Gil-Jiménez *et al. EURASIP Journal on Image and Video Processing* (2016) 2016:8
DOI 10.1186/s13640-016-0108-7

EURASIP Journal on Image and Video Processing
a SpringerOpen Journal

RESEARCH Open Access

Geometric bounding box interpolation: an alternative for efficient video annotation

Pedro Gil-Jiménez*, Hilario Gómez-Moreno, Roberto López-Sastre and Saturnino Maldonado-Bascón

Abstract

In video annotation, instead of annotating every frame of a trajectory, usually only a sparse set of annotations is provided by the user: typically its endpoints plus some key intermediate frames, interpolating the remaining annotations between these key frames in order to reduce the cost of the video labeling. While a number of video annotation tools have been proposed, some of which are freely available, and bounding box interpolation is mainly based on image processing techniques whose performance is highly dependent on image quality, occlusions, etc. We propose an alternative method to interpolate bounding box annotations, based on cubic splines and the geometric properties of the elements involved, rather than image processing techniques.

The algorithm proposed is compared with other bounding box interpolation methods described in the literature, using a set of selected videos modeling different types of object and camera motion. Experiments show that the accuracy of the interpolated bounding boxes is higher than the accuracy of the other evaluated methods, especially when considering rigid objects. The main goal of this paper is related with the bounding box interpolation step, and we believe that our design can be integrated seamlessly with any annotation tool already developed.

Keywords: Video annotation, Bounding box interpolation, Projective geometry, Cubic splines

1 Introduction

The growth in image and video processing demands larger quantities of annotated training data. A still image can be annotated at different levels. At the image level, annotations consist of simply indicating the objects present in the image, without any spatial information. At the pixel level, each pixel must be assigned to a label. Intermediate solutions may consist of drawing polygons surrounding the object or simple geometric shapes, such as rectangles or ellipses. The complexity, and thus, the cost or time required to annotate an image, depends on the level at which we need to annotate it.

Large-scale image databases are useful for supervised learning of visual object appearance. There are many publicly available databases. For instance, the PASCAL VOC Challenge [1] provides a standard dataset of annotated images at the bounding box level, for object categorization and detection. The images are collected from the flickr photo-sharing web site, and annotations are done at a single annotation "party", to ensure consistency. The ImageNet database [2] offers a collection of tens of millions of annotated images hierarchically organized, in this case, at the image level.

Many applications has been proposed in the literature for the task of image annotation, both for personal use and for massive annotations. In [3], Russell et al. proposed *LabelMe*, a web-based open tool that allows polygon-based image annotations. This tool can be accessed freely by any researcher, either to annotate their own images or to download the already annotated image collections. In [4], von Ahn and Dabbish took advantage of the *Games With A Purpose* (GWAP) philosophy to develop an online game which allows image annotation while the users are enjoying the game. In this case, images can only be annotated at the image level. The *Human Intelligence Task* (HIT) service provided by *Amazon Mechanical Turk* was explored by Sorokin and Forsyth [5] to outsource human annotation services at a low cost. This approach enables the annotation of large image databases at the bounding box level in a short time.

Video annotation can also be done at different levels. For instance, ANVIL [6] is a free tool that allows video

*Correspondence: pedro.gil@uah.es
Signal Theory and Communication Department, Universidad de Alcala, Crta Madrid-Barcelona km 33.600, Alcala de Henares, Spain

annotations at the frame level, that is, without any spatial information. However, when object location is required, the temporal evolution of the annotated objects must also be provided. Manually annotating the object location for all the frames in the video sequence is a tedious task. This is typically solved by allowing the user to provide only a limited set of annotations for a given object, then propagating this to the rest of the frames. The frames where the annotations are supplied are called *key frames*, and this set must usually include the first and last frames where the object is visible, in order for the system to know the endpoints of the object's trajectory.

The number of key frames needed depends on many factors, such as the object type (rigid or non-rigid), trajectory length (number of frames), type of object motion (straight, curved, or chaotic), and the accuracy required for the propagated annotations. The number and position of these key frames can be fixed or defined by the user. Vondrick et al. [7] discovered that 'a fixed key frame schedule is significantly faster than a user-defined key frame schedule.' However, the key frame frequency needs to be adjusted depending on the type of motion to annotate. When an object moves in a chaotic trajectory, more annotations are needed in order to accurately handle the chaotic motion. In this paper, however, we are not concerned with this issue and will use both schedules indiscriminately.

Propagation across frames has typically been performed by means of image and video processing techniques. In [8], Badrinarayanan et al. proposed a propagation scheme, given the pixel-level annotation of the first and last frames of a video sequence. The algorithm is based on a hidden Markov model and labels every pixel in all the frames between the endpoints. In [9], Liu et al. proposed a contour tracker to track the contour provided by the user in the first frame. The user can specify a time-varying depth value for each annotated object, intended to handle occlusions and 3D linear interpolation. The user may also be requested for an inspection to correct tracker mistakes. Contour tracking is also used in [10] to track annotated curves in a video sequence for *rotoscoping*, i.e., the process of tracking shapes throughout a video sequence. In [11], Kalal et al. proposed a long-term tracker of a bounding box provided by the user in the first frame. The system is able to automatically re-detect the object after a period when it is not in the field of view or after tracking failure, by learning the object appearance from the previous frames.

Propagation can also be done by means of the geometric properties of the elements involved. In this case, the user provides a set of simple (rectangle, ellipse, etc.) or complex shapes, and the system computes the coordinates of the polygon for the remaining frames. For this case, the main techniques described in the literature will be analyzed in Section 2, where different publicly available annotation tools will be introduced. Section 3 describes the alternative procedure proposed in this paper. Section 4 describes the experiments performed to compare the existing software applications. Section 5 concludes the paper.

2 Related works

In this paper, we propose an alternative method to propagate bounding box annotations between key frames. For that reason, we will analyze here the most popular software packages for video annotation at the bounding box level, focusing on the propagation algorithms proposed in each.

2.1 Available software

VIdeo Performance Evaluation Resource (ViPER) is one of the earliest video annotation tools provided by the research community [12]. The initial goal was to create a flexible ground truth format and a tool for ground truth generation and sharing. It is a JAVA application that allows the user to define object categories (people, cars, animals, etc.) and different attributes, including spatial ones (bounding box, bounding ellipse, points, etc.). The software includes a default linear interpolation utility. For this interpolator, if we define $\mathbf{b}_i$ to be the coordinates of a bounding box at $t = t_i$, then

$$\mathbf{b}_i = \begin{bmatrix} x_i & y_i & w_i & h_i \end{bmatrix}, \tag{1}$$

where (x_i, y_i) are the coordinates of the center of the bounding box and (w_i, h_i) are its width and height. Given two consecutive annotations $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$, the coordinates of the bounding box for a given time $t = t_j$ are

$$\mathbf{b}_j = \left(\frac{t_{i+1} - t_j}{t_{i+1} - t_i}\right)\mathbf{b}_i + \left(\frac{t_j - t_i}{t_{i+1} - t_i}\right)\mathbf{b}_{i+1}. \tag{2}$$

Linear interpolation can be defined as the simplest approach [7] for label propagation. This method has two main drawbacks. On the one hand, linear interpolation can only handle straight trajectories, since interpolation between adjacent segments are computed independently. On the other hand, it can not accurately capture scale transformations due to perspective effects. A recent update can be found in [13], which allows semiautomatic video annotation. Nevertheless, the system is based on a foreground/background motion detection, which limits its use to videos recorded with static cameras.

Vondrick et al. proposed in [7] a user interface to be deployed on Amazon Mechanical Turk (MT) to outsource bounding box annotations using its HIT services. In their work, they analyzed several aspects related to this HIT service, such as interface design, the evaluation of a worker's talents, and the optimal key frame frequency. Focusing on the bounding box propagation technique, they proposed two different algorithms, the simplest one being the linear interpolation discussed above. The other method is based on dynamic programming, using a visual model of the annotated object. They extract the model from the N bounding boxes provided, compute a feature descriptor composed of HOG and color features, and train a linear SVM classifier from positive and negative (background) samples. The main drawback is that the algorithm is highly dependent on the image quality, and the system can even fail in the case of object occlusions. The system includes a constant to allow the visual interpolator to gracefully degrade to linear interpolation when the former becomes inaccurate.

Instead of relying on visual properties, the interpolation can be done using the geometric properties of the elements involved. In this case, when interpolating coordinates, we can use image or spatial coordinates.

