Published in Image Processing On Line on 2017–07–14.
Submitted on 2015–07–27, accepted on 2017–06–09.
ISSN 2105–1232 © 2017 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2017.148

Vanishing Point Detection in Urban Scenes Using Point Alignments

José Lezama[1], Gregory Randall[1], Rafael Grompone von Gioi[2]

[1] IIE, Universidad de la República, Uruguay ({jlezama, randall}@fing.edu.uy)
[2] CMLA, ENS Cachan, France (grompone@cmla.ens-cachan.fr)

Abstract

We present a method for the automatic detection of vanishing points in urban scenes based on finding point alignments in a dual space, where converging lines in the image are mapped to aligned points. To compute this mapping the recently introduced PClines transformation is used. A robust point alignment detector is run to detect clusters of aligned points in the dual space. Finally, a post-processing step discriminates relevant from spurious vanishing point detections with two options: using a simple hypothesis of three orthogonal vanishing points (Manhattan-world) or the hypothesis that one vertical and multiple horizontal vanishing points exist. Qualitative and quantitative experimental results are shown. On two public standard datasets, the method achieves state-of-the-art performances. Finally, an optional procedure for accelerating the method is presented.

Source Code

The MATLAB and ANSI C reviewed source code for this algorithm is available from the web page of this article[1]. Compilation and usage instructions are included in a `README.txt` file. Auxiliary files for computing benchmarking scores in York Urban and Eurasian Cities datasets are also included.

Keywords: vanishing points; Manhattan world; PClines; a contrario; point alignments

1 Introduction

Under the pinhole camera model, straight lines in 3D space are transformed into straight lines in 2D. Moreover, parallel lines in 3D space are projected into lines that converge to a point (perhaps at infinity) known as a *vanishing point* (VP). In the presence of parallel lines, as is common in urban and human-made environments, VPs provide crucial information about the 3D structure of the scene and have multiple applications such as camera calibration, single-view 3D scene reconstruction, autonomous navigation, and semantic scene parsing, to mention a few.

[1] https://doi.org/10.5201/ipol.2017.148

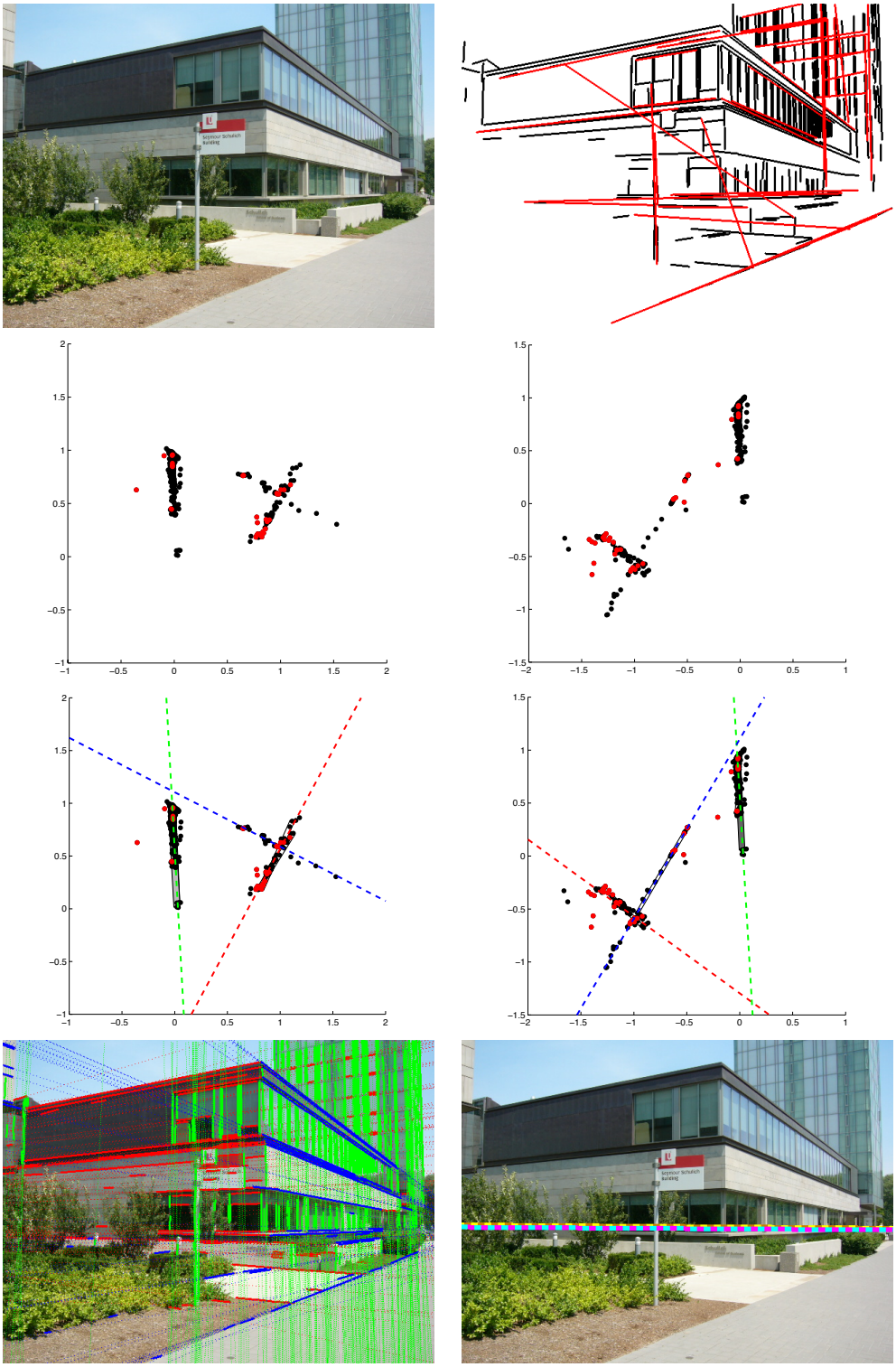


Figure 1: Main steps of the proposed method. **Top-Left:** Input image. **Top-Right:** Steps 1 and 2. Line segments (black) and alignments of line segments endpoints (red). **2nd Row:** Step 3. Lines as points in *straight* (left) and *twisted* (right) PClines spaces. Red dots correspond to alignments of line segments endpoints. **3rd Row:** Step 4. Aligned points detections (shaded rectangles) and the ground truth (colored dashed lines). **Bottom-Left:** Step 6. Final VP associations by enforcing orthogonality. **Bottom-Right:** Step 6. Horizon line estimation (yellow-brown: ground truth, cyan-magenta: ours).

The method presented in this work builds on the advances of various previous works to obtain

an accurate VP detection algorithm. The method uses the line segments detected with the LSD algorithm [9] as the basic oriented elements. The line segments are converted to a dual space, using the PClines point-to-line mappings described by Dubská et al. [6]. Each line in the image space is mapped to a point in the dual space so that converging lines in the image become aligned points. An unsupervised point alignment detector [13] is used twice: First, to group aligned line segments into longer and more precise ones. Second, to find sets of collinear points in the dual space, which correspond to sets of converging lines in the image, thus candidates for VPs.

Once a set of candidate vanishing points is obtained from the clusters in the dual space, the result can be refined by assuming that the scene is a typical urban scene where constraints from the real world can be imposed. In this article we use two possible hypotheses. The first one is the so-called "Manhattan-world" hypothesis, which assumes that the image is dominated by three orthogonal vanishing points, one vertical and two horizontal. The second one assumes that one vertical and an unknown quantity of horizontal, non necessarily mutually-orthogonal VPs exist. In the rest of this document we shall refer to the second hypothesis as "non-Manhattan-world".

The measure of statistical significance [13] of the alignments in the dual space, as well as the 3D constraints from these hypotheses, are used to select the final set of VPs. The core ideas of this method were originally presented in [12]. In this article, we review the method, describe its implementation details, and propose an auxiliary procedure for accelerating the point alignments detection. A related vanishing point detection method using the *a contrario* framework is presented in [2].

This document is organized as follows. In Section 2 we present the fundamentals of the method. In Section 3 we describe the implementation details. In Section 4 we describe an optional procedure to accelerate the method, at the cost of some quality downgrading. Finally in Section 5 we present and comment experimental results.

2 Method Description

The proposed algorithm works in six steps, illustrated in Figure 1 and described in detail in the rest of this document:

1. Detect line segments in the image using LSD [9].
2. Find and group aligned line segment endpoints using the unsupervised point alignment detector* [13].
3. Transform line segments in the image into points in the PClines *straight* and *twisted* dual spaces [6]. In the PClines dual spaces, converging lines become aligned points. Two different dual spaces are used to cope with the unbounded nature of the mapping.
4. Detect point alignments in both dual spaces using the unsupervised point alignment detector* [13]. Detected point alignments, which correspond to converging line segments in the image, are considered as VP candidates.
5. Identify redundant detections (VPs detected in both dual spaces) and refine the position of the candidate VPs.
6. (Optionally) Select the final set of VPs and estimate the horizon line based on one of two hypotheses: Manhattan or non-Manhattan world.

* An optional accelerated version of this procedure is described in Section 4.

2.1 Point Alignment Detector

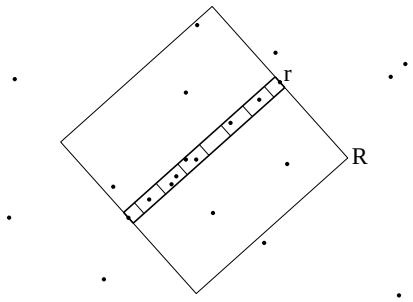


Figure 2: Representation of the candidate rectangle. Figure taken from [13]. In this case, the candidate rectangle r is divided into $c = 6$ boxes, 5 of which are occupied. The local density estimation window is R.

The proposed method uses the point alignment detector introduced in [13]. In this section we shall give a quick and basic summary of the method. For more details, we kindly refer the reader to the original paper [13] and the article detailing its implementation [14]. In the VP detection procedure, this method will be used twice. First, to find alignments of segment endpoints that can contribute to discover extra cues on vanishing directions. Second, to find alignments of points in the dual space, that correspond to converging line segments in the image. Note that the alignment detector does not require the number of clusters to be known in advance, so it can work with images with any number of vanishing points.

Given a set of 2D points, this detector defines point alignments as rectangular clusters. Candidate rectangles are obtained by considering each possible pair of points as endpoints and a set of possible widths. Then, each candidate rectangle is divided into boxes and the boxes occupied by at least one point are counted. Based on the *a contrario* methodology, the idea behind the algorithm is to measure the expected number of occurrences of such an event under a null hypothesis H_0 of independent and uniformly distributed random points. When this expectation is small, the event is termed non accidental and detected.

Let $\mathbf{x}$ be a set of N points. Let r be a candidate rectangle divided into c equal boxes, and R a rectangle surrounding r, used for local point density estimation, see Figure 2. If $b(r, c, \mathbf{x})$ is the observed number of occupied boxes, the expectation of occurrences of such an event under H_0 is approximated by the associated Number of False Alarms (NFA) [13],

$$\mathrm{NFA}(r, R, c, \mathbf{x}) = \frac{N(N-1)}{2} JLC \cdot \mathcal{B}\big(c, b(r, c, \mathbf{x}), p(R, c)\big), \tag{1}$$

where J, L and C are the number of different rectangle widths, local window widths and number of boxes tested for each rectangle r, $\mathcal{B}$ is the tail of the binomial distribution, and $p(R, c)$ is the probability for a box of being occupied taking into account the point density in the local window R. When the NFA of an observed configuration is large, this means that such an event is to be expected under the *a contrario* model and therefore it is irrelevant. On the other hand, when the NFA is small, the event is rare and probably meaningful. A threshold ε is fixed for the NFA. Rectangles with $\mathrm{NFA}(r, R, c, \mathbf{x}) \leq \varepsilon$ are considered ε-meaningful and constitute the detection result of the algorithm (previous to a redundancy reduction step). In [13] it is proved that this threshold effectively bounds the expected number of times that such an event would occur under the *a contrario* hypothesis H_0.

Once all the ε-meaningful alignments are obtained, the algorithm applies a *masking principle* to resolve the redundancy of detections. One detection B is said to be "masked" by another detection

A if, when the elements (points) of A are taken out from B, B is no longer ε-meaningful. In this step, the method keeps the most meaningful detections in terms of the NFA, and discards those that are "masked" by them. Figures 1, 3, 9, 10, 11, 12 and 13 show alignments detected by this algorithm (parallel black lines). In Section 4 a method for accelerating the search of point alignments is discussed. We refer the reader to [13] and [14] for further detail on the masking principle.

2.2 Segment Endpoint Alignments

The aim of this step is to exploit the alignment of features that a line segment detector alone cannot capture. For example, a regular arrangement of equally sized vertical poles along a road reveals a vanishing direction that would be unnoticed by the line segment detector, which would only discover the individual poles. A similar behavior with the vertical borders of windows can be seen in Figure 3. To exploit this type of cue, the point alignment detector of Section 2.1 is used to find alignments among the endpoints of the line segments detected by LSD [9]. In addition, this step increases the direction accuracy of short line segments by grouping them and brings an improvement to the VP detection performance[2].

This kind of strategies for avoiding short line segments and favoring longer, more precise lines, was already used in the literature [1, 10, 8, 17]. Those techniques could be applied in the present case, with the possible exceptions of [10, 8] due to the excessive computational cost. These methods work well for merging short and aligned line segments into longer ones. Nevertheless, the proposed method has the advantage of also detecting alignments which are orthogonal to the actual line segments found, see Figure 3.

The procedure is as follows. First, line segments are grouped by length and orientation. The idea is that joinable line segments should be similar in length and orientation. This also reduces the number of input points to the alignment detector, which significantly reduces the computation time. A single threshold τ is used on the length, and angular steps of 30° are used to group them by orientation, with a 10° overlap between groups. These values have been found to work well empirically. The objective is to connect line segments that share the same orientation – e.g. the borders of the windows of a building – or endpoints of parallel line segments – e.g. an array of vertical structures. For each group of line segments, the point alignment detector is run over the line segment endpoints. This produces a new set of line segments. At the end of this stage, short (and therefore inaccurate) line segments are discarded. The final list of segments is composed of the long line segments and the line segments found from the alignment of endpoints among both the short and long ones. Figure 3 shows some examples of the grouping by length and orientation and the resulting detections. It can be seen that grouping the borders of windows creates long line segments that follow the direction of the building structure. It is worth noting that some false or redundant detections are not harmful to the overall method, since they will only contribute to a few extra points in the dual space.

2.3 PClines Parameterization

The problem of finding converging lines in the image can be posed as a point alignment detection problem by parameterizing the lines as points in a suitable dual space, where converging lines are mapped to aligned points. In such a space, the problem of VP detection has a simple interpretation: the detection of elongated clusters of points. Our method uses the PClines parameterization [6], see Figure 4. The *straight* version uses a parallel coordinate system, where the horizontal x axis in the image is represented as the vertical v axis in the dual space, and the vertical y axis in the image is

[2]On average 2% with the metric used in Section 5.

