Image and Vision Computing 50 (2016) 14–26

Contents lists available at ScienceDirect

Image and Vision Computing

journal homepage: www.elsevier.com/locate/imavis

Estimating the focus of expansion in a video sequence using the trajectories of interest points☆

Pedro Gil-Jiménez*, Hilario Gómez-Moreno,
Roberto J. López-Sastre, Alberto Bermejillo-Martín-Romo

Dpto. de Teoría de la Señal y Comunicaciones, Universidad de Alcalá, 28805, Alcalá de Henares, (Madrid), Spain

ARTICLE INFO

Article history:
Received 2 July 2015
Received in revised form 22 December 2015
Accepted 23 March 2016
Available online 31 March 2016

Keywords:
Focus of expansion
Optical flow
Interest point
Point trajectory
Cross ratio

ABSTRACT

In this paper, we present a new algorithm for the computation of the focus of expansion in a video sequence. Although several algorithms have been proposed in the literature for its computation, almost all of them are based on the optical flow vectors between a pair of consecutive frames, so being very sensitive to noise, optical flow errors and camera vibrations. Our algorithm is based on the computation of the vanishing point of point trajectories, thus integrating information for more than two consecutive frames. It can improve performance in the presence of erroneous correspondences and occlusions in the field of view of the camera. The algorithm has been tested with virtual sequences generated with Blender, as well as some real sequences from both, the public KITTI benchmark, and a number of challenging video sequences also proposed in this paper. For comparison purposes, some algorithms from the literature have also been implemented. The results show that the algorithm has proven to be very robust, outperforming the compared algorithms, specially in outdoor scenes, where the lack of texture can make optical flow algorithms yield inaccurate results. Timing evaluation proves that the proposed algorithm can reach up to 15fps, showing its suitability for real-time applications.

1. Introduction

When a camera moves across a rigid scene, the apparent motion of the imaged points can be used to infer the relative shift of the camera with respect to the scene. For the general case, the problem consists in the computation of the translational and rotational vectors, and is called *ego-motion* [1]. The computation of ego-motion plays an important role in some vision systems, such as Visual Odometry, 3D reconstruction, time-to-impact estimation or obstacle detection and avoidance.

When the rotational component is null, that is, the camera moves in a straight line, the problem reduces to the computation of the translational vector, and the image of this vector on the image plane is called the *Focus of Expansion* (FoE) when the camera moves forwards, or the *Focus of Contraction* (FoC) when it moves backwards, see Fig. 1. Although FoE and FoC refer to opposite directions, their properties are very similar, and we will only refer to the former, recalling that the differences in the computation of the later are minimal.

For its computation, many algorithms have been proposed. In the classic approach, the focus of expansion is computed from the optical flow field between two time-varying frames, which can be obtained from several algorithms [2]. However, technical challenges still exist for general scenes. Typical inaccuracies raise in unconstrained environments, such as road scenes where a large proportion of the image appears untextured, for instance, the sky or a textureless pavement, and optical flow vectors for these areas do not exist or are erroneous. Another source of error can be caused by vibrating platforms. Although many vision benchmarks are available for research, such as the *KITTI Benchmark* [3] or the *CMU Visual Localization Data Set* [4], these sequences are recorded with complex camera setups to prevent such problems. For more basic setups, however, any instability can cause the FoE computation to decrease its accuracy.

To face these problems, we propose a new method based on the estimation of the vanishing point for multi-frame interest point trajectories. The paper is structured as follows: In Section 2, a brief overview of different FoE estimation approaches is provided. In Section 3 we introduce our algorithm for the FoE computation based on the trajectories of interest points. A comparison of our algorithm with other works is given in Section 4, while Section 5 concludes the paper.

☆ This paper has been recommended for acceptance by Cornelia M Fermuller.

* Corresponding author.
E-mail address: pedro.gil@uah.es (P. Gil-Jiménez).

http://dx.doi.org/10.1016/j.imavis.2016.03.007
0262-8856/

2. Related works

Existing techniques to estimate the FoE can be grouped into two main approaches, namely the *continuous methods* and the *discrete methods*. Algorithms from the continuous group employ dense optical flow fields, as in [5–7]. In [8], Sazbon et al. recall that, for a camera moving in a rigid scene, the FoE is characterized by a null flow vector, with the optical flow field radially diverging from it. Thus, only the angular component is enough for the estimation of the FoE, ignoring the magnitude component of the optical flow field. In their work, it is proposed the use of a specially designed matched filter which can work with a low-quality estimation of the optical flow. The matched filter is used to refine the FoE location after a rough estimation in a first phase. However, the algorithm proposed in this work requires that the flow estimation strongly covers the area near the FoE, but this is not generally the case for general video sequences.

The main disadvantage of continuous methods is that dense optical flow is computationally expensive. Furthermore, scenes with lack of texture on a large proportion of the image can yield inaccurate results, since optical flow fields for these areas are likely to be erroneous. To solve these problems, discrete methods use stronger correspondences between image features, such as points or lines, that can be computed from sparse optical flow algorithms.

If several correspondences between points from two consecutive frames are available, the *Fundamental Matrix* F can be computed, and the FoE will correspond with the null-vector of F [9] (p. 245). Being the basic approach for the FoE estimation, the Fundamental Matrix is very inaccurate, and has not been widely used. In [10], the essential matrix is used instead. However, the results show that the iterative Levenberg–Marquardt algorithm is needed to improve results from linear algorithms, increasing thus the computational complexity.

Another approach consists in the computation of the intersection point of all the lines defined by the optical flow vectors. Since noise makes all the lines not intersecting at the same point, a minimization criterion is needed. In [11], Suhr et al. accumulate the lines defined by the optical flow. After it, the largest peak would correspond with the required FoE. In [12] Wu et al. compare different minimization criteria, more specifically, the algebraic method, which is a linear problem, and the geometric method, which is non-linear and numerically more expensive. It is worth noting here that, for the geometric method, the *Cross Ratio*, which is the main tool used in our work, is also employed there, although in a fundamentally different way. In their work, it is employed to generate the so called *inherent constrains* between a pair of points in two consecutive frames. If the inherit constrain fails, that is, the *Cross Ratio* does not hold, the two corresponding pair can not be considered as *true* correspondences, and are eliminated from the FoE computation.

In [13], Bak et al. define the C-Velocity over a planar surface imaged by a camera. When the plane is aligned with the image axis, the optical flow vectors on the plane can be used to estimate the FoE. Although simple, this method can only be used when actual planes are present in the image. For that reason, the use of the algorithm is limited to urban scenes, where planar facades and the road can be used as planes to compute the C-Velocity. In [14], Born projects the optical flow vectors onto the horizontal and vertical axis. These components form a line and the point of intersection with the image axis is the required FoE. A linear regression is needed to compute the line parameters.

Although discrete methods have been usually preferred over continuous ones, these methods also show some disadvantages. On the one hand, finding strong features and correspondences can be a difficult task, and sometimes it could not be present in the scene. Furthermore, these methods are normally less robust because they use local instead of global information.

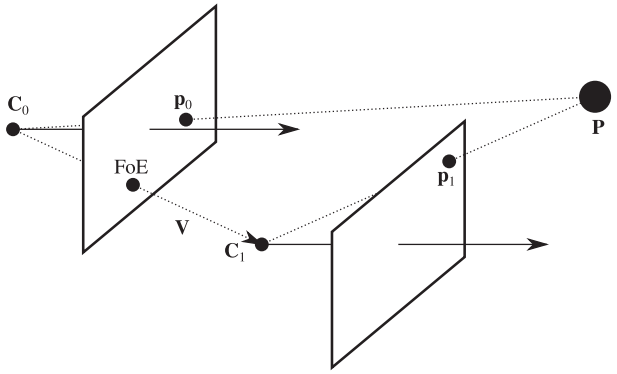


Fig. 1. Focus of expansion of a translating camera. The center of projection is located in $\mathbf{C}_0$ at $t = t_0$, and moves to $\mathbf{C}_1$ at $t = t_1$ with velocity $\mathbf{V} = (V_x, V_y, V_z)^{\mathrm{T}}$. A static point $\mathbf{P} = (P_x, P_y, P_z)^{\mathrm{T}}$ is projected to $\mathbf{p}_0$ and $\mathbf{p}_1$ at t_0 and t_1 respectively. The focus of expansion is the image of the vector $\mathbf{V}$.