Consider a point $\mathbf{Q}$ moving in front of a camera, as in Fig. 1. At $t = t_i$, the point will be located at $\mathbf{Q}_i$ in spatial coordinates, and its image will be at $\mathbf{q}_i$, and similarly for $t = t_{i+1}$. If we suppose constant motion for the point, at $t_j = (t_i + t_{i+1})/2$, the point will be located at $\mathbf{Q}_j$, at the intermediate position between the points in space and its image at $\mathbf{q}_j$. However, if we interpolate using image coordinates, the point $\hat{\mathbf{q}}_j$ will be located at the intermediate position between the image points, and as can be seen in Fig. 1, the coordinates $\mathbf{q}_j$ and $\hat{\mathbf{q}}_j$ do not coincide, due to the effects of perspective.

Perspective effects get canceled only when the object moves parallel to the image plane. These effects increase as the component of the motion normal to the image plane increases. For instance, in Fig. 2, we can see two annotated trajectories. Object `1` moves almost parallel to the image plane, so that the perspective effects are minimal and interpolation using image coordinates can be an accurate solution.

However, parallel motion is not the general situation. For object `2` in Fig. 2, the variation in depth is large. In this situation, the last assumption does not hold, and the accuracy in the bounding box interpolation decreases, as we will see later, in Section 4.

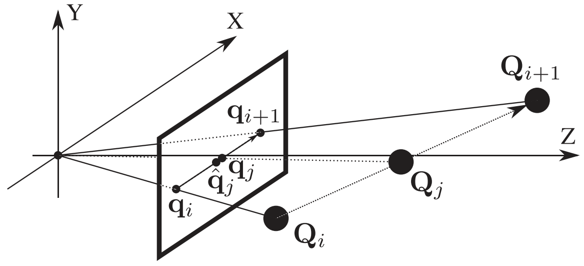

Fig. 1 Interpolation of the intermediate position between $\mathbf{Q}_i$ and $\mathbf{Q}_{i+1}$ using image ($\hat{\mathbf{q}}_j$) and spatial ($\mathbf{q}_j$) coordinates

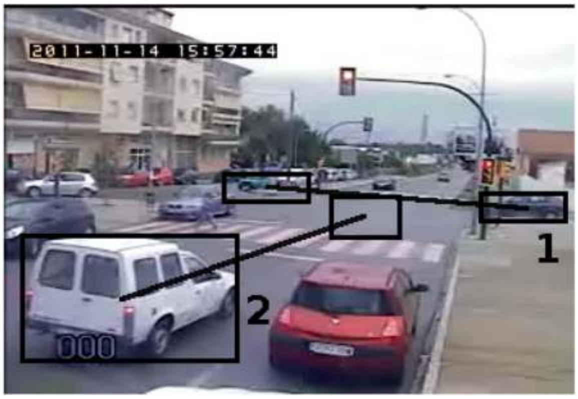

Fig. 2 Perspective effects for an object which moves parallel (*1*) and for an object which does not move parallel (*2*) to the image plane

To take into account this effect, Yuen et al. [14] pointed out that 'a constant velocity in space does not project to a constant velocity in the image plane, due to perspective effects.' In their work, they proposed an alternative method to project the coordinates of the annotated object to the image plane for any time between two key frames.

Assuming that an object in space is moving with constant motion between two key frames, any point of the object will therefore move in a straight line. The line in the image plane joining the same point of the object in both key frames will be the image of the line followed by this point in space. If all the points of the object move in parallel trajectories, all points in space will meet at infinity, and so the cross point in the image plane of all lines will be the vanishing point for this trajectory. The coordinates of any point of the object for a given time t is computed using

$$(x(t), y(t)) = \left(\frac{x_0 + \lambda(t)x_{\mathrm{v}}}{\lambda(t) + 1}, \frac{y_0 + \lambda(t)y_{\mathrm{v}}}{\lambda(t) + 1} \right) \tag{3}$$

where (x_0, y_0) are the coordinates of the point in the frame at $t = 0$, and $p_{\mathrm{v}} = (x_{\mathrm{v}}, y_{\mathrm{v}})$ are the coordinates of the vanishing point computed as described above. $\lambda(t)$ is the displacement of the point along the direction of the motion in space, and since constant motion is assumed, we have $\lambda(t) = vt$, where v is the velocity of the point in space and can be computed, given a second key frame at $t = t_1$, using

$$v = \frac{x_1 - x_0}{t_1 \cdot (x_v - x_1)}. \tag{4}$$

This approach has two main drawbacks. Suppose that we have two bounding boxes of an object at $t = t_i$ and $t = t_{i+1}$, as in Fig. 3, and we trace the lines joining the four pair of corners. As stated before, the cross point would correspond to the vanishing point of the trajectory. However, in the general situation, annotation errors and changes in object aspect ratio will make the lines fail

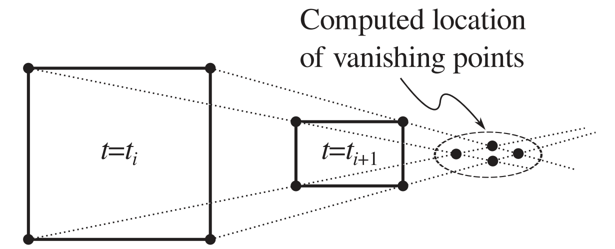

Fig. 3 Errors in computation of the vanishing point from a pair of bounding boxes

to meet at the same point. For some points in the last figure, some of the cross points are even located outside the image plane. Furthermore, for motion parallel to the image plane, the vanishing point will lie at infinity, so that in (3), there is an indetermination that must be resolved some other way. The second drawback is that this schema can only be employed between a pair of key frames, while assuming straight paths in between, which is not a correct assumption for object motion in general.

The second drawback is solved by Lee et al. in [15, 16] by interpolation using cubic splines [17]. In their work, they proposed a tool for collaborative video annotation, mainly intended for interactive services (i.e., smart TV) in the multimedia industry. They also proposed a dynamic sampling-based cubic spline interpolation (DSCSI) algorithm to reduce the size of the trajectory data on a "clickable video". Focusing on the interpolation part, the object position is computed using cubic spline interpolation (CSI) from a sparse set of annotations. Nevertheless, the interpolation is performed using image coordinates instead of spatial coordinates, so that, as explained above, it is not able to accurately model object trajectories with large depth variations.

To address these problems, in this paper, a new interpolation algorithm that combines a 3D reconstruction schema is proposed, similar to the one proposed in [14], with the cubic spline interpolation used in [16]. However, the main difference with respect to [14] is that the algorithm proposed in our paper performs a continuous reconstruction in the spatial coordinates, as will be described in Section 3.3. This improvement allows us to use cubic splines to interpolate the spatial bounding box coordinates, instead of using linear interpolation, to model object trajectories. The main difference with respect to [16] is that the proposed 3D reconstruction allows us to model more accurately the trajectory of objects when they exhibit large depth variations.

3 Geometric interpolation

In this section, we will describe the process designed to compute the bounding box coordinates for any frame from the known bounding box annotations. The overall schema is depicted in Fig. 4. Suppose we have two annotations $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ of an object, the key idea is finding the coordinates $\mathbf{B}_i$ and $\mathbf{B}_{i+1}$ of the bounding box in space. Having that, computing the coordinates $\mathbf{b}_j$ of the bounding box in the image plane for any time $t = t_j$ is straightforward, assuming we know the motion model of the object in space.

We will demonstrate that using only bounding box coordinates, we can perform a 3D reconstruction of the bounding box positions. If the camera's internal parameters are not known, i.e., we are working with an uncalibrated camera, the 3D positions can be computed only up to a scale transformation. Nevertheless, we will see that actual 3D coordinates are irrelevant for bounding box interpolation, and so, no camera calibration is needed.

Prior to defining the interpolation schema proposed in this paper, we introduce briefly the basic concepts related with central projection and the camera model needed. The notation has been taken from [18] where possible. We will first describe the interpolation technique for a simpler two-dimensional case and then extend the concepts to three dimensions.

3.1 Two-dimensional camera model

In the two-dimensional case, central projection is based on the *line camera* shown in Fig. 5. This camera projects points on the plane XZ onto an *image line*. Let the center of projection be the point $\mathbf{C} = (X_C, Z_C, 1)^T$, expressed in homogeneous coordinates, and let $\mathbf{l} = (a, b, c)^T$ be the homogeneous vector representing the image line. The distance from the line $\mathbf{l}$ to the point $\mathbf{C}$ is f, which is called the *focal length* of the camera. The line from the center of projection, normal to the image line, is called the *principal axis* of the camera, and the cross point between the image line and the principal axis is called the *principal point*.