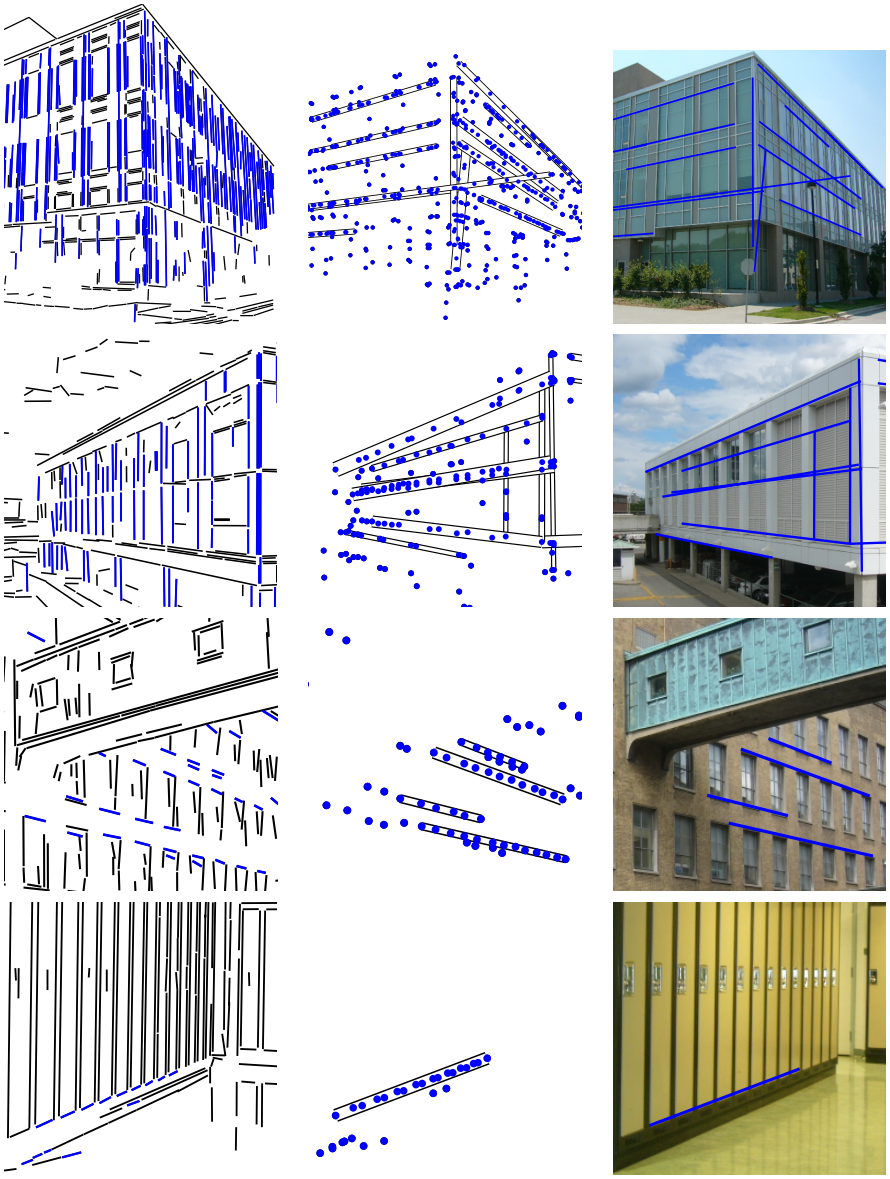


Figure 3: Examples of detected alignments of line segment endpoints. Best viewed in electronic format. **Left**: original line segments, highlighting in blue one group of length and orientation. **Center**: line segment endpoints and the detected alignments for that group (parallel black lines). **Right**: additional oriented features obtained by the method (blue lines). In the top two rows, parallel line segment endpoints are connected to recover structural directions that were not captured by the line segment detector alone. In the bottom rows, short, inaccurate line segments are joined into longer, more accurate ones. Figure taken from [12].

represented as a vertical line passing through $u = d$ in the dual space. Thus, a point $A = (A_x, A_y)$ in the image is represented in the dual space by a line passing by $(0, A_x)$ and (d, A_y). A line joining multiple points in the image is represented in the dual space as a point that lies at the intersection of the lines representing those points. Note that points in dual space can be arbitrarily far away from the origin [6]. To overcome this unboundedness problem, the *twisted* version of the PClines transform is also used, where the x axis stays in the ordinates axis but the $-y$ axis is transformed into a vertical line at $u = -d$. The value of d is set to 1. The second row of Figure 1 shows example results of the transformation for real data. The detailed algorithm is presented in Section 3.4.

By using both the *straight* and *twisted* transforms, it is guaranteed that all vanishing points in the image are represented as a bounded set of points in at least one of the two spaces. The alignment

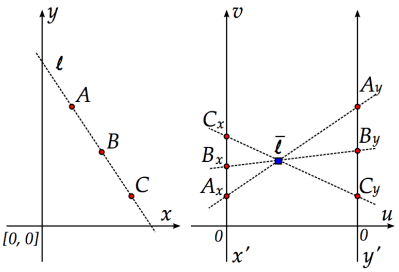


Figure 4: Schematic of the *straight* PClines transform. **Left**: three points and a line in the image $(x,\ y)$ space. **Right**: their representation in the PClines $(u,\ v)$ *straight* dual space. This figure was taken from [6].

detector works in a bounded rectangular domain, so a convenient rectangle has to be chosen in each space, and discard for each of them the points falling outside. To estimate a convenient rectangle for each space, we conducted the following two simple experiments.

First we generated 100,000 random line segments and converted them using PClines (Figure 5). The line segments' endpoints were drawn independently from a uniform distribution on the image domain. We found that most transformed lines fall in the $[-1,\ 2] \times [-1,\ 2]$ range for the straight space, and $[-2,\ 1] \times [-1.5,\ 1.5]$ range for the twisted space. We repeated this experiment for the 96,423 line segments extracted from the York Urban and Eurasian Cities VP datasets (Section 5.2) and observed that their distribution is different but the range of interest is the same as before. Figure 6 shows the data for the points in the datasets.

Based on these observations, the limits of the dual domains where the point alignment detector will be executed are set to $[-1,\ 2] \times [-1,\ 2]$ for the *straight* transform and $[-2,\ 1] \times [-1.5,\ 1.5]$ for the *twisted* transform.[3] Points that fall outside each domain are discarded.

Once the alignment detector is run in each dual space, each alignment found determines a candidate vanishing point $\mathbf{v}_i$ in the image, with an associated meaningfulness NFA_i (Section 2.1). In the next step, the location of the VPs will be refined, and redundant VP detections (detections produced in both dual spaces) will be removed.

2.4 Refinement

Each rectangle candidate of the point alignment detector (Section 2.1), applied in PClines space, is defined by a pair of points. Thus, it represents in the image the intersection of two lines, which forms an initial candidate VP, that needs to be refined. The refinement is done in the following way. Given a line segment $\mathbf{l}$ and a VP candidate $\mathbf{v}_i$, the consistency between $\mathbf{l}$ and $\mathbf{v}_i$ is considered as the angle between the direction of $\mathbf{l}$ and the ideal line passing through $\mathbf{v}_i$ and the midpoint of $\mathbf{l}$ (see Figure 7). This consistency measure is often used in the literature [5, 15]. The subset of line segments consistent with $\mathbf{v}_i$ is obtained by setting a threshold θ on this angle. Finally, the function for updating the VP estimate of [3] is used on this subset, which minimizes the weighted sum of the square of perpendicular distances from the VPs to the lines defined by the line segments. The weights are given by the segments lengths. Due to the good quality of the initial clusters, a single refinement iteration is enough.

After refining the location of the VP in the image, the VPs that were detected in both dual spaces are identified with a distance threshold δ and only the one with the best NFA value is kept. The implementation details of the candidates refinement and redundant detections removal are presented in Section 3.6.

[3]Note that in [12] the *twisted* space range used was $[-2,\ 1] \times [-2,\ 1]$.

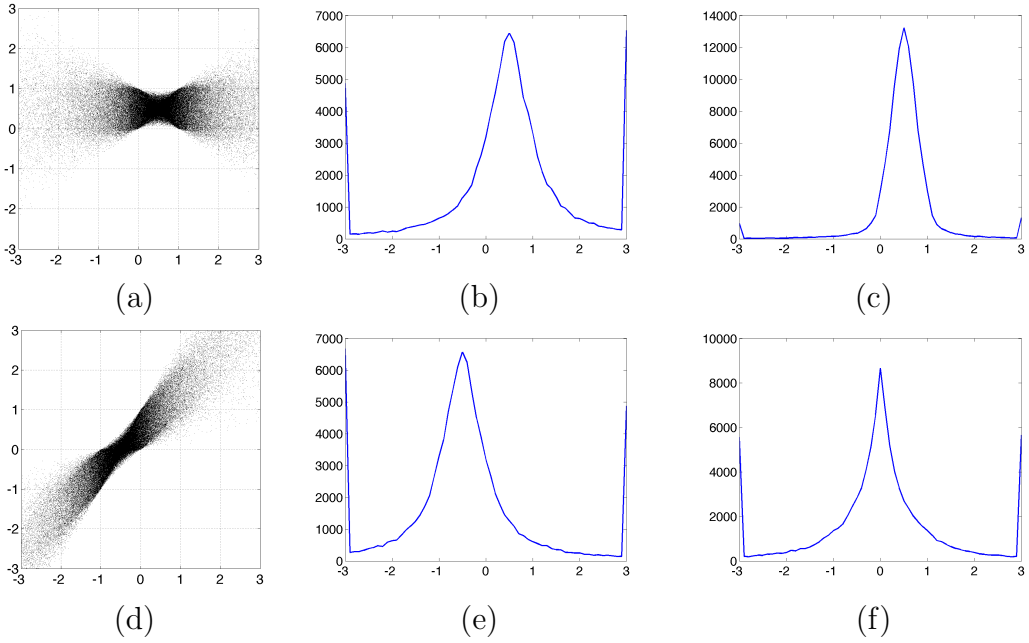


Figure 5: We converted 100,000 random lines in [0, 1]x[0, 1] using PCLines and plotted the obtained points in the *straight* and *twisted* spaces. We computed the histograms of the abscissas and ordinates to study the range where it is reasonable to look for alignments in the dual space. **(a)** Points in the *straight* space. **(b)** Histogram of the abscissa of the points in the *straight* space. The majority of them fall in the [-1, 2] range. **(c)** Histogram of the ordinate of the points in the *straight* space. The majority of them fall in the [-1, 2] range. **(d)** Points in the *twisted* space. **(e)** Histogram of the abscissa of the points in the *twisted* space. The majority of them fall in the [-2, 1] range. **(f)** Histogram of the ordinate of the points in the *twisted* space. The majority of them fall in the [-1.5, 1.5] range.

2.5 Geometrical Constraints from 3D World

Figure 8 (b) shows the intermediate result of the algorithm, where all the obtained candidate VPs are shown. In what follows, geometrical constraints valid for typical urban scenes are imposed to refine the final set of detected VPs. Figure 8 (c) shows an example result of enforcing such constraints. Typically, two hypotheses about the urban 3D world that the scene is depicting can be assumed.

The first one is the "Manhattan-world" hypothesis, which assumes that the image is dominated by three orthogonal vanishing points: one vertical and two horizontal. This assumption greatly simplifies the search for valid VPs, since it allows to discard VPs that do not comply with the orthogonality constraint and also to produce a third VP by imposing orthogonality when only two VPs are available.

The second, more relaxed hypothesis consists of considering one vertical VP and an unknown quantity of horizontal ones, that are not necessarily mutually orthogonal. In that case, the algorithm proceeds by identifying the vertical vanishing point and based on the vertical VP it determines the horizontal VPs. The procedure followed by the method under these two cases is detailed in the rest of this section.

In each case, using the obtained vanishing points, the method will estimate the horizon line in the image.

2.5.1 Manhattan World

When the camera parameters are known or have been correctly estimated, the candidate VPs can be represented in the Gaussian sphere. The Gaussian sphere is a unit sphere whose center is located at the center of the camera. A vanishing point in the image corresponds to a point in the Gaussian

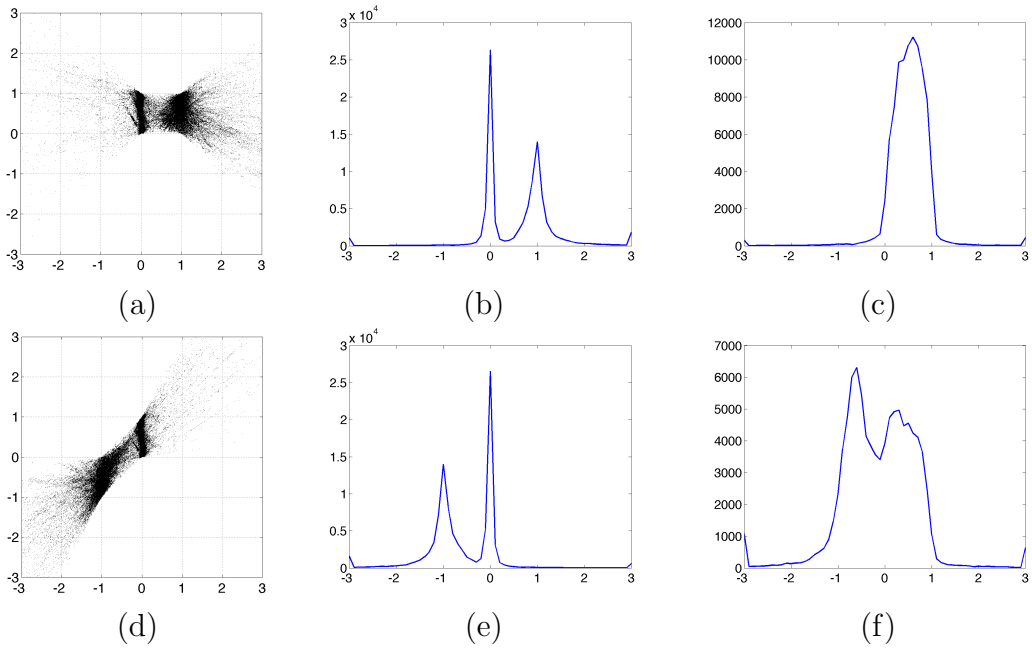


Figure 6: We converted all the straight segments in the YUD and ECD benchmark datasets (96,423 in total) using PCLines and plotted the obtained points in the *straight* and *twisted* spaces. We repeated the experiment of Figure 5. We observe that the same ranges apply for the majority of the points. **(a)** Points in the *straight* space. **(b)** Histogram of the abscissa of the points in the *straight* space. **(c)** Histogram of the ordinate of the points in the *straight* space. **(d)** Points in the *twisted* space. **(e)** Histogram of the abscissa of the points in the *twisted* space. **(f)** Histogram of the ordinate of the points in the *twisted* space.