An alternative way to compute the FoE is by means of algorithms that compute the 3D camera motion, which are normally referred to as *Visual Odometry* (VO) systems. In this case, the goal is the computation of both the rotational matrix R and the translational vector $\mathbf{v}$, such that the relation between camera positions at two consecutive frames is [15]:

$$\mathrm{T} = \begin{pmatrix} \mathrm{R} & \mathbf{v} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{pmatrix}. \tag{1}$$

When the camera motion is a pure translation, R reduces to the identity matrix, and the image of the vector $\mathbf{v}$ is the FoE, as was shown in Fig. 1. In these systems, for the computation of R and $\mathbf{v}$, typically the *Essential Matrix* E is first computed from a set of correspondences between two consecutive frames. The relationship between those elements is given by:

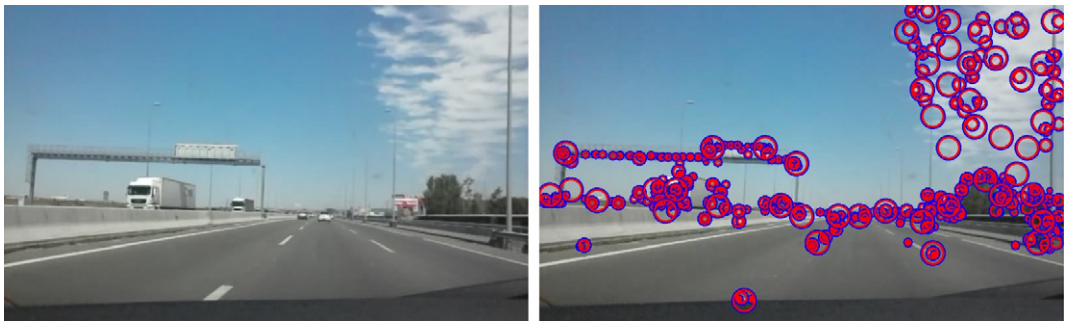
$$\mathrm{E} = [\mathbf{t}]_{\times}\mathrm{R} \tag{2}$$

where $[\mathbf{t}]_{\times}$ is the matrix representation of the *Cross Product* with $\mathbf{t}$. Thus, the computation of the *Essential Matrix* is a fundamental step in these kind of systems. Many works have been proposed in the VO field. In [16], Forster et al. designed a VO system for Micro Aerial Vehicles (MAVs) using a downward-looking camera. Although it shows accurate results, it is mainly designed for planar surfaces. In the work by Geiger et al. [17], R and $\mathbf{t}$ is computed by iterative minimization, using the Gauss–Newton algorithm, of the projection error of the detected points to the image planes. On top of that procedure, a standard Kalman Filter is used to improve the estimations.

It is worth noting that typically such VO systems are not only designed for motion estimation, but these systems are also able to simultaneously perform a 3D reconstruction of the scenario, a technique called *Simultaneous Localization and Mapping* (SLAM). Nonetheless, we are only interested in the motion estimation block.

As we will see in Section 4, some of these algorithms have been implemented (or downloaded from the paper web page if available), along with the algorithm proposed in this paper, for comparison purposes. For this task, we have employed both, virtual video sequences generated with Blender[1] and a series of challenging video sequences recorded with an on-board camera mounted on a vehicle.

[1] http://www.blender.org/.



(a) Original frame. (b) Points detected at four different scales.

Fig. 2. Performance of the Multi-scale *Harris Interest Point* Detector.

3. System overview

The proposed system can be divided into two main blocks. The first block takes the video sequence as input and continuously extracts *Point Trajectories* by means of the *Optical Flow* computation for the *Interest Points* detected in the image, as will be described in Section 3.1. At this stage, each point trajectory will correspond with the track, relative to the camera motion, of a given point in the scene.

In the second block, each trajectory is projected into its vanishing point by means of the *Cross Ratio* property. This allows the system to compute the focus of expansion of the video sequence, as will be seen in Section 3.2.

3.1. Interest Point Trajectories

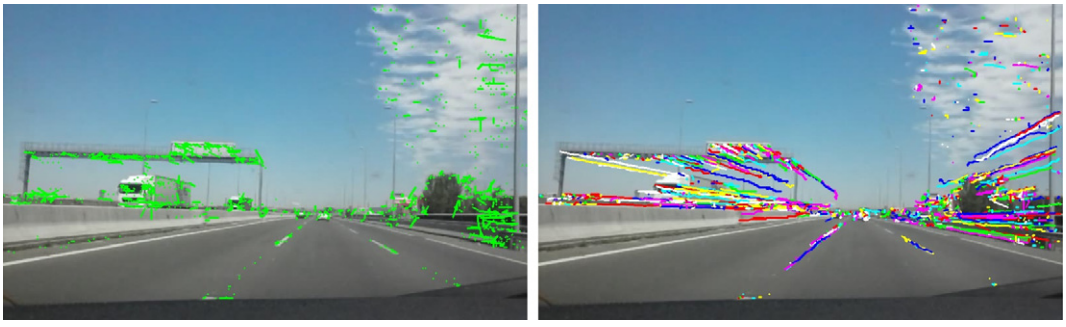
The main goal of the first block is the computation of trajectories for points belonging to the static scene. To accomplish this task, the block is based on two internal algorithms, *Interest Point* detection and *Optical Flow* computation. For the detection of the *Interest Points* many point detectors have been defined in the literature, in [18] a complete overview can be found. In our case, we have used one of the more classic, yet efficient detectors, the *Harris Detector*, proposed in [19]. In Fig. 2 we can see a visual example of the performance of the detector. In (a), the original frame of a video sequence is shown. In (b), we can see the output of the Harris Detector applied to that frame at different scales. The points are drawn with circles, centered at the point's coordinates, and whose radius is related to the scale at which the point was detected. In this case, only four scales were considered (full resolution, half resolution, one-fourth and one-eighth), although other scales could be used.

Once the *Interest Points* have been detected, the next step is the estimation of the motion for each point between two consecutive frames. One of the more common techniques for this task is the *Optical Flow*. Among the available *Optical Flow* algorithms, one of the most efficient implementations is the classical *Lucas–Kanade* method [20], which is the one used in our system. In Fig. 3 (a) we can see an example of the output of the optical flow block for the frame shown in Fig. 2. Note the inaccuracy of the results when working with outdoor images, which makes, as we will see in Section 4, decrease the performance of the algorithms based exclusively in optical flow vectors.

The last step of this block is the computation of the point trajectories. Point trajectories are constructed linking together the consecutive segments corresponding to the motion vectors computed for a particular point, until the point stops being visible to the camera. Fig. 3 (b) shows an example of the trajectories extracted for a given frame in a real scene. Recall that trajectories are generated and destroyed continuously, and that this figure only shows *alive* trajectories for the current frame.

3.2. Trajectory Projection

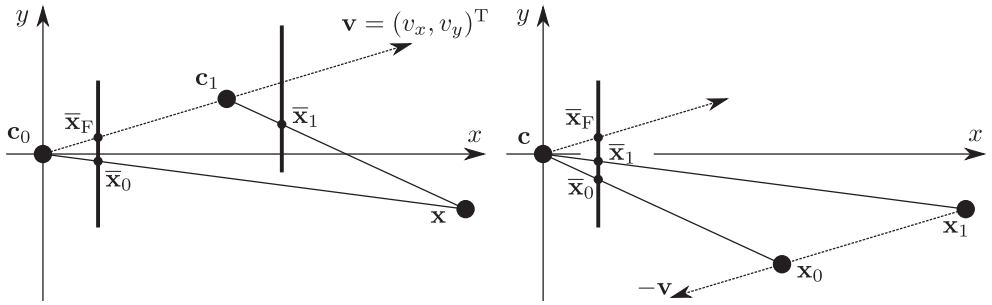
In this section, we will see how we can use the point trajectories extracted by the previous block to compute the focus of expansion



(a) *Lucas-Kanade* optical flow computation for the *Interest Points* shown in Fig. 2 (b).

(b) Extracted trajectories (best viewed in color). The different colors are used only for visualization purposes.

Fig. 3. Computation of point trajectories on real sequences. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)



(a) A one dimension projective camera moving with translational velocity $\mathbf{v}$.

(b) Relative motion of a point $\mathbf{x}$ with respect to the camera.

Fig. 4. One dimension geometry for a camera moving on a static scene.

for a given sequence. For simplicity, we will develop the explanation for a 1 dimension camera moving in a 2D plane. The extrapolation for a 3D scenario will be straightforward. The notation throughout this section has been taken from [9] where possible. More specifically, we will use the notation $\overline{\mathbf{x}}$ to represent a one dimension point, while $\mathbf{x}$ represents a two dimensions point, and $\mathbf{X}$ a three dimensions point, all expressed in homogeneous coordinates.

Let suppose a one dimension camera $\mathbf{c}$ moving relative to a rigid environment with translational velocity $\mathbf{v} = (v_x, v_y)^{\mathrm{T}}$, as in Fig. 4 (a), and a static point $\mathbf{x}$. The camera will image this point to $\overline{\mathbf{x}}_0$ at $t = t_0$, and $\overline{\mathbf{x}}_1$ at $t = t_1$. Then, the point will appear to move with respect to the camera with velocity $-\mathbf{v}$, and the focus of expansion $\overline{\mathbf{x}}_{\mathrm{F}}$ will be the vanishing point of the line described by this point, that is, the image of the point at $t = -\infty$, see Fig. 4 (b). Note that for a projective camera, the image of the point at $t = -\infty$ is the same than the image at $t = \infty$, and so, from now on, we will consider $\pm\infty$ indifferently. Next, we will see how we can compute the vanishing point of a trajectory from a minimum of three points by means of the *cross ratio*.