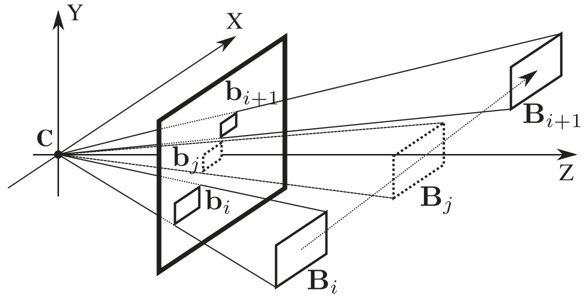

Fig. 4 Overview of 3D interpolation technique. Given annotations $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$, the computation of $\mathbf{b}_j$ is performed by a 3D reconstruction of those given annotations and the projection onto the image plane of the interpolated annotation in space ($\mathbf{B}_j$)

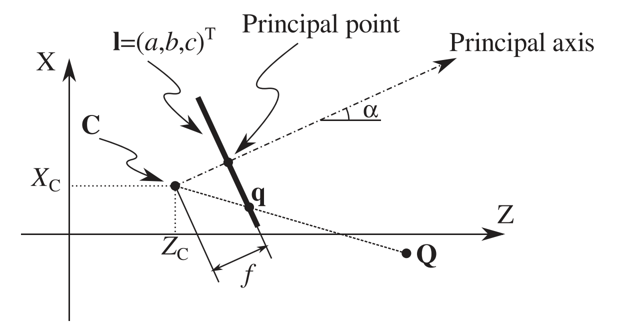

Fig. 5 Model for a two-dimensional projective camera

Under this model, a point $\mathbf{Q} = (X_{\mathrm{Q}}, Z_{\mathrm{Q}}, 1)^{\mathrm{T}}$ in the plane is mapped to the point $\mathbf{q} = (x_{\mathrm{q}}, w_{\mathrm{q}})^{\mathrm{T}}$ in homogeneous coordinates in the image line. This point will be located where a line joining the point $\mathbf{Q}$ with the center of projection $\mathbf{C}$ meets the image line. The homogeneous coordinates of $\mathbf{q}$ can be computed from the *projection matrix* P:

$$\mathrm{P} = \mathrm{K} \cdot \mathrm{R} \cdot [\,\mathrm{I} \mid -\widetilde{\mathbf{C}}] \tag{5}$$

where K is the 2 × 2 camera *internal parameter* matrix, R is the 2 × 2 camera *rotation* matrix, I is the 2 × 2 identity matrix, and $\widetilde{\mathbf{C}} = \{X_{\mathrm{C}}, Z_{\mathrm{C}}\}$ is the center of projection expressed in inhomogeneous coordinates. In order to simplify the problem, suppose that the camera is placed at a canonical position, that is, suppose that the center of projection is the origin of the coordinate system, $\mathbf{C} = (0, 0, 1)^{\mathrm{T}}$, and that the principal axis is the Z axis ($\alpha = 0$), as in Fig. 6. Furthermore, we can have the principal point be the origin of the image coordinate system. In this situation, P reduces to

$$\mathrm{P} = \begin{pmatrix} f & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}. \tag{6}$$

A point $\mathbf{Q} = \left(X_{\mathrm{Q}}, Z_{\mathrm{Q}}, 1\right)^{\mathrm{T}}$ in the plane is projected onto the image line at

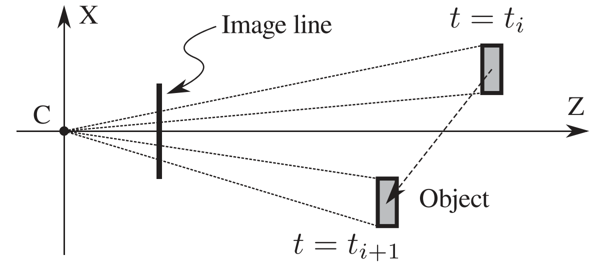

Fig. 6 An object moving in the plane *XZ*, with the camera placed at a canonical position

$$\mathbf{q} = \begin{pmatrix} x_{\mathrm{q}} \\ w_{\mathrm{q}} \end{pmatrix} = \mathrm{P} \cdot \mathbf{Q} = \begin{pmatrix} f \cdot X_{\mathrm{Q}} \\ Z_{\mathrm{Q}} \end{pmatrix} \tag{7}$$

which can be expressed in inhomogeneous coordinates as

$$\widetilde{q} = \frac{f \cdot X_{\mathrm{Q}}}{Z_{\mathrm{Q}}}. \tag{8}$$

3.2 Reconstruction for a single segment

Now, consider an object moving in front of a line camera from $t = t_i$ to $t = t_{i+1}$, as shown in Fig. 6. In this section, we will consider a trajectory defined by only one segment, that is, only two key frames, at the start and endpoints of the trajectory, without intermediate key frames. In the following sections, we will extend the method to any number of segments.

If we call a the lower edge of the object, and b the upper one, as can be seen in Fig. 7, the edges of the object will have inhomogeneous coordinates $\widetilde{\mathbf{Q}}_i^{\mathrm{a}} = \{X_i^{\mathrm{a}}, Z_i\}$ and $\widetilde{\mathbf{Q}}_i^{\mathrm{b}} = \{X_i^{\mathrm{b}}, Z_i\}$, respectively. Note that we have assumed the object to be parallel to the image line, thus the coordinate Z for both points is the same. Under a camera matrix P, the point a is imaged onto the image line at

$$\widetilde{q}_i^{\mathrm{a}} = \frac{f \cdot X_i^{\mathrm{a}}}{Z_i} \tag{9}$$

as shown in Fig. 8. If the shift of the object from t_i to t_{i+1} is $\Delta_i = \{\Delta X_i, \Delta Z_i\}$, then the coordinates for point a at $t = t_{i+1}$ will be

$$\widetilde{\mathbf{Q}}_{i+1}^{\mathrm{a}} = \widetilde{\mathbf{Q}}_i^{\mathrm{a}} + \Delta_i \tag{10}$$

and will be imaged by the camera at

$$\widetilde{q}_{i+1}^{\mathrm{a}} = \frac{f \cdot (X_i^{\mathrm{a}} + \Delta X_i)}{Z_i + \Delta Z_i} \tag{11}$$

and similarly for point b. Gathering all the equations, we have

$$\begin{cases} \widetilde{q}_i^{\mathrm{a}} = \frac{f \cdot X_i^{\mathrm{a}}}{Z_i} \\ \widetilde{q}_i^{\mathrm{b}} = \frac{f \cdot X_i^{\mathrm{b}}}{Z_i} \\ \widetilde{q}_{i+1}^{\mathrm{a}} = \frac{f \cdot (X_i^{\mathrm{a}} + \Delta X_i)}{Z_i + \Delta Z_i} \\ \widetilde{q}_{i+1}^{\mathrm{b}} = \frac{f \cdot \left(X_i^{\mathrm{b}} + \Delta X_i\right)}{Z_i + \Delta Z_i}. \end{cases} \tag{12}$$

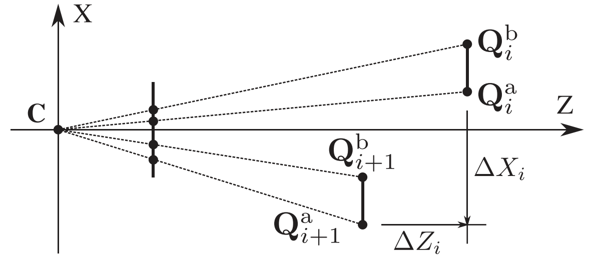
X
$\mathbf{Q}_i^{\mathrm{b}}$
$\mathbf{Q}_i^{\mathrm{a}}$
C
Z
$\mathbf{Q}_{i+1}^{\mathrm{b}}$
ΔX_i
$\mathbf{Q}_{i+1}^{\mathrm{a}}$
ΔZ_i