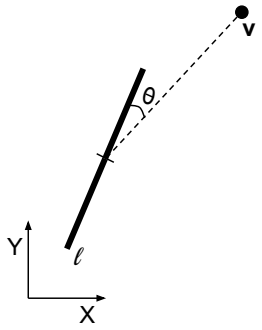
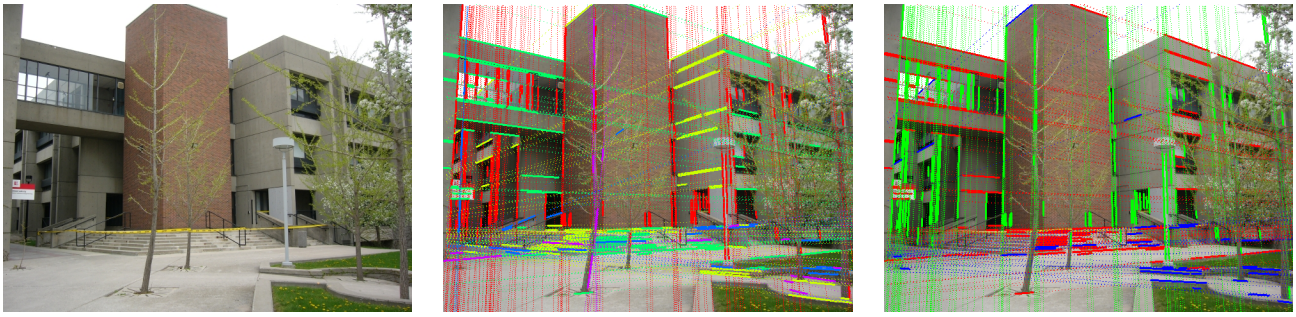


Figure 7: The angular error between a line segment $\mathbf{l}$ and a vanishing point $\mathbf{v}$ is computed as the angle between the line segment direction and the line formed by the line segment midpoint and the vanishing point.

sphere (a 3D unitary vector), which is the intersection between surface of the sphere and the ray joining the camera center and the vanishing point. To perform this conversion, the geometry of the camera needs to be known, in particular the principal point and the focal length in pixel units. The simple operations involved in this conversion are explained in detail in Section 3.7.1. In what follows, when we refer to orthogonal VPs or to the angle between VPs, we will be referring to their representation in the Gaussian sphere. The representation space we are referring to will be clear in each context.

The "Manhattan-world" hypothesis assumes that the image contains three orthogonal VPs. To identify orthogonal triplets of VPs, the candidate vanishing points are converted to vectors in the Gaussian sphere, and orthogonality is determined based on a threshold on the angles between the vectors. Once the orthogonal triplets of VPs have been found, the one with the lowest sum of NFA



(a) (b) (c)

Figure 8: Orthogonal constraints from the 3D world are applied to refine detections. **(a)** original image. **(b)** All candidate VPs (5). **(c)** final VPs after imposing Manhattan-world orthogonality constraints.

values of the detections is selected as the valid VP triplet. When no orthogonal triplet is found, the most significant orthogonal pair is taken (in terms of summed NFA) and the third VP is obtained by the cross product of the two orthogonal vectors.

To compute the horizon line, the vertical VP is identified as the one with the largest absolute vertical coordinate. The remaining two VPs are the horizontal ones. The horizon line is simply the line passing through the horizontal VPs. Of course, this trivial solution requires the camera to be not far from horizontal with respect to the captured urban scene.

2.5.2 Non-Manhattan World

The Manhattan-world hypothesis is in general valid for simple urban scenes. However, for more complex scenes, this assumption is too strong and must be relaxed. A more relaxed hypothesis is the existence of one vertical and an unknown quantity of horizontal VPs, not necessarily mutually-orthogonal. We shall refer to this hypothesis as "non-Manhattan world". It is important to note that the unsupervised alignment detector used to compute candidate VPs in section 2.3 does not require the number of clusters to be known *a priori*, it is automatically computed. Thanks to this, the method can manage images with any number of VPs.

The procedure followed by the algorithm under this assumption is described next. It starts by identifying which of the candidate VPs is the vertical VP. Then, based on the estimated vertical VP and a set of constraints, it identifies the horizontal VPs.

To determine the vertical VP, the first step is to compute a set of vertical VP candidates, among all VPs. The vertical distance from a VP to the center of the image, and the angle with respect to the vertical axis of the image[4] are evaluated to form the subset of possible vertical VPs. Among this subset, the most meaningful VP (with lowest NFA) is kept as the zenith or vertical VP. Here again, in order for this heuristic to work the target scene is expected not to be heavily rotated with respect to the 3D horizontal plane.

The rest of the VPs are considered as candidates for horizontal VPs. To compute the final set of relevant horizontal VPs, these are filtered out based on two criteria: First, the VPs are converted to the Gaussian sphere and the VPs that are far from being orthogonal to the vertical VP are discarded. Note that to do this the focal length in pixel units f and the principal point must be known. However, these are used only to discard non-horizontal VPs, so an approximate value is generally sufficient. Second, the horizontal VPs that are obtained from clusters of parallel lines in the image – for which the problem of setting the horizon line height is undetermined – are also discarded, based on a threshold on the distance from the VP to the image center.

[4]We refer to the vertical axis as a vertical line passing through the image center.

Because the horizon line is perpendicular to the line connecting the principal point and the vertical VP, the problem of estimating the horizon line from a set of candidate horizontal VPs is one-dimensional. The horizon line is obtained by a weighted vote, where each horizontal VP casts a vote and the weights are based on the NFA (Section 2.1) of each VP detection. The weight w_i for the VP $\mathbf{v}_i$ is

$$w_i = \left(\frac{(-\log_{10} \mathrm{NFA}_i)}{\sum_j (-\log_{10} \mathrm{NFA}_j)} \right)^2 . \tag{2}$$

The implementation details of this procedure are described in Section 3.7.2.

3 Algorithm

3.1 Main Algorithm

Algorithm 1 presents the pseudo code for the main VP detection procedure. Each subroutine is detailed and explained in the rest of this section. Parameters are detailed in each subroutine and in Table 1. The following notation will be used:

- bold lowercase letters for vectors (e.g. $\mathbf{v}$)
- bold uppercase letters for lists (e.g. $\mathbf{L}$)
- uppercase letters for matrices (e.g. Q).
- $x_{\mathbf{v}}$ and $y_{\mathbf{v}}$ for the horizontal and vertical coordinates of a point $\mathbf{v}$ in the image, respectively.

Algorithm 1: Main body of the algorithm

input : An image I. Parameters τ, θ, δ, ζ, f, $\mathbf{p}$, γ_S, γ_R, ω, λ, κ, (see Table 1).
output: A list $\mathbf{V}$ of detected vanishing points and $\mathbf{l}_{hor}$, the estimated horizon line.

$\mathbf{L} \leftarrow \texttt{LSD}(I)$
$\mathbf{L} \leftarrow \texttt{denoise\_lines}(\mathbf{L}, \tau)$ *// Algorithm 2*
$\mathbf{X}_{straight}, \mathbf{X}_{twisted} \leftarrow \texttt{pclines\_transform}(\mathbf{L})$ *// Algorithm 3*
$\mathbf{A}_{straight} \leftarrow \texttt{detect\_alignments}(\mathbf{X}_{straight})$ *// Algorithm 5*
$\mathbf{A}_{twisted} \leftarrow \texttt{detect\_alignments}(\mathbf{X}_{twisted})$
$\mathbf{V}_{straight} \leftarrow \texttt{inverse\_pclines\_transform}(\mathbf{A}_{straight})$ *// Algorithm 4*
$\mathbf{V}_{twisted} \leftarrow \texttt{inverse\_pclines\_transform}(\mathbf{A}_{twisted})$
$\mathbf{V} \leftarrow \mathbf{V}_{straight} \bigcup \mathbf{V}_{twisted}$
$\mathbf{V} \leftarrow \texttt{refine\_detections}(\mathbf{V}, \theta, \delta, \zeta)$ *// Algorithm 6*
/ $\mathbf{V}$ is an intermediate result containing VPs before imposing orthogonality constraints */*
if *Manhattan-world* **then**
  $\mathbf{V}, \mathbf{l}_{hor} \leftarrow \texttt{manhattan\_constraints}(\mathbf{V}, f, \mathbf{p}, \gamma_S)$ *// Algorithm 9*
else
  $\mathbf{V}, \mathbf{l}_{hor} \leftarrow \texttt{non-manhattan\_constraints}(\mathbf{V}, f, \mathbf{p}, \gamma_R, \omega, \lambda, \kappa)$ *// Algorithm 14*
end

The input to the algorithm is the image I. The output is a list $\mathbf{V}$ of relevant vanishing points locations, and the estimated horizon line $\mathbf{l}_{hor}$. The method has eleven parameters. Parameters f and $\mathbf{p}$ are given by the geometry of the camera. The threshold τ is applied on the length of line segments,

and is relative to the image size. Thresholds θ, δ and ζ are used to refine the position of the candidate VPs and remove redundant detections. Parameters γ_S and γ_R are angular parameters that are used to determine orthogonality or non-orthogonality in the Gaussian sphere. The parameters ω, λ and κ are thresholds used to discard or validate candidate VPs. The use of each parameter will be detailed in the section corresponding to each subroutine.

Lines 1 and 2 perform the line segment detection and denoising (grouping and discarding short line segments). Lines 3 to 5 perform the mapping of the line segments into points in the dual spaces, and the detection of alignments in them. Lines 6 to 8 convert the alignments found in the dual space to points in the image, which are the candidate VPs. The procedure in line 9 refines the position in the image of each candidate VP. In line 10, the algorithm is divided into two possible paths: assuming the Manhattan-world hypothesis or not.

3.2 Line Segment Detection

The `LSD` method in line 1 of Algorithm 1 runs the Line Segment Detector of [9] on the image I. The output is a list of line segments identified by their endpoints $((x_1, y_1), (x_2, y_2))$.

3.3 Line Segment Denoising

The aim of this step is to remove short, noisy line segments, and to capture the direction of higher level structures formed by groupings of line segment endpoints. Algorithm 2 presents the pseudo code for this procedure. To achieve its goal, this procedure uses the alignment detector of [13, 14] to find alignments of line segment endpoints. The line segments are grouped by length and orientation. First, they are divided into "short" and "long" segments using a threshold τ. Then, they are subdivided by orientation in angular slots of 30° with 10° overlap.

The alignment detection algorithm is run on the endpoints of the line segments contained in each slot (lines 5 to 7 and lines 9 to 11). The list of final line segments is composed by the long segments and the segments found as endpoint alignments (lines 3, 8 and 12). The benefits of this stage are twofold: First, noisy short segments are removed. Second, additional cues on vanishing directions are obtained when parallel line segments are grouped.

Algorithm 2: `denoise_lines`: Line segments denoising

input : A list of line segments $\mathbf{L}$. A length threshold τ.
output: A list $\mathbf{L}'$ of denoised line segments.

1 $\mathbf{L}_{short} \leftarrow \{l : l \in \mathbf{L}, \texttt{length}(l) \leq \tau\}$
2 $\mathbf{L}_{long} \leftarrow \{l : l \in \mathbf{L}, \texttt{length}(l) > \tau\}$
3 $\mathbf{L}' \leftarrow \mathbf{L}_{long}$
4 **for** $\alpha \in \{0°, 30°, 60°, 90°, 120°, 150°\}$ **do**
5 $\quad \mathbf{L}^{\alpha}_{short} \leftarrow \{l : l \in \mathbf{L}_{\mathbf{short}}, \alpha - 20° \leq \texttt{ang}(l) < \alpha + 20°\}$
6 $\quad \mathbf{x}^{\alpha}_{short} \leftarrow \texttt{endpoints}(\mathbf{L}^{\alpha}_{short})$
7 $\quad \mathbf{A}^{\alpha}_{short} \leftarrow \texttt{detect\_alignments}(\mathbf{x}^{\alpha}_{short})$ *// Algorithm 5*
8 $\quad$ append $\mathbf{A}^{\alpha}_{short}$ to $\mathbf{L}'$
9 $\quad \mathbf{L}^{\alpha}_{long} \leftarrow \{l : l \in \mathbf{L}_{\mathbf{long}}, \alpha - 20° \leq \texttt{ang}(l) < \alpha + 20°\}$
10 $\quad \mathbf{x}^{\alpha}_{long} \leftarrow \texttt{endpoints}(\mathbf{L}^{\alpha}_{long})$
11 $\quad \mathbf{A}^{\alpha}_{long} \leftarrow \texttt{detect\_alignments}(\mathbf{x}^{\alpha}_{long})$ *// Algorithm 5*
12 $\quad$ append $\mathbf{A}^{\alpha}_{long}$ to $\mathbf{L}'$
13 **end**

3.4 Dual Space Transformation

To map lines in the image into points in a dual space, the method uses the PCLines transformation [6]. The fundamentals of this transformation are presented in [6], Section 2. The simple operations required to compute the mapping into the *straight* and *twisted* spaces are described in Algorithm 3 for the direct transform (image to dual space) and in Algorithm 4 for its inverse (dual space to image). Note that in the direct transform, an output domain is fixed, and points falling outside it are discarded (Algorithm 3, lines 14, 17). This is because the output points will be passed to the alignment detector, which can only operate on a restricted domain (Section 3.5). The domains are fixed to $[-1,\ 2] \times [-1,\ 2]$ in the *straight* space and $[-2,\ 1] \times [-1.5,\ 1.5]$ in the *twisted* space.

By using both *straight* and *twisted* spaces, it is guaranteed that each group of converging line segments in the image will appear in at least one of the dual spaces as a bounded set of points. A simple analysis of the transformation shows that lines with an orientation close to 45° are mapped into points close to infinity in the *straight* transform, but are bounded for the *twisted* version. On the contrary, lines with an orientation of -45° are mapped to infinity in the *twisted* transform, but not in the *straight* one.

Algorithm 3: `pclines_transform`: PCLines transformation of line segments in the image into points in dual space.

input : A list of line segments $\mathbf{L}$, $d = 1$. Height and width of the image, H and W.
output: Two lists $\mathbf{X}_{straight}$ and $\mathbf{X}_{twisted}$ of points in the dual space.

1 $\mathbf{X}_{straight} \leftarrow \emptyset$
2 $\mathbf{X}_{twisted} \leftarrow \emptyset$
3 **for** $\mathbf{l} \in \mathbf{L}$ **do**
4 $\quad (x_1, y_1), (x_2, y_2) \leftarrow$ `endpoints(l)`
5 $\quad x_1 \leftarrow \frac{x_1}{W}, y_1 \leftarrow \frac{y_1}{H}, x_2 \leftarrow \frac{x_2}{W}, y_2 \leftarrow \frac{y_2}{H}$
6 $\quad d_y \leftarrow y_2 - y_1$
7 $\quad d_x \leftarrow x_2 - x_1$
8 $\quad m \leftarrow \frac{d_y}{d_x}$
9 $\quad b \leftarrow \frac{(y_1 \cdot x_2 - y_2 \cdot x_1)}{d_x}$
10 $\quad u_{straight} \leftarrow \frac{d}{(1-m)}$
11 $\quad v_{straight} \leftarrow \frac{b}{(1-m)}$
12 $\quad u_{twisted} \leftarrow \frac{-d}{(1+m)}$
13 $\quad v_{twisted} \leftarrow \frac{-b}{(1+m)}$
14 $\quad$ **if** $(u_{straight}, v_{straight}) \in [-1, 2] \times [-1, 2]$ **then**
15 $\quad\quad$ Append $(u_{straight}, v_{straight})$ to $\mathbf{X}_{straight}$
16 $\quad$ **end**
17 $\quad$ **if** $(u_{twisted}, v_{twisted}) \in [-2, 1] \times [-1.5, 1.5]$ **then**
18 $\quad\quad$ Append $(u_{twisted}, v_{twisted})$ to $\mathbf{X}_{twisted}$
19 $\quad$ **end**
20 **end**

3.5 Alignment Detection in Dual Space

To find clusters of aligned points in the dual space, the method uses the unsupervised alignment detector of [13], whose implementation is detailed in [14]. A relaxed threshold of $\varepsilon = 10$ in the number

Algorithm 4: `inverse_pclines_transform`: Inverse transformation of lines in PCLines space to points in the image

input : A list of lines $\mathbf{L} = \{\mathbf{l} : v = m \cdot u + b\}$. The PCLines space identifier (*straight* or *twisted*). Parameter $d = 1$. Height and width of the image, H and W.
output: A list of points $\mathbf{X} = \{(x, y)\}$ in the image domain.