3.2.1. The Cross Ratio

The cross ratio is the basic projective invariant of IP^1 [9](p. 45). For a set of four collinear points, the cross ratio is defined as:

$$\mathrm{Cross}(\overline{\mathbf{x}}_0', \overline{\mathbf{x}}_1', \overline{\mathbf{x}}_2', \overline{\mathbf{x}}_3') = \frac{|\overline{\mathbf{x}}_0'\overline{\mathbf{x}}_1'|\,|\overline{\mathbf{x}}_2'\overline{\mathbf{x}}_3'|}{|\overline{\mathbf{x}}_0'\overline{\mathbf{x}}_2'|\,|\overline{\mathbf{x}}_1'\overline{\mathbf{x}}_3'|}, \tag{3}$$

where the one dimension point $\overline{\mathbf{x}}_i' = (x_{i1}', x_{i2}')^{\mathrm{T}}$ are expressed in homogeneous coordinates, and:

$$|\overline{\mathbf{x}}_i\overline{\mathbf{x}}_j| = \det\begin{bmatrix} x_{i1} & x_{j1} \\ x_{i2} & x_{j2} \end{bmatrix} = x_{i1}x_{j2} - x_{i2}x_{j1}. \tag{4}$$

In the projective one dimension camera in Fig. 5, the cross ratio for the points imaged by the camera keeps constant, that is:

$$\mathrm{Cross}(\overline{\mathbf{x}}_0, \overline{\mathbf{x}}_1, \overline{\mathbf{x}}_2, \overline{\mathbf{x}}) = \mathrm{Cross}(\overline{\mathbf{x}}_0', \overline{\mathbf{x}}_1', \overline{\mathbf{x}}_2', \overline{\mathbf{x}}_3'). \tag{5}$$

If we suppose a camera moving at constant velocity $\mathbf{v}$, the trajectory of a point $\mathbf{x}$ relative to the camera will draw a line, and $\overline{\mathbf{x}}_i'$ will correspond to the position of the point at $t = t_i$ in that line. In this situation, the vanishing point of this line is the image of the point at $t = \pm\infty$. Considering, without loss of generality, $|\mathbf{v}| = 1$, $t_0 = 0$ and $\overline{\mathbf{x}}_0' = (0, 1)^{\mathrm{T}}$, the homogeneous coordinates of the points in the line at different times will be:

$$\overline{\mathbf{x}}_0' = \begin{pmatrix} 0 \\ 1 \end{pmatrix},\ \overline{\mathbf{x}}_1' = \begin{pmatrix} t_1 \\ 1 \end{pmatrix},\ \overline{\mathbf{x}}_2' = \begin{pmatrix} t_2 \\ 1 \end{pmatrix},\ \overline{\mathbf{x}}_3' = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{6}$$

where $\overline{\mathbf{x}}_3' = (1, 0)^{\mathrm{T}}$ is an ideal point (the point for $t = \pm\infty$) on the line. Substituting in Eq. (3):

$$\mathrm{Cross}\quad(\overline{\mathbf{x}}_0', \overline{\mathbf{x}}_1', \overline{\mathbf{x}}_2', \overline{\mathbf{x}}_3') = \frac{\begin{vmatrix} 0 & t_1 \\ 1 & 1 \end{vmatrix}\begin{vmatrix} t_2 & 1 \\ 1 & 0 \end{vmatrix}}{\begin{vmatrix} 0 & t_2 \\ 1 & 1 \end{vmatrix}\begin{vmatrix} t_1 & 1 \\ 1 & 0 \end{vmatrix}} = \frac{t_1}{t_2}. \tag{7}$$

Since the cross ratio keeps constant under any projective transformation, given any three finite positions $\overline{\mathbf{x}}_0$, $\overline{\mathbf{x}}_1$ and $\overline{\mathbf{x}}_2$, with $\mathbf{x}_i = (p_i, 1)^{\mathrm{T}}$, of a point at different instants in the *image line* of the one dimension camera, the fourth position can be computed using Eq. (7):

$$\mathrm{Cross}\quad(\overline{\mathbf{x}}_0, \overline{\mathbf{x}}_1, \overline{\mathbf{x}}_2, \overline{\mathbf{x}}) = \frac{\begin{vmatrix} 0 & p_1 \\ 1 & 1 \end{vmatrix}\begin{vmatrix} p_2 & p_3 \\ 1 & 1 \end{vmatrix}}{\begin{vmatrix} 0 & p_2 \\ 1 & 1 \end{vmatrix}\begin{vmatrix} p_1 & p_3 \\ 1 & 1 \end{vmatrix}} = \frac{p_1(p_2 - p_3)}{p_2(p_1 - p_3)} = \frac{t_1}{t_2} \tag{8}$$

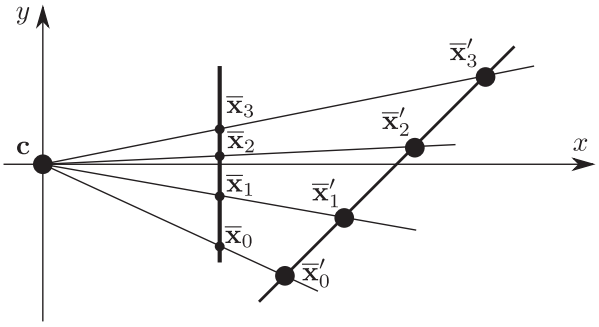


Fig. 5. Cross ratio for a set of four collinear points. As the relation between the points $\{\overline{\mathbf{x}}_0', \ldots \overline{\mathbf{x}}_3'\}$ and $\{\overline{\mathbf{x}}_0 \ldots \overline{\mathbf{x}}_3\}$ is a projective transformation, the cross ratio for both sets is the same.

where we have considered $p_0 = 0$ for simplicity, that is, we have set $\bar{\mathbf{x}}_0$ the origin of coordinates for the *image line*, so that p_i is the distance from $\bar{\mathbf{x}}_i$ to $\bar{\mathbf{x}}_0$. After a little computation, we have:

$$p_3 = \frac{p_1 p_2 (1-k)}{p_1 - k p_2}, \tag{9}$$

where $k = t_1/t_2$. Since Eq. (7) was computed for the last point at $t = \infty$, p_3 will be the location of the vanishing point for the line described by the point, that is, the searched focus of expansion, let's call this point $p_3 = p_F$. Finally, defining $r = p_1/p_2$:

$$p_F = p_2 \frac{r(1-k)}{r-k}, \tag{10}$$

where p_F is the distance from $\bar{\mathbf{x}}_0$ to the vanishing point $\bar{\mathbf{x}}_F$ in the *image line* of the one dimension camera. Generalization for a 3D environment is straightforward. A 3D point $\mathbf{X}$ will be imaged by a projective 2D camera to $\mathbf{x}_i$ at $t = t_i$. As long as the camera moves at constant velocity $\mathbf{V} = (V_x, V_y, V_z)^T$, the trajectory described by the imaged points $\mathbf{x}_i$ will be a line, and the FoE again corresponds to the vanishing point of this line.

3.2.2. Error sensitivity

In this section, we will check the sensitivity of the process to errors in image measurements. Since the coordinates of trajectory nodes correspond to interest point locations, we will assume that measure errors are mainly due to errors in the *Harris Detector*, independently of image coordinates. Let suppose that the absolute error in the location of an interest point is $\Delta\mathbf{x}$, and that errors for different points are independent. When computing the vanishing point, we need a set of three image points with its corresponding errors $\Delta\mathbf{x}_0$, $\Delta\mathbf{x}_1$ and $\Delta\mathbf{x}_2$. Note that in Eq. (10), p_2 is the Euclidean distance between $\mathbf{x}_0$ and $\mathbf{x}_2$, and so, the relative error for p_2 can be approximated to:

$$\delta p_2 \approx \frac{\Delta p_2}{p_2}. \tag{11}$$

Thus, the absolute error of the vanishing point with respect to p_2 will be:

$$\Delta p_F = \frac{\partial p_F}{\partial p_2}\Delta p_2 = \frac{r(1-k)}{r-k}\Delta p_2 = \frac{p_F}{p_2}\Delta p_2, \tag{12}$$

and its relative error:

$$\delta p_F = \frac{\Delta p_F}{p_F} = \frac{\Delta p_2}{p_2}, \tag{13}$$

that is, the relative error decreases as the distance between $\mathbf{x}_0$ and $\mathbf{x}_2$ increases. This result agrees with our intuitive notion that longer trajectories are better for computation than shorter ones.

