify events by comparing them to videos of prototype events.

Previous Work

Several researchers already looked into the problem of video synchronization. However, none of them deals with the general case of moving cameras and general 3D scenes. Stein [11] proposed a first method to align video sequences, assuming the cameras are static and the images are related by a homography (i.e., either a planar scene or two cameras placed so close to one another that parallax effects can be ignored). Also Caspi and Irani [1] make the same assumption. They align the video sequences both spatially and temporarily in a feature-less way, minimizing the sum of squared differences between the sequences. They also developed a trajectory-based scheme, that works with projective epipolar geometry [2]. In their examples, cameras are placed extremely far apart from one another, and subframe accurate synchronization is obtained. However, both cameras still need to be static (or moving rigidly with respect to one another), such that they are always linked by the same epipolar geometry. The same holds for the method recently proposed by Tresadern et al. [12]. Finally, the method proposed by Rao et al. [8] also deals with general 3D scenes and can cope with non-constant time delays. This allows to synchronize videos of similar (but not necessarily identical) events, such as recordings of human activities. They partially compensate for moving cameras, by subtracting the coordinates of a static point in the background. However, this only compensates for translations in the image plane, not for generally translating and rotating cameras.

To the best of our knowledge, we are the first to tackle the problem of automatic video synchronization for inde-
pendently moving cameras and general 3D scenes. In this paper, we go after a single time delay or time offset \( t_d \) (assuming both cameras have the same frame rate), as this is the case that happens most often in practice. However, it can easily be extended to deal with different frame rates, using a linear temporal transformation (offset plus scalefactor).

This may seem a rather simple problem at first: given the vast amount of input data, one gets the feeling that retrieving this one parameter \( t_d \) cannot be so difficult. However, when working with moving cameras, the number of hidden parameters increases drastically: both internal and external camera parameters may change from frame to frame, and since they influence our measurements (image coordinates of the point correspondences), they should be determined or at least compensated for, for every frame one wants to use.

The remainder of the paper is organized as follows. First, we explain how we plan to synchronize two video sequences of the same event in section 2.1. Next, in section 3 we have a closer look at some practical implementation issues. Then comes section 4 showing preliminary experimental results. Section 5 concludes the paper.

2 Video synchronization

2.1 Solution Strategy - 2D or 3D ?

A first important observation consists of the fact that although the scene points may be moving non-rigidly, for perfectly corresponding frames (i.e., frames taken at exactly the same time instant \( t \)) they can be considered as a rigid configuration. As a result, the problem of video synchronization can be reformulated in terms of checking the rigidity of a set of points. A general solution strategy to our problem could then be summarized as follows. For each combination of frames (one from the first sequence and one from the second sequence) we compute a rigidity measure. The results are stored in a compatibility matrix \( M \). A diagonal line of low values will then appear at positions \( M(i, i + t_d) \), with \( t_d \) the time delay between the two sequences.

The most straightforward method to check the rigidity of a set of points in two views consists of computing the best fitting epipolar geometry and checking the residual errors (i.e. the distances from the points to their respective epipolar lines). From now on, we refer to this method or any variation thereof (as e.g. used by [1, 8, 12]) as 2D analysis. Alternatively, one can also consider the problem in 3D, by backprojecting the points as lines into the 3D world and looking for intersecting lines. This is referred to as 3D analysis from now on. Of course, both methods are closely related and the difference between them is subtle, albeit not unimportant. At first, a 2D analysis may seem the most appropriate choice for tackling the problem at hand, as in 3D one has to take into account the effect of skew (see section 3.2), whereas in 2D the error measure is the distance from a point to its epipolar line, and this can be compared to image measurement noise, i.e. a meaningful distance. However, here we argue that nevertheless a 3D analysis is the better choice.

When selecting a solution strategy, the following issues need to be taken into account:

- **Effect of sampling** In practice, due to a discrete sampling over time, the perfectly corresponding frame mostly falls in between two frames in the other sequence. As such, it can only be approximated by the closest frame or, via interpolation techniques, from the two closest frames. This has an effect on the accuracy of the point correspondences. The ideal solution strategy should not be too sensitive to such inaccuracies, and degrade smoothly as the deviations get larger. This problem becomes even more acute with moving cameras, as the motions are typically larger and one can no longer integrate out the effect by combining multiple frames, as done in [1, 8, 12]. Note that in this context a distinction needs to be made between (1) trying to estimate the time delay with subframe accuracy (as done e.g. by [8, 12]), and (2) realizing that the time delay is probably not integer, (i.e. the closest frame does not correspond exactly), and taking this into account when selecting a solution strategy. In this paper, we try to do (2), not (1).

