Geometric Urban Geo-Localization

Mayank Bansal[1,2]
[1]Center for Vision Technologies, SRI International
Princeton, NJ, USA
mayank.bansal@sri.com

Kostas Daniilidis[2*]
[2]GRASP Lab, University of Pennsylvania
Philadelphia, PA, USA
kostas@cis.upenn.edu

Abstract

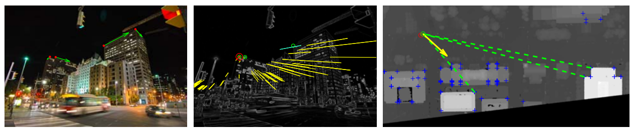
We propose a purely geometric correspondence-free approach to urban geo-localization using 3D point-ray features extracted from the Digital Elevation Map of an urban environment. We derive a novel formulation for estimating the camera pose locus using 3D-to-2D correspondence of a single point and a single direction alone. We show how this allows us to compute putative correspondences between building corners in the DEM and the query image by exhaustively combining pairs of point-ray features. Then, we employ the two-point method to estimate both the camera pose and compute correspondences between buildings in the DEM and the query image. Finally, we show that the computed camera poses can be efficiently ranked by a simple skyline projection step using building edges from the DEM. Our experimental evaluation illustrates the promise of a purely geometric approach to the urban geo-localization problem.

1. Introduction

In this paper, we study the challenging problem of geo-locating a street-level image using only the corners and roof-line edges of the buildings visible in the image and matching them geometrically to a database of 3D corners and direction vectors extracted from an elevation map without using any appearance information from the database.

Recent work on geo-localization using a model database has relied largely on rendering-based techniques [12, 10, 1]. Typically, these techniques render the 3D model on a uniform grid in the ground-plane, compute features for each rendering and then match these against the query features to retrieve candidate locations. While these techniques have proven efficient and effective for geolocating in a mountainous terrain [1], their adaptation to urban environments has not had the same level of success. The reason for this is the

*The authors are grateful for support through the following grants: NSF-IIP-0742304, NSF-OIA-1028009, ARL MAST-CTA W911NF-08-2-0004, ARL RCTA W911NF-10-2-0016, NSF-DGE-0966142, and NSF-IIS-1317788.



computational overhead of rendering the building models at a fine enough resolution such that the rendering can closely match the query image.

In this paper, we take a different approach to the urban geolocalization problem. Instead of pre-rendering the model, we extract purely geometric entities from the model and the query image and then use them directly to solve for the camera pose. To handle this combinatorial problem in a tractable manner, we propose a novel framework for correspondenceless pose-estimation in a 3D-to-2D setup that employs a minimal solver without suffering from the combinatorial explosion typical in such setups. In particular, we employ the two-point method for estimating the pose of a calibrated camera with known vertical direction in the image. Without a set of extracted 3D-to-2D correspondences, employing even a 2-point algorithm for pose estimation is prohibitively expensive. In a geolocation setup, assuming we have identified m building corners in the query image and have n 3D-building corner points in a database, the cost of testing all minimal configurations is $O(m^2n^2)$. In addition, the number of correct correspondences is at most m which makes any direct voting-based method infeasible.

Therefore, we propose a stratified approach that uses a lower number of constraints to compute a partial solution which is then used to generate putative correspondences on which the 2-point method can be applied. Specifically, first we use a single point and line direction (ray) correspondence to solve for the camera pose partially. In this partial solution, the camera rotation (pan) is recovered and the translation is expressed as a locus along a 3D-line segment. We show that the projection of 3D points using this parametric camera pose generates line-segments in the image. The perpendicular distance of the image points from these segments can be used to identify putative correspondence of the 2D-points with the 3D-points corresponding to these line segments. This is a novel insight that uses a partial solution to establish putative correspondences without using any appearance information. The standard 2-point algorithm can now be applied in a RANSAC setting to this putative set to generate a hypothesis camera pose. In this framework, we test at most $O(m^2 n)$ minimal configurations which is a substantial cost saving in typical problem instances where $n >> m$. In the geolocalization setting we address in this paper, n corresponds to the total number of building corners in a database which is substantially larger than the number of building corners detected in a single query image (m).

Our paper distinguishes from the state of the art in the following contributions:

1. A novel formulation for vertical pose estimation using a point-line pair.
2. Degeneracy conditions for the point-line problem.
3. A novel framework for stratified pose estimation using point-line and 2-point algorithms.
4. An application of this framework to geo-localization without appearance correspondences.
5. The fact that we avoid any visibility information or rendering.

2. Related Work

Image-based geo-localization has largely been approached as an appearance matching problem between a query image and a database of geo-tagged images. Zamir and Shah [14] employed a structured dataset of $360°$ panoramic imagery from Google Street-view to create an index of SIFT features which is used to geolocate a query image. However, their method requires an extensive dataset to be available and be indexed. SIFT feature matching was also employed for urban localization by Zhang and Kosecka[15]. Hays and Efros[3] propose a data-driven approach to single-image localization which also uses scene features a large dataset. More recent work on image-based geolocalization has also looked at using non-ground-level database imagery. Although direct feature correspondence is not employed in these approaches, appearance information is still used either in a bag-of-words or a feature learning framework. Examples of these two frameworks include work on using self-similarity bag-of-words features for matching to oblique aerial imagery [2], multiple feature learning for matching to satellite and land cover attribute imagery [8] and static camera localization by correlating with satellite imagery [5].