Next, we compute the sensitivity to the intermediate point $\mathbf{x}_1$ chosen, or more precisely, the ratio $k = t_1/t_2$. In this case, the absolute error of p_F with respect to k is:

$$\Delta p_F = \frac{\partial p_F}{\partial k}\Delta k = p_2 \frac{r(1-r)}{(r-k)^2}\Delta k, \tag{14}$$

and the relative error:

$$\delta p_F = \frac{\Delta p_F}{p_F} = \frac{1-r}{(r-k)(1-k)}\Delta k. \tag{15}$$

From Eq. (10), we can solve for r:

$$r = \frac{p_F k}{p_F - p_2(1-k)} = \frac{mk}{m-1+k}, \tag{16}$$

where $m = p_F/p_2$. Finally, substituting in Eq. (15):

$$\delta p_F = \frac{1 - \frac{mk}{m-1+k}}{\left(\frac{mk}{m-1+k} - k\right)(1-k)}\Delta k = \frac{m-1}{k(1-k)}\Delta k. \tag{17}$$

Taking into account that $0 < k < 1$, the relative error of p_F with respect to k have a minimum at $k = 0.5$, and the error tends to infinity when k approaches 0 and 1, that is, when the point $\mathbf{x}_1$ is close to either $\mathbf{x}_0$ or $\mathbf{x}_2$.

In brief, the relative error of the computed vanishing point reduces as the value of p_2, that is, the distance between the exterior points $\mathbf{x}_0$ and $\mathbf{x}_2$, increases. Similarly, the relative error reaches the minimum when $k = t_1/t_2 = 0.5$.

3.2.3. FoE extraction

From the previous discussion, we can assert that any point trajectory belonging to a static point with at least three nodes can be used to compute its vanishing point, and hence, the focus of expansion of a video sequence. However, noise and processing errors can make the point not drawing a straight line. For that reason, prior to FoE computation, we need to choose the more appropriate trajectories for this task. To this end, for a given trajectory composed of $n+1$ nodes $T = \{\mathbf{x}_0, \mathbf{x}_1, \ldots \mathbf{x}_n\}$, a line $\mathbf{l} = (l_1, l_2, l_3)^T$ is defined using the first ($\mathbf{x}_0 = \{p_0^x, p_0^y\}$) and the last ($\mathbf{x}_n = \{p_n^x, p_n^y\}$) nodes (see Fig. 6), and the mean geometric error for the rest of the nodes is computed:

$$e = \frac{1}{n-1}\sum_{i=1}^{n-1}\frac{l_1 p_i^x + l_2 p_i^y + l_3}{\sqrt{l_1^2 + l_2^2}}. \tag{18}$$

Since, for a given frame, a number of trajectories from the video sequence are available, the FoE can be computed as the weighted mean of the vanishing point computed for each trajectory. The geometric error computed in Eq. (18) will be used, along with other parameters, to weight each computed point, as will be described in Section 3.2.4, using Eq. (19). Nevertheless, a threshold th_e can be set to prevent the system from computing the vanishing point for unsuitable trajectories. If a given trajectory reaches that threshold, the first point (the oldest one) is ignored, and a new geometric error is computed, in this case with its n remaining nodes. The process continues until either the geometric error is below this threshold or the trajectory does not have enough nodes to compute its vanishing point.

Once a set of $n+1$ nodes is found to be suitable for the computation of its vanishing point following the process described before, all the nodes are projected onto the line defined by the first and last

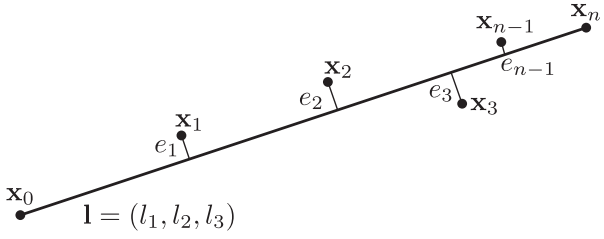


Fig. 6. Geometric error for a set of $n+1$ points and a line defined between the exterior points $\mathbf{x}_0$ and $\mathbf{x}_n$.

(a) Vanishing points computed for a given frame (the gray level represents the weighting value ω).

(b) Computed focus of expansion. The FoE is represented as the meeting point of the four lines drawn.

Fig. 7. Computation of the focus of expansion.

node, as in Fig. 6. At this point, $n-1$ vanishing points can be computed, using the first node as p_0 (origin of coordinates of the line) in Eq. (10), the last node as p_2, and one of the $n-1$ intermediate nodes as p_1. Furthermore, $n-2$ vanishing points can be computed using the second, the last and one of the $n-2$ intermediate nodes, and so on. Note that computing more vanishing points using the penultimate node as p_2 is not needed, since that computation was performed in the previous frame of the video sequence.

Thus, for a suitable trajectory of $n+1$ nodes, we can compute up to $(n^2-n)/2$ different vanishing points. To get a single vanishing point for the trajectory, the centroid for all the points can be computed. However, not all the possible point combinations have the same sensitivity to measure errors when applying Eq. (10), as seen before. To this end, we can weight each vanishing point, computing its relative error with respect to the actual points employed in the computation.

3.2.4. FoE computation

For a given frame, the FoE is computed as the weighted centroid of all the vanishing points $\mathbf{x}_F$ computed from all the available trajectories. The weight for each vanishing point is computed, according to previous discussion, as:

$$w = w_d \cdot w_k \cdot w_e, \tag{19}$$

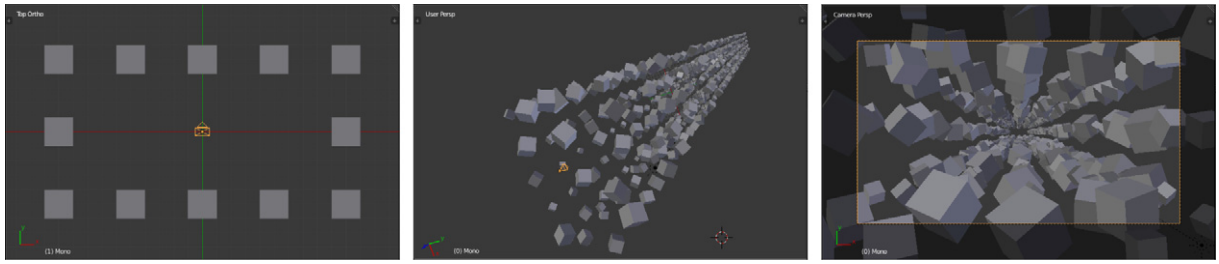
where $w_d = p_2$, $w_k = k(1-k)$, and $w_e = th_e - e$, being e the geometric error computed according to Eq. (18). In Fig. 7 (a) we can see the cloud of points representing all the vanishing points computed for all the available trajectories for the current frame shown in Fig. 3 (b). In this figure, the weighting value w is represented by its gray level, with black points representing low values and white points representing high values.

Table 1
Main parameters of the Blender project designed for the rendering of the virtual video data set. We have considered for distances 1 blender unit equal to 1 m, and a frame rate of 25 fps.

Image width	640 pixels
Image height	480 pixels
Horizontal field of view	49.13°
Focal length	700 pixels
Scenario length	250 m
Sequence length	500 frames
Seq. duration	25 s

Finally, the focus of expansion can be computed as the weighted centroid of all the vanishing points. However, to make the computation robust to outliers generated from erroneous trajectories, the *mean-shift* algorithm is employed. Following the original algorithm from Comaniciu [21], the mean shift vector $\mathbf{m}$ is computed using the weighted vanishing points $\mathbf{x}_i$, according to:

$$\mathbf{m}^{(i+1} = \frac{\sum_{i=1}^{N} \mathbf{x}_i w_i g\left(\left\| \frac{\mathbf{x}_i - \mathbf{x}^{(i}}{h} \right\|^2\right)}{\sum_{i=1}^{N} w_i g\left(\left\| \frac{\mathbf{x}_i - \mathbf{x}^{(i}}{h} \right\|^2\right)} - \mathbf{x}^{(i}, \tag{20}$$



(a) (b) (c)

Fig. 8. Different views of the *Blender 3D View* window. (a): Orthogonal view, from the Z axis, of the rows of cubes before random rotation, scaling and shift. (b): Perspective view of the generated *tube*. Note the camera placed at the start of the *tube*, drawn in orange. (c): The *tube* from the camera point of view. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

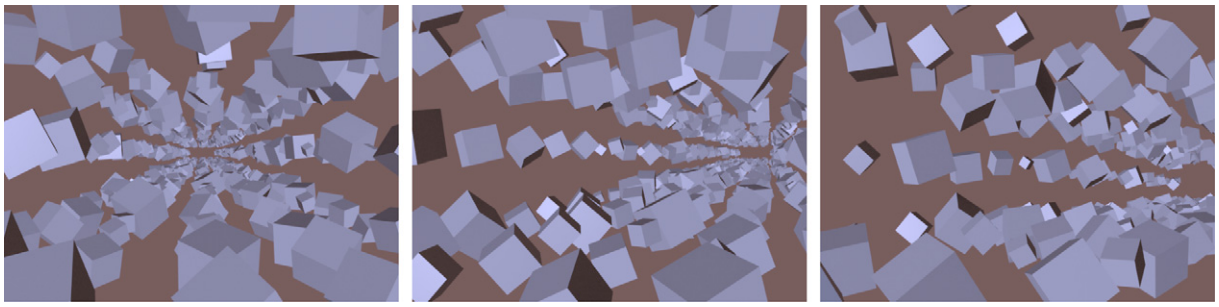


Fig. 9. Sample frames for sequences with different pan (α) and tilt (β) angles.