$\mathbf{X} \leftarrow \emptyset$
for $\mathbf{l} : v = m \cdot u + b \in \mathbf{L}$ **do**
  $x \leftarrow b$
  if *straight* **then**
    $y \leftarrow d \cdot m + b$
  else if *twisted* **then**
    $y \leftarrow -(-d \cdot m + b)$
  end
  Append $(x \cdot W, y \cdot H)$ to $\mathbf{X}$
end

of false alarms is always used (it allows in theory some 10 false alarms per image). Since future steps in the algorithm will refine relevant detections and remove spurious ones, false positives are not a serious problem. Furthermore, the relaxed detection threshold can help by finding alignments at the limit of detection. The exact same version of the algorithm and its parameters is used in Algorithm 1, lines 4 and 5 and in Algorithm 2, lines 7 and 11. Algorithm 5 presents a short pseudo code for this procedure. Note that the domain of the input points must be defined and passed to the algorithm. When working in the image, as in Algorithm 2, the domain is the same as the image domain ($[0, W] \times [0, H]$). When working in the *straight* and *twisted* dual spaces, the domains are $[-1, 2] \times [-1, 2]$ and $[-2, 1] \times [-1.5, 1.5]$ respectively, as explained in Section 3.4. For clarity, this has not been explicited in the calls to `detect_alignments` in Algorithms 1 and 2.

Algorithm 5: `detect_alignments`: Point alignments detection.

input : A list of points $\mathbf{X}$, a rectangular domain D, the Number of False Alarms threshold $\varepsilon = 10$.
output: A list $\mathbf{A}$ of alignment detections.

$\mathbf{A} \leftarrow$ `detect_alignments`$(\mathbf{X}, D, \varepsilon)$ *// Detailed in [14]*

Lines corresponding to alignments detected in the dual space are converted to points in the image space by using the inverse PCLines transform described in Algorithm 4.

3.6 Vanishing Point Detections Refinement

The point alignments detections in both PCLines dual spaces constitute the first candidates for vanishing points (recall that lines in the dual space correspond to points in the image). Because of the nature of the point alignment detector, the direction of the detected alignments is given by the two points at the extremes. This direction is of course biased and needs to be refined. The refinement procedure consists of two parts, as described in Algorithm 6. First, the candidate VPs are refined using the line segment information in the image domain. This process is described in Algorithm 7. Second, the set of candidate VPs is searched for redundant detections. This process is detailed in Algorithm 8.

To refine a candidate VP the following two steps are taken. First, the group of line segments converging to the candidate VP is searched for. This search is done only among the line segments detected by LSD (not the ones obtained as alignments of endpoints in Section 3.3). The angle between a line segment and the line passing by the midpoint of the line segment and the candidate VP is considered (see Figure 7). Thresholding this angle, the list of line segments converging to the candidate VP is obtained (Algorithm 7, lines 3 to 8). Considering this subset of line segments, the position of the candidate VP is refined by using the VP update rule of [3]. This update rule finds the point in the image space minimizing the weighted sum of the geometric distances between the VP and the line segments. We kindly refer the reader to [3] for the detailed development of the minimization problem and only give the resulting formula here. The update process is described in lines 11 to 24 of Algorithm 7.

Given N segments $\mathbf{l}_i = [a_i, b_i, c_i]^T$, the following 3x3 matrix Q, which is derived from the formula of the orthogonal distance is computed,

$$Q = \sum_{i=1}^{N} \rho_i^2 \frac{\mathbf{l_i} \mathbf{l_i}^T}{\mathbf{l_i}^T I_s \mathbf{l_i}}, \tag{3}$$

where $I_s = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ and ρ_i are the normalized lengths of the line segments, so the longest line segment has length 1. For a point $\mathbf{v}$ in homogeneous coordinates, we need to find the minimum of $\mathbf{v}^T Q \mathbf{v}$ (sum of squared distances of $\mathbf{v}$ to lines $\mathbf{l_i}$), under the constraint $\mathbf{v}^T \mathbf{p} = 1$, with $\mathbf{p} = (0\ 0\ 1)^T$. Writing the Lagrange formulation of the problem, we need to find a minimum of $\mathbf{v}^T Q \mathbf{v} - \lambda(\mathbf{v}^T \mathbf{p} - 1)$, whose gradient is $2Q\mathbf{v} - \lambda\mathbf{p}$. Hence, the estimated vanishing point $\mathbf{v}$ is given, in homogeneous coordinates, by the solution of the right null-space problem

$$[2Q,\ -\mathbf{p}] \begin{pmatrix} \mathbf{v} \\ \lambda \end{pmatrix} = \mathbf{0}, \tag{4}$$

(Algorithm 7, line 20). Note that $[2Q,\ -\mathbf{p}]$ is a 3x4 matrix and $\begin{pmatrix} \mathbf{v} \\ \lambda \end{pmatrix}$ is a 4x1 vector, because $\mathbf{v}$ is in homogeneous coordinates. In MATLAB, the function `null()` is used to obtain the result.

It could happen that the matrix Q is ill conditioned, for example when the line segments in the group are mostly parallel. The method detects ill conditioned cases by measuring the normalized distance between the original VP and the refined VP. If it is above a certain threshold ζ, the original VP is kept (Algorithm 7, lines 21 to 24).

The second step in the refinement is the removal of the redundant detections, described in Algorithm 8. These redundancies can have two causes. One type is caused by detections corresponding to the same VP found both in the *straight* and *twisted spaces*. The alignment detection algorithm can also produce some redundant detections when a noisy cluster of aligned points triggers multiple thin detections. The identification of redundant detections is done by running single-link agglomerative clustering on the VP candidates in the image (Algorithm 8, line 3). The single-link agglomerative clustering is a common clustering procedure and we shall only briefly describe it. In the single-link agglomerative clustering, each VP starts in a cluster of its own. The clusters are then sequentially combined into larger clusters. At each step, the two clusters separated by the shortest distance are combined. The shortest distance is the minimum distance between two elements (one in each cluster). The algorithm stops when the shortest distance is above a threshold δ. For the detection of redundant VPs, the distances used are normalized, so that

$$d(\mathbf{v}_i, \mathbf{v}_j) = \frac{||\mathbf{v}_i - \mathbf{v}_j||}{\max(||\mathbf{v}_i||, ||\mathbf{v}_j||)}. \tag{5}$$

For each cluster found, only the VP with lowest NFA (most significant) is kept (line 5).

Algorithm 6: `refine_detections`: Post-processing of alignment detections.

input : A list of candidate VPs $\mathbf{V}$. An angle threshold θ and distance thresholds δ and ζ
output: A list $\mathbf{V}'$ of refined candidate vanishing points.

1 $\mathbf{V}' \leftarrow$ `refine_vps`$(\mathbf{V}, \mathbf{L}, \theta, \zeta)$; *// Algorithm 7*
2 $\mathbf{V}' \leftarrow$ `remove_redundancy`$(\mathbf{V}', \delta)$; *// Algorithm 8*

Algorithm 7: `refine_vps`: Refine VP detections.

input : A list $\mathbf{V}$ of candidate VPs. A list $\mathbf{L}$ of line segments. An angle threshold θ and a distance threshold ζ.
output: A list $\mathbf{V}'$ of refined candidate VPs.

1 $\mathbf{V}' \leftarrow \emptyset$
2 **for** $\mathbf{v} \in \mathbf{V}$ **do**
3 $\quad \mathbf{L}' \leftarrow \emptyset$
4 $\quad$ **for** $\mathbf{l} \in \mathbf{L}$ **do**
5 $\quad\quad \mathbf{m} \leftarrow$ `mid_point`$(\mathbf{l})$
6 $\quad\quad \mathbf{l}' \leftarrow \overline{\mathbf{mv}}$ *// Line joining* $\mathbf{m}$ *and* $\mathbf{v}$.
7 $\quad\quad$ **if** *angle*$(\mathbf{l}, \mathbf{l}') < \theta$ **then**
8 $\quad\quad\quad$ Append $\mathbf{l}$ to $\mathbf{L}'$
9 $\quad\quad$ **end**
10 $\quad$ **end**
11 $\quad Q \leftarrow 3 \times 3$ matrix with all zeroes
12 $\quad I_s \leftarrow 3 \times 3$ matrix from (3)
13 $\quad M \leftarrow \max($`lengths`$(\mathbf{L}'))$
14 $\quad$ **for** $\mathbf{l} \in \mathbf{L}'$ **do**
15 $\quad\quad w \leftarrow$ `length`$(\mathbf{l})/M$
16 $\quad\quad Q \leftarrow Q + w^2 \cdot \frac{\mathbf{l}\,\mathbf{l}^T}{\mathbf{l}^T I_s \mathbf{l}}$
17 $\quad$ **end**
18 $\quad \mathbf{p} \leftarrow [0, 0, 1]^T$
19 $\quad A \leftarrow [2Q,\ -\mathbf{p}]$
20 $\quad \mathbf{v}' \leftarrow$ `null`(A) *// Computes the null space and keeps the first 3 components, see Equation* (4).
21 $\quad$ **if** $\frac{||\mathbf{v}-\mathbf{v}'||}{||\mathbf{v}||} < \zeta$ **then**
22 $\quad\quad$ Append $\mathbf{v}'$ to $\mathbf{V}'$
23 $\quad$ **else**
24 $\quad\quad$ Append $\mathbf{v}$ to $\mathbf{V}'$ *// Reject ill-conditioned case.*
25 $\quad$ **end**
26 **end**

3.7 Orthogonality Constraints and Horizon Line Estimation

Once a list of refined candidate VPs is obtained, the last part of the VP detection algorithm consists in selecting the subset of VPs that best correspond to the real vanishing points in the image. To relax this problem, two possible hypotheses are introduced. One is the "Manhattan-world" hypothesis,

Algorithm 8: `remove_redundancy`: Remove redundant VP detections.

input : A lists of VP candidates, $\mathbf{V}$. A distance threshold δ
output: A single list $\mathbf{V}$ of non-redundant VP candidates.

1 $\mathbf{V} \leftarrow \mathbf{V}_{straight} \bigcup \mathbf{V}_{twisted}$
2 $\mathbf{V}' \leftarrow \emptyset$
3 $\mathcal{C} \leftarrow$ `single_link_agglomerative_clustering`$(\mathbf{V}, \delta)$
4 **for** $\mathbf{C} \in \mathcal{C}$ **do**
5 $\quad \mathbf{v}' \leftarrow \arg\min_{\mathbf{v} \in \mathbf{C}} \text{NFA}(\mathbf{v})$
6 $\quad$ Append $\mathbf{v}'$ to $\mathbf{V}'$
7 **end**

which implies that only three orthogonal vanishing points are present in the image. The second one, which we will refer to as "non-Manhattan-world" is the existence of one vertical and multiple, non necessarily mutually-orthogonal horizontal VPs. The final step of the algorithm has two variants, one for each hypothesis. The two variants are described in the rest of this section.

3.7.1 Manhattan World

Algorithm 9 presents the pseudo code for the procedure followed by the method when the Manhattan-world hypothesis can be assumed. The procedure starts by converting the candidate VPs in the image to unit vectors in the Gaussian sphere using the `image_to_gaussian_sphere` routine described in Algorithm 12. To do this, it is necessary to know or estimate the focal length in pixel units and the position of the principal point or image center. After this conversion, triplets of orthogonal VPs are searched for using the `orthogonal_triplets` routine described in Algorithm 10 and the triplet with the lowest combined NFA is selected as the final result (Algorithm 9, line 8). (Recall that each candidate VP has an NFA associated which is the NFA of the corresponding alignment detected in the dual space). When no orthogonal triplets are found (lines 9 to 15), the most significant orthogonal pair is taken using the procedure in Algorithm 11, and the third vector is obtained by the cross product, and then refined using the `refine_vps` procedure of Algorithm 7.

In line 19, the VPs are converted back to image coordinates using the `gaussian_sphere_to_image` procedure described in Algorithm 13. In lines 22 to 24 the vertical VP is identified as the one with the largest vertical coordinate and the remaining two horizontal VPs are used to trace the horizon line.

3.7.2 Non-Manhattan World

We refer to a "non-Manhattan" world scenario as a particular relaxation of the simple "Manhattan" world model. This relaxed hypothesis assumes the image has one vertical VP and multiple horizontal VPs, which are orthogonal to the vertical VP, but not necessarily orthogonal between them. Examples of these scenes can be seen in Figure 13. Under this assumption, the method starts by estimating which of the candidate VPs is the vertical one, and then selecting a possible set of horizontal VPs. This procedure is detailed in Algorithm 14.

In line 1 of Algorithm 14, the `vertical_vp_candidates` routine, detailed in Algorithm 15, is used to select a subset of the VPs that are candidates for the vertical VP. This routine is based on two thresholds on the image coordinates of the VPs. Given a candidate VP $\mathbf{v}$, its vertical coordinate $y_{\mathbf{v}}$ must be outside the image domain (larger than H), and a threshold ω is used for the angle between a vertical line passing through the image center, ($\mathbf{l}_{ver}$ in line 1 of Algorithm 15) and the line connecting

Algorithm 9: `manhattan_constraints`: Final VP and horizon line estimation under the Manhattan-world hypothesis.

input : A list of candidate VPs in image domain $\mathbf{V}$. A threshold γ_S. f, the focal length in pixel units. $\mathbf{p}$, the principal point.

output: A list $\mathbf{V}$ of candidate vanishing points in image domain. The horizon line $\mathbf{l}_{hor}$.

