Geolocalization using Skylines from Omni-Images

Srikumar Ramalingam[1] Sofien Bouaziz[1&2] Peter Sturm[3] Matthew Brand[1]

[1]Mitsubishi Electric Research Lab (MERL), Cambridge, MA, USA
[2]Virtual Reality Lab, Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland
[3]INRIA Grenoble – Rhône-Alpes and Laboratoire Jean Kuntzmann, Grenoble, France

Abstract

We propose a novel method to accurately estimate the global position of a moving car using an omnidirectional camera and untextured 3D city models. The camera is oriented upwards to capture images of the immediate skyline, which is generally unique and serves as a fingerprint for a specific location in a city. Our goal is to estimate global position by matching skylines extracted from omni-directional images to skyline segments from coarse 3D city models. Our contributions include a sky segmentation algorithm for omni-directional images using graph cuts and a novel approach for matching omni-image skylines to 3D models.

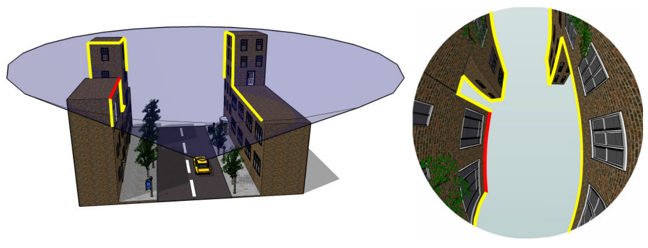

(a) (b)

Figure 1. *An illustration of our motivating idea. (a) A car automounted with an omni-directional camera facing towards the sky. (b) The image captured by the camera with skylines marked. We match these skylines to those synthesized from 3D city models to compute the geospatial location.*

1. Introduction and Previous Work

The skyline has long been a source of fascination for photographers, thought to identify the city as uniquely as a fingerprint. In this work, we take a step further and investigate its uniqueness for any given geospatial location. We propose a simple, robust and fast method to compute the geospatial location by matching the skyline imaged on a fisheye image with the skyline synthesized from a coarse 3D model of a large city. In other words, we solve the pose estimation problem for an omnidirectional camera in the actual world coordinate system. Pose estimation, although extremely challenging due to several degeneracy problems and inaccurate feature matching, is a well researched topic in computer vision. However most existing solutions are only proven on the smaller scale of the laboratory setup.

In the last few years, there has been an increasing interest in inferring geolocation from images [18, 25, 24, 8, 6]. In [18], Robertson and Cipolla showed that it is possible to obtain geospatial localization by matching a query image with an image database using vanishing vertical direction. Zhang and Kosecka showed accurate results in the ICCV 2005 computer vision contest (”Where am I?”) using SIFT features [25]. Jacobs et al. used a novel approach to geolocate a webcam by correlating its images with satellite weather imagery at the same time. Hays and Efros used millions of GPS-tagged images from the web for georeferencing a new image [6]. In contrast to most of these approaches that leverage on the availability of these georeferenced images, we use coarse 3D models from the web for geospatial localization: like georeferenced images, a large repository of coarse 3D models already exists for major cities in the world. Several 3D reconstruction algorithms have been proposed for the reconstruction of large urban scenes [23, 2, 4]. Koch and Teller proposed a localization method using a known 3D model and a wide angle camera for indoor scenes by matching lines from the 3D model with the lines in images [10]. In contrast to their work, our work relies only on the skylines for geolocalization. Although, the idea of using skylines has been explored earlier in [11, 19], our approach is significantly different from them. An expensive infrared camera is used in [11] instead of a visible one. There are several important differences between our approach and [19]. First, a human user input is required to extract the skyline in the case of [11], whereas we develop an automatic algorithm using graph cuts. Second, a hash table is precomputed to match the skylines with the 3D model, whereas our algorithm synthesizes fisheye images on the fly for matching. We show our basic idea of matching skylines from 3D models and omni-directional images in Figure 1.