where $\mathbf{x}^{(i}$ is the current FoE location for the ith iteration, and $\mathbf{x}^{(i+1} = \mathbf{x}^{(i} + \mathbf{m}^{(i+1}$. Also, $g(x)$ is the derivate of the *Kernel Profile* as defined in [21], where the profile employed was the *Normal Kernel*. Experiments have shown that a number of 10 iterations is enough for the convergence of the algorithm. For each frame, the initial value $\mathbf{x}^{(0}$ is set to the computed FoE for the previous frame. For the first time, however, it is set to the weighted centroid using all the available vanishing points. Recall that the first FoE can only be computed when we have at least one trajectory with three nodes. Fig. 7 (b) shows the focus of expansion computed for a real scene. Note how the FoE has been located at the vanishing point of the road, proving the applicability of the proposed algorithm.

4. Results and discussion

In order to test the algorithm described in this paper, the proposed system has been implemented in Java.[2] For the *Harris Detector* and the *Lucas–Kanade algorithm*, we employed the implementations available in the OpenCV 2.4.8 library. For comparison purposes, the following algorithms have been used:

1. *Trajectory Projections (TRP)*: This is the implementation of the system described in this paper.
2. *Flowfield Projections (FLP)*: Following the work described in [14], the optical flow vectors are projected onto the image axis, and a line is computed for each one. The FoE location is the point of intersection of these lines with the image axis.
3. *Intersection Point (IPG)*: In this case, the FoE is the intersection point of all the lines defined by the optical flow vectors. Since noise makes all the lines not meeting at the same point, a minimization criteria is needed. In this case, we use the Simplified Geometric Algorithm (SGA), as described in [12].
4. *Visual Odometry (VO)*: In order to give a reference between algorithms that compute the FoE directly in the image plane (TRP, FLP and IPG) and Visual Odometry algorithms, which allows its computation from the translational vector, the work proposed in [17] has been downloaded and included in the comparison.

Although the original works based solely in optical flow vectors (FLP and IPG) use the raw output of the Lucas–Kanade algorithm, we improved their performance by constructing more accurate *flow vectors* using two non-consecutive nodes of each trajectory. That is, for an interval of N frames, flow vectors are constructed using trajectory nodes at frames M and $M - N$ for the computation of the FoE in frame M. This way, flow vectors are more accurate and the FoE is more stable. For the frame interval N, an exhaustive search was performed, finally choosing a value of 5 frames. For the algorithm proposed in this paper, however, this previous step is not required.

To compare the performance of these systems, we have designed two different data sets. First, a virtual scenario has been built using Blender 2.69 and Python scripting. In this case, camera parameters, such as focal length, camera motion and orientation can be accurately defined. The second data set is composed of several video sequences recorded with an on-board camera.

4.1. Virtual video data set

Blender is a free and open source 3D creation suite. It allows modeling and animation of 3D scenarios. In this case, we have used Blender to create a simple scenario, and animate a camera to simulate camera translations when rendering the video sequence. More specifically, the scenario has been constructed with a series of cubes forming a *tube*, and a camera moving inside it. Fig. 8 (a) shows one row of this composition, with the camera in the middle, drawn in orange. The composition is extruded along the Z axis to form the *tube*, and the camera will travel from the beginning of the *tube* to its end. Furthermore, all the cubes are randomly rotated, scaled and shifted from its initial position. Fig. 8 (b) shows the final scenario, while Fig. 8 (c) displays it from the camera point of view. Finally, the camera is animated and a video sequence is rendered from that. Table 1 shows the main parameters of the

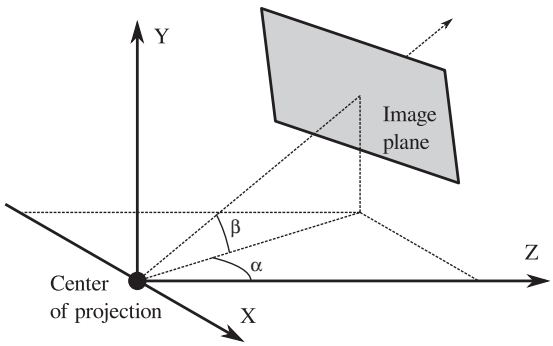


Fig. 10. Pan and tilt angles for a camera orientation.

[2] A Java application, and all the videos employed for testing can be downloaded from http://agamenon.tsc.uah.es/Investigacion/gram/papers/FoE.

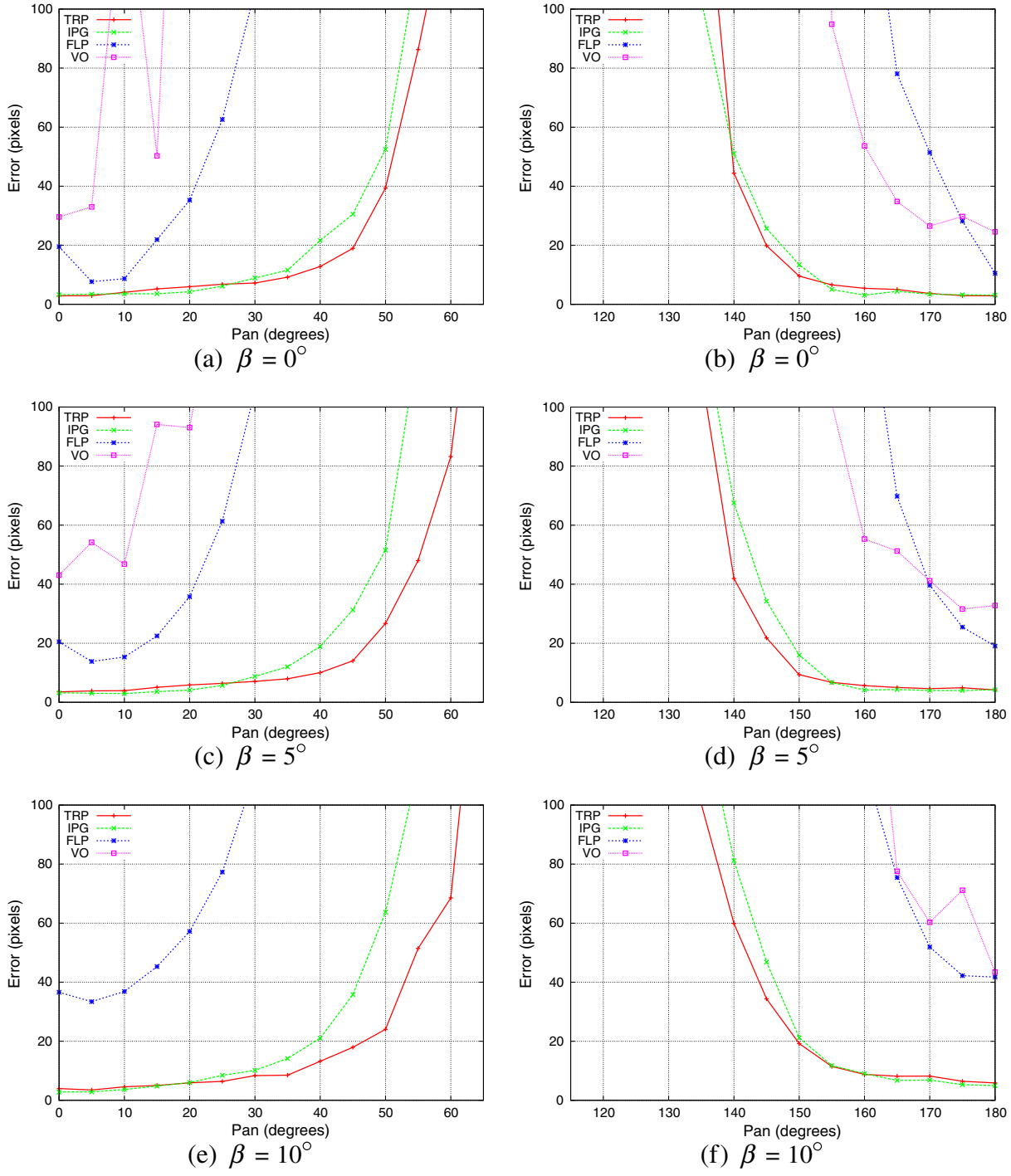


Fig. 11. Average Euclidean distance to the true FoE, in pixels, as a function of the pan angle (α) for different values of the tilt angle (β). For Panels (a), (c) and (e), the camera is moving forwards, while for (b), (d) and (f) it moves backwards.