- **Efficiency** The straightforward general solution strategy described above performs an exhaustive search over every pair of frames. This combinatorial nature makes it computationally expensive, and unfit for longer video sequences or for applications involving a database of video sequences.

Note how the above two constraints are in a sense contradicting: the effect of sampling could easily be attenuated by adding more interpolated views, but this at the same time makes the combinatorics even heavier.

In the light of these considerations, we believe that a 3D analysis is definitely the better choice. Backprojecting points as lines in 3D is a 1-view process, whereas estimating epipolar geometry (or any derived 2D method) inherently involves two views. This not only influences the efficiency (allowing a combinatorics-free implementation, section 3.3), but also matters when taking into account the effect of sampling. Indeed, when working in 3D, the backprojected lines are correct, and one can reason geometrically about the interpolating surfaces through these sets of lines (as in fig. 1), and where they might intersect. In a 2D analysis, this would involve interpolating between epipolar geometries which themselves are already corrupted as they are computed based on not-exactly-corresponding frames.
Also the link between 2D residual errors and image measurement noise no longer seems to make much sense.

2.2 Backprojecting points into 3D

Backprojecting a point into the 3D physical world is only possible in a calibrated setup, i.e., when the camera parameters are known. Clearly, this is by far not the case in a general video synchronization setup. However, we do not need a Euclidean reconstruction. Instead, we can express points and lines in 3D relative to a coordinate axes frame fixed to some scene points, effectively avoiding the need for calibration. If we assume scaled orthographic projection, four scene points suffice to define such a reference frame [5].

Given four scene points \( \{ \mathbf{P}_i, i = 0..3 \} \), any fifth point \( \mathbf{P} \) can be expressed as a linear combination of these points:

\[
\mathbf{P} = \mathbf{P}_0 + X(\mathbf{P}_1 - \mathbf{P}_0) + Y(\mathbf{P}_2 - \mathbf{P}_0) + Z(\mathbf{P}_3 - \mathbf{P}_0)
\]

with \( X, Y \) and \( Z \) the coordinates of the point \( \mathbf{P} \) expressed in the new coordinate axes frame. After projection onto an image, a similar relation still holds (note that we use capital letters to denote 3D points and small letters for 2D points):

\[
\mathbf{p} = \mathbf{p}_0 + X(\mathbf{p}_1 - \mathbf{p}_0) + Y(\mathbf{p}_2 - \mathbf{p}_0) + Z(\mathbf{p}_3 - \mathbf{p}_0)
\]

Suppose we know the coordinates of the points in an image. Then we can consider the above equation as the equation of a line in 3D, with unknowns \((X, Y, Z)\). This is the backprojected line, expressed relative to the coordinate axes frame fixed to the scene points \( \{ \mathbf{P}_i, i = 0..3 \} \).

2.3 Finding corresponding frames

If we repeat the above process for each frame in each video sequence, we obtain two sets of lines in 3D \( S_i = \{ l_i(t_i), t_i = 1..n_i \} \), with \( t_i \) the time relative to the start of the sequence, \( n_i \) the number of frames in the sequence, and the subscript \( i \) referring to the first or second sequence. Note that the scene points to which the coordinate axes frame is attached need not be static: they can freely move around with respect to the camera, with respect to the fifth point, with respect to the real world or even with respect to one another, just as the fifth point.

We do not know the 3D coordinates \((X(t), Y(t), Z(t))\) of the fifth point, but we know that at each time instant \( t = t_{si} + t_i \) (with \( t_{si} \) the time when that video sequence was started) the point should be on the backprojected lines \( l_1(t - t_{si}) \) and \( l_2(t - t_{s2}) \). This is illustrated in figure 1. It shows the 3D trajectory of the fifth point (relative to the coordinate axes frame attached to \( \mathbf{P}_0..\mathbf{P}_3 \)), as well as the backprojected lines from both video sequences. To keep the figure simple, all lines within a set are drawn parallel. Note though that this is not true in general. More precisely, the orientation of the lines will change when either the camera rotates or when one of the points to which the coordinate axes frame is attached moves relative to the camera.

If the time delay \( t_d = t_{s2} - t_{s1} \) is an integer, corresponding frames between the two video sequences can be found as the intersection of a line of \( S_1 \) with a line of \( S_2 \). If, on the other hand, the time delay is non-integer, as will usually be the case, there are no exact corresponding frames and no exact line intersections. Instead, we have to interpolate the image coordinates of two subsequent frames or, alternatively, interpolate the surface in between the backprojected lines (assuming a smooth motion of both the scene points and the cameras). Instead of line intersections, we then compute the intersection of a line of \( S_1 \) with the interpolated surface spanned by the lines of \( S_2 \) and vice versa. From these intersections, we can find the closest frames in the other video sequence.