Our main goal is to obtain highly accurate geolocation for an image. It has been shown formally that omni-directional cameras can give much better accuracy in motion estimation than perspective cameras [9, 1]. In the case of small rigid motions, two different motions can yield nearly identical motion fields for classical perspective cameras, which is not the case for omnidirectional images. Omnidirectional cameras have also proven advantageous in applications such as video conferencing, augmented reality, and surveillance.

In this work we used a fisheye lens with a field of view of about 183°. However, we would like to propose a method that could work with all kinds of omni-directional cameras. For that reason, we chose to use a generic calibration approach which treats every camera as a mapping between a pixel and its corresponding projection rays. This general model has been used in various works [5, 12, 13, 14, 17, 20], and is best described in [5], where properties other than geometric ones are also considered. It was recently shown that the generic calibration algorithm outperforms standard parametric approaches in the case of very high distortions [17, 3]. We use the calibration information to synthesize virtual fisheye views from the 3D model and match these with real fisheye images. To the best of our knowledge, we are not aware of any work in vision which synthesizes omni-directional images from 3D models for pose estimation. The synthesis is done using a pixel shader program implemented on a Graphics processing unit (GPU) for generating a large number of fisheye images in real time.

The various buildings blocks of this system are already novel contributions in their respective domains. First we propose a graph cuts based algorithm embedded with parameter learning for robustly detecting sky in omni-directional images. The most closely related work, which already gives excellent results, is the geometric labeling problem to classify a given image into sky, buildings and ground [7, 16]. However, some of their prior assumptions do not hold true in our model. First, the camera is not perspective. Second, the image is not taken by a person with the optical axis approximately parallel to the ground. In fact, the optical axis is perpendicular to the ground. Third, their algorithm expects the sky to be mostly on the top of the image. On the contrary, our images always have the sky at the center of the image. As a result of these differences, we chose to develop a tailor-made sky detection algorithm for upward looking omni-directional cameras. We then propose a novel method to match skylines from omni-directional images by synthesizing fisheye images using existing 3D models and calibration information, and its application to geolocalization.

2. Overview of our Algorithm

We show the various stages of our algorithm in figure 2. The first step is to detect the region corresponding to sky in the omndirectional images. In order to do this we use a graph cuts based algorithm, which is generally used to minimize a discrete quadratic pseudo-boolean energy function. We give the details of the features used to compute unary and pairwise costs of the energy function. The parameters of the function are usually fixed manually. As we expect our algorithm to work at various lighting conditions and places, we use a parameter learning algorithm to automatically compute the best set of parameters. The exact details of the algorithm are shown in section 2.1.

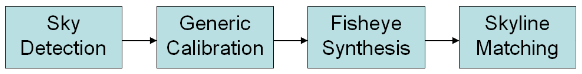


Figure 2. *The various stages of our algorithm.*

The second stage is the calibration algorithm, which is used to compute the 3D projection rays corresponding to every image pixel in the omni-directional image. We also propose enhancements of the calibration algorithm, using plumb-line constraints to significantly improve its performance, cf. section 2.2. Third, we show a novel method to synthesize virtual fisheye views at various locations in a coarse 3D model. This uses pixel shaders executed on a GPU, and is very fast.

The next stage is the skyline matching algorithm where we compute the chamfer distance between the synthesized and real fisheye images. Finally in section 3, we show experimental results on fisheye images captured in the city of Boston. We also show a perturbation analysis on the shape of the skylines to study the robustness of our method and its superiority over perspective images.

2.1. Sky Detection

Given an omni-directional image, which is circular in the case of a fisheye model, we want to classify the pixels into sky and rest. This can be seen as a segmentation problem with two labels. The features that can be used for this segmentation can vary from simple RGB colorspace components to a wide variety of features like gradients, straight lines, vanishing points, etc. Our approach has two modules: a parameter learning method and a discrete optimization algorithm. In our problem we use graph cuts, which is both fast and highly successful in various vision problems like stereo, segmentation, and image restoration [21]. An energy function, involving binary variables in unary and pairwise terms, is represented using a weighted graph whose minimum cut (computed using the maxflow algorithm) yields an energy-minimizing partition of the variables. The minimum cut separates the set of nodes in the graph into two sets, one belonging to a source node and one belonging to a sink node. Formally, let $\mathbf{G} = (V, E)$ be a directed graph with non-negative edge weights and two special nodes, namely, the source S and the sink T. The st-mincut algorithm partitions the set of vertices in V into two disjoint sets V_S and V_T, such that $S \in V_S$ and $T \in V_T$. The special nodes S and T correspond to sky and rest respectively.