Blender project. The Python script designed for modeling and rendering the whole data set can be downloaded from the project web page.

In the first experiment, we tested the performance of the evaluated algorithms to camera orientation while the camera is traveling at constant speed inside the scenario. To this end, two key frames are inserted at the beginning and end of the sequence, so that the camera traverses the whole scenario. The sequences were generated for different camera pan and tilt angles, and all the algorithms were executed for all these sequences. Fig. 9 shows some sample frames for different pan and tilt angles.

Note that the location of the FoE can be computed for a particular video sequence from the camera parameters. More specifically, if α and β are the pan and tilt angles respectively for a particular orientation of the camera, as shown in Fig. 10, the camera projection matrix P will be:

$$\mathrm{P} = \mathrm{K} \cdot (\mathrm{R_Y}(\alpha) \cdot \mathrm{R_X}(\beta))^{-1} \cdot [\mathrm{I}|\mathbf{0}] \tag{21}$$

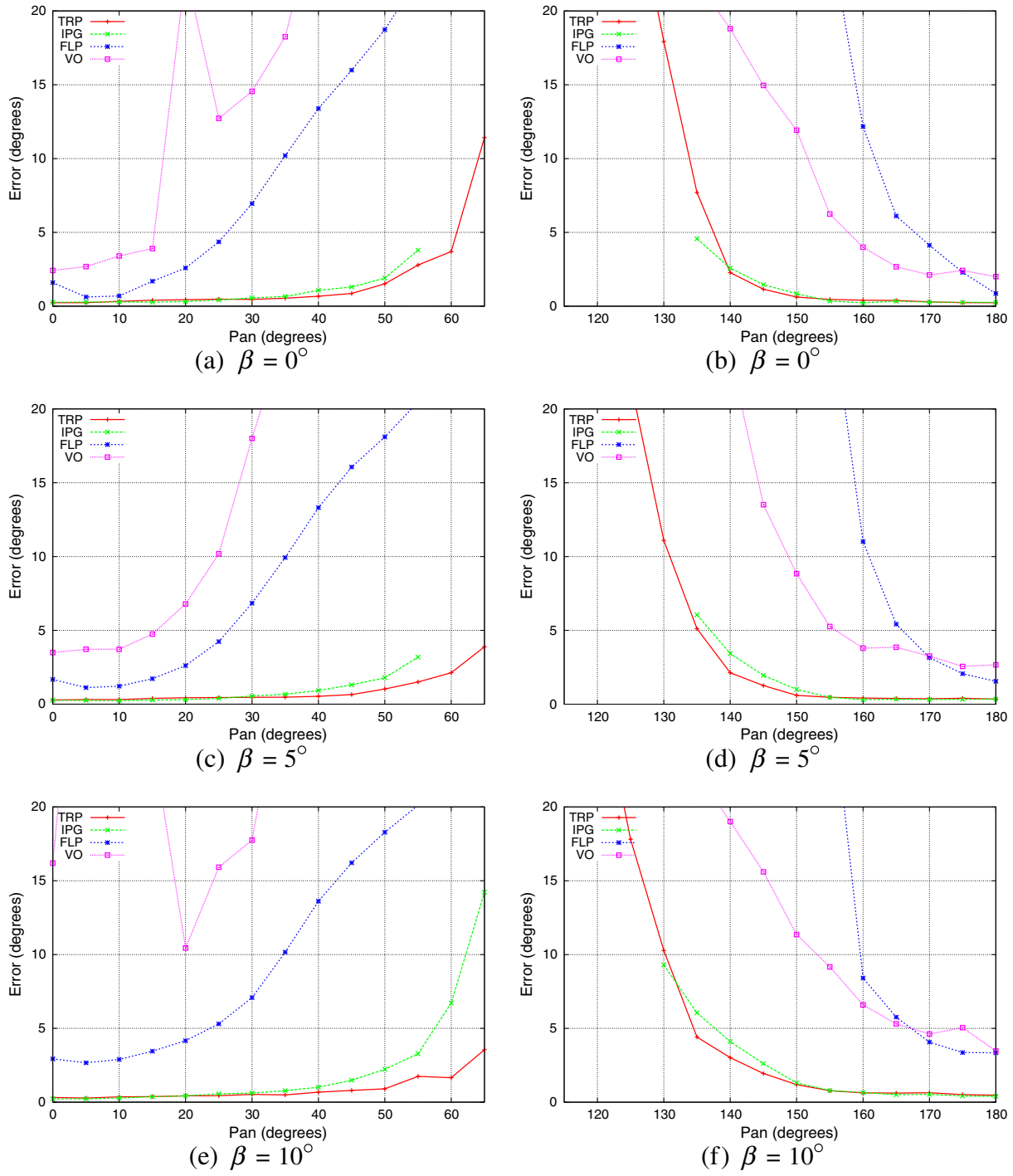


Fig. 12. Error angle, in degrees, as a function of the pan angle (α) for different values of the tilt angle (β). For Panels (a), (c) and (e), the camera is moving forwards, while for (b), (d) and (f) it moves backwards.

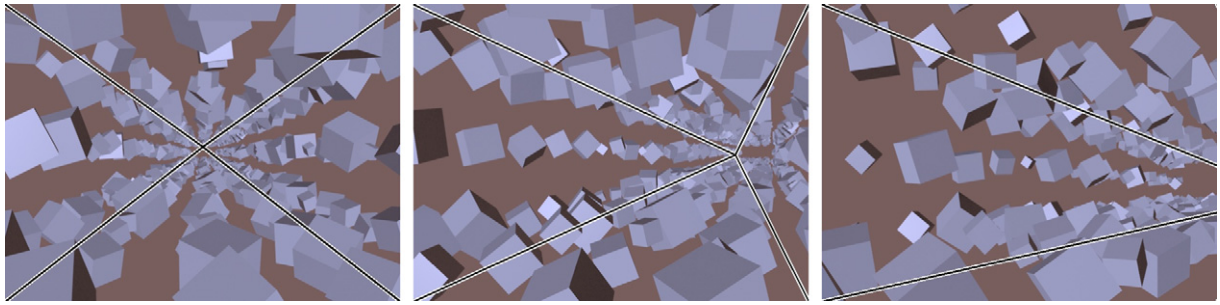
where R_Y is a matrix rotation around the Y axis, R_X a matrix rotation around the X axis, I is the 3×3 identity matrix and K is the *camera calibration matrix*, that can be computed from the parameters given in Table 1 (with zero skew and square pixels). Since the camera is programmed to move along the Z axis, the translational vector $\mathbf{v} = (0,0,1)^{\mathrm{T}}$ will be projected to the new camera as:

$$\mathbf{x}_{\mathrm{F_G}} = \mathrm{K} \cdot (\mathrm{R_Y}(\alpha) \cdot \mathrm{R_X}(\beta))^{-1} \cdot \mathbf{v} \tag{22}$$

where $\mathbf{x}_{\mathrm{F_G}}$ is the *Ground Truth* FoE. After a little computation, we get:

$$\begin{aligned} x_{\mathrm{F_G}} &= x_{\mathrm{c}} + f \cdot \frac{\tan\alpha}{\cos\beta} \\ y_{\mathrm{F_G}} &= y_{\mathrm{c}} + f \cdot \tan\beta \end{aligned} \tag{23}$$

where f is the focal length of the camera and $p_{\mathrm{c}} = (x_{\mathrm{c}}, y_{\mathrm{c}})$ are the coordinates of the principal point of the camera (which in this



(a) $\alpha = 0^\circ$, $\beta = 0^\circ$. (b) $\alpha = 20^\circ$, $\beta = 0^\circ$. (c) $\alpha = 30^\circ$, $\beta = 5^\circ$.

Fig. 13. Sample results for sequences with different pan (α) and tilt (β) angles.