1 $\mathbf{V}_u \leftarrow \emptyset$
2 **for** $\mathbf{v} \in \mathbf{V}$ **do**
3 $\quad \mathbf{v}_u \leftarrow$ `image_to_gaussian_sphere`$(\mathbf{v}, f, \mathbf{p})$ *// Algorithm 12*
4 $\quad$ Append $\mathbf{v}_u$ to $\mathbf{V}_u$
5 **end**
6 $\mathbf{T} \leftarrow$ `orthogonal_triplets`$(\mathbf{V_u}, \gamma_S)$ *// Algorithm 10*
7 **if** $\mathbf{T} \neq \emptyset$ **then**
8 $\quad \{\mathbf{v}_u^{(1)}, \mathbf{v}_u^{(2)}, \mathbf{v}_u^{(3)}\} \leftarrow \arg\min_{\mathbf{t} \in \mathbf{T}} \big(\text{score}(\mathbf{t})\big)$
9 **else**
$\quad$ *// No orthogonal triplet found, use pairs*
10 $\quad \mathbf{P} \leftarrow$ `orthogonal_pairs`$(\mathbf{V_u}, \gamma_S)$ *// Algorithm 11*
11 $\quad \{\mathbf{v}_u^{(1)}, \mathbf{v}_u^{(2)}\} \leftarrow \arg\min_{\mathbf{o} \in \mathbf{P}} \big(\text{score}(\mathbf{o})\big)$
12 $\quad \mathbf{v}_u^{(3)} \leftarrow \mathbf{v}_u^{(1)} \times \mathbf{v}_u^{(2)}$
13 $\quad \mathbf{v}_{img}^{(3)} \leftarrow$ `gaussian_sphere_to_image`$(\mathbf{v}_u^{(3)}, f, \mathbf{p})$ *// Algorithm 13*
14 $\quad \mathbf{v}_{img}^{(3)} \leftarrow$ `refine_vps`$(\mathbf{v}_{img}^{(3)})$ *// Algorithm 7*
15 $\quad \mathbf{v}_u^{(3)} \leftarrow$ `image_to_gaussian_sphere`$(\mathbf{v}_{img}^{(3)}, f, \mathbf{p})$
16 **end**
17 $\mathbf{V} \leftarrow \emptyset$
18 **for** $i \in \{1, 2, 3\}$ **do**
19 $\quad \mathbf{v}_{img}^{(i)} \leftarrow$ `gaussian_sphere_to_image`$(\mathbf{v}_u^{(i)}, f, \mathbf{p})$
20 $\quad$ Append $\mathbf{v}_{img}^{(i)}$ to $\mathbf{V}$
21 **end**
22 $\mathbf{v}^{ver} \leftarrow \arg\max_{\mathbf{v} \in \mathbf{V}}(y_{\mathbf{v}})$ *// $y_{\mathbf{v}}$ is the ordinate of $\mathbf{v}$*
23 $\{\mathbf{v}^{hor1}, \mathbf{v}^{hor2}\} \leftarrow \mathbf{V} \setminus \{\mathbf{v}^{ver}\}$
24 $\mathbf{l}_{hor} \leftarrow \overline{\mathbf{v}^{hor1}\mathbf{v}^{hor2}}$

the image center with the candidate VP ($\mathbf{l_v}$ in line 5 of Algorithm 15). If both conditions are met the VP is added to the list of candidates for the vertical VP (line 6 of Algorithm 15). Of course, this is a trivial procedure and it would fail if the image is heavily rotated, but it works well for approximately horizontal urban scenes, for which the algorithm is intended. Finally, among all the vertical VP candidates, the one with the lowest NFA is chosen (Algorithm 14, line 2).

Once the vertical VP has been determined the rest of the VPs are considered as horizontal VP candidates (line 3). Then, two lists are created. The first one, $\mathbf{V}_{hor}^{ortho}$ (line 5), will contain the horizontal VP candidates that are orthogonal to the vertical VP. The second list, $\mathbf{V}_{hor}^{finite}$ (line 12) will contain the horizontal candidate VPs that do not lie at "infinity".

The computation of the list of VPs orthogonal to the vertical VP, $\mathbf{V}_{hor}^{ortho}$, is described in lines 4 to 9 of Algorithm 14. The VPs are converted to the Gaussian sphere, based on known or estimated values for the focal length in pixel units, and the principal point. Using a threshold γ_R on the angle between vectors in the Gaussian sphere, the VPs that are far from being orthogonal to the vertical

Algorithm 10: `orthogonal_triplets`

input : A list of unit norm vectors $\mathbf{V}_u$ (with associated NFA values). A threshold γ_S.
output: A list $\mathbf{T}$ of orthogonal triplets of vectors.

$\mathbf{T} \leftarrow \emptyset$
for $\mathbf{v}_u^{(i)} \in \mathbf{V}_u$ **do**
  for $\mathbf{v}_u^{(j)} \in \mathbf{V}_u \setminus \mathbf{v}_u^{(i)}$ **do**
    for $\mathbf{v}_u^{(k)} \in \mathbf{V}_u \setminus \mathbf{v}_u^{(i)}, \mathbf{v}_u^{(j)}$ **do**
      if $\max(|\mathbf{v}_u^{(i)} \cdot \mathbf{v}_u^{(j)}|, |\mathbf{v}_u^{(i)} \cdot \mathbf{v}_u^{(k)}|, |\mathbf{v}_u^{(j)} \cdot \mathbf{v}_u^{(k)}|) < \gamma_S$ **then**
        $\mathbf{t}^{(i,j,k)} \leftarrow \left(\mathbf{v}_u^{(i)}, \mathbf{v}_u^{(j)}, \mathbf{v}_u^{(k)}\right)$
        $\text{score}(\mathbf{t}^{(i,j,k)}) = \text{NFA}(\mathbf{v}_u^{(i)}) + \text{NFA}(\mathbf{v}_u^{(j)}) + \text{NFA}(\mathbf{v}_u^{(k)})$
        Append $\mathbf{t}^{(i,j,k)}$ to $\mathbf{T}$
      end
    end
  end
end

Algorithm 11: `orthogonal_pairs`

input : A list of unit norm vectors $\mathbf{V}_u$ (with associated NFA values). A threshold γ_S.
output: A list $\mathbf{P}$ of orthogonal pairs of vectors.

$\mathbf{T} \leftarrow \emptyset$
for $\mathbf{v}_u^{(i)} \in \mathbf{V}_u$ **do**
  for $\mathbf{v}_u^{(j)} \in \mathbf{V}_u \setminus \mathbf{v}_u^{(i)}$ **do**
    if $|\mathbf{v}_u^{(i)} \cdot \mathbf{v}_u^{(j)}| < \gamma_S$ **then**
      $\mathbf{p}^{(i,j)} \leftarrow \left(\mathbf{v}_u^{(i)}, \mathbf{v}_u^{(j)}\right)$
      $\text{score}(\mathbf{p}^{(i,j)}) = \text{NFA}(\mathbf{v}_u^{(i)}) + \text{NFA}(\mathbf{v}_u^{(j)})$
      Append $\mathbf{p}^{(i,j)}$ to $\mathbf{P}$
    end
  end
end

Algorithm 12: `image_to_gaussian_sphere` : Convert VP candidates in the image to unit vectors in the Gaussian sphere

input : A VP in image coordinates $(x_\mathbf{v}, y_\mathbf{v})$. The focal ratio f in pixel units and the principal point $\mathbf{p}$.
output: The corresponding unit-norm vector $\mathbf{v}_u$ in the Gaussian sphere.

$\mathbf{v}_u \leftarrow \left(x_\mathbf{v} - x_\mathbf{p}, \quad y_\mathbf{v} - y_\mathbf{p}, \quad f\right)^T$
$\mathbf{v}_u \leftarrow \frac{\mathbf{v}_u}{||\mathbf{v}_u||}$

VP are discarded (lines 6 to 9).

The computation of the list of non-infinite VPs, $\mathbf{V}_{hor}^{finite}$, is described in lines 12 to 19. The aim is to identify VPs caused by sets of parallel lines, whose vertical position is undetermined. This is done with a threshold λ on the distance between the VP and the image center $\mathbf{p}$ (lines 13 to 15).

Algorithm 13: `gaussian_sphere_to_image` : Convert VP candidates as unit vectors in the Gaussian sphere to points in the image.

input : A unit-length vector $\mathbf{v}_u$ in the Gaussian sphere. The focal ratio f in pixel units and the principal point $\mathbf{p}$.

output: The corresponding VP in image coordinates $\mathbf{v}_{img}$

1 $\mathbf{v}_{img} \leftarrow \left(f \cdot \frac{\mathbf{v}_u(1)}{\mathbf{v}_u(3)} + x_{\mathbf{p}}, f \cdot \frac{\mathbf{v}_u(2)}{\mathbf{v}_u(3)} + y_{\mathbf{p}}\right)^T$

This threshold is proportional to the image size (see Table 1). Note that it could happen that no VP is finite under this criterion. In that case the result is given by taking the VP closest to the image center (lines 18 to 19).

The list of final horizontal VPs is the intersection of the two aforementioned lists, namely those VPs that are both orthogonal to the vertical VP and finite. In the extreme case that the lists are disjoint, only one horizontal VP is considered, the one with the lowest NFA. This is described in lines 21 to 24.

Once the final subset of horizontal VP candidates is determined, the horizon line is obtained by a weighted average: each candidate horizontal VP casts a vote for the horizon line. This procedure is detailed in Algorithm 16. For each VP, its proposed horizon line is the line passing by itself, perpendicular to the line formed by the image center and the vertical VP (line 5). Since all lines are parallel, this problem is actually one-dimensional: what is being determined is the height of the horizon line, h_{hor} (line 2). The weight of each vote is given by the NFA associated to the VP, more precisely as $-log_{10}(\text{NFA})$, normalized over the sum of this value for all remaining VPs (line 4).

Once a first estimation of the horizon line is obtained by the weighted average, a threshold κ operates on the distance between individual candidate horizon lines and the estimated horizon line, removing outliers (line 8). Note that the distance between these lines is naturally defined because they are all parallel. After the outliers are removed, the weighted average is recomputed (line 9).

3.8 Analysis of the Method's Parameters

There are eleven parameters in the VP detection method, which we divide into five groups. Table 1 provides their description and values. Group I is the line segment distance threshold for line segment denoising and grouping. Group II contains the three parameters for refining the raw VP detections. The two parameters in group III are parameters of the camera. Groups IV and V contain the parameters for imposing the 3D world assumptions under the two hypotheses: Manhattan and non-Manhattan world respectively. The alignment detector is always used as described in [13] with $\varepsilon = 10$, without further modification or adjustment.

This is an exhaustive list of the parameters that can affect the performance of the algorithm. Some of the parameters are more sensitive to the final result than others, as will be described below. When evaluating the algorithm on standard benchmark datasets (Section 5.2), we found that a few of them needed to be manually adjusted according to the type of images to be processed (see Table 2). Many of them did not need to be adjusted for each dataset. A procedure for automatically determining parameter values according to the type of input image remains to be developed.

The most sensitive parameter of the method is the focal length f. This value can sometimes be obtained if the camera model is known, or from the information included in the EXIF[5] header for some type of image formats. The EXIF header can provide the focal length in mm, the CCD size in mm and the CCD resolution in pixels, which can be used to compute the focal length in pixel units. If the camera is available but its characteristics unknown, f can be estimated by camera

[5] http://en.wikipedia.org/wiki/Exchangeable_image_file_format

Algorithm 14: `non-manhattan_constraints`: Final VP and horizon line estimation without the Manhattan-world hypothesis.

input : A list of candidate VPs in image domain $\mathbf{V}$. f, the focal length in pixel units. $\mathbf{p}$, the principal point. Thresholds γ_R, ω, λ and κ

output: A list $\mathbf{V}$ of candidate vanishing points. The horizon line $\mathbf{l}_{hor}$.

$\mathbf{V}_{ver} \leftarrow$ `vertical_vp_candidates`$(\mathbf{v}, \omega, \mathbf{p})$ *// Algorithm 15*
$\mathbf{v}^{(v)} \leftarrow \arg\min_{\mathbf{v} \in \mathbf{V}_{ver}} \big(\text{NFA}(\mathbf{v})\big)$
$\mathbf{V}_{hor} \leftarrow \mathbf{V} \setminus \mathbf{V}_{ver}$
$\mathbf{v}_u^{(v)} \leftarrow$ `image_to_gaussian_sphere`$(\mathbf{v}^{(v)}, f, \mathbf{p})$ *// Algorithm 12*
$\mathbf{V}_{hor}^{ortho} \leftarrow \emptyset$
for $\mathbf{v}^{(h)} \in \mathbf{V}_{hor}$ **do**
  $\mathbf{v}_u^{(h)} \leftarrow$ `image_to_gaussian_sphere`$(\mathbf{v}^{(h)}, f, \mathbf{p})$
  if $|\mathbf{v}_u^{(v)} \cdot \mathbf{v}_u^{(h)}| < \gamma_R$ **then**
    Append $\mathbf{v}^{(h)}$ to $\mathbf{V}_{hor}^{ortho}$
  end
end
$\mathbf{V}_{hor}^{finite} \leftarrow \emptyset$
for $\mathbf{v}^{(h)} \in \mathbf{V}_{hor}$ **do**
  if $||\mathbf{v}^{(h)} - \mathbf{p}|| < \lambda$ **then**
    Append $\mathbf{v}^{(h)}$ to $\mathbf{V}_{hor}^{finite}$
  end
end
if $\mathbf{V}_{hor}^{finite} = \emptyset$ **then**
  $\mathbf{V}_{hor}^{finite} \leftarrow \arg\min_{\mathbf{v}^{(h)} \in \mathbf{V}_{hor}} \big(||\mathbf{v}^{(h)} - \mathbf{p}||\big)$
end
if $\mathbf{V}_{hor}^{ortho} \bigcap \mathbf{V}_{hor}^{finite} \neq \emptyset$ **then**
  $\mathbf{V} \leftarrow \mathbf{V}_{hor}^{ortho} \bigcap \mathbf{V}_{hor}^{finite}$
else
  $\mathbf{V} \leftarrow \arg\min_{\mathbf{v} \in \mathbf{V}_{hor}} \big(\text{NFA}(\mathbf{v})\big)$
end
Append $\mathbf{v}^{(v)}$ to $\mathbf{V}$
$\mathbf{l}_{hor} \leftarrow$ `horizon_line_voting`$(\mathbf{V}, \kappa)$ *// Algorithm 16*

calibration [11]. Otherwise, a general rule of thumb is to choose $f \in [0.3W, 3W]$, supposing W is the largest size of the image.

Also needed to compute the VP in the Gaussian sphere, the principal point of the camera, $\mathbf{p}$, can be safely assumed to be at the image center $[W/2, H/2]$, if the image was not cropped from a bigger image.

The threshold τ for the length of line segments can have a big impact in the final result, if there happens to be a significant group of structures in the image whose length is close to the threshold. This threshold should vary with the image size. The chosen value of $\frac{\sqrt{W+H}}{1.71}$ is not strictly scale invariant, but, as most of the parameters in Table 1, it has been found empirically using the benchmarking datasets of Section 5.2.

Algorithm 15: Obtain vertical VP candidates.

input : A list of candidate VPs in image domain $\mathbf{V}$. An angular threshold ω. The principal point $\mathbf{p}$.

output: A list $\mathbf{V}_{ver}$ of candidate vertical vanishing points.