Fig. 7 Point coordinates as the object moves from $t = t_i$ to $t = t_{i+1}$

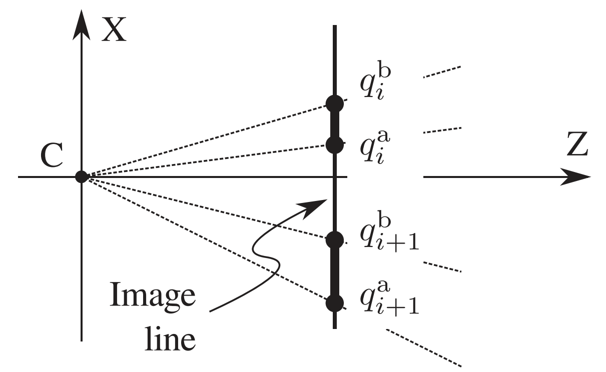

Fig. 8 Projection of the object points onto the camera image line

Before going on, we will analyze the effect of the focal length *f*. As can be seen in Fig. 9, for an image point $\mathbf{q}_i$ at $t = t_i$, if we assume the point depth to be $Z = Z_i$ and use a camera with focal length *f*, the point is back projected to $\widetilde{\mathbf{Q}}_i = \{X_i, Z_i\}$. Now, changing the camera focal length to $f' = k_{\mathrm{E}} \cdot f$ while keeping the same depth Z_i, the point coordinates will change to $\widetilde{\mathbf{Q}}_i' = \{X_i', Z_i\}$. By similar triangles, we have

$$\frac{\widetilde{q}_i}{f} = \frac{X_i}{Z_i} \tag{13}$$

and

$$\frac{\widetilde{q}_i}{f'} = \frac{X_i'}{Z_i} \tag{14}$$

which gives $X_i = k_{\mathrm{E}} \cdot X_i'$, and the same for $t = t_{i+1}$. Since our final interest is not the reconstruction itself, but the computation of the images of intermediate positions, it can easily be seen that the interpolation is independent of the particular focal length chosen. For instance, if we interpolate the image coordinates for the point at the middle position between t_i and t_{i+1}, assuming constant motion, the coordinates for the point when focal length is f will be

$$\widetilde{\mathbf{Q}}_j = \left\{ \frac{X_i + X_{i+1}}{2}, \frac{Z_i + Z_{i+1}}{2} \right\} \tag{15}$$

which will be imaged at

$$\widetilde{q}_j = f \left(\frac{X_i + X_{i+1}}{Z_i + Z_{i+1}} \right). \tag{16}$$

In the same way, if the focal length is f', the interpolated position will be

$$\widetilde{q}_j' = f' \left(\frac{X_i' + X_{i+1}'}{Z_i + Z_{i+1}} \right) = f' \left(\frac{X_i + X_{i+1}}{k_{\mathrm{E}} \cdot (Z_i + Z_{i+1})} \right) = \widetilde{q}_j, \tag{17}$$

that is, both points are imaged onto the same image coordinate. Figure 9 shows a graphical example.

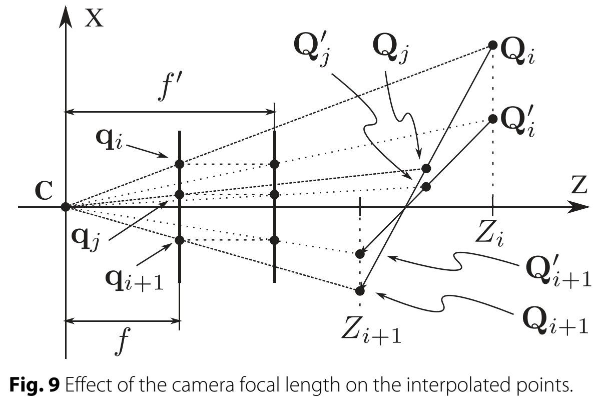

Fig. 9 Effect of the camera focal length on the interpolated points. The inhomogeneous coordinate $\widetilde{q}_j$ of the interpolated point is independent of the particular focal length *f* chosen

The last proof shows that the coordinate of the interpolated points is independent of the particular focal length chosen, and therefore, the internal camera parameters are not needed. Thus, to simplify the problem, we choose $f = 1$. Reorganizing the initial equations,

$$\begin{cases} \widetilde{q}_i^{\mathrm{a}} Z_i - X_i^{\mathrm{a}} = 0 \\ \widetilde{q}_i^{\mathrm{b}} Z_i - X_i^{\mathrm{b}} = 0 \\ \widetilde{q}_{i+1}^{\mathrm{a}} Z_i + \widetilde{q}_{i+1}^{\mathrm{a}} \Delta Z_i - X_i^{\mathrm{a}} - \Delta X_i = 0 \\ \widetilde{q}_{i+1}^{\mathrm{b}} Z_i + \widetilde{q}_{i+1}^{\mathrm{b}} \Delta Z_i - X_i^{\mathrm{b}} - \Delta X_i = 0 \end{cases}. \tag{18}$$

The last result forms a system of linear equations in the unknowns $\mathbf{B}_i =[X_i^{\mathrm{a}}, X_i^{\mathrm{b}}, Z_i, \Delta X_i, \Delta Z_i]$, which can be computed from its null space. Since (18) is a homogeneous system of equations, $\mathbf{B}_i$ can be found up to a constant factor k_{s}. The meaning from the geometric point of view is that the space is reconstructed up a scale transformation. However, under central projection, two points related by a scale transformation are imaged onto the same image coordinates. This means that the particular constant factor k_{s} chosen for $\mathbf{B}_i$ is not important, since only the relation between all variables is important. We will come back to this concept when analyzing trajectories composed of more than one segment.

With a little computation from (18), we get

$$\begin{cases} \left(\widetilde{q}_{i+1}^{\mathrm{a}} - \widetilde{q}_i^{\mathrm{a}}\right) Z_i + \widetilde{q}_{i+1}^{\mathrm{a}} \Delta Z_i - \Delta X_i = 0 \\ \left(\widetilde{q}_{i+1}^{\mathrm{b}} - \widetilde{q}_i^{\mathrm{b}}\right) Z_i + \widetilde{q}_{i+1}^{\mathrm{b}} \Delta Z_i - \Delta X_i = 0 \end{cases} \tag{19}$$

and

$$\begin{cases} X_i^{\mathrm{a}} = \widetilde{q}_i^{\mathrm{a}} Z_i \\ X_i^{\mathrm{b}} = \widetilde{q}_i^{\mathrm{b}} Z_i \end{cases}. \tag{20}$$

Solving (19) using the cross product, we obtain

$$\begin{cases} Z_i = \widetilde{q}_{i+1}^{\mathrm{b}} - \widetilde{q}_{i+1}^{\mathrm{a}} = s_{i+1} \\ \Delta Z_i = \left(\widetilde{q}_i^{\mathrm{b}} - \widetilde{q}_i^{\mathrm{a}}\right) - \left(\widetilde{q}_{i+1}^{\mathrm{b}} - \widetilde{q}_{i+1}^{\mathrm{a}}\right) = s_i - s_{i+1} \\ \Delta X_i = \widetilde{q}_i^{\mathrm{b}} \widetilde{q}_{i+1}^{\mathrm{a}} - \widetilde{q}_i^{\mathrm{a}} \widetilde{q}_{i+1}^{\mathrm{b}} \end{cases} \tag{21}$$

where $s = \widetilde{q}^{\mathrm{b}} - \widetilde{q}^{\mathrm{a}}$ is the size of the object in the image line. Finally, (20) can be solved using these last results:

$$\begin{cases} X_i^{\mathrm{a}} = \widetilde{q}_i^{\mathrm{a}} \cdot s_{i+1} \\ X_i^{\mathrm{b}} = \widetilde{q}_i^{\mathrm{b}} \cdot s_{i+1} \end{cases} . \tag{22}$$

3.3 Reconstruction from multiple segments

The results obtained in the previous section allow us to interpolate the coordinates for any frame between two key frames. For a trajectory composed of N key frames, we could apply the same process independently for the $N - 1$ segments. However, since the computations are independent, the results would give disjoint reconstructed segments. To see this, suppose we have three different annotations for an object, $\mathbf{b}_i$, $\mathbf{b}_{i+1}$, and $\mathbf{b}_{i+2}$, as in Fig. 10.

The reconstruction for the segment defined by $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ gives $\mathbf{B}_i$ and $\mathbf{B}_{i+1}$. Likewise, for points $\mathbf{b}_{i+1}$ and $\mathbf{b}_{i+2}$ we have $\mathbf{B}'_{i+1}$ and $\mathbf{B}'_{i+2}$. The reconstructed segments will be, in general, disjoint. Although both reconstructions are correct, they are only useful if further interpolations are performed independently for each segment, i.e. assuming straight paths between key frames. However, typical object motion is better modeled with interpolation schemes which need more samples. As we will see later, we will use cubic spline interpolation, a method which uses more than two samples to compute any interpolation. To this end, we need the reconstructed trajectory to be continuous along the whole path.