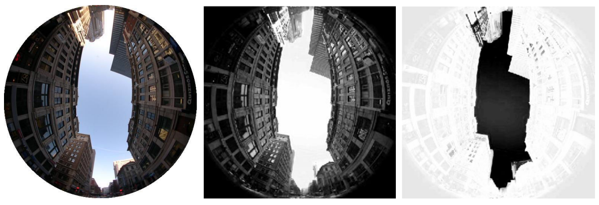

(a) (b) (c)

Figure 3. *(a) Original image. (b) Likelihood for the sky. (c) Likelihood for the rest of the image. The brighter values correspond to higher likelihood.*

We briefly introduce the energy function, whose parameters we are interested in learning. Let $x_i \in \mathbb{B} = \{0, 1\}$ where $i = 1, 2, ..n$ represent boolean random variables. In our problem, each of these variables could represent the boolean labels (sky and rest) of the pixels in the omnidirectional image. We use quadratic pseudo-boolean functions for representing our energy functions. These are nothing but energy functions of boolean variables that map a boolean vector to a real value and thus the name pseudo-boolean. Let θ denote the parameters in our energy function. The parameter vector θ consists of the unary terms $\theta_{i;a}$ and the pairwise terms $\theta_{ij;ab}$, where $i, j = 1, 2, .., n$ and $a, b \in \mathbb{B}$. These parameters are also referred to as unary and pairwise potentials. In contrast to many vision algorithms where these parameters are manually fixed, we compute them automatically. The unary parameter $\theta_{i;a}$ can be seen as a pseudo-boolean function $f : \mathbb{B} \to \mathbb{R}$ that gives the cost when $x_i = a$. Similarly, a pairwise parameter $\theta_{ij;ab}$ is a quadratic pseudo-boolean function $f : \mathbb{B}^2 \to \mathbb{R}$ that gives the cost when $x_i = a$ and $x_j = b$. The function mapping partitions to energies is then

$$\begin{aligned} E(\mathbf{x}|\theta) &= \sum_{i \in V} \{\theta_{i;0}(1 - x_i) + \theta_{i;1}(x_i)\} + \\ & \sum_{(i,j) \in E} \{\theta_{ij;00}(1 - x_i)(1 - x_j) \\ & +\theta_{ij;01}(1 - x_i)x_j + \theta_{ij;10}x_i(1 - x_j) \\ & +\theta_{ij;11}x_i x_j\} \qquad (1) \end{aligned}$$

Our goal is to learn the parameters (Θ) automatically for our problem. We optimize the discriminating power of the model by estimating parameters that maximize the difference between ground truth labellings and all other labellings of a small number of manually labeled examples. Our method is similar to the maximum-margin network learning method using graph cuts [22]; we generate “near-miss” labeling, then estimate a parameter vector that maximizes the margin separating true labellings from the near misses in a convex optimization. This is done repeatedly; the process converges to an optimal parameter vector in a small number of iterations. The theoretical properties of our algorithm will be published in a companion paper.

In our current implementation problem we obtain the unary likelihood for sky and rest using their color values. We first estimate a Gaussian model for the classes sky and rest and compute the mean and covariance using manually segmented ground truth images. For a new test image we obtain the unary likelihood by computing the Mahalanobis distance to the sky and non-sky classes as shown in figures 3(b) and (c). We will reformulate our energy function by decomposing the unary parameters $\theta_{i;a}$ as follows:

$$\theta_{i;a} = \theta_a^p + l_i \theta_a^l \qquad (2)$$

Once we have the likelihood cost l_i for every node the unary parameters θ_a^p and θ_a^l are dependent only the label a. Similarly we assume that the pairwise parameters $\theta_{ij;ab}$ are also independent of the associated nodes i and j and replace them by θ_{ab}. Due to the problem nature, we assume that the pairwise matrix is symmetric. i.e. $\theta_{01} = \theta_{10}$. We denote the new parameter vector which we want to learn as Θ.