$\mathbf{V}_{ver} \leftarrow \emptyset$
$\mathbf{l}_{ver} \leftarrow$ vertical line passing trough $\mathbf{p}$
for $\mathbf{v} \in \mathbf{V}$ **do**
  $\mathbf{v}_{centered} \leftarrow \mathbf{v} - \mathbf{p}$
  $\mathbf{l}_{\mathbf{v}} \leftarrow \overline{\mathbf{p}\mathbf{v}_{centered}}$
  if $\mathit{angle}(\mathbf{l}_{\mathbf{v}}, \mathbf{l}_{ver}) < \omega$ *AND* $|y_{\mathbf{v}_{centered}}| > H$ **then**
    Append $\mathbf{v}$ to $\mathbf{V}_{ver}$
  end
end

Algorithm 16: `horizon_line_voting`: Obtain horizon line by votes.

input : A list of horizontal VPs $\mathbf{V}_{hor}$, a vertical VP $\mathbf{v}_{ver}$ and their associated NFAs. The image center $\mathbf{p}$ and threshold κ.

output: A line $\mathbf{l}_{hor}$ representing the horizon

1 $\mathbf{l}_{ver} \leftarrow \overline{\mathbf{p}\mathbf{v}_{ver}}$
2 $h_{hor} \leftarrow 0$
3 **for** $\mathbf{v}^{(i)} \in \mathbf{V}_{hor}$ **do**
4 $w^{(i)} = \left(\frac{-log_{10}(\mathrm{NFA}_i)}{\sum_j -log_{10}(\mathrm{NFA}_j)}\right)^2$
5 $\mathbf{l}_{hor}^{(i)} =$ line through $\mathbf{v}^{(i)}$ perpendicular to $\mathbf{l}_{ver}$
6 $h_{hor} \leftarrow h_{hor} + w^{(i)} \cdot \mathtt{height}(\mathbf{l}_{hor}^{(i)})$
7 **end**
8 $\mathbf{V}_{hor} = \{\mathbf{v}^{(i)} \in \mathbf{V}_{hor} \mid ||\mathbf{l}_{hor}^{(i)} - \mathbf{l}_{hor}|| < \kappa\}$ *// Keep only VPs close to average.*
9 Repeat lines 2 to 6. *// Re-estimate average.*
10 $\mathbf{l}_{hor} \leftarrow$ line perpendicular to $\mathbf{l}_{ver}$ with height h_{hor}

The refinement angle threshold θ is set to a very small angle, so that the segments chosen for refining a candidate VP do not largely differ from the points forming the alignment found by the alignment detector. This threshold can have an important impact in the final location of the refined VP. The threshold ζ is used to determine when the refinement has gone too far, although its sensitivity is limited, since ill conditioned cases are a minority.

The angular threshold γ_S is used to determine whether a pair of unit vectors are close to orthogonal, in the case where orthogonality is searched for. On the other hand, γ_R is a more relaxed threshold, used to determine that two vectors are far from being orthogonal. The sensitivity of these two parameters is in some way related to f. When f is known or correctly estimated, the mapping of vectors into the Gaussian sphere should be correct, so orthogonality and non-orthogonality between vectors should be clearly determined. The values presented in Table 1 have been found empirically using the benchmarking datasets of Section 5.2.

The angular parameter ω is used to determine if a candidate VP could possibly be the vertical VP of the image. The idea is that a vertical VP should be in a close angle with respect to the vertical direction. Of course, in order for this trivial heuristic to work properly the image has to be approximately horizontal.

Group	Name	Description	Algorithm	Value
I	τ	Line segments length threshold	2	$\frac{\sqrt{W+H}}{1.71}$
II	θ	VP to line segment angle threshold	6, 7	$2°$
	ζ	Variation threshold to detect ill-conditioning	6, 7	0.3
	δ	Distance threshold for redundant VP detections	6, 8	0.0001
III	f	Focal length in pixel units	9, 12, 13, 14	$\max(W, H)$
	$\mathbf{p}$	Principal point or image center	9, 14, 16	$(H/2, W/2)$
IV	γ_S	Angle threshold for orthogonality	9, 10, 11	$87.5°$
V	γ_R	Angle threshold for non-orthogonality	14	$77.5°$
	ω	Angle threshold for vertical VPs	14, 15	$50°$
	λ	Distance threshold for infinite VPs	14	$3.6 \cdot W$
	κ	Distance threshold for outlier VPs	14, 16	$0.14 \cdot H$

Table 1: Parameters of the proposed method, with suggested values for general images. The parameters are divided into five groups (see text). W and H are the image width and height, respectively.

Two threshold parameters remain. The threshold λ is a simple distance threshold used to determine if a point lies at "infinity". Naturally, this value needs to be proportional to the image size. This parameter can be sensitive in the case where all the cues for horizontal VPs in the image come from parallel structures (the intersection of the lines lies at infinity). For example, when a building façade is photographed very closely, and no perspective is produced. The threshold κ is used to determine outliers in the estimation of the horizon line, and operates on the distance between each VP and the estimated horizon line.

4 Acceleration of the Point Alignments Detection

In terms of computational complexity, the bottleneck of the proposed method is the alignment detection stage. The alignment detector of [13] performs an exhaustive search over all pair of points and for each pair it tests different template configurations to compute possible alignment candidates. The counting of points inside each candidate rectangle makes the exhaustive version of the algorithm to have $\mathcal{O}(Cn^3)$ complexity, where n corresponds to the number of points and C to all the different possible shapes for the alignment templates.

One way to accelerate the method is to use a fast algorithm to propose alignment candidates, instead of evaluating every pair of points in the dataset. To this end, we propose to use the unsupervised Gaussian Mixtures algorithm of Figueiredo et al. [7], exploiting the fact that an elongated cluster can be approximated by a thin Gaussian. This algorithm detects Gaussians of any shape and can automatically discover the number of Gaussian clusters in the dataset. The clusters found by this algorithm can be considered as potential alignment candidates and then be evaluated by the *a contrario* criterion of [13].

The computational complexity of Figueiredo's algorithm is similar to the EM algorithm in 2D, which is approximately $\mathcal{O}(nk)$, where k is the number of iterations. Since the algorithm converges in at most a few hundreds of iterations, this is usually much faster than $\mathcal{O}(n^3)$. To cope with the non-deterministic nature of the algorithm (given by its random initialization), the algorithm can be run multiple times to obtain different lists of candidates. In the proposed method it is run three times, and a single list of candidates is produced by merging the output of all runs.

Algorithm 17 presents the pseudo code for the accelerated version. For each Gaussian detected by Figueiredo's algorithm, we consider a line segment along the longest axis of the Gaussian, centered

at the mean and with length twice the square-root of the largest eigenvalue of the co-variance matrix of the Gaussian (lines 6 and 7). The set of endpoints of all such segments are then passed to the alignment detector algorithm, which will compute candidate alignments using only those endpoints (line 11). The `GMM` procedure in line 3 uses the original code of [7], downloaded from the author's web page[6].

The `detect_alignments_with_candidate_endpoints` procedure in line 11 of Algorithm 17 implements the alignment detector of [13, 14], except that instead of going through each pair of points to form candidate alignments, it only considers pairs of candidate endpoints obtained with the Gaussian mixtures algorithm. Section 5.3 presents an experimental analysis of the effect of using this acceleration.

Algorithm 17: Use Figueiredo et al. Gaussian Mixtures algorithm to generate alignment candidates.

input : A list of points $\mathbf{X}$. The number of times the `GMM` algorithm is run, k.
output: A list $\mathbf{A}$ of alignment detections.

1 $\mathbf{X}_c \leftarrow \emptyset$
2 **for** $i = 1 : k$ **do**
3 $\quad (\mu_j, \Sigma_j)_{j=1\ldots n} \leftarrow$ `GMM(`$\mathbf{X}$`)`
4 $\quad$ **for** $j = 1 : n$ **do**
5 $\quad\quad (\mathbf{u}, \sigma, \mathbf{v})_{1,2} \leftarrow$ `SVD(`Σ_j`)`
6 $\quad\quad \mathbf{x}_1 \leftarrow \mu_j + \mathbf{u}_1 \cdot 2 \cdot \sqrt{\sigma_1}$
7 $\quad\quad \mathbf{x}_2 \leftarrow \mu_j - \mathbf{u}_1 \cdot 2 \cdot \sqrt{\sigma_1}$
8 $\quad\quad$ Append $\mathbf{x}_1$ and $\mathbf{x}_2$ to $\mathbf{X}_c$
9 $\quad$ **end**
10 **end**
11 $\mathbf{A} \leftarrow$ `detect_alignments_with_candidate_endpoints(`$\mathbf{X}, \mathbf{X}_c$`)`

5 Experiments

In this experimental section we present example results of the method, as well as a systematic performance evaluation on two standard public datasets for vanishing points detection.

5.1 Example Results

For the experiments presented in this section, we used the C code of [13] for alignment detection and the C code of [9] for line segment detection. The rest of the algorithm was implemented in MATLAB. Processing a 640x480 image takes an average of 22 seconds in a 2.4 Ghz Intel Core i5 laptop with 8 GB of RAM. The bottleneck of the method is the computation of point alignments in *straight* and *twisted* dual domains, which accounts for more than 90% of the processing time. This can be accelerated as explained in Section 4.

Figures 9, 10 and 11 show example results of the method. The resulting image shows, on the top-right corner, the grouping of line segments in the image corresponding to each VP and the estimated horizon line. On the bottom rows, the PClines *straight* and *twisted* spaces are shown, with the point alignments that have been detected, and the representation of the final VPs. Figures 9 and 10, show urban scenes with a regular layout that adjusts well to the Manhattan-world hypothesis. In both

[6] http://www.lx.it.pt/~mtf/

cases, this hypothesis has been assumed. In Figure 11, the buildings are not arranged in a regular grid, and multiple horizontal vanishing points exist. By imposing the "non-Manhattan" world hypothesis, the algorithm finds multiple horizontal vanishing points. For further experimentation, the reader is invited to try the online demo in the web page of this article[7].

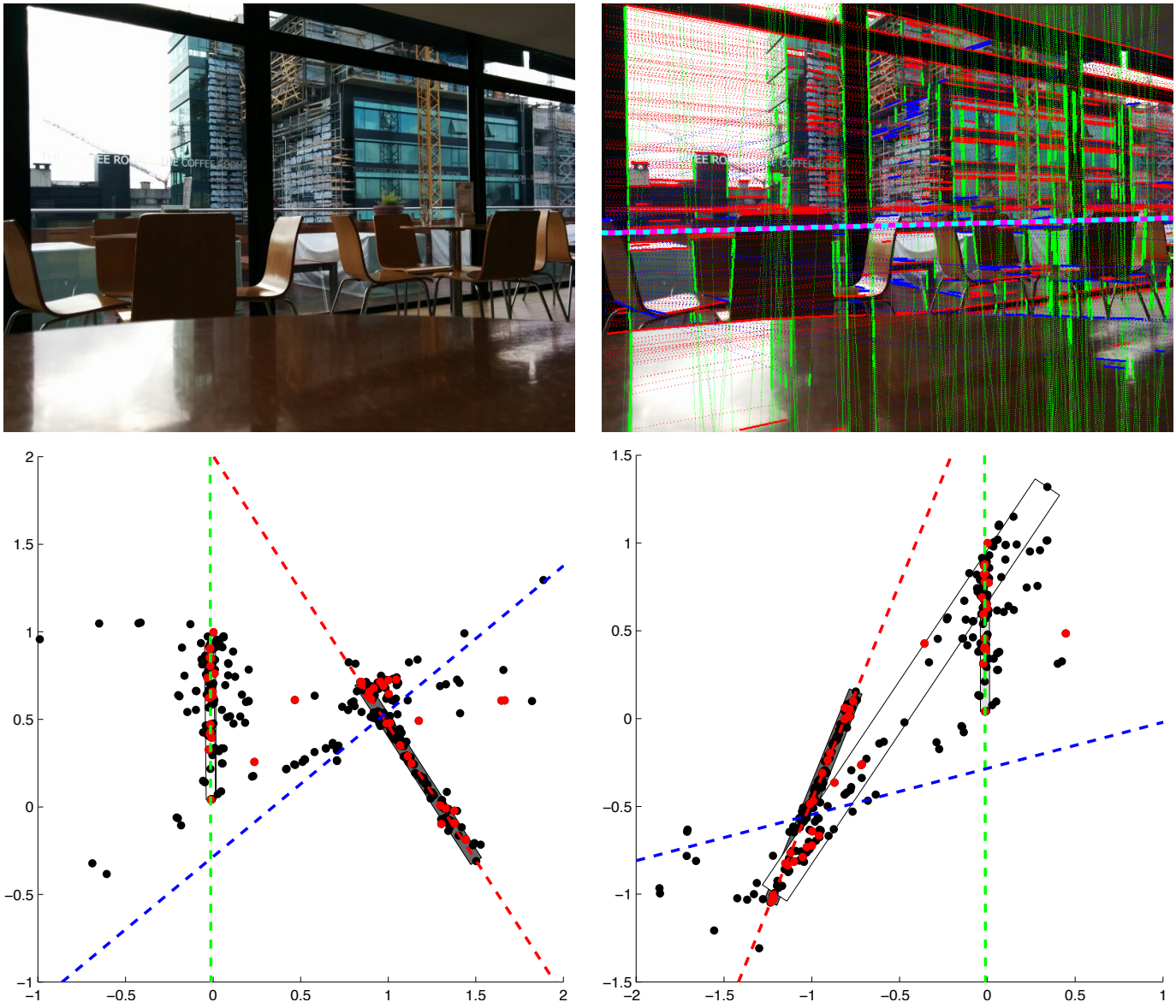


Figure 9: Example result of the method. Best viewed in electronic format. **Top left:** Original image. **Top right:** Algorithm result, line segments corresponding to each final VP detection and horizon line. **Bottom:** PClines *straight* (**left**) and *twisted* (**right**) dual spaces. Alignments (shadowed rectangles) and final VPs (colored dashed lines). In this case, the leftmost VP, represented by the blue dashed line and the blue segments in the result, was not found by the alignment detector in any of the dual spaces. However, it has been determined by applying the Manhattan-world hypothesis, which implies that the third VP must be orthogonal to the two VPs detected. The same hypothesis allows to reject the VP detection seen as a large rectangle in the center of the *twisted* space. For this example f was set to 1.08.

[7] https://doi.org/10.5201/ipol.2017.148

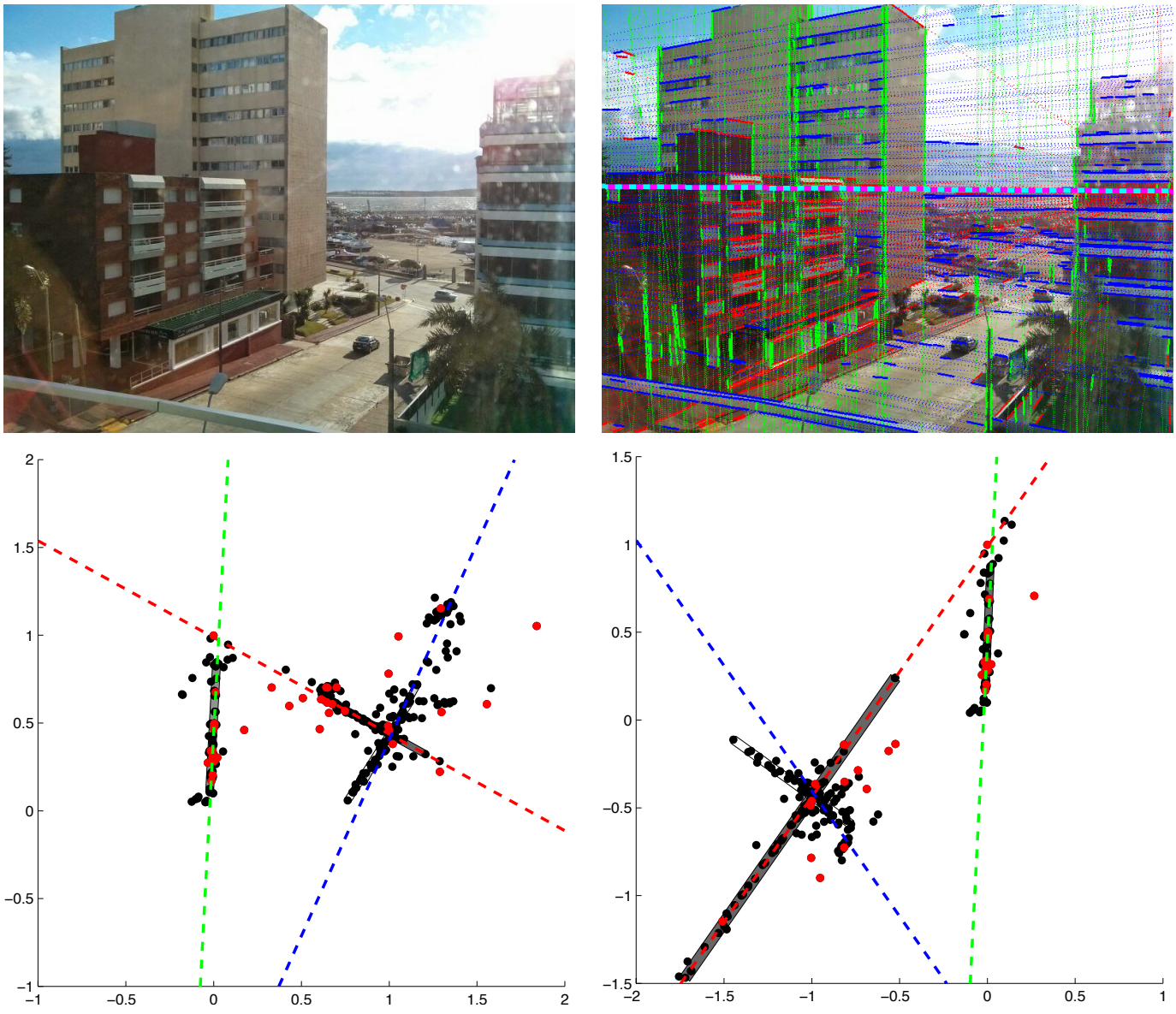


Figure 10: Example result of the method. Best viewed in electronic format. **Top left:** Original image. **Top right:** Algorithm result, line segments corresponding to each final VP detection and horizon line. **Bottom:** PClines *straight* (**left**) and *twisted* (**right**) dual spaces. Alignments (shadowed rectangles) and final VPs (colored dashed lines). In this case, the alignment detections corresponding to all VPs have been detected by the alignment detector. The difference in direction between the detections rectangles and the dashed lines shows that the refinement step has fine-tuned the final VP locations. For this example f was set to 1.08.