Continuity can be achieved easily taking into account that reconstruction can only be computed up to a scale transformation. We can adjust the scale factor k_{s}, introduced before, for the second segment to have $\hat{\mathbf{B}}'_{i+1} = \mathbf{B}_{i+1}$:

$$\hat{\mathbf{B}}'_{i+1} = k_i^{\mathrm{s}} \cdot \mathbf{B}'_{i+1} = \mathbf{B}_{i+1} \tag{23}$$

where $k_i^{\mathrm{s}} = Z_{i+1}/Z'_{i+1}$ is the scale factor to be applied to the second segment, that is, $\hat{\mathbf{B}}'_{i+1} = k_i^{\mathrm{s}} \cdot \mathbf{B}'_{i+1}$ and $\hat{\mathbf{B}}'_{i+2} = k_i^{\mathrm{s}} \cdot \mathbf{B}'_{i+2}$. Figure 11 shows how, applying this scale correction, we can make the trajectory continuous. Furthermore, for the next segment, a new scale factor $k_{i+1}^{\mathrm{s}} = Z'_{i+2}/Z''_{i+2}$ must be computed, so that the coordinates for that segment will be $\hat{\mathbf{B}}''_{i+2} = k_i^{\mathrm{s}} \cdot k_{i+1}^{\mathrm{s}} \cdot \mathbf{B}''_{i+2} = k_i^{\mathrm{s}} \cdot \mathbf{B}'_{i+2}$, and so on.

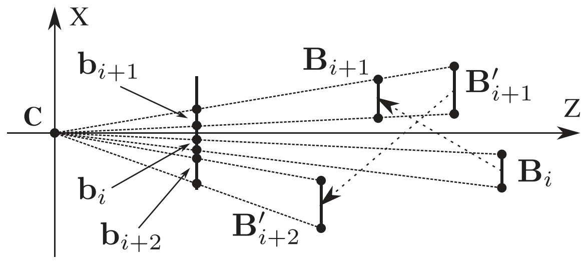

Fig. 10 Independent reconstruction for two segments of an object trajectory

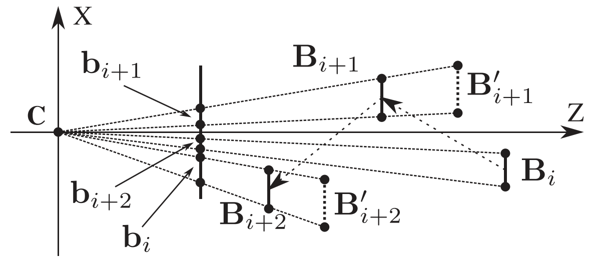

Fig. 11 Scale correction for the second segment to build a continuous trajectory

3.4 Three-dimensional case

Extrapolating the problem to three dimensions is not straightforward. A first approach could be to perform the reconstruction independently for the horizontal and vertical coordinates, but this would yield different depth values (Z_i and ΔZ_i) for the horizontal and vertical reconstructions. Although we can adjust the scale factor k_{s} of one of the coordinates to make the depth values Z_i equal for both coordinates, the other value, ΔZ_i, will, in general, be different and so will be the object depth for $t = t_{i+1}$, but, obviously, the depth position should be the same for both coordinates along the whole path, since it is the same point.

The depth values for the horizontal and vertical coordinates will only be equal when the bounding box aspect ratio is kept constant between annotations. Thus, to improve the performance, we have to correct the aspect ratio variation between key frames. To see this, suppose given an object annotation for $t = t_i$ and $t = t_{i+1}$ as in Fig. 12. As the object moves away from the camera, its size will decrease and inversely. If the aspect ratio of the annotated rectangle was kept constant between the key frames, this would be the only variation on both height and width.

This can easily be checked if we look at the equations in (21) for the computation of Z and ΔZ. Applying the

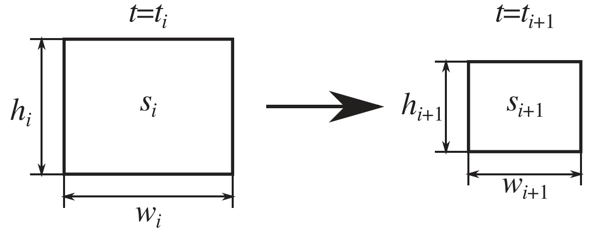

Fig. 12 Horizontal and vertical variables for an object in two consecutive annotations

equation for the computation of the depth information, substituting $s \rightarrow w$ for the horizontal coordinate, and $s \rightarrow h$ for the vertical coordinate, we have $Z_i^x = w_{i+1}$ and $\Delta Z_i^x = w_i - w_{i+1}$ for the x coordinate and $Z_i^y = h_{i+1}$ and $\Delta Z_i^y = h_i - h_{i+1}$ for the y coordinate. We can use the scale correction factor k_s introduced before to have $\hat{Z'}_i^y = k_i^s \cdot Z_i^y = Z_i^x$, that is, the depth position at the beginning of the segment will be the same for the horizontal and vertical reconstructions. In this situation, the depth shift for the vertical reconstruction will be

$$\Delta \hat{Z'}_i^y = k_i^\mathrm{s} \cdot \Delta Z_i^y = \frac{Z_i^x}{Z_i^y} \cdot \Delta Z_i^y = \frac{w_{i+1}}{h_{i+1}}(h_i - h_{i+1}). \quad (24)$$

If both rectangles have the same aspect ratio,

$$\frac{h_{i+1}}{h_i} = \frac{w_{i+1}}{w_i} \quad (25)$$

and after a little calculation, we obtain

$$\Delta \hat{Z'}_i^y = \frac{w_i}{h_i}(h_i - h_{i+1}) = w_i - w_{i+1} = \Delta Z_i^x, \quad (26)$$

that is, the computed depths for the x and y coordinates are the same, as desired. However, if the aspect ratio is not maintained, the last equation does not hold, and so, the depth shifts for the horizontal and vertical reconstructions do not coincide.

The aspect ratio of an imaged object is only maintained when considering planar rigid objects that move without rotating. But even in this hypothetical case, errors in object annotations will generally result in the bounding box aspect ratio not being maintained. When annotating non-rigid objects, the problem gets worse. For instance, consider a person standing up in one key frame, and the same person crouching in the next key frame. In that situation, the bounding box height will halve, whereas the width will be the same, or even increase.

In order to improve performance, we need to take such aspect ratio variations into account. Again, consider the object annotations given in Fig. 12. As we saw before, one of the factors that make the height and width of the object change is depth variation. If we denote by k_i^d the size variation from t_i to t_{i+1} due to the change in depth and compute it from the geometrical mean of its width and height,

$$k_i^\mathrm{d} = \sqrt{\frac{h_{i+1} \cdot w_{i+1}}{h_i \cdot w_i}}, \quad (27)$$

the variation in height will be

$$h_{i+1} = k_i^\mathrm{d} \cdot k_i^\mathrm{h} \cdot h_i, \quad (28)$$

and similarly for its width. In the last equation, k_i^h (and k_i^w for the width) is the variation in height not included in the size variation due to the change in depth k_i^d. This factor can be computed by

$$k_i^\mathrm{h} = \frac{h_{i+1}}{k_i^\mathrm{d} \cdot h_i} = \sqrt{\frac{w_i \cdot h_{i+1}}{w_{i+1} \cdot h_i}} \quad (29)$$

for the height, and

$$k_i^\mathrm{w} = \frac{w_{i+1}}{k_i^\mathrm{d} w_i} = \sqrt{\frac{w_{i+1} \cdot h_i}{w_i \cdot h_{i+1}}} = \frac{1}{k_i^\mathrm{h}}. \quad (30)$$

That is, the factors are inversely proportional. Thus, to complete the system, we only have to take into account the last variable k_i^h along the trajectory, modifying the width and height of the bounding box of the annotated rectangle correspondingly. Since its value can change along the different segments of the trajectory, interpolation of this variable between key frames needs to be considered to eliminate abrupt size changes between segments.

3.5 Bounding box interpolation

The process described so far allows us to perform, up to a scale transformation, a three-dimensional reconstruction of the annotated trajectory. The last step is the design of the interpolation schema to compute the bounding box coordinates for any frame. First of all, thanks to the reconstruction process described, the interpolation is not performed over the image plane coordinates but over the reconstructed spatial coordinates. Once we have the interpolated 3D coordinates, we project them over the image plane to get the new bounding box coordinates. In Section 3.1, we chose the camera to have its center of projection at the origin of the system of coordinates, with its principal axis the Z axis and with focal length equal to 1. Since this is the camera employed in the reconstruction, it will be the one to project the interpolated coordinates to the image plane.

In order to model the actual object motion, instead of using linear interpolation between key frames, we use cubic spline interpolation, as can be seen in Fig. 13. Since we are using spatial coordinates, the three coordinates X, Y, and Z need to be interpolated. Furthermore, to eliminate abrupt changes in object size, the aspect ratio correction k_i^h also needs to be interpolated. So, a four cubic spline interpolators need to be implemented. Algorithm 1 summarizes the whole process needed to interpolate bounding box coordinates from a sparse set of annotations.