$$\Theta = \left[\begin{array}{ccccccc} \theta_0^p & \theta_0^l & \theta_1^p & \theta_1^l & \theta_{00} & \theta_{01} & \theta_{11} \end{array} \right] \qquad (3)$$

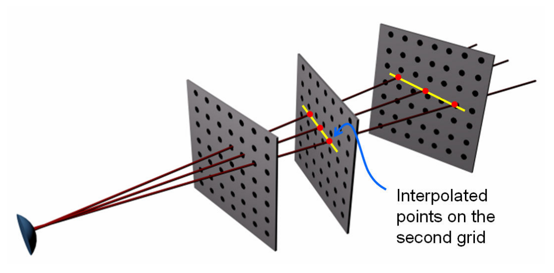
The parameter vector Θ is then estimated via the standard convex program for linear SVMs, using vectors of unary and pairwise statistics from true labellings and near miss labellings as positive and negative examples. In order to guarantee that the estimated model supports optimal inference, we augment the convex program with a constraint on the pairwise terms in the parameter vector that guarantees submodularity. In the binary case the constraint is simply

$$\theta_{00} + \theta_{11} \leq 2\theta_{01} \qquad (4)$$

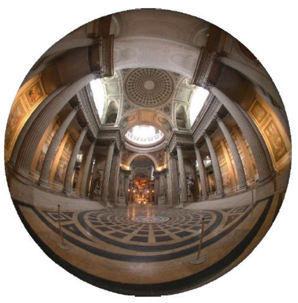
The submodularity condition is the discrete analogues of convex functions in continuous domains.

2.2. Calibration

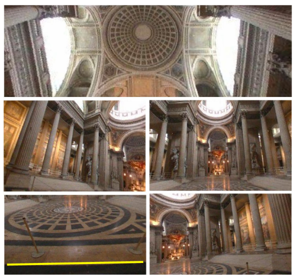
We briefly explain the main idea behind the generic calibration approach [17] (see Figure 4). Three images of a calibration grid are captured by the general camera that needs to be calibrated. The images are taken from unknown viewpoints. Every image pixel observes three 3D points in different calibration grids. These 3D points are obtained in three different coordinate systems. However, these points are collinear if they are expressed in the same coordinate



(a)



(b)



(c)

Figure 4. *(a) Generic calibration using three calibration grids. (b) and (c) show an original fisheye image and samples of distortion corrected regions, obtained using the calibration result.*

system. That constraint enables us to compute the motion between the views and eventually the projection rays for the image pixels.

Let $\mathbf{Q}$, $\mathbf{Q}'$ and $\mathbf{Q}''$ refer to the known points on the first, second and the third calibration grids respectively. Let the coordinate system of the first calibration grid be the reference frame. The pose of the second and the third calibration grids with respect to the first one be given by $(\mathsf{R}', \mathbf{t}')$ and $(\mathsf{R}", \mathbf{t}")$ respectively. Let O represent the unknown optical center. We stack the three points $\mathbf{O}$, $\mathbf{Q}$ and $\mathbf{Q}'$ (after transforming it to the reference frame) in the following matrix.

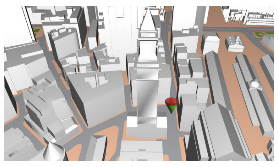
$$\begin{pmatrix} O_1 & Q_1 & R'_{11}Q'_1 + R'_{12}Q'_2 + R'_{13}Q'_3 + t'_1Q'_4 \\ O_2 & Q_2 & R'_{21}Q'_1 + R'_{22}Q'_2 + R'_{23}Q'_3 + t'_2Q'_4 \\ O_3 & Q_3 & R'_{31}Q'_1 + R'_{32}Q'_2 + R'_{33}Q'_3 + t'_3Q'_4 \\ O_4 & Q_4 & t'_4Q'_4 \end{pmatrix} \quad (5)$$

The collinearity constraint will force the determinant of any submatrix, whose size is size 3×3, of the above matrix to vanish. In other words, we obtain four constraints by removing one row at a time. A similar collinearity constraint can be obtained from the 3D points on first and third calibration grids. Using all these constraints we can extract the motion variables $(\mathsf{R}', \mathbf{t}')$ and $(\mathsf{R}", \mathbf{t}")$. See [15] for details on extracting the individual motion parameters. In practical calibration approaches, it is not possible to extract the projection rays for every pixel. Homography or bilinear interpolation can be used to compute the projection rays for other pixels.