Digital elevation models (DEM) and 3D models of the environment have shown promise for the geo-localization problem as well. Baatz *et al*. [1] described a framework for geolocating queries in a mountainous terrain by matching against skylines pre-rendered from digital elevation models. Ramalingam *et al*. [11] present a formulation for computing camera pose using minimal configurations of points and lines, and use this to geolocate a query using the 3D model of a city. However, their approach demands the availability of an initial correspondence between one query image and the 3D model. This correspondence is used to setup 3D-to-2D constraints which are then propagated to a new query image using image-to-image appearance matching. Thus, they do not address the geo-localization problem in the traditional sense and implicitly use image appearance. Skylines precomputed from a 3D model have been used for urban geolocalization of an omni-camera in [12]. However, the approach has shown more promise for keeping track of the camera location rather than for initialization.

In terms of the problem setup, our work is most closely related to the urban geo-localization setup of Matei *et al*. [10]. They used a LIDAR scan of the environment to create a DEM which is rendered exhaustively from multiple locations and viewpoints. Features extracted from these renderings are matched against query features to generate candidate camera locations. In this work, we employ a DEM as the starting point of our database as well. However, instead of rendering apriori in a quantized pose space, we extract sparse *PointRay* features which are purely geometric and allow us to compute candidate query poses without any appearance matching. We verify each candidate pose by comparing the building skyline visible in the query with the skyline rendered from the candidate viewpoint. This rendering step is very efficiently performed by linear algebraic means and involves projection of building contours from the DEM and does not need any depth culling computation.

Closed-form minimal solutions to the absolute pose problem for a vertical camera were first proposed by Kukelova *et al*. [7]. We propose a novel formulation for the geo-localization problem using this minimal solver and derive a stratified approach that can work in a correspondence and appearance-free setup by solving for the partial pose using a point-ray correspondence. The 2-point absolute pose framework is also employed by Saurer *et al*. [13] for visual odometry under vertical camera assumption. They also confirm that the vertical direction measurement from an off-the-shelf IMU is accurate enough for the 2-pt pose estimation algorithm.

Correspondenceless estimation of pose or scene structure has been addressed in [9]. However, their method relies on appearance matches between SIFT descriptors while our approach works purely with geometric entities.

The paper is organized as follows. In section-3, we formulate the geometric problem for estimating the pose of a vertical calibrated camera in terms of minimal set of points and direction correspondences. Section-4 presents details of our proposed algorithm for geo-localization including procedures for detecting query and database features. Section-5 presents experimental results.

3. With Correspondence Information

3.1. Preliminaries

Let $\bar{\mathbf{p}} = (\bar{u}, \bar{v}, 1)^T$ be the (homogeneous) image projection of a 3D point $\mathbf{P} = (X, Y, Z)^T$. Then the projection equation is given by:

$$K(R\mathbf{P} + \mathbf{t}) \cong \bar{\mathbf{p}} \tag{1}$$

where K is the camera internal matrix and R and $\mathbf{t} = (t_x, t_y, t_z)^T$ are the camera rotation and translation respectively relative to the world. Defining $\mathbf{p} = K^{-1}\bar{\mathbf{p}}$ and converting the projective equivalence to equality by taking a cross product, we get:

$$[\mathbf{p}]_\times (R\mathbf{P} + \mathbf{t}) = 0 \tag{2}$$

where $\mathbf{p} = (u, v, 1)^T$ are the normalized coordinates of the image point and $[\mathbf{p}]_\times$ is the skew-symmetric matrix representing cross-product with the vector $\mathbf{p}$.

If the Y-axis of the camera is aligned with the Y-axis (and the gravity vector in the geo-localization setup) of the world coordinate system, the rotation matrix R can be simplified to:

$$R = \begin{pmatrix} \frac{1-q^2}{1+q^2} & 0 & \frac{-2q}{1+q^2} \\ 0 & 1 & 0 \\ \frac{2q}{1+q^2} & 0 & \frac{1-q^2}{1+q^2} \end{pmatrix} \tag{3}$$

where $q = \tan(\theta/2)$ and θ is the unknown camera pan angle. Substituting in equation (2) and denoting $\mathbf{q} = (q^2, q, 1)^T$, we get:

$$\mathbf{f}(u, \mathbf{P})^T \mathbf{q} = (u\bar{t}_z - \bar{t}_x) \tag{4}$$

$$\mathbf{g}(v, \mathbf{P})^T \mathbf{q} = (\bar{t}_y - v\bar{t}_z) \tag{5}$$

where,

$$\mathbf{f}(u, \mathbf{P}) = \begin{pmatrix} uZ - X \\ -2(uX + Z) \\ X - uZ \end{pmatrix}, \quad \mathbf{g}(v, \mathbf{P}) = \begin{pmatrix} -Y - vZ \\ 2vX \\ vZ - Y \end{pmatrix} \tag{6}$$

and $\bar{\mathbf{t}} = (\bar{t}_x, \bar{t}_y, \bar{t}_z)^T = (1 + q^2)\mathbf{t}$.

3.2. Correspondence of two points

Proposition 1. *Given correspondence of any two points, not in a degenerate configuration, there are two possible solutions for the camera pose*[7].