5.2 Quantitative Evaluation

A commonly used measure to assess the performance of vanishing points detection is the horizon line estimation error. This measure is defined as the maximum distance in the image domain between the estimated horizon line and the ground truth, divided by the image height. In this section, we use this measure to establish the performance of the method in two standard and widely used datasets.

The first dataset is the York Urban Dataset (YUD) [5]. It includes 102 images of outdoor and indoor scenes, the camera parameters, and the ground truth VPs. All the images satisfy the Manhattan world assumption, so the ground truth consists of 3 orthogonal VPs. Example results for this dataset are shown in Figure 12.

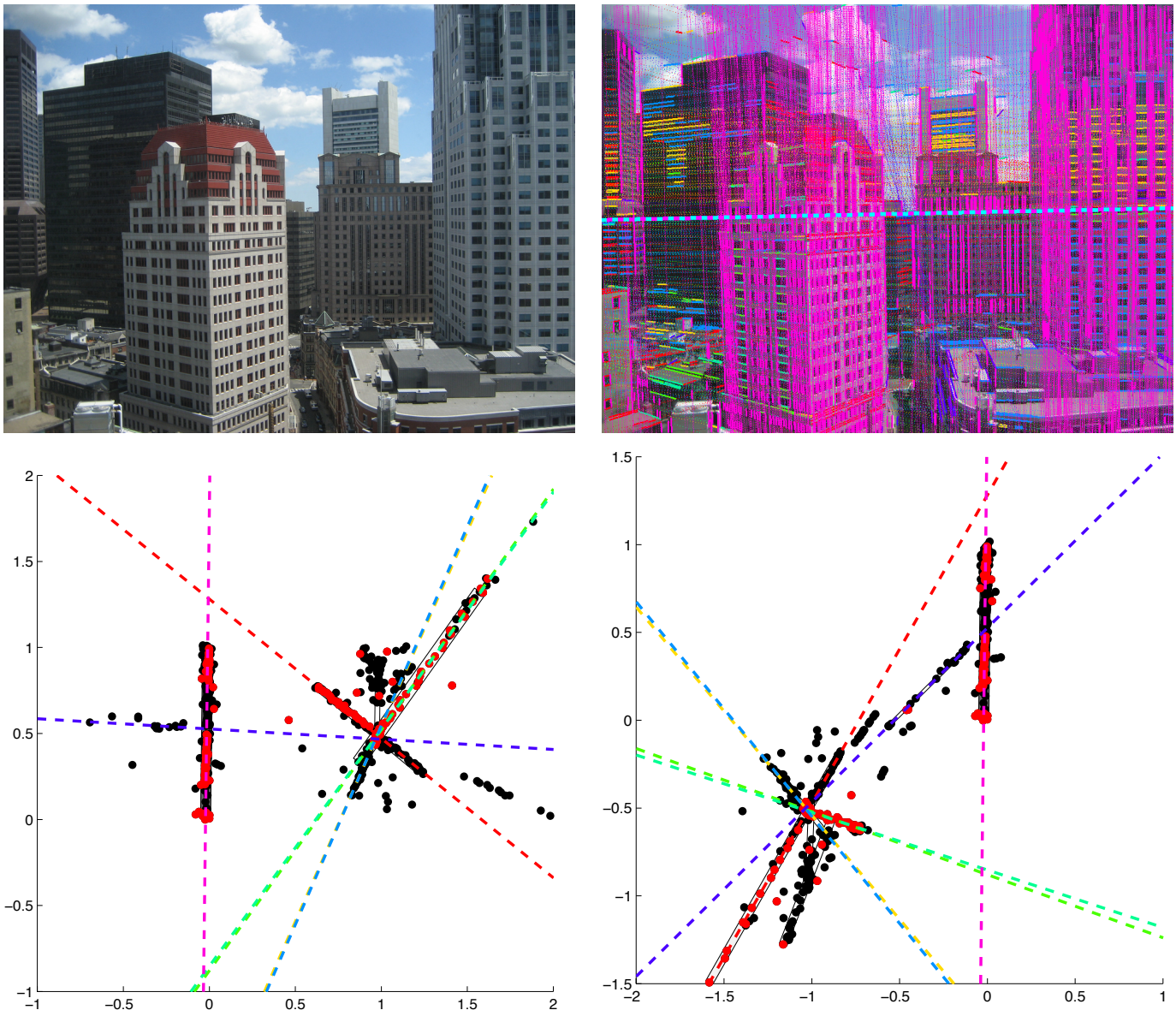


Figure 11: Example result of the method when the Manhattan-world hypothesis is not applicable. Best viewed in electronic format. **Top left:** Original image. **Top right:** Algorithm result, line segments corresponding to each final VP detection and horizon line. **Bottom:** PClines *straight* (**left**) and *twisted* (**right**) dual space. Alignments (shadowed rectangles) and final VPs (colored dashed lines). In this example multiple horizontal VPs have been found. On the dual space, one can see that the lines corresponding to the multiple horizontal VPs intersect close to a point which represents the horizon line in the image. There is one misdetection in the *straight* space and two misdetections in the *twisted* space that do not correspond to a true VP, but have been discarded by the algorithm in the refinement step. For this example f was set to 1.

The second dataset is the Eurasian Cities Dataset (ECD) [4], which presents a much more challenging dataset of 103 urban scenes that do not satisfy the Manhattan world assumption in general. They depict different architectural and urban styles and are taken by different cameras. Under these conditions, the method described in Section 2.5.2 is used. Example results for this dataset are shown in Figure 13.

Following the protocol of [4] and other recent works [17, 16, 18], the first 25 images of each dataset are used to adjust the parameters of the method and the evaluation is performed on the remaining images. The performance score is measured as the area under the curve (AUC) of the cumulative histogram of the horizon line detection error. Table 2 presents the values for the parameters, opti-

Name	Algorithm	Value for YUD	value for ECD
τ	2	$\frac{\sqrt{W+H}}{1.71}$	$\frac{\sqrt{W+H}}{1.71}$
θ	6, 7	$2°$	$0.6°$
ζ	6, 7	0.1	0.3
δ	6, 8	0.0001	0.0001
f	9, 12, 13, 14	$1.08W$	$1.08W$
$\mathbf{p}$	9, 14, 16	$(307.55, 251.45)$	$(W/2, H/2)$
γ_S	9, 10, 11	$87.5°$	N/A
γ_R	14	N/A	$77.5°$
ω	14, 15	N/A	$50°$
λ	14	N/A	$3.6W$
κ	14, 16	N/A	$0.14H$

Table 2: Parameters of the proposed method optimized for YUD and ECD datasets. H and W are the height and width of the image. A brief description of the parameters can be found in Table 1. Note that the focal length ratio f for YUD is slightly different from the one provided in the dataset.

YUD *train*	YUD *test*	YUD *train* (non-Man.)	YUD *test* (non-Man.)	ECD *train*	ECD *test*
96.13%	95.48%	90.44%	93.70%	88.95%	90.11%

Table 3: Quantitative results of the algorithm as the area under the curve (AUC) of the horizon line estimation error. *Train* indicates the set of the first 25 images used to adjust the parameters. *Test* indicates the remaining images in the dataset. The third and fourth columns present the result of the algorithm when not applying the Manhattan-world on YUD (applying the exact same method as for ECD). At the moment of publication these results were state-of-the-art [12].

mized by grid search on the first 25 images of each dataset. The performance scores obtained in both training and testing subsets for both datasets are reported in Table 3. It also includes the result of applying the non-Manhattan version of the algorithm on YUD. All the results in Table 3 have been obtained with the non-accelerated version of the algorithm. In the companion code to this article, two auxiliary scripts are provided for evaluating the performance on YUD and ECD.

5.3 Accelerated Version

Dataset	$k = 3$	$k = 6$	$k = 9$	$k = 12$	no acceleration
ECD	$83.06 \pm 0.90\%$	$86.01 \pm 0.92\%$	$85.53 \pm 1.66\%$	$86.40 \pm 0.31\%$	90.11%
YUD	$93.02 \pm 0.36\%$	$94.21 \pm 0.36\%$	$94.30 \pm 0.50\%$	$94.73 \pm 0.36\%$	95.48%

Table 4: Evaluation of the acceleration of the point alignment detection using the algorithm of [7] to generate candidate alignments. The parameter k is the number of times the algorithm is run to create a list of possible clusters of aligned points. The performance shown is the AUC score as described in Section 5.2 averaged on 10 runs.

In this subsection we present an experimental analysis of the effects of using the accelerated version of the point alignment detector instead of the one based on exhaustive search. As pointed out in Section 4, the Gaussian mixtures algorithm of [7] (which we will refer to as `GMM`) has a random initialization, which makes its result stochastic. Thus, the result of the accelerated vanishing point detector, which depends on the result of the `GMM` procedure, will also be stochastic. This behavior is illustrated in Figure 14. Each run of `GMM` produces a different result. However, the most important

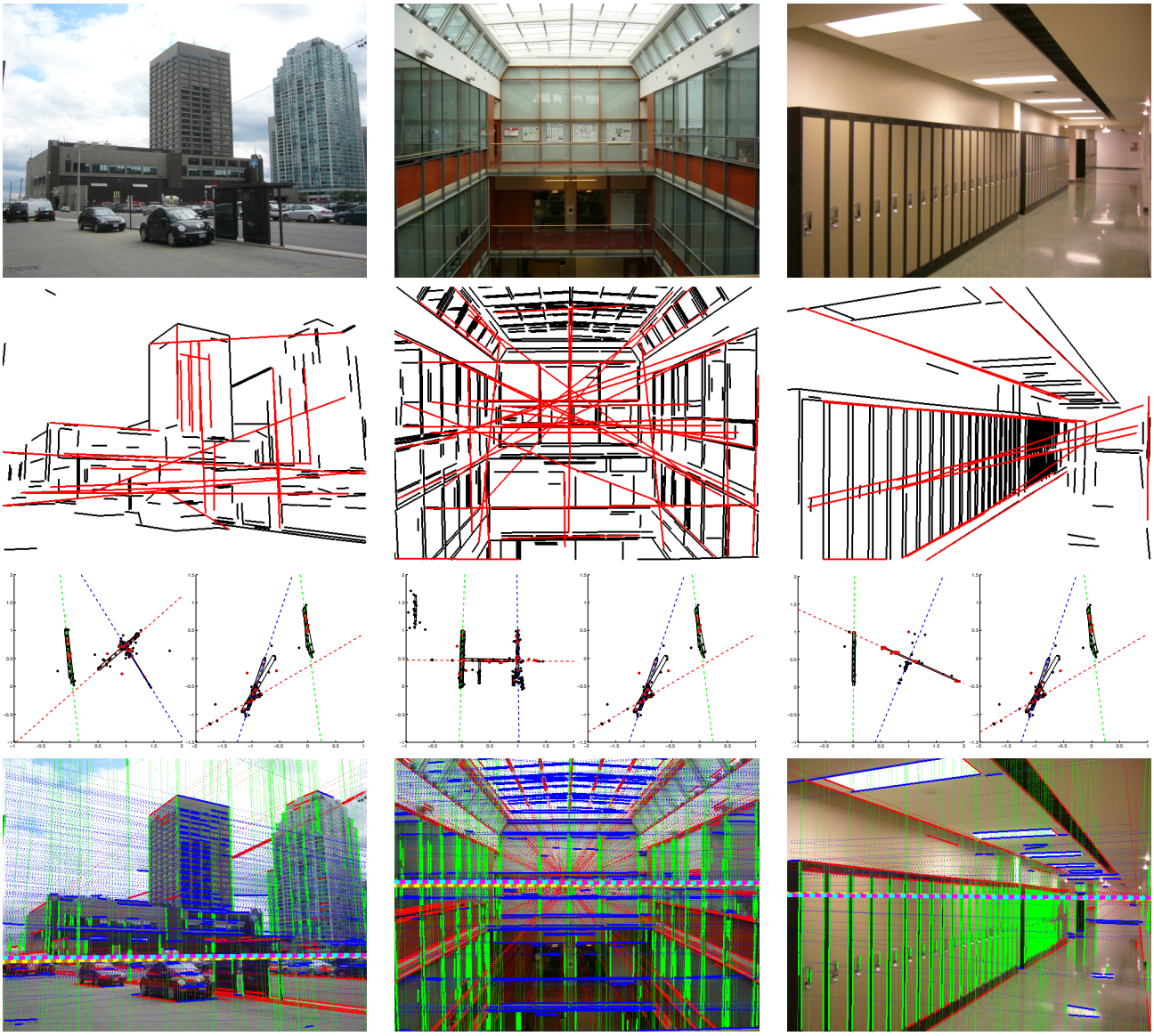


Figure 12: Example results of the method on selected images from YUD. **Top Row:** Original image. **2nd Row:** Line segments (black) and alignments of line segments endpoints (red). **3rd Row:** PClines *straight* and *twisted* dual spaces and point alignment detections (parallel black lines). The ground truth is represented with colored dashed lines. **Bottom Row:** Line segments corresponding to each final VP detection and horizon line (yellow-orange: ground truth, magenta-cyan: ours). In the last row, line segments from endpoint alignments have been removed for clarity. Note that the refinement and redundancy steps are not represented in this figure.