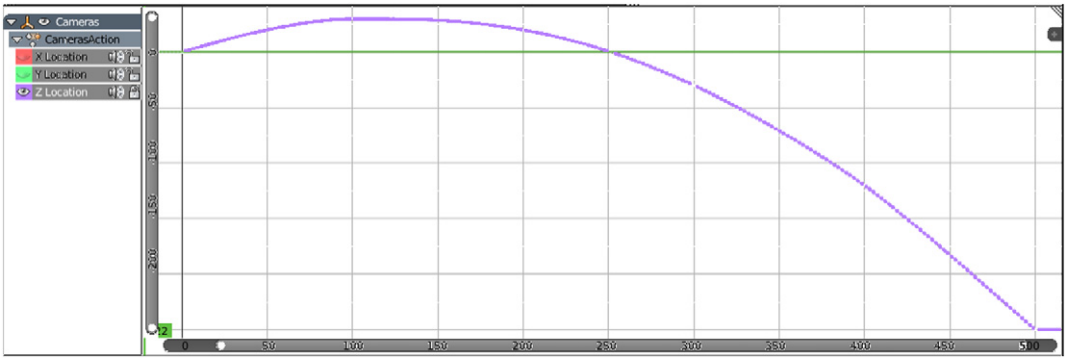


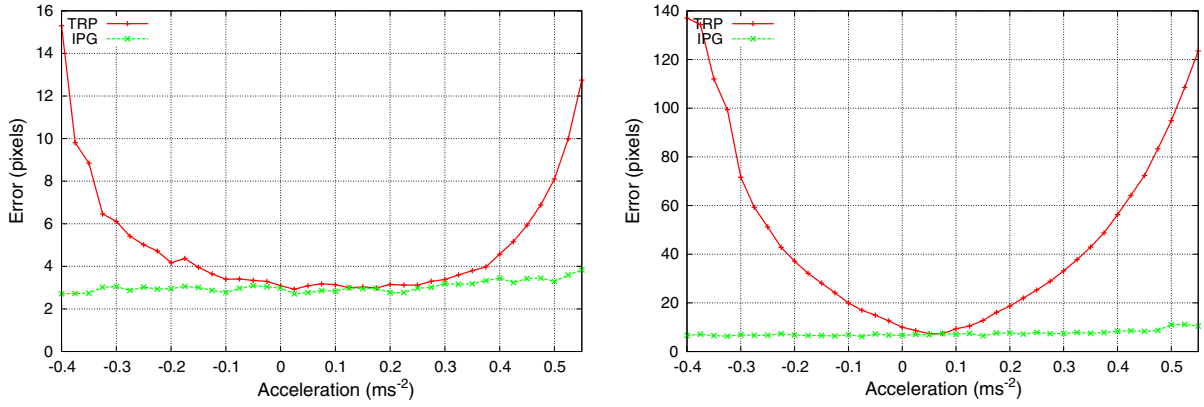
Fig. 14. Snapshot of the Blender *Graph Editor* showing the Z coordinate for the camera as a function of time, for a particular acceleration a_z. The initial velocity v_z is computed to get the camera located at the end of the *tube* in the last frame of the sequence. Note that, for some values of a_z, the initial velocity v_z can make the camera move in opposite direction.

case is the image center). The evaluated parameters are the average Euclidean distance to the true FoE, in pixels, and its corresponding angle, in degrees, for the whole video sequence. For the computation of the angle, we used the well known *cosine* formula:

$$\cos\theta = \frac{\mathbf{X}_{F_G} \cdot \mathbf{X}_F}{\| \mathbf{X}_{F_G} \| \cdot \| \mathbf{X}_F \|} \tag{24}$$

where $\mathbf{X}_{F_G} = (x_{F_G}, y_{F_G}, f)^T$ is the 3D coordinates of the true FoE in the *Camera Coordinates System* and $\mathbf{X}_F = (x_F, y_F, f)^T$ the estimated one.

In Fig. 13 some visual results of the performance of the proposed algorithm are shown. On the other hand, a numerical comparison between the algorithms evaluated is provided in Fig. 11. Panels (a) and (b) show the average Euclidean distance to the true FoE in pixels as a function of the pan angle of the camera α, for a tilt angle β equal to 0. Note that for pan angles between 90° and 180° the



(a) $\alpha = 0^\circ$ (b) $\alpha = 30^\circ$

Fig. 15. Average Euclidean distance to the true FoE, in pixels, as a function of the camera acceleration a_z, in $m \cdot s^{-2}$ for different values of the pan angle (α).

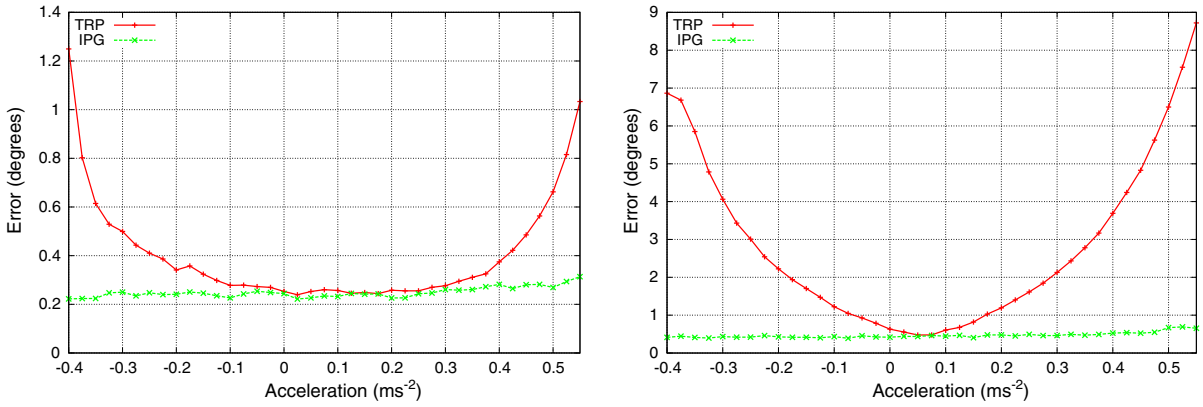


Fig. 16. Error angle, in degrees, as a function of the camera acceleration a_z, in $\mathrm{m} \cdot \mathrm{s}^{-2}$ for different values of the pan angle (α).

camera is looking backwards, and so, the *Focus of Contraction* is to be computed, instead of the FoE. Furthermore, when the pan angle is 90°, the image plane is parallel to the motion vector, and the FoE lies at infinity. We found experimentally that for angles close to 90° none of the algorithms yielded valid results. For those reasons, the whole range of the pan angle has been divided into two different graphics, one from 0° to 65°, and the other from 115° to 180°. Fig. 11 (c) and (d) shows the same results for a tilt angle equal to 5°, and (e) and (f) for a tilt angle of 10°.

As can be seen from these results, the algorithm based in Visual Odometry (VO) yields poor results. This is obvious, because it is not an algorithm designed for the computation of the FoE, although it can be computed from its output. Nevertheless, it can serve for comparisons between VO and FoE algorithms. Likewise, the algorithm based on Flowfield Projections (FLP) also yields inaccurate results. On the other hand, the algorithm proposed in this paper (TRP) and the one based on Intersection Point (IPG) perform similarly. The Euclidean error increases for all the algorithms as the image plane of the camera tends to be parallel to the motion vector, since the FoE tends to be located far away from the image origin of coordinates. For that reason, results are more meaningful if given in terms of the angle error. Fig. 12 are the equivalent results, showing the angle error. In these figures, it can easily be seen that for pan angles approaching 90°, that is, when the FoE lies outside the image plane, the proposed algorithm performs better than the rest.

In Section 3.2.1, we assumed constant velocity for the camera. In the next experiment, we will test the performance of the evaluated systems to camera accelerations. To this end, using the same scenario designed in the previous experiment, we programmed the camera to move with an *Uniformly Accelerated Motion*, thus the camera center will be located at:

$$\mathbf{C}(t) = \mathbf{v}t + \frac{\mathbf{a}t^2}{2} \tag{25}$$

where $\mathbf{C}(t) = (0, 0, z)^{\mathrm{T}}$ are the coordinates of the camera center of projection, $\mathbf{v} = (0, 0, v_z)^{\mathrm{T}}$ is the translation vector, and $\mathbf{a} = (0, 0, a_z)$ its acceleration. Fig. 14 shows the camera location z in the Blender *Graph Editor* for a particular value of a_z.

Fig. 15 (a) shows the average Euclidean distance to the true FoE as a function of the camera acceleration, for a pan angle equal to 0° while Fig. 15 (b) for a pan angle equal to 30°, and tilt angles equal to 0 in both cases. Fig. 16 (a) and (b) shows the same result in terms of the error angle. Note that, from Table 1, the camera average speed is $12.5\mathrm{m} \cdot \mathrm{s}^{-1}$. As can be seen, since the hypothesis of constant velocity does not hold for these sequences, the proposed algorithm does not perform well, thus, showing a limitation of this algorithm. Comparisons have been performed only with the IPG algorithm, since the other two algorithms yielded worse results.

4.2. Real video data set

To test the performance of the evaluated systems on real images, a set of video sequences extracted from the KITTI benchmark, along with some videos recorded with an on-board camera have been used. For the last case, we employed a LG L40 and an iPhone 4 smart-phones installed in the windscreen of a vehicle using a standard windshield mount. In all the cases, the vehicle was driven on a straight road at approximately constant speed, with the camera pointing forwards. In this situation, the focus of expansion, as long as the vehicle follows the road, is located on the vanishing point of the road, as in Fig. 7, and can be easily annotated manually.

The set of videos includes different situations, from high textured scenes, to very challenging, low textured ones. An overview of the videos and their main characteristics are given in Table 2. In Fig. 17 an example for each group is given. For the scenes of the group 4, the vanishing point of the road is not visible, and so the location of their FoE has been annotated manually by projecting scene lines, like road marks or the line defined by the street lamps.