Proof. Let $\mathbf{P}_1 = (X_1, Y_1, Z_1)^T$ and $\mathbf{P}_2 = (X_2, Y_2, Z_2)^T$ be the two 3D points and $(u_1, v_1)^T$ and $(u_2, v_2)^T$ be the corresponding image points. Then, from equation (4), we get the following pair of equations:

$$\mathbf{f}(u_1, \mathbf{P}_1)^T \mathbf{q} = (u_1\bar{t}_z - \bar{t}_x) \tag{7}$$

$$\mathbf{f}(u_2, \mathbf{P}_2)^T \mathbf{q} = (u_2\bar{t}_z - \bar{t}_x) \tag{8}$$

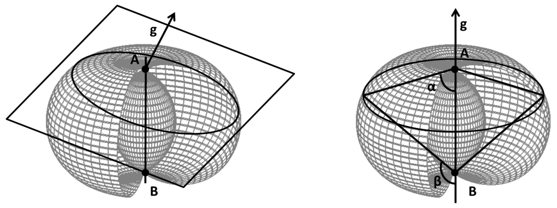


Figure 1. Degenerate cases for the two point method

Eliminating $\mathbf{t}_x$ between equations (8) and (7), we get:

$$(\mathbf{f}(u_1, \mathbf{P}_1) - \mathbf{f}(u_2, \mathbf{P}_2))^T \mathbf{q} = (u_1 - u_2)\bar{t}_z \tag{9}$$

Similarly, from equation (5), we get:

$$\mathbf{g}(v_1, \mathbf{P}_1)^T \mathbf{q} = (\bar{t}_y - v_1\bar{t}_z) \tag{10}$$

$$\mathbf{g}(v_2, \mathbf{P}_2)^T \mathbf{q} = (\bar{t}_y - v_2\bar{t}_z) \tag{11}$$

which, after eliminating $\bar{t}_y$ gives:

$$(\mathbf{g}(v_1, \mathbf{P}_1) - \mathbf{g}(v_2, \mathbf{P}_2))^T \mathbf{q} = (v_2 - v_1)\bar{t}_z \tag{12}$$

Eliminating $\bar{t}_z$ from equations (9) and (12), we get a quadratic in q which can be solved to get two solutions for the camera rotation parameter q. Substituting this value in either equation (9) or (12), we can solve for $\bar{t}_z$ and then for $\bar{t}_x$ and $\bar{t}_y$ from the remaining equations leading to two solutions for the camera pose. □

Degeneracies. *The two-point method is degenerate if and only if either a) the two points lie in the XZ-plane passing through the camera center, or b) the two points lie on the same vertical line.*

Proof. Assume two points A and B at known positions in a 3D world frame. It is well known that the locus of camera centers O which view these points at the same relative non-zero angle (relative bearing) is a toroid. This is easy to see since in the 2D case the locus is a circle which when rotated about axis AB becomes a toroid. It is also known that with one more point (P3P), three toroids are created which in the general case intersect at 8 points.

If the third point is at infinity then we have the case of a known direction $\mathbf{g}$ w.r.t. the world (like gravity). Assume that the angle between the known direction and the ray to A is α. Then the locus of camera centers that see A under angle α w.r.t. the known direction is a cone $((\mathbf{X} - \mathbf{A})^T \mathbf{g})^2 = (\mathbf{X} - \mathbf{A})^2 \cos^2 \alpha$. A second cone is created for the constraint that point B is seen under angle β w.r.t. the known direction $\mathbf{g}$. In the general case, the toroid intersects with the two cones in two points.

This is not the case when the cone degenerates to a plane (case 1) or when the intersections of each cone with the toroid are identical (case 2), yielding in both cases a one-parameter family of solutions (Fig. 1).

Case 1: When the angle α is $90°$, the cone degenerates into a plane. This is not a problem when this happens with one of the points since the number of intersections of a toroid, a cone, and a plane, is finite. In the case of gravity, this is the case when one of the two points is in the middle row of the image. However, if this happens to both points $\alpha = \beta = 90°$ then the two planes coincide and the intersection with the toroid is a circle. This is the only case when the intersection of the two "cones" is a two-parameter solution.

Case 2: If the line AB happens to be parallel to the direction of reference, then the two cones intersect at a circle. This circle is contained in the toroid because at each position of this circle the relative bearing $A\hat{O}B$ is constant and equal to $|\beta - \alpha|$. This is not the case with points in the general intersection between two cones because in this case the angles α and β are spanned in different planes. This is also the only case (except case 1) where the intersection is a curve because this is the only case when the intersection between the cones is a circle. In any other case the intersection between two cones is a conic section and the only conic section satisfying the constancy of relative bearing is the circle. $\square$

Proposition 2. *Given correspondence of two points at the same elevation from the ground, the camera location can be determined without the knowledge of the elevation of the points up to an unknown t_y i.e. the camera elevation will remain undetermined.*

Proof. Let the unknown elevation be $Y = Y_1 = Y_2$. Then, from equation (6), $\mathbf{g}(v_1, \mathbf{P}_1) - \mathbf{g}(v_2, \mathbf{P}_2)$ is independent of Y. Since equation (9) is also independent of Y, we can now solve for q and then $\bar{t}_z$ and $\bar{t}_x$ using the same steps as in proposition (1). From equation (10) (or (11)), t_y can be expressed as follows:

$$t_y = v_1 t_z + \frac{q^2(-v_1 Z_1) + q(2v_1 X_1) + v_1 Z_1}{1 + q^2} - Y \quad (13)$$

This places the camera vertical translation t_y at a fixed offset relative to the unknown elevation Y of the 3D points. $\square$

3.3. Correspondence of one line

Proposition 3. *For a camera with its vertical axis aligned with the world vertical, a single line correspondence is sufficient to determine two possible solutions for the unknown camera rotation (pan angle).*