3 Technical Details

3.1 Finding corresponding point trajectories

Before we can apply the above scheme for synchronizing two videos, we have to establish point correspondences. To this end, we first match affine invariant regions between the first frame of each sequence [7, 13, 14]. Next, we extract corner features within the matched invariant regions in one of these frames, and track them through the entire sequence using the KLT feature tracker [10]. The corner features within each invariant region are then transferred to the other sequence by applying the affine transformation mapping the invariant region onto its corresponding region. This allows to track the same features also in the other sequence.

The above scheme assumes that there is sufficient overlap between the first frames of each sequence, and that the effect of occlusions is limited, such that a sufficient amount of features can be tracked throughout the entire sequences. However, it can easily be made more robust by including

![Figure 1: Using four points to define an affine reference frame, the trajectory of a fifth point is backprojected into 3D from both cameras. For corresponding frames, the reprojected lines should intersect.](image)
more matching, e.g. matching every \( n \)-th frame of each video sequence.

Of all the points that have been tracked successfully throughout both sequences, we then have to select a subset of 5 points, making sure they are not moving rigidly and they do not include mismatches. For the time being, this selection is still performed manually – although we are currently working on a fully automatic, robust implementation.

3.2 Computing the Line-Surface Intersection

Interpolating the surface spanned by the backprojected lines is not straightforward, since subsequent lines are not necessarily coplanar. We implemented two different schemes to approximate the intersection point and find the closest frame in the other sequence.

The first and simplest method simply looks for the closest line. In fact, this is theoretically invalid, since one cannot compare distances when using an affine reference frame. Indeed, when ordering lines according to their distance, one could get a different result depending on the chosen coordinate axes frame. However, intersections are invariant under affine transformations, so a distance of zero remains zero, and also distances close to zero probably stay close to zero when shifting to a different coordinate axes frame.

The second method is computationally more expensive, but avoids the need for computing distances by directly computing intersections between the line and a set of planes that approximate the interpolating surface. This is illustrated in figure 2. The part of the (unknown) interpolating surface in between two subsequent backprojected lines (shown left) lies with a high probability somewhere within the boundingbox spanned by the four planes shown at the right. A backprojected line from the other sequence cannot intersect the interpolated surface in between the two backprojected lines without also intersecting the boundingbox, which can easily be checked for.

Experimental validation showed that the simpler implementation based on distances always gave the same outcome, in a fraction of the computation time needed for the boundingbox-based method. So we stick to the simpler distance-based method, possibly repeating the same procedure for a few different choices of coordinate axes frame to increase robustness in case of doubt (see also section 4).

3.3 A non-combinatorial implementation

As mentioned earlier, a big advantage of working in 3D rather than 2D is that it is much more efficient, as the lines can be computed for each sequence separately. The only step that involves combinations of frames consists of computing a distance between lines in 3D to find the closest one, and even this step can be implemented in a non-combinatorial way. To this end, we discretize the 3D space into a set of cells, with attached to each cell a list of lines of \( S_1 \) passing through that cell. In most cases the list will be empty, but it can also have one or more elements. For each line \( l_2 \) of \( S_2 \), we then check the cells through which it passes. Each time a non-empty list is found, we compute the distance between \( l_2 \) and the lines in the list, and keep the line with the smallest overall distance. Clearly, such a non-combinatorial implementation only makes sense if one has to find corresponding parts in very long video sequences, or if one wants to find similar events in a video database. In most other cases, the most time-consuming part is the preprocessing (matching and tracking) rather than the actual video synchronization, which runs almost instantaneously.

4 Experimental Results

As a first example, we synchronized two sequences of the movie ‘Run, Lola, run’ (‘Lola Rennt’, Tykwer, 1999). They both feature a train passing over a bridge. We can assume that both trains have more or less the same speed. Some example frames are shown in figure 3. Both cameras are non-static, with the main camera motion downwards. The trajectories of a few tracked points are shown in figure 3 right. Superimposed on the images, one can see one of the coordinate axes frames used, as well as a tracked point on the train. Figure 4 shows the computed distances between the backprojected lines based on this configuration. There is a clear diagonal line of small distances in the upper right of the figure, running over corresponding, i.e. synchronized frames. Further to the left, there is another region of lower values, caused by accidental line intersections. However, the distances here are not as small as for the diagonal further to the right. Moreover, the accidental line intersections do not occur according to a straight diagonal line. As a result, the system is still able to find the correct solution in spite of the accidental line intersections. This is illustrated by figure 5, which shows the average distance between backprojected lines as a function of the time delay. This corresponds to averaging the distances over diagonals in figure 4.
Correct synchronization line intersections

Correct synchronization

Figure 4: Compatibility matrix showing the distance between backprojected lines for frame \( i \) from the first sequence (horizontal axis) and frame \( j \) from the second sequence (vertical axis). Darker pixels represent smaller distances.