Algorithm 1: Algorithm for the interpolation of bounding box coordinates combining 3D reconstruction and cubic splines.

```
function Interpolation ($T, j$);
Input  : $T = [\mathbf{b}_0, \mathbf{b}_1, \ldots \mathbf{b}_N]$ where
         $\mathbf{b}_i = [x_i, y_i, w_i, h_i]$: list of $N + 1$ bounding
         box annotations for a given trajectory
         $j$: Required interpolation time
Output: $\mathbf{b}_j$: bounding box coordinates for time $j$
if $\mathbf{b}_j \in T$ then
    return $\mathbf{b}_j$
else
    Perform 3D reconstruction to obtain
    $\mathrm{T}' = [\mathbf{B}_0, \mathbf{B}_1, \ldots \mathbf{B}_N]$, where
    $\mathbf{B}_i = [X_i, Y_i, Z_i . k_i^{\mathrm{h}}]$
    Compute $\mathbf{B}_j = [X_i, Y_i, Z_i]$ from $\mathbf{T}'$ using a 4
    cubic spline interpolator
    Project $\mathbf{B}_j$ to the image plane to obtain $\mathbf{b}_j$,
    using projection matrix $\mathrm{P} = [\mathrm{I}|0]$
    return $\mathbf{b}_j$
end
```

Although cubic splines can accurately model general object motion, they are not able to model abrupt motion changes, like a ball bouncing on the floor. This can easily be solved by allowing the user to insert *trajectory breaks* at these points, so that the interpolation is computed independently at either side of the break. Figure 14 shows the improvement when breaks are included in the trajectory annotation.

4 Results and discussion

In order to test the interpolation schema described in this paper, the proposed algorithm has been implemented in Python, using OpenCV. For the cubic spline interpolation, we used the functions implemented in the SciPy library [19]. For comparison purposes, the following algorithms have been implemented and tested:

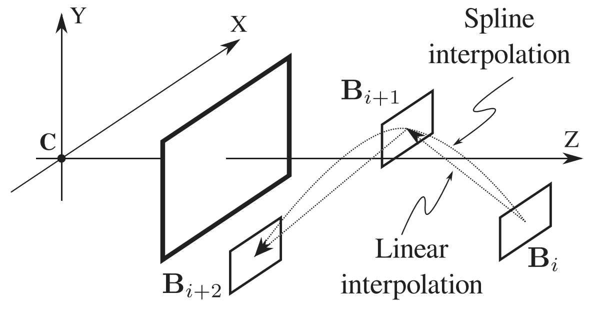

Fig. 13 Comparison between linear and spline interpolation over the reconstructed spatial coordinates

1. *Geometric cubic spline interpolation* (GC): This is the algorithm described in this paper.
2. *Geometric interpolation* (GI): A simplified version of the algorithm described, without using cubic spline interpolation.
3. *Linear interpolation* (LI): A linear interpolator, as defined in Eq. (2). This is the interpolation used in ViPER [12].
4. *LabelMe* (LM): Although the original work was designed for general polygonal annotations [14], the algorithm can be easily adapted to work with rectangles. For the computation of the vanishing point p_{v}, we used the cross point between the lines defined by the top-left and the bottom-right corners of consecutive rectangles. Velocity and point coordinates are computed according to (4) and (3).
5. *Cubic spline interpolation* (CS): Following the original work by Lee et al. [15], the center coordinates of the bounding box and the rectangle size in the image plane as well are interpolated between key frames using the cubic spline functions implemented in the SciPy library.

To test the performance of the interpolation algorithms, 19 short (a few seconds long) video clips were selected and manually annotated. The clips were extracted from surveillance and broadcasted video and personal cameras. All the videos, annotations, source code, and a Python application can be downloaded from the project web page [20]. The objects were carefully annotated in every frame (which can involve more than one hundred frames each). Table 1 summarizes the main properties of the videos selected for the test. The selection includes different object and/or camera motion types (see column *motion type* in this table), to check the algorithms against a number of situations. These include the following:

(a) Rigid object moving in straight path, static camera
(b) Rigid object moving in curved path, static camera
(c) Static object, with camera panning
(d) Static object, with camera translation
(e) Static object, with camera zoom
(f) Non-rigid objects

Also, Fig. 15 shows some sample images of the video data set.

The accuracy of the bounding box interpolation has been evaluated by a decimation followed by a further interpolation of the decimated samples, according to the following process. Once a complete trajectory is read, one half of the samples (the odd or the even samples) are removed, and their coordinates are interpolated using the implemented algorithms. For each computed bounding box $\mathbf{b}_i$, we calculate the *overlap error* as

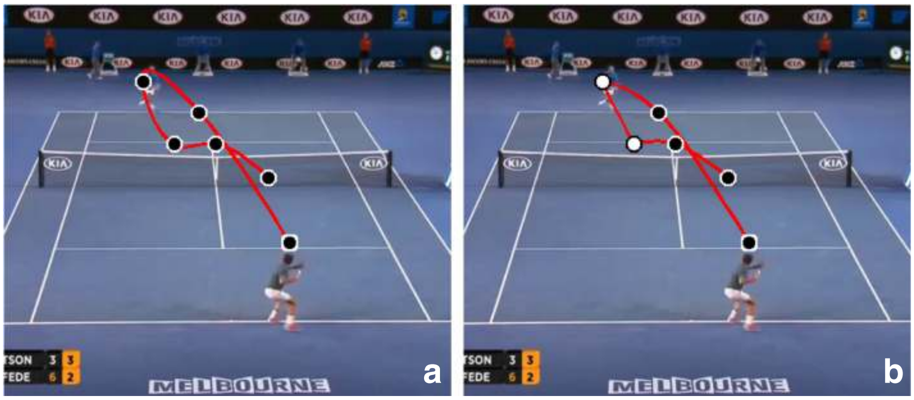

Fig. 14 Comparison of trajectories using cubic spline interpolation **a** without and **b** with *trajectory breaks*. Normal key frames are marked with *black circles*, while breaks are marked with *white circles*

$$e_i = \left(\mathbf{b}_i^{\mathrm{p}} \cup \mathbf{b}_i^{\mathrm{g}}\right) - \left(\mathbf{b}_i^{\mathrm{p}} \cap \mathbf{b}_i^{\mathrm{g}}\right) \tag{31}$$

where the first term denotes the union of the interpolated bounding box with the annotated one (or ground truth) and the second their intersection. For a given trajectory, the interpolation error is computed as the average of all partial overlap errors:

$$e_n = \frac{1}{M} \sum_M e_i \tag{32}$$

Table 1 Video data set summary

Motion	Resolution ($W \times H$)	Trajectory length	Bounding box size ($h \cdot w$)	
type	(pixels)	(frames)	Min (pixels)	Max (pixels)
a	352 × 240	41	560	12,212
a	352 × 240	28	616	4888
a	352 × 240	53	660	1856
b	352 × 240	55	1440	5508
b	352 × 240	76	1392	4800
b	352 × 240	52	7488	16,020
b	352 × 240	188	2912	17,472
c	640 × 360	34	6030	7056
c	960 × 540	29	12,920	15,675
c	640 × 360	17	3942	5270
d	720 × 576	89	240	4640
d	640 × 480	164	1176	6360
d	640 × 360	47	432	2480
d	640 × 480	97	440	2808
e	640 × 360	67	680	3936
e	640 × 480	33	13,720	57,200
f	640 × 360	69	2072	3192
f	640 × 360	69	2184	4292
f	640 × 360	198	9396	33,200

See text for the description of the column motion type

where the subscript n indicates the interval between a pair of key frames. In this case, since we removed one half of the samples of the trajectory, $n = 1$ sample. Note that since we have to remove one half of the samples, we can create one test trajectory by removing the odd samples and a second one removing the even samples. So, the computation of e_1 is finally obtained as the mean value of e_1^0 and e_1^1, where the superscript 0 means the error when the test trajectory is obtained by keeping the even samples (that is, we start keeping the sample in 0, remove the sample in 1, keep the sample in 2, and so on), and similarly for the superscript 1. Likewise, e_2 is computed by constructing a new test trajectory removing $n = 2$ out of three samples. In this case, we can have e_2^0, e_2^1, and e_2^2, and so the error e_2 is computed as the mean of these three values. We performed this computation until the interval $n = 20$.