Proof. Let $\mathbf{L} = (L_x, L_y, L_z)^T$ be the direction vector of a 3D line and $\mathbf{n} = (n_1, n_2, n_3)^T$ be the homogeneous representation of the corresponding line observed in the image. Then, the following result expresses the relationship between $\mathbf{n}$, $\mathbf{L}$ and the rotation matrix R of the camera in the world coordinate system:

$$\mathbf{n}^T R \mathbf{L} = 0 \quad (14)$$

For a vertically aligned camera, the rotation matrix R can be set from equation (3) leading to the following quadratic equation in the unknown camera rotation q:

$$\mathbf{l}^T \mathbf{q} = 0 \quad (15)$$

where,

$$\mathbf{l} = (-L_x n_1 + L_y n_2 - L_z n_3, 2L_x n_3 - 2L_z n_1, \mathbf{L}^T \mathbf{n})^T \quad (16)$$

The above equation leads to two different solutions for the unknown q. $\square$

Degeneracies. *For a camera with its vertical axis aligned with the world vertical, estimation of the camera rotation (pan) using a line correspondence is degenerate if and only if either a) the line is vertical in the world or b) the line is in the XZ-plane passing through the camera center.*

Proof. The equation (15) is degenerate when the quadratic coefficients $\mathbf{l}$ are all zero. Setting $\mathbf{l} = 0$ leads to the following four conditions: a) $\mathbf{L} = 0$, b) $\mathbf{n} = 0$, c) $\{n_2 = 0, L_z = 0, L_x = 0\}$, and d) $\{n_1 = 0, n_3 = 0, L_y = 0\}$.

In (c), $n_2 = 0 \Rightarrow \mathbf{n} = (\alpha, 0, \beta)$, which implies that the line is vertical in the image. Also, $L_z = L_x = 0$ implies that it is vertical in the world as well.

In (d), $n_1 = n_3 = 0$ implies the image line $v = 0$ which is a horizontal line passing through the image center. This implies that $Y = -t_y = c_y$ from the camera projection equation. Coupled with the condition $L_y = 0$, this is a line in the XZ-plane passing through the camera center. $\square$

3.4. Correspondence of a point and a line

Proposition 4. *Given the correspondence of a point and 3D direction vector, the camera pose can be determined upto an unknown location on a 3D line.*

Proof. From the line correspondence, first we estimate the unknown camera pan angle using proposition (3). Then, we use the point correspondence in equations (7) and (10) to establish the locus of the camera center. Substituting the values of the known point coordinates and the estimated value of q, we get equations of the form:

$$t_x = u_1 t_z + \alpha \quad (17)$$
$$t_y = v_1 t_z + \beta \quad (18)$$

where α and β are functions of (u_1, v_1) and (X_1, Y_1, Z_1) and are known. Thus, the locus of the camera translation $\mathbf{t}$

is a 3D line.

$$\mathbf{t} = \frac{t_y}{v_1}\begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} + \frac{1}{v_1}\begin{pmatrix} \alpha v_1 - \beta u_1 \\ 0 \\ -\beta \end{pmatrix} \tag{19}$$

with orientation given by the vector $(u_1, v_1, 1)^T$. □

Degeneracy. The above formulation is degenerate when $v_1 = 0$ (which is when the point $\mathbf{P}_1$ is on the XZ-plane passing through the camera center). In this case, we can solve for t_y directly as $t_y = \beta$ and the locus of the translation is given by equation (17). This locus is a line on the XZ-plane passing through the camera center. Note that in this case the camera location is not constrained even with the knowledge of the camera height.

Corollary 1. *A camera with known rotation and location on a 3D line projects another 3D point to a line in the image which passes through the first point.*

Proof. Let $\mathbf{P}_2 = (X_2, Y_2, Z_2)^T$ be the second 3D point (first point being the reference point used to determine the locus of the camera location). Its projection in the image is then given by:

$$\mathbf{p}_2 \cong R\mathbf{P}_2 + \mathbf{t} \tag{20}$$

Substituting the locus from equation (19), we get:

$$\mathbf{p}_2 \cong R\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \end{pmatrix} + \frac{t_y}{v_1}\begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} + \frac{1}{v_1}\begin{pmatrix} \alpha v_1 - \beta u_1 \\ 0 \\ -\beta \end{pmatrix} \tag{21}$$

The point $\mathbf{p}_2$ traces a line in the image as t_y is varied. As $t_y \to \infty$, $\mathbf{p}_2 \to (u_1, v_1, 1)^T$. This corrolary is depicted on a real example in Fig. 16. □

4. Correspondenceless Geolocalization

With the machinery established in the previous section, we are in a position to describe an algorithm for geolocating a street-level image using a database of 3D building corners extracted from a Digital Elevation Map (DEM) of the environment.

Let $\mathbf{P}_j \in \Re^3$ with $j \in \{1, \ldots, n\}$ be the set of n 3D building corners extracted from the DEM. Each corner is also associated with the 3D direction vector $\mathbf{L}_j \in \Re^3$ along the roofline of the building [1]. The pairing of each point and the direction vector is referred to as a *PointRay* feature. The set of *PointRay* features extracted from the DEM is represented by the database feature set $\mathcal{D} = \{(\mathbf{P}_j, \mathbf{L}_j), j = 1, \ldots, n\}$. An algorithm for creating this feature set from a DEM will be discussed in section-4.2.