Dataset	$k = 3$	$k = 6$	$k = 9$	$k = 12$	no acceleration
ECD	8.9 ± 2.8 secs	17.13 ± 6.1 secs	26.3 ± 10.4 secs	36.3 ± 15.4 secs	40.1 ± 71.7 secs
YUD	4.0 ± 1.3 secs	6.9 ± 2.9 secs	9.8 ± 4.6 secs	13.0 ± 6.6 secs	7.02 ± 9.1 secs

Table 5: Computation time of the vanishing points detection algorithm for one image, using the algorithm of [7] to generate candidate point alignments. The acceleration parameter k is the number of times the Gaussian mixtures algorithm is run to create a list of possible clusters of aligned points. The time shown is the average computation time, evaluated on the whole dataset and divided by the number of images in it, averaged on 10 runs. The CPU used is a 2.6GHz Intel Xeon E5 with 126GB of RAM.

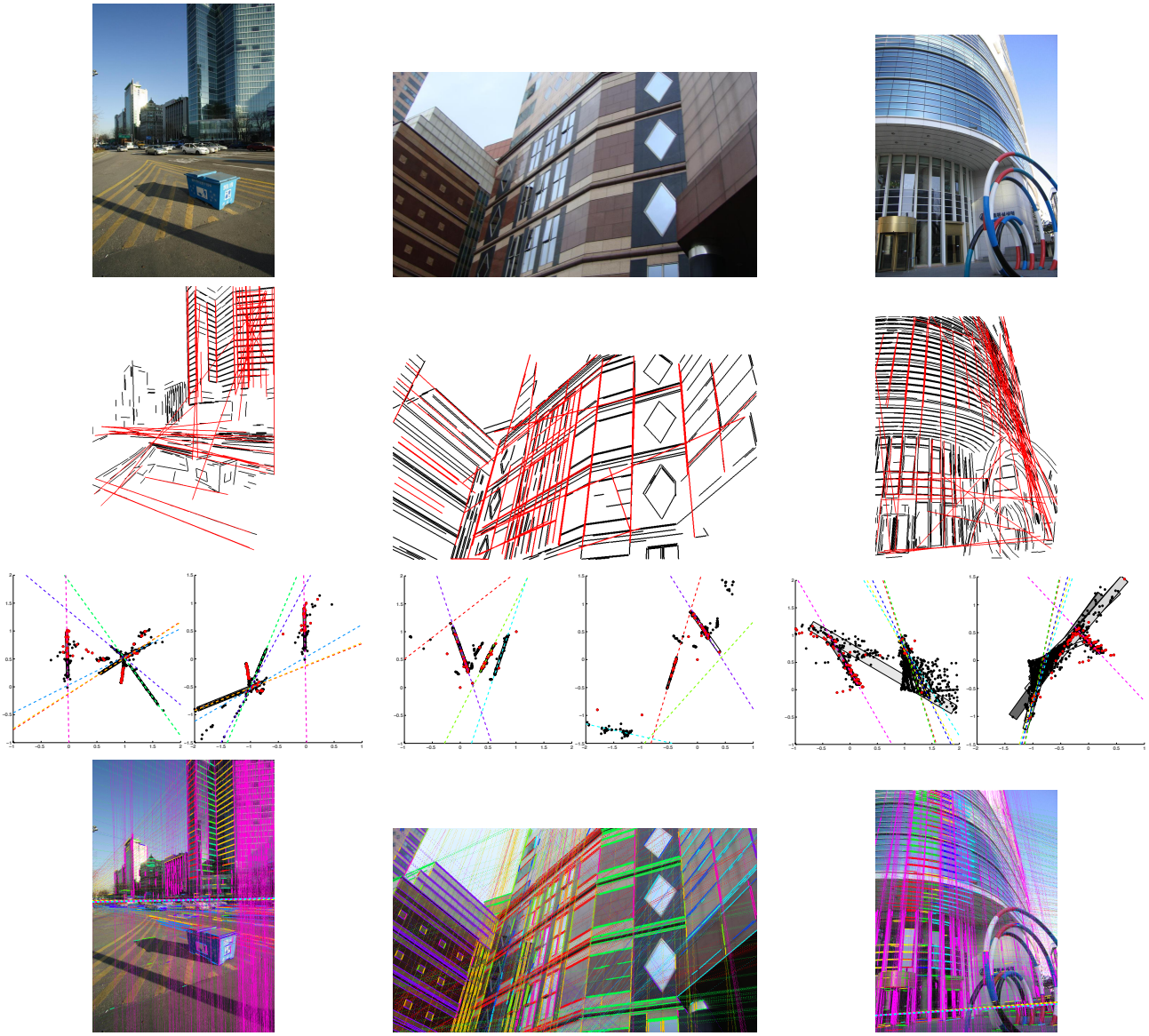


Figure 13: Example results of the method on selected images from ECD. **Top Row:** Original image. **2nd Row:** Line segments (black) and alignments of line segments endpoints (red). **3rd Row:** PClines *straight* and *twisted* dual spaces and point alignment detections (parallel black lines). The ground truth is represented with colored dashed lines. **Bottom Row:** Line segments corresponding to each final VP detection and horizon line (yellow-orange: ground truth, magenta-cyan: ours). In the last row, line segments from endpoint alignments have been removed for clarity. Note that the refinement and redundancy steps are not represented in this figure.

clusters are usually captured, albeit with minor deviations. Once the ellipses shown in the top row of Figure 14 have been obtained, the alignment detector algorithm is run considering as possible endpoints for alignments, only the extremes of these ellipses. The bottom row of Figure 14 shows the detections thus obtained. One can see that the endpoints of the alignments correspond to extremes of the ellipses. Figure 15 shows the effect of the random result of `GMM` in the final estimation of the horizon line for an example image from YUD.

To obtain a more precise result, one may run the `GMM` procedure many times, producing a bigger list of ellipses and thus more candidates for point alignments. Naturally, as the algorithm is run more times, the probability of finding the good clusters increases, as does the computation time. An analysis of the cost in both performance and computation time of running multiple instances

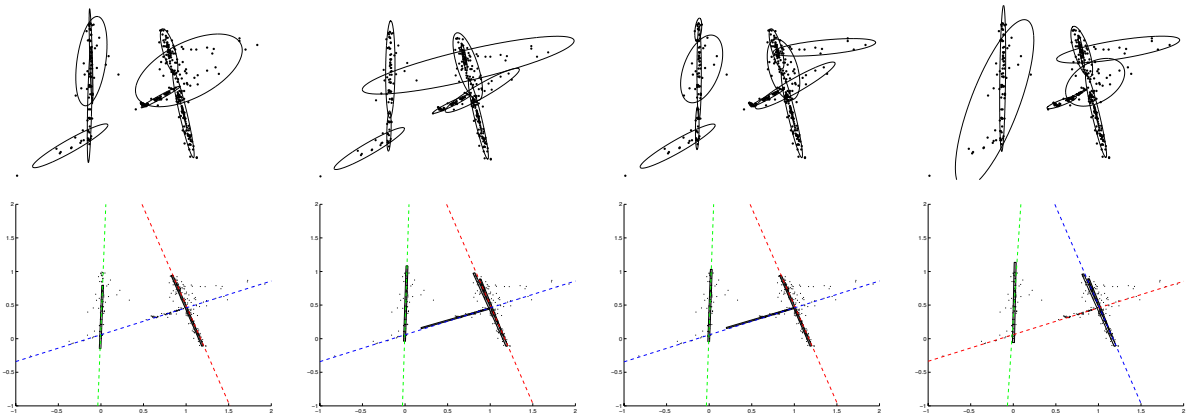


Figure 14: **Top:** Result of four different runs of the GMM routine in the PClines *straight* space for the image shown in Figure 15. Ellipses represent the obtained Gaussians. Each run of the GMM algorithm produces a different result, because of the random initialization. **Bottom:** Shadowed rectangles represent the alignments detections found using as candidate endpoints the extremes of the ellipses in the top row. Dashed lines represent the obtained vanishing points.

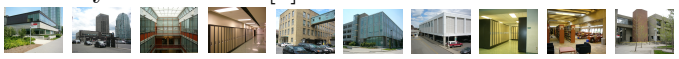
of the GMM procedure to generate alignment candidates is presented in Tables 4 and 5. In Table 4, the same measure of performance used in Section 5.2, namely the horizon line estimation error, is used. The parameter k indicates the number of times the GMM procedure is run to produce candidate alignment endpoints. Since the algorithm is non-deterministic, it is run 10 times for each value of k, and the performance results are averaged. As expected, the more instances are run, the better the performance of the algorithm at a higher computation time cost. Table 5 presents the average computation time of the accelerated algorithm for one image, for each value of k.

Image Credits

Images by the authors except:

 Buildings by Michael Coté[8]. Under Creative Commons license type "Attribution"[9].

 By M. Dubská [6].

 By P. Denis [5].

 By O. Barinova [4].

[8] https://flic.kr/p/4WVZHb

[9] https://creativecommons.org/licenses/by/2.0/legalcode

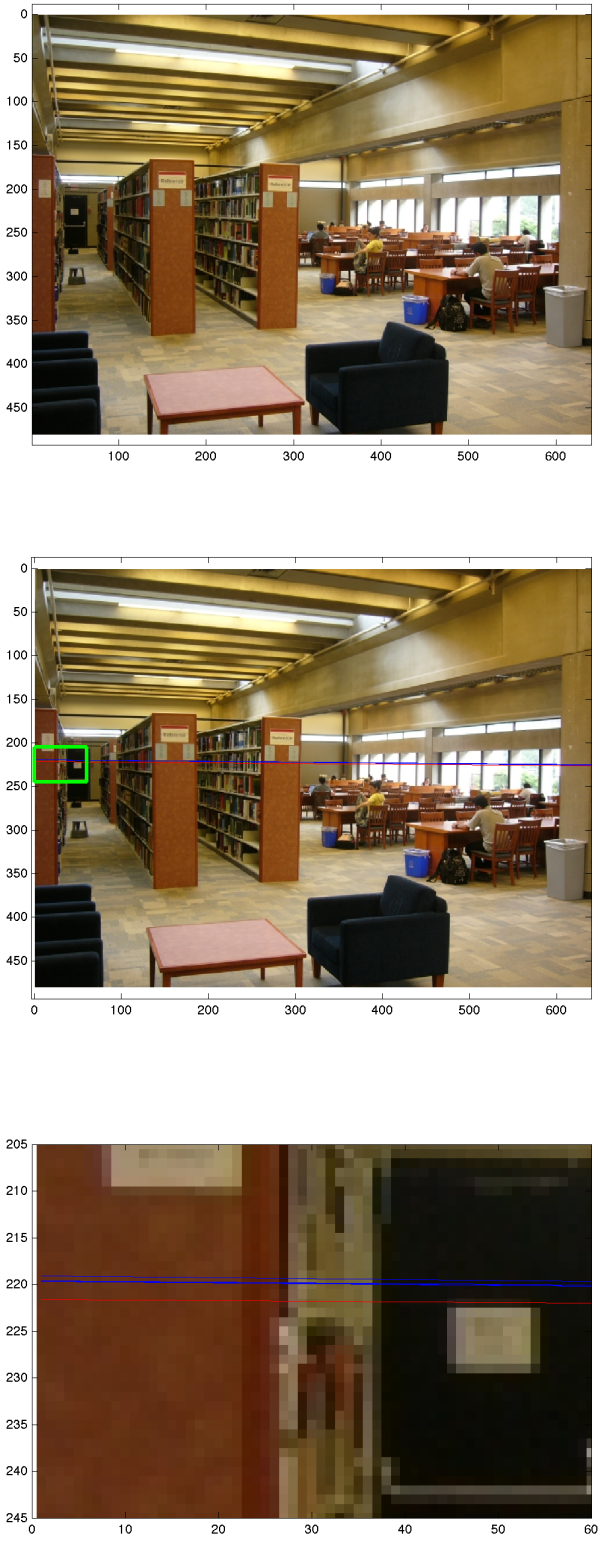


Figure 15: Example results of the accelerated version of the VP detection algorithm. Best viewed in electronic format. **Top:** Original image. **Center:** In red: Ground Truth horizon line. In blue: horizon lines obtained in each of the runs shown in Figure 14, superposed. Each run of the Gaussian Mixtures algorithm produces a different but similar result, because of the random initialization. **Bottom** Detail of the green rectangle in the middle image.

References

[1] A. Almansa, *Echantillonnage, interpolation et détection. Applications en imagerie satellitaire*, PhD thesis, ENS Cachan, 2002.

[2] A. Almansa, A. Desolneux, and S. Vamech, *Vanishing point detection without any a priori information*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 25 (2003), pp. 502–507. https://doi.org/10.1109/TPAMI.2003.1190575.

[3] M. Antunes and J.P. Barreto, *A global approach for the detection of vanishing points and mutually orthogonal vanishing directions*, in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2013. https://doi.org/10.1109/CVPR.2013.176.

[4] O. Barinova, V. Lempitsky, E. Tretiak, and P. Kohli, *Geometric image parsing in man-made environments*, in Proceedings of European Conference on Computer Vision, 2010.

[5] P. Denis, J.H. Elder, and F.J. Estrada, *Efficient edge-based methods for estimating Manhattan frames in urban imagery*, in Proceedings of European Conference on Computer Vision, 2008.

[6] M. Dubská, A. Herout, and J. Havel, *PClines - line detection using parallel coordinates*, in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2011.

[7] M. A. T. Figueiredo and A. K. Jain, *Unsupervised learning of finite mixture models*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 24 (2002), pp. 381–396.

[8] R. Grompone von Gioi, J. Jakubowicz, J.M. Morel, and G. Randall, *On straight line segment detection*, Journal of Mathematical Imaging and Vision, 32 (2008), pp. 313–347. https://doi.org/10.1007/s10851-008-0102-5.

[9] R. Grompone von Gioi, J. Jakubowicz, J. M. Morel, and G. Randall, *LSD: a line segment detector*, Image Processing On Line, (2012). https://doi.org/10.5201/ipol.2012.gjmr-lsd.

[10] R. Grompone von Gioi, J. Jakubowicz, and G. Randall, *Multisegment detection*, Proceedings of IEEE International Conference on Image Processing, (2007).

[11] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*, Cambridge university press, 2003.

[12] J. Lezama, R. Grompone von Gioi, G. Randall, and J. M. Morel, *Finding vanishing points via point alignments in image primal and dual domains*, in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 509–515.

[13] J. Lezama, J-M. Morel, G. Randall, and R. Grompone von Gioi, *A contrario 2d point alignment detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 37 (2015), pp. 499–512.

[14] J. Lezama, G. Randall, J-M. Morel, and R. Grompone von Gioi, *An unsupervised point alignment detection algorithm*, Image Processing On Line, 5 (2015), pp. 296–310. https://doi.org/10.5201/ipol.2015.126.

[15] C. Rother, *A new approach to vanishing point detection in architectural environments*, Image and Vision Computing, 20 (2002), pp. 647–655.

[16] A. Vedaldi and A. Zisserman, *Self-similar sketch*, in Proceedings of European Conference on Computer Vision, 2012.

[17] H. Wildenauer and A. Hanbury, *Robust camera self-calibration from monocular images of Manhattan worlds*, in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2012. https://doi.org/10.1109/CVPR.2012.6248008.

[18] Y. Xu, S. Oh, and A. Hoogs, *A minimum error vanishing point detection approach for uncalibrated monocular images of man-made environments*, Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, (2013).