In Fig. 16, we can see the overall performance of the interpolators evaluated as a function of the interval n. In this case, the error has been computed as the average of the error e for all the videos used in this experiment. Note that, for $n = 1$, an error is present for all the interpolators. This error is mainly due to the process employed to annotate the videos manually, which adds a random shift to the object trajectory. Obviously, as the interval n increases, the average error also increases. However, the best performance is achieved for the interpolators based on depth recovery to compute the interpolated bounding boxes, since many of the objects annotated exhibit great variations of depth. In this sense, the algorithm proposed in this paper (GC) yields the best performance. The algorithm proposed by Yuen et al. (LM) and the simplified version of the algorithm proposed in this paper without cubic spline interpolation (GI) obtain worse results. In fact, for most of the cases, both algorithms are quite similar, since they are based on depth recovery and straight interpolation between key frames. Lastly, those algorithms not

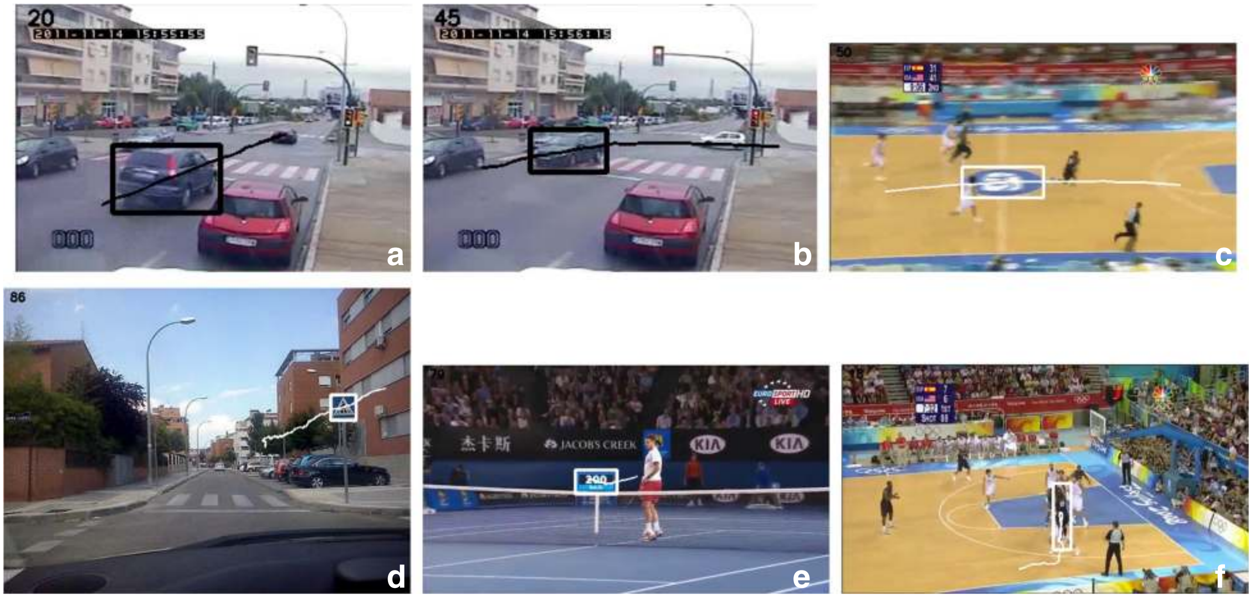

Fig. 15 a–f Sample images of the video data set used in the experiments. See text for description

based on depth recovery but on interpolation from image coordinates, (LI) and (CS), yield the worst results. Table 2 shows the numerical results for the overall performance of the interpolators for different values of the interval *n*.

In order to make a more detailed analysis of the performance of the interpolators, we next present partial results considering the different types of object and camera motions. Figure 17 shows the mean error using only the trajectories of rigid objects moving in a straight path with great depth variations. In this case, the algorithms using depth recovery performed similarly and better than those using interpolation from the image coordinates.

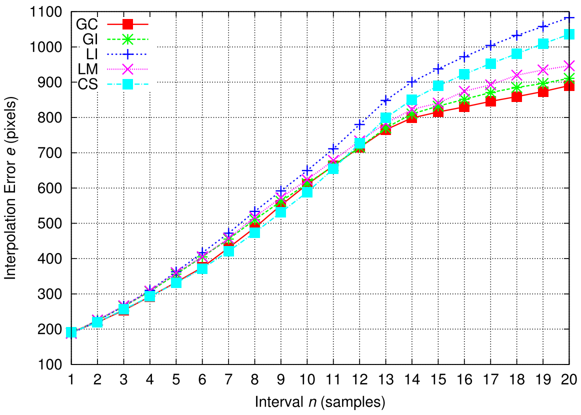

Fig. 16 Overall interpolation error

However, when the object follows a curved path, algorithms using cubic splines can model the object motion better. As can be seen in Fig. 18, the inclusion of a cubic spline interpolation improves the performance. In this case, the algorithm proposed in this paper (GC) yields the best results, since the objects also exhibits depth variations. The algorithms which compute straight paths between key frames decrease their performance.

When considering static objects but non-static cameras, the results are quite similar. If we analyze the trajectory on the image plane of a static object recorded with an on-board camera, we get the results shown in Fig. 19. In this case, the situation is equivalent to an object in front of a static camera moving in the opposite direction, so that the same comments made before are valid here, that is, the algorithms using depth recovery perform better than those interpolating from image coordinates. However, since the videos are recorded with on-board cameras, camera vibration adds a random shift to the object trajectories, which degrades the performance of all the interpolators.

Table 2 Overall interpolation error e_n (pixels)

Interval *n* (samples)	GC	GI	LI	LM	CS
5	332.9	356.0	362.5	360.3	331.3
10	610.2	612.5	649.7	623.8	588.2
15	815.5	832.0	937.9	840.8	889.4
20	890.4	910.9	1083.2	946.4	1035.6

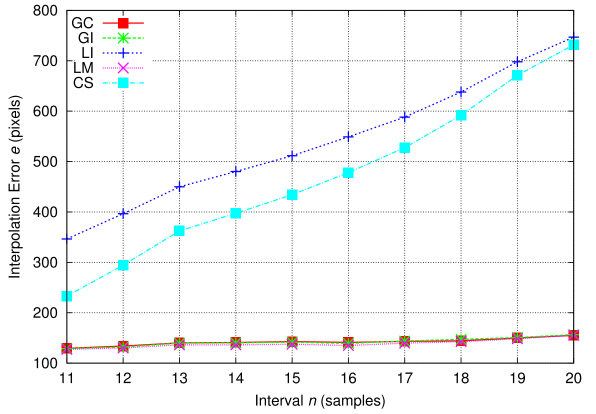

Fig. 17 Results for straight paths with depth variations, static camera

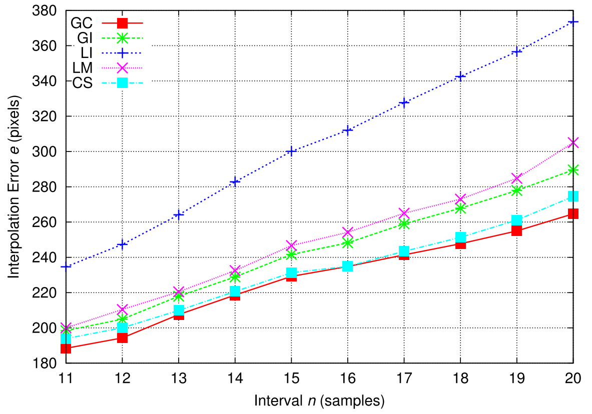

Fig. 19 Results for static object, with camera translation

Next, we obtained results for a static object while the camera is panning. In this case, since neither the object depth nor the size changes significantly, the algorithms based on image plane interpolation perform better than those based on depth recovery, as can be seen in Fig. 20. The main reason that makes the algorithms based on depth recovery decrease their performance is the nature of the annotations. In this case, although the object size in the image plane does not change, the annotations do, because they were done manually. This random change degrades the performance of those algorithms.

Next, we analyze the performance when the camera is zooming. In this case, the variation in the image plane of the object size due to the change in the camera's focal length is not equivalent to the change in depth. In fact, an alternative study to the one developed in this paper can be done to model this type of trajectory. Nevertheless, Fig. 21 shows that it can be modeled accurately with the algorithms which use depth recovery (GC, GI, and LM).

The algorithm proposed in this paper is mainly intended to work with rigid objects. On the other hand, when working with non-rigid objects, like humans, geometric assumptions do not hold, and the results get worse. This is due to the fact that a change in object size is erroneously interpreted as a change in object depth, and non-rigid objects can change their size in the image without changing their depth in space. For instance, the image of a man while crouching can reduce its bounding box height while maintaining the same distance from the camera. In order to analyze the performance for non-rigid object annotation, Fig. 22 shows the results for this kind of object. As can be seen, the results are a little worse than before, although all interpolators analyzed perform similarly, with a small advantage for the algorithms based on cubic splines.