Figure 5: Scores obtained for different time delays, by averaging the distances over diagonals in figure 4.

The graph goes to zero at 89: the time delay between the two video sequences. Corresponding frames based on this time delay are shown in the two lower rows of figure 3.

Figure 6 shows similar graphs starting from different point configurations (each time rescaled between [0,1], since the actual values for different affine coordinate axes frames can vary greatly). The correct time delay always stands out clearly. This holds not only if the coordinate axes frame consists of static points (solid curves), as in the example of figure 3, but also in case of a moving coordinate axes frame. This means one does not need to know in advance which points are static (as long as at least one of them is moving).

Comparing the solid curves, one can clearly see that different coordinate axes frames lead to different computed distances (even though it is actually the same lines that are being used). This is due to the affine skewing. However, the values for the correct time delay are systematically close to zero, as expected.

Sometimes, there are also other points where a curve tends to zero. In our experiment, the minima at these locations were always higher than the one at \( t_d = 89 \), except for one case. For robustness, we suggest that in case of a second minimum with similar value as the global one (due to accidental line intersections, as shown in figure 4), one best tries out another point configuration to disambiguate the situation. None of the other minima is found consistently over different point configurations, so after trying out a few more configurations (possibly just a permutation of the original one) it is always possible to select the correct time delay. This effect of accidental line intersections cannot be avoided, but typically happens more often in small baseline cases, where the cameras and image points do not move over large distances, resulting in shallow intersection angles and limited variation within a set of lines, as is the case for this example. Other problems can be expected when the coordinate axes frame becomes degenerate, with all four points defining it lying in a plane. However, this case can be detected and again solved by simply shifting to a different point configuration (or, if it is just for a few frames, disregarding these frames in the further computations).

As a second example, look at some sample frames of two video sequences of a toy train shown in figure 7. The rightmost images show the point tracks, giving an idea of the camera motions (mainly moving downwards, but less smoothly than in the previous example). For this pair of sequences, we have ground truth information available (using a photo-flash to mark some frames right before and after the actual recording). From this, we know the actual time delay between both sequences is 45 frames. Superimposed on the images in figure 7, one can again see an example coordinate axes frame, as well as some other tracked points we used in this experiment. Based on this coordinate axes frame and the leftmost point on the foreground, we computed the distances between backprojected lines, as shown in figure 8. The average distance as a function of the estimated time delay is shown in figure 9. It reaches a minimum value for a time delay of -49 frames. This result is also shown by
the dashed line superimposed on figure 8, together with the line given by our groundtruth information, shown as a solid line. The deviation between both lines can be explained by the fact that the distance function shows a rather shallow minimum, making it sensitive to noise, tracking errors, perspective deformations, etc. However, if we repeat the same experiment for different point configurations (always selected from the 7 points shown in figure 8 and making sure the point on the train is always included) and average out the result, we obtain a final time delay of 45.53, accurately matching the groundtruth information.

5 Summary and Conclusions

In this contribution, we proposed a novel method to synchronize two video sequences. The strength of the method lies in the fact that it can deal with arbitrarily moving cameras and general 3D scenes. The basic algorithm only needs a configuration of 5 non-rigidly moving points, matched and tracked throughout both sequences. To increase robustness and accuracy, however, it is better to combine the outputs of several (not necessarily fully independent) sets of 5 points.

Apart from making the system more robust against occlusions (by introducing more matching to recover from lost features) and extending the framework to full projective, future research will focus mainly on the following points.

First, the issue on how to automatically select the point configuration to be used should be considered. Indeed, one needs to be sure that at least one of the points is moving relatively to the others. Factorization methods, such as [4] probably offer a way out in this respect. Also robustness to outliers (mismatched or mistracked points) still needs to
be included. More complex temporal transformations, up to non-constant time delays as in the work of [8] for static cameras, (e.g. using dynamic programming techniques) are another possible extension, as is subframe accurate synchronization.

Acknowledgements
We gratefully acknowledge financial support by the European project VIBES, the GOA project Marvel of K.U.Leuven Research Council, and the Fund for Scientific Research - Flanders (FWO-Vlaanderen).

References