Similarly, we process the query image to detect a number of building corners and associate them with roofline edges (in the image) to generate a set of 2D *PointRay* features $(\mathbf{p}_i, \mathbf{l}_i)$ where $\mathbf{p}_i \in \Re^2$ is the point coordinate and $\mathbf{l}_i \in \Re^3$ are the line coefficients. This leads to the query feature set $\mathcal{Q} = \{(\mathbf{p}_i, \mathbf{l}_i), i = 1, \ldots, m\}$. An algorithm for creating this feature set from a query image will be discussed in section-4.2.

We begin by describing an algorithm that naively uses the 2-point algorithm in Proposition-1 to recover camera pose using $\mathcal{D}$ and $\mathcal{Q}$.

Algorithm-0. We select a pair of features from $\mathcal{Q}$ and a pair of features from $\mathcal{D}$ and employ the 2-point algorithm to compute pose hypothesis $(R, \mathbf{t})$. Then, we project all 3D-points $\mathbf{P}_j$ into the image using the computed camera $(R, \mathbf{t})$, and count the number of image features $\mathbf{p}_i$ for which a 3D-point projects within a threshold ϵ. The number of inliers can be used as a score for this pose. Since there is no apriori correspondence information, the above process has to be repeated for all potential minimal (2-point) configurations i.e. $\binom{m}{2}\binom{n}{2} \approx O(m^2n^2)$ times. Clearly, a skyline rendering-based verification step is not feasible with such a large candidate pose space.

4.1. Stratified Geo-localization Algorithm

We select a single feature from $\mathcal{Q}$, associate it with a single feature from $\mathcal{D}$ and employ the point-line algorithm in Proposition-4 to compute a pose hypothesis $(R, \mathbf{t}_\alpha)$ (α parameterizes the 3D camera height). Then, similar to Algorithm-0, we project all 3D-points $\mathbf{P}_j$ into the image using the computed camera $(R, \mathbf{t}_\alpha)$. However, in this case, by corrolary-1, each point projects to a line-segment in the image. For each point $\mathbf{p}_i$, we search for the closest line-segment (within a reprojection error threshold $\epsilon = 10$ pixels) and associate the corresponding 3D point with this query feature. This set of putative correspondences is now used to solve the 2-point problem (Proposition-1) in a RANSAC-based setting to generate a refined pose. Similar to Algorithm-0, a score can be associated with this pose by counting the number of inliers or by a separate scoring function. The above process is repeated for each pair of features from $\mathcal{Q}$ and $\mathcal{D}$ i.e. mn times.

Algorithm-1 describes the proposed geo-location algorithm. The function $\mathrm{PosePointRay}(\mathbf{p}_i, \mathbf{l}_i, \mathbf{P}_j, \mathbf{L}_j)$ applies Proposition-4 to recover the camera rotation R and translation locus $\mathbf{t}_\alpha$ from a single point and direction correspondence. The function $\mathrm{Project}(\mathbf{P}_k, K, R, \mathbf{t}_{\alpha_1}, \mathbf{t}_{\alpha_2})$ projects the 3D-point $\mathbf{P}_k$ using the camera matrices $K[R|\mathbf{t}_{\alpha_1}]$ and $K[R|\mathbf{t}_{\alpha_2}]$ to a line segment in the image $\mathbf{s}_k$.

The parameters α_1 and α_2 specify an interval for allowable camera height. In our implementation, we keep

[1]Each corner is associated with two roofline edges and hence two different direction vectors. Thus, each corner is represented twice in the set $\{\mathbf{P}_j\}$ once for each direction vector.

this fairly loose to compensate for image noise and variable ground elevation at each location. Our implementation uses $\alpha_1 = Y_{ground} - 5$m and $\alpha_2 = Y_{ground} + 20$m where Y_{ground} is the estimated average ground elevation obtained from the DEM.

The for loop on line-number-7 builds a set of correspondences $\mathcal{C}$ by looking for the nearest line-segment from the set $\{\mathbf{s}_i\}$ for reach query point $\mathbf{p}_k$. The function $\mathrm{Nearest}(\mathbf{p}_k, \{\mathbf{s}_i\}, \epsilon)$ returns the index of the 3D-point that is closest.

The function $\mathrm{PoseTwoPoints}(\mathbf{p}_i, \mathbf{P}_j, \mathcal{C})$ uses the two-point algorithm in Proposition-1 using $(\mathbf{p}_i, \mathbf{P}_j)$ as a fixed correspondence and successively trying out each candidate correspondence from $\mathcal{C}$. For each choice, the algorithm produces two solutions from which the rotation which is closer to the original estimate R is selected. Since $\mathcal{C}$ can have at most m elements, each call to $\mathrm{PoseTwoPoints}()$ invokes Proposition-1 at most m times. However, it outputs only one solution pose $(R', \mathbf{t}')$ – the pose for which maximum number of elements of $\mathcal{C}$ have a low image projection error.