5 Conclusions

In this paper, we have presented an alternative method that interpolates the bounding box annotations between

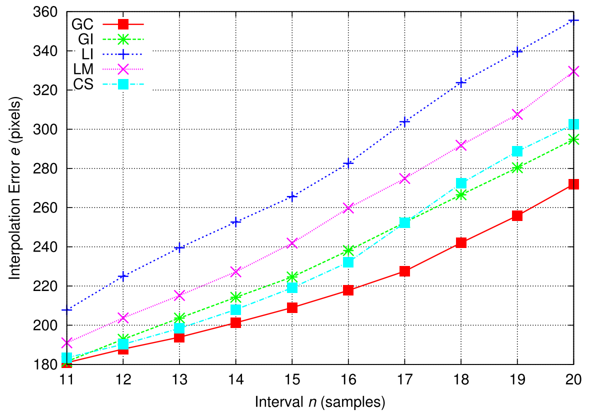

Fig. 18 Results for curved paths, static camera

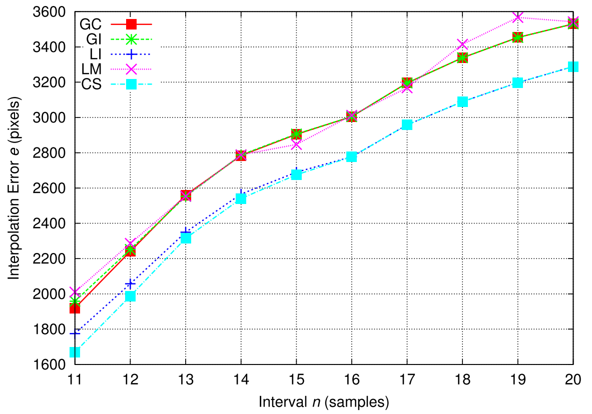

Fig. 20 Results for static object, with camera panning

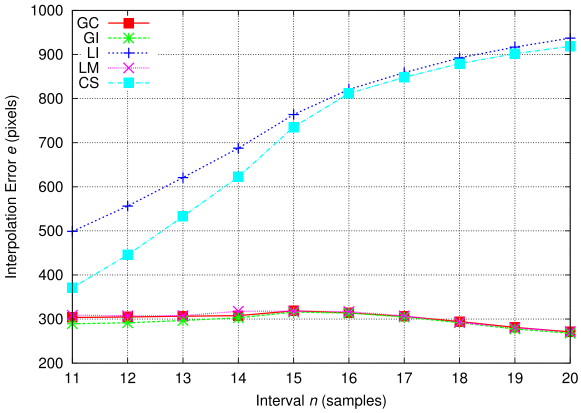

Fig. 21 Results for static object, with camera zoom

key frames for video labeling. The method is based on a 3D reconstruction of the bounding box using the provided annotations, based on the geometric properties of the elements involved. Once the 3D coordinates are computed, the interpolation of the bounding box for the remaining frames is performed in spatial coordinates, using cubic spline interpolation, and finally projected onto the image plane. The algorithm has been evaluated using a selected set of video clips that includes different types of object and camera motions and compared with other interpolation algorithms proposed in the literature. The results show a good performance, especially when considering rigid objects moving in trajectories with great variation in depth, where the accuracy of the interpolated bounding boxes is higher than the other evaluated algorithms. Since bounding box interpolation is a specific part of any annotation tool, we believe that the proposed algorithm can be a good alternative for all these tools when accurate object annotations are required.

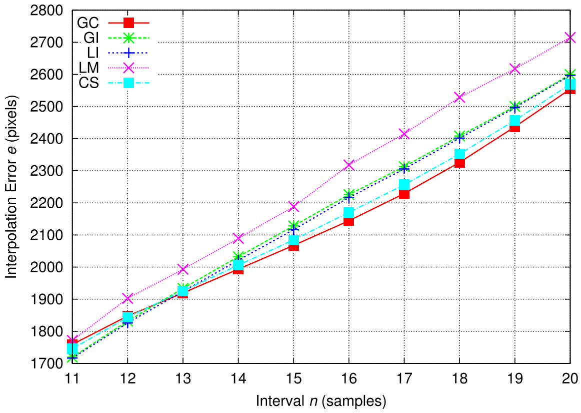

Fig. 22 Results for non-rigid objects

Competing interests

The authors declare that they have no competing interests.

Acknowledgements

This research was supported by projects CCG2014/EXP-055 and TEC2013-45183-R.

Received: 26 May 2015 Accepted: 7 February 2016
Published online: 22 February 2016

References

1. M Everingham, LV Gool, C Williams, J Winn, A Zisserman, The PASCAL visual object classes (VOC) challenge. Int. J. Comput. Vis. **88**, 303–338 (2010)
2. J Deng, W Dong, R Socher, LJ Li, K Li, L Fei-Fei, in *IEEE Conf. Computer Vision and Pattern Recognition*. ImageNet: a large-scale hierarchical image database (IEEE, New York, 2009)
3. B Russell, A Torralba, K Murphy, W Freeman, LabelMe: a datadata and web-based tool for image annotation. Int. J. Comput. Vis. **77**(1), 157–173 (2008)
4. L von Ahn, L Dabbish, in *Proceedings of the Conference on Human Factors in Computing Systems*. Labeling IImage with a computer game (ACM, New York, 2004)
5. A Sorokin, D Forsyth, in *IEEE Conference on Computer Vision and Pattern Recognition Workshops*. Utility data annotation with Amazon Mechanical Turk (EEE, 2008), pp. 1–8
6. M Kipp, in *Proceedings of the 7th European Conference on Speech Communication and Technology*. Anvil - A generic annotation tool for multimodal dialogue (INTERSPEECH, Aalborg, 2001), pp. 1367–1370
7. C Vondrick, DCR Patterson, Efficient scaling up crowdsourced video annotation. Int. J. Comput. Vis. **101**(1), 184–204 (2012)
8. V Badrinarayanan, F Galasso, R Cipolla, in *IEEE Conf. on Computer Vision and Pattern Recognition*. Label propagation in video sequences (IEEE, New York, 2010)
9. C Liu, WT Freeman, EH Adelson, Y Weiss, in *IEEE Conference on Computer Vision and Pattern Recognition*. Human-assisted motion annotation (IEEE, New York, 2008)
10. A Agarwala, A Hertzmann, DH Salesin, SM Seitz, in *ACM Transactions on Graphics*. Keyframe-based tracking of rotoscoping and animation (ACM, New York, 2004)
11. Z Kalal, K Mikolajczyk, J Matas, Tracking-learning-detection. Pattern Anal. Mach. Intell. **34**(7), 1409–1422 (2012)
12. D Doermann, D Mihalcik, in *Int. Conf. on Pattern Recognition. vol. 4*. Tools and techniques for video performance evaluation (IEEE, New York, 2000)
13. MA Serrano, J García, MA Patricio, JM Molina, in *Distributed Computing and Artificial Intelligence. vol. 79*. Interactive video annotation tool (Springer-Verlag, Berlin - Heidelberg, 2010), pp. 325–332
14. J Yuen, B Russell, C Liu, A Torralba, in *IEEE Int. Conf. Computer Vision*. LabelMe video: building a video database with human annotations (IEEE, New York, 2009)
15. JH Lee, KS Lee, GS Jo, in *Proceedings of the International Conference on Information Science and Applications (ICISA)*. Representation method of the moving object trajectories by interpolation with dynamic sampling (IEEE, New York, 2013), pp. 1–4
16. KS Lee, AN Rosli, IA Supandi, GS Jo, Dynamic sampling-based interpolation algorithm for representation of clickable moving object in collaborative video annotation. Neurocomputing. **146**, 291–300 (2014)
17. M Unser, Splines: a perfect fit for signal and image processing. Signal Process. Mag. **16**(6), 22–38 (1999)
18. R Hartley, A Zisermann, *Multiple View Geometry in Computer Vision*, 2nd ed. (Cambridge University Press, Cambridge, 2003)
19. E Jones, T Oliphant, P Peterson, et al., SciPy: Open source scientific tools for Python (2001). http://www.scipy.org/. Accessed 15 Feb 2016
20. P Gil-Jiménez, TrATVid annotation tool. http://agamenon.tsc.uah.es/Investigacion/gram/papers/Annotation. Accessed 16 Nov 2015