Again, the evaluated parameters are the average Euclidean distance to the true FoE, in pixels, and its corresponding angle, for the whole video sequence. For the KITTI benchmark, the horizontal *angle of view* is close to 90° (see [22]). For the recorded videos, the horizontal *angle of view* of the LG L40 back camera is approximately 52°, while for the iPhone 4, approximately 37°.

Table 3 shows the average error, for the different video groups and the different algorithms. As we can see, the method based on the computation of the intersection point (IPG) performs similarly that

Table 2
Videos employed in the experiments. The *Texture* column refers to the average amount of texture on the video frames. The column *FoE location* denotes whether the FoE is located inside the image plane or not.

	Texture	FoE location	# videos	# frames
Group 1	Whole	Inside	10	2103
Group 2	Half	Inside	10	5491
Group 3	Minimal	Inside	6	10,191
Group 4	Whole	Outside	5	12,807

(a) Group 1

(b) Group 2

(c) Group 3

(d) Group 4

Fig. 17. Sample frames for the different groups employed in the evaluation. For group 4, a graphical representation of the FoE annotation process is also given.

the algorithm proposed in this paper for the less challenging scenes (group 1). On the other hand, the algorithm based on the Flowfield Projections (FLP) performs worse. The VO algorithm has not been included in this experiment, since most of the results yielded by the algorithm were wrong.

Nevertheless, the IPG algorithm starts decreasing its performance as the amount of image texture reduces, as can be seen in Table 3 for the videos of groups 2 and 3. For instance, group 3 is composed of videos with texture mainly in the middle of the image, with a textureless sky and road. In this situation, lines defined by the optical flow vectors tend to be horizontal, and the computation of the intersection point of quasi-parallel lines becomes inaccurate. For the proposed algorithm, on the other hand, the relative angles between the trajectories is unimportant, since vanishing points are computed independently.

Results get very inaccurate when the FoE is located outside the field of view of the camera, as it is the case for the videos of the group 4. In this case, only the proposed algorithm is able to locate the focus of expansion correctly. In Fig. 18 some successful graphical results are shown.

Table 3
Average error in pixels (and its corresponding angle in degrees) for the sequences employed in the comparisons, for the four algorithms evaluated.

	TRP	IPG	FLP
Overall	23 pixels (1.4°)	31 pixels (3.5°)	151 pixels (10.1°)
Group 1	16 pixels (0.9°)	18 pixels (1.1°)	81 pixels (3.7°)
Group 2	21 pixels (1.2°)	28 pixels (1.7°)	59 pixels (3.2°)
Group 3	29 pixels (1.7°)	39 pixels (2.5°)	54 pixels (3.3°)
Group 4	26 pixels (1.8°)	122 pixels (10.3°)	323 pixels (20.5°)

Finally, we evaluated the real-time applicability of the proposed algorithm. In Table 4 we report the average computation time per image, along with the individual times employed by each block. Times were obtained by averaging the processing time for all the videos. Since the videos for the KITTI benchmark and the new proposed videos have different resolution, results are given separately. Tests were run on an i5 32 bits at 3.2 GHz on a Linux Kernel 3.13.0. As we can see, the proposed algorithm can reach up to 15 fps on a 800×480 pixels (WVGA) image resolution.

5. Conclusions and future work

In this paper, we have proposed a new algorithm for the computation of the focus of expansion for a video sequence. The algorithm uses the trajectories of interest points to compute the vanishing point for each trajectory, by means of the *cross ratio* property. A number of challenging videos, both virtual ones generated with Blender and real ones recorded with a smart phone, along with some sequences of the public KITTI benchmark have been used to test the algorithm. Results showed that the proposed algorithm can improve the performance of classical algorithms for challenging sequences. This is specially relevant for low textured scenes, for instance, traffic sequences with textureless top (sky) and bottom (road), and for the

Fig. 18. Sample results for different video sequences.

Table 4
Average time per image employed by the proposed algorithm, for different image sizes.

Image size (pixels)	FoE (ms)	Trajectories (ms)	Total (ms)	Frame rate (fps)
1226 × 370	4.34	78.15	82.49	12.12
800 × 480	8.22	56.59	64.81	15.43

cases where camera orientation with respect to the moving platform makes the focus of expansion lie outside the image plane. Furthermore, a Java implementation of the proposed algorithm can reach up to 15 fps on WVGA (800 × 480 pixels) video resolution, showing its viability for real-time applications. Limitations of the proposed algorithm arise when camera motion is not constant. Thus, when camera acceleration is significant, the algorithm starts decreasing its performance.

The proposed paper allows us to glimpse the potential of using the trajectories of interest points for ego-motion tasks. Although we have focus our work on the computation of the focus of expansion, thus limiting the applicability to straight trajectories, work will concentrate in widening the study to not straight paths, allowing us to compute not only translation, but also the rotational vector related with the camera ego-motion.

Acknowledgments

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

References

[1] F. Raudies, H. Neumann, A Review and evaluation of methods estimating ego-motion, Comput. Vis. Image Underst. 116 (2012) 606–633.
[2] J.L. Barron, D.J. Fleet, S.S. Beauchemin, Performance of optical flow techniques, Int. J. Comput. Vis. 12 (1994) 43–77.
[3] A. Geiger, P. Lenz, R. Urtasun, Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite, Conference on Computer Vision and Pattern Recognition (CVPR), 2012.
[4] H. Badino, D. Huber, T. Kanade, The CMU Visual Localization Data Set., Computer Vision Group 2011.
[5] S. Negahdaripour, B.K.P. Horn, A direct method for locating the focus of expansion, Comput. Vis. Graphics Image Process. 46 (3) (1989) 303–326.
[6] R. Sharma, Y. Aloimonos, Early detection of independent motion from active control of normal image flow patterns, IEEE Trans. Syst. Man Cybern. B Cybern. 26 (1) (1996) 42–52.
[7] C. Herdtweck, C. Curio, Monocular Heading Estimation in Non-stationary Urban Environment, IEEE International Conference on Multisensor Fusion and Information Integration, IEEE, 2012.
[8] D. Sazbon, H. Rotstein, E. Rivlin, Finding the focus of expansion and estimating range using optical flow images and a matched filter, Mach. Vis. Appl. 15 (2004) 229–236.
[9] R. Hartley, A. Zisermann, Multiple View Geometry in Computer Vision, second ed., Cambridge University Press 2003.
[10] F. Woelk, R. Koch, Robust monocular detection of independent motion by a moving observer, International Workshop of Complex Motion, LNCS, 3417, Springer-Verlag 2007, pp. 209–222.
[11] J.K. Suhr, H.G. Jung, K. Bae, J. Kim, Outlier rejection for camera on intelligent vehicles, Pattern Recogn. Lett. 29 (2008) 828–840.
[12] F.C. Wu, L. Wang, Z.Y. Hu, FOE estimation: can image measurement errors be totally "Corrected" by the geometric method?, Pattern Recogn. 40 (2007) 1971–1980.
[13] A. Bak, S. Bouchafa, D. Aubert, Focus of expansion localization through inverse C-Velocity, International Conference in Image Analysis and Processing, Lecture Notes in Computer Science 6978, Springer-Verlag 2011, pp. 484–493.
[14] C. Born, Determining the focus of expansion by means of flowfield projections, In Proc. Deutsche Arbeitsgemeinschaft fur Mustererkennung, 1994. pp. 711–719.
[15] D. Scaramuzza, F. Fraundorfer, Visual odometry: part I— the first 30 years and fundamentals., IEEE Robot. Autom. Mag. 18 (4) (2011)
[16] C. Forster, M. Pizzoli, D. Scaramuzza, SVO: fast semi-direct monocular visual odometry, IEEE International Conference on Robotics and Automation (ICRA), 2014.
[17] Andreas. Geiger, Julius. Ziegler, Christoph. Stiller, StereoScan: dense 3d reconstruction in real-time, IEEE Intelligent Vehicles Symposium, Baden-Baden, Germany, 2011.
[18] T. Tuytelaars, K. Mikolajczyk, Local invariant feature detectors: a survey, Found. Trends. Comput. Graph. Vis. 3 (3) (2008) 177–280.
[19] C. Harris, M. Stephens, A combined corner and edge detector, Alvey Vision Conference, 15, 1988. pp. 147–151. Manchester, UK.
[20] B.D. Lucas, T. Kanade, An iterative image registration technique with an application to stereo vision, Proceedings of the 7th international Joint Conference on Artificial Intelligence, 1981.
[21] D. Comaniciu, V. Ramesh, P. Meer, Real-time tracking of non-rigid objects using mean shift, Proc. 2000 IEEE Conf. Computer Vision and Pattern Recognition, 2, 2000. pp. 142–149.
[22] A. Geiger, P. Lenz, C. Stiller, R. Urtasun, Vision meets robotics: the KITTI dataset, Int. J. Robot. Res. 32 (11) (2013) 1231–1237.