The run-time complexity of this algorithm is $O(m^2 n)$ and it can produce an output list $\mathcal{P}_0$ of length at most $2mn$. Next, we describe the functions Filter and ScoreAndRank.

```
input      : Query features $\{(\mathbf{p}_i, \mathbf{l}_i), i = 1, \ldots, m\}$.
             Database features
             $\{(\mathbf{P}_j, \mathbf{L}_j), j = 1, \ldots, n\}$.
parameters: Camera height interval $[\alpha_1, \alpha_2]$.
output     : Camera pose set
             $\mathcal{P} = \{(R_k, \mathbf{t}_k), k = 1, \ldots, \leq 2mn\}$
             with associated scores $S_k$.
1  $\mathcal{P}_0 \leftarrow \emptyset$;
2  for $i \leftarrow 1$ to $m$ do
3      for $j \leftarrow 1$ to $n$ do
4          $(R, \mathbf{t}_\alpha) \leftarrow$ PosePointRay($\mathbf{p}_i, \mathbf{l}_i, \mathbf{P}_j, \mathbf{L}_j$);
5          $\mathbf{s}_k \leftarrow$ Project($\mathbf{P}_k, K, R, \mathbf{t}_{\alpha_1}, \mathbf{t}_{\alpha_2}$)
           $\forall k \in \{1, \ldots, n\}$;
6          $\mathcal{C} \leftarrow \emptyset$;
7          for $k \leftarrow 1$ to $m$ do
8              $f_k \leftarrow$ Nearest($p_k, \{\mathbf{s}_i\}, \epsilon$);
9              $\mathcal{C} \leftarrow \mathcal{C} \cup (p_k, P_{f_k})$;
10         end
11         $(R', \mathbf{t}') \leftarrow$ PoseTwoPoints($\mathbf{p}_i, \mathbf{P}_j, \mathcal{C}$);
12         $\mathcal{P}_0 \leftarrow \mathcal{P}_0 \cup \{(R', \mathbf{t}')\}$;
13     end
14 end
15 $\mathcal{P}_1 \leftarrow$ Filter($\mathcal{P}_0$);
16 $\mathcal{P} \leftarrow$ ScoreAndRank($\mathcal{P}_1$);
```

Algorithm 1: Proposed Geo-localization Algorithm

4.1.1 Candidate Filtering

The candidate pose list $\mathcal{P}_0$, in the worst case, can be of length $2mn$. Since the camera height was restricted to a fairly loose interval using the parameters α_1 and α_2, there are potentially a large number of recovered poses which do not agree with the ground-level at the lat-long location of the pose. However, the function Filter removes such poses and generates the output list $\mathcal{P}_1$ as follows. First, we filter out all camera poses that land outside the extent of the DEM. Next, we look at the XZ-location of each camera pose and look-up the ground elevation from the DEM. If the camera height c_y is within a threshold h_{cam}m of the ground elevation then we keep this pose, otherwise we filter it out. The threshold h_{cam} allows us to seamlessly model specific geo-localization scenarios. For example, when the query imagery is only taken by walking pedestrians, we can set this threshold to say $h_{cam} = 2.5$m to filter out poses with camera placed higher than 2.5m.

4.1.2 Scoring and Ranking

For geo-localization purposes, each pose $(R, \mathbf{t})$ in the list $\mathcal{P}_1$ needs to be associated with a score so that a ranked list can be created and further verified either using appearance or using a specialized rendering-based matcher. We propse two algorithms for scoring each pose and present experimental comparison of this two methods in section-5.

Inlier edge scoring. For this scoring step , we look at the set of inlier correspondences that generated each pose $(R, \mathbf{t})$. Since each building corner is associated with 3 building edge directions, we project these lines for each inlier into the image and score the correspondence based on the edge-strength accumulated by the lines in the query gradient map. The sum of scores over all inlier correspondences constitutes the score for the pose. Fig. 2(c) shows an example of the line projection from the inlier correspondences. The intuition behind this scoring strategy is to penalize poor features detected on the query which may not have enough edge-support when measured using projected building edges.

Skyline match verification. The skyline-based verification step is similar to the use of urban-skylines for pose estimation by Ramalingam *et al.* [12]. However, instead of pre-rendering the urban skylines from the continuous pose space, we only render the skyline at the camera poses in the filtered list. In addition, this rendering step is very efficient as it can be carried out using linear algebra alone. We take the 3D point set corresponding to the DEM edge map $\mathcal{E}$ (see section-4.2) and project it to the image using the camera matrix. Keeping only the points which fall into the query image, we determine the highest point that projects at each image column by a simple linear search. This gives us the v coordinate for the rendered skyline at each column of the image. Fig. 2(d) shows an example of the skyline rendered from a correctly computed pose. For the query image, we directly use the sky-mask from the Geometric Context [4] algorithm to generate a skyline. At this stage, we can use the two skylines to refine the estimated pose. However, in this paper, we only use the skylines to generate a score for each pose – the scoring function measures the area overlap between the sky regions from the rendered and the query image using an intersection over union criterion.

Fig. 2 walks through the entire algorithm on a query from our evaluation set.

4.2. Feature Extraction

Database Features. The database feature set $\mathcal{D}$ is generated from a 3D-model or a LIDAR scan as follows. First, we project the data using an orthographic camera onto the ground-plane and keep the height information of the highest point at each pixel. This generates a digital elevation map (DEM) with the elevation information represented by the pixel intensity. We use the median elevation within the DEM as an approximate ground elevation estimate Y_{ground} and use it to mask out the ground regions. This generates building regions as disconnected components which are futher processed one at a time. Within each component, we use the video-compass [6] algorithm to detect long line-segments, associate each line-segment with the elevation information from the DEM underneath, and then intersect them pairwise to generate corner locations in the DEM image. Thus, each corner is associated with a 3D location $\mathbf{P}_j$ in the world coordinate system. The line-segments that intersect at this corner are converted to direction vectors $\mathbf{L}_j$ thus generating pairs $(\mathbf{P}_j, \mathbf{L}_j)$. The set of pairs from all building components is collected into the set $\mathcal{D}$ of database features. The canny edge map used for line extraction is also associated with elevation information from the DEM and is stored as a set of 3D coordinates $\mathcal{E}$ for use in the skyline rendering step of our algorithm. Note that the above algorithm is much simpler than the typical processing that is required to extract full building models from a DEM[10]. This is a distinct advantage of our *PointRay* feature which does not need full building outlines to be extracted.