2.3. Fisheye Synthesis

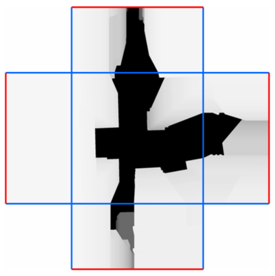
In figure 5 we show the various stages in synthesizing a fisheye image. OpenGL and DirectX pipelines do not support non-linear projections. In order to synthesize a fisheye image we first generate five binary perspective images, corresponding to the views in the cubemap shown in figure 5(c). As our algorithm uses 3D coarse models without any texture, we generate a cubemap simply by rendering a 3D model colored entirely in white; the resulting binary image is black in sky regions. We then use our calibrated ray-table to map the cubemap to a fisheye image, as shown in figure 5(d). The black region is the predicted shape of the sky. Some of the fisheye images synthesized at different places in a 3D model of Boston's financial center are shown in figure 5(e).

As the previous section indicated, our ray-calibrated view synthesis has the advantage that it does not "bake in" error that would arise from using a parametric lens model. In addition, a pixel shader program is implemented in GPU to generate these fisheye images at a very fast rate. This allows us to generate accurate fisheye images on the fly; there is no need to store images in a large database.

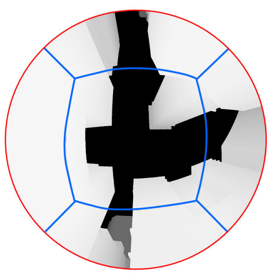


(a)

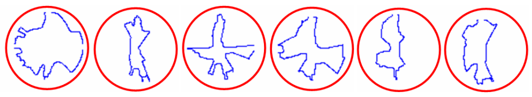
(b)



(c)



(d)



(e)

Figure 5. *Our algorithm to construct a fisheye image. (a) and (b) show the 3D model and aerial image of an urban scene where we synthesize a fisheye image. (c) The cubemap generated from a given point in the 3D model. Note that in practice, we generate binary images; the slight shading here is just for visualization. (d) Fisheye image created from the cubemap. (e) Examples of skylines extracted from the fisheye images.*

2.4. Skyline Matching

Using the graph cut sky detection algorithm we segment the sky in fisheye images and obtain the skyline as shown in figure 6(a). The predicted skylines corresponding to different locations in the 3D model are obtained using the method suggested in the previous section. Since the real-image segmentation and the predicted image are both high quality, a simple chamfer distance suffices to score the skyline match. During the chamfer matching, we vary the pose parameters of our virtual fisheye camera and obtain new skylines at various locations in the 3D model. This is done using a multi-resolution strategy: we start with synthesizing and matching fisheye images which are distributed at intervals of 5 meters and refine the location until we have achieved a positive match that is precise up to several centimeters. We show some matching results in figure 8.

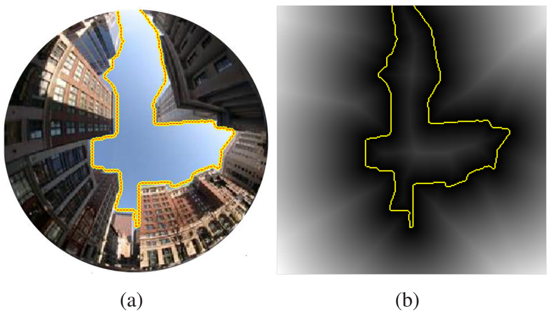

(a) (b)

Figure 6. *(a) The skyline detected in a real fisheye image. (b) The chamfer distance map of the skyline of (a), with the best match of synthesized skylines superimposed on it.*

3. Experiments

Our