Query Features. Before computing the query features, we rectify the query image (to a vertical camera orientation) using vanishing-points. We use the video-compass algorithm from [6] to fit line segments to the canny edge-map of the query image. This gives us a set of candidate line-segments from which we select candidate "sky-hugging" segments by scoring each segment using a sky-probability map computed by the Geometric-Context algorithm [4]. The resulting segments are then intersected pairwise to generate candidate building corners. For each corner $\mathbf{p}_i$, we thus also obtain the line segment(s) $\mathbf{l}_i$ that generated this corner and are along building rooflines by construction. In practice, for each corner we also verify its proximity to the end-points of the intersecting line-segments, and remove multiple corners by non-maximal suppression. The pairs $(\mathbf{p}_i, \mathbf{l}_i)$ that pass the verification are collected into the final feature set $\mathcal{Q}$. Fig. 2(a) shows an example of the extracted *PointRay* features.

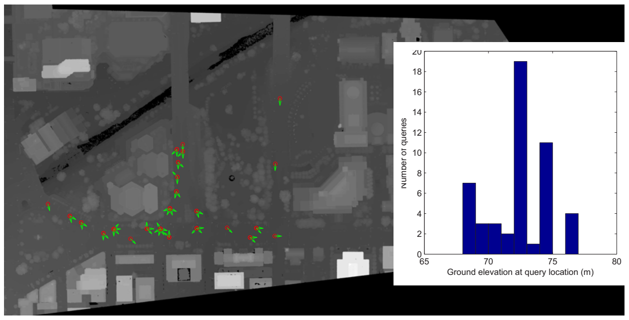


Figure 3. DEM from Ottawa with query locations overlaid. Each location is shown as a red circle with green arrows depicting the look-at vectors corresponding to each query image. The inset shows the elevation distribution for the query set.

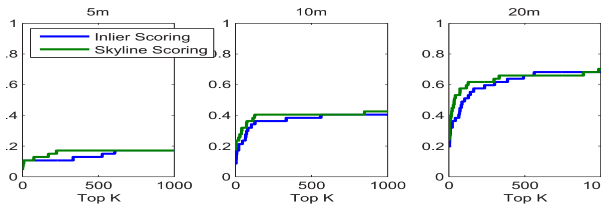
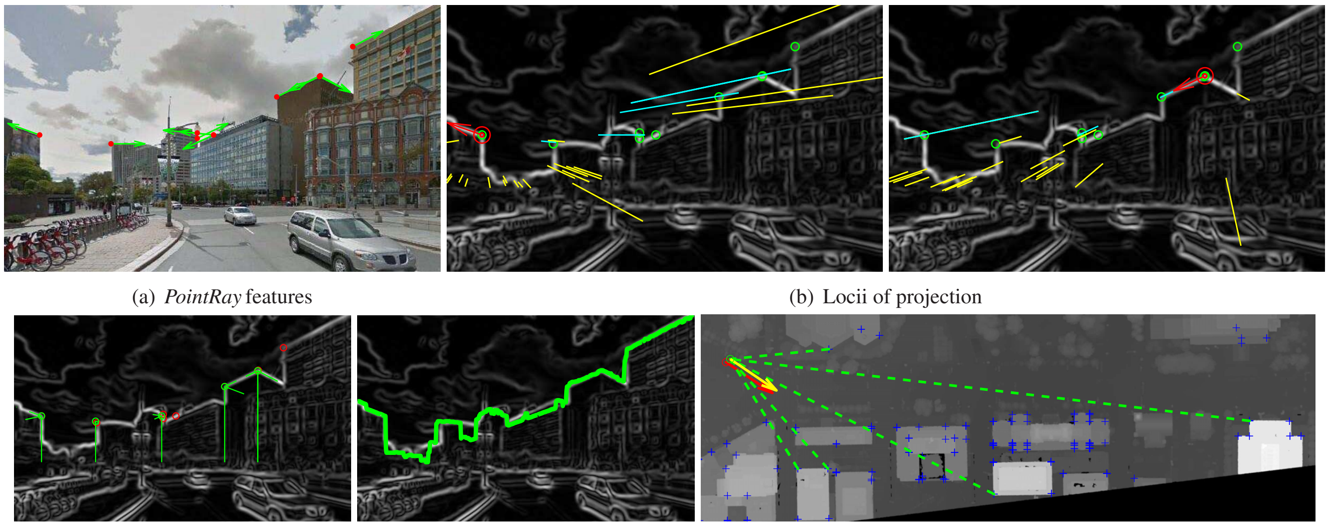


Figure 4. Geo-localization performance evaluation.

5. Experiments

We use the publicly available dataset of aerial LIDAR scans of Ottawa, Ontario, Canada. We created a DEM at a ground resolution of 0.25m per pixel for a 1Km $\times$ 0.5Km area from this data (see Fig. 3) and ran the feature extraction algorithm (section-4.2) resulting in approximately $n = 600$ 3D *PointRay* features.

For query data, we use Google Street-view imagery downloadable at specified latitude-longitude locations. 50 images from several locations within the extent of the DEM map at several different camera pan angles were downloaded (see Fig. 3 for the query pose variety). The camera tilt was set at zero degrees to simulate a vertical camera. The camera internal parameters were fixed at a field-of-view of $90°$ and a resolution of 640×640. This is the maximum available resolution from Gooogle for general users and the poor quality (pixellation, ringing, blur) of this imagery poses numerous challenges for line and corner extraction algorithms. For each query, we automatically extract the query features using the algorithm in section-4.2. We found that detection of *PointRay* features is quite robust to image quality issues and is most severely impacted by poor sky segmentation from the Geometric Context algorithm when it gets confused by tall construction equipment in the scene.



(a) *PointRay* features (b) Locii of projection

(c) Inliers from the two-point fit (d) Skyline match verification (e) Localization on the DEM

Figure 2. Query processing: a) Each red-dot and green-arrow pair make up one *PointRay* feature $(\mathbf{p}_i, \mathbf{l}_i)$. b) The *PointRay* feature $(\mathbf{p}_i, \mathbf{l}_i)$ shown in red when corresponded with the correct 3D *PointRay* feature from the database results in a camera pose locus. By corollary-(1), the camera in this locus maps the remaining database corners to line segments shown in yellow and cyan. The cyan segments represent the inlier set $\mathcal{C}$ because of their proximity to the query corners (shown as green circles). We show two cases here where different locii are created by using a different (but correct) reference *PointRay* feature correspondence. c) Green circles denote the inliers after the two-point method is applied to $\mathcal{C}$; the green lines depict projected 3D edges using the computed camera pose. d) DEM contours rendered using the hypothesis camera pose generate the green skyline which matches correctly with the perceived skyline. e) DEM showing the reference corners (blue +), recovered camera pose (yellow), ground-truth camera pose (red) and the recovered inliers (green).

For each query image, we run our geo-localization algorithm and generate the list of candidate poses $\mathcal{P}$ sorted by decreasing scores. Given the precise ground-truth location and orientation for each query, we label as true-positive each returned pose that is within a distance threshold τ of the ground-truth location and has a look-at vector within 25° of the ground-truth rotation. Fig. 4 presents the results for three different distance thresholds, τ: 5m, 10m and 20m. For each value of top-K, the precision is measured as the fraction of queries that returned a true-positive pose within the first top-K elements of the output list $\mathcal{P}$. The two curves within each plot compare the results of using the two scoring strategies outlined before. Using geometry alone, we achieve significant performance considering that for our typical case of $m = 15$ and $n = 600$, we potentially create a ranked list of $2mn$ i.e. 18000 poses and still obtain within $20m$ localization in the top-100 ranks.

The inset in Fig. 3 shows the distribution of ground elevation at the ground-truth locations of the 50 queries in our test set. The variation clearly indicates the importance of being able to operate with a flexible camera height assumption in our algorithm. Any algorithm that assumes a fixed camera height (at some nominal ground-plane elevation) would not be able to deal with this variation.

Acknowledgment

The authors thank Charles Karney at SRI International for help with the LIDAR-to-DEM algorithm.

References

[1] G. Baatz, O. Saurer, K. Köser, and M. Pollefeys. Large scale visual geo-localization of images in mountainous terrain. In *ECCV*. 2012. 1, 2

[2] M. Bansal, K. Daniilidis, and H. Sawhney. Ultra-wide baseline facade matching for geo-localization. In *ECCV Workshops*, 2012. 2

[3] J. Hays and A. A. Efros. Im2gps: estimating geographic information from a single image. In *CVPR*, 2008. 2

[4] D. Hoiem, A. A. Efros, and M. Hebert. Geometric context from a single image. In *ICCV*, 2005. 7

[5] N. Jacobs, S. Satkin, N. Roman, R. Speyer, and R. Pless. Geolocating static cameras. In *ICCV*, 2007. 2

[6] J. Košecká and W. Zhang. Video compass. In *ECCV*. 2006. 7

[7] Z. Kukelova, M. Bujnak, and T. Pajdla. Closed-form solutions to minimal absolute pose problems with known vertical direction. In *ACCV*. 2010. 2, 3

[8] T.-Y. Lin, S. Belongie, and J. Hays. Cross-view image geolocalization. In *CVPR*. 2013. 2

[9] A. Makadia, C. Geyer, and K. Daniilidis. Correspondence-free structure from motion. *IJCV*, 75(3):311–327, 2007. 2

[10] B. C. Matei, N. Vander Valk, Z. Zhu, H. Cheng, and H. S. Sawhney. Image to lidar matching for geotagging in urban environments. In *WACV*, 2013. 1, 2, 7

[11] S. Ramalingam, S. Bouaziz, and P. Sturm. Pose estimation using both points and lines for geo-localization. In *ICRA*, 2011. 2

[12] S. Ramalingam, S. Bouaziz, P. Sturm, and M. Brand. Geolocalization using skylines from omni-images. In *ICCV Workshops*, 2009. 1, 2, 6

[13] O. Saurer, F. Fraundorfer, and M. Pollefeys. Homography based visual odometry with known vertical direction and weak manhattan world assumption. *ViCoMoR*, 2012. 2

[14] A. R. Zamir and M. Shah. Accurate image localization based on google maps street view. In *ECCV*. 2010. 2

[15] W. Zhang and J. Kosecka. Image based localization in urban environments. In *3D Data Processing, Visualization, and Transmission, Third International Symposium on*, 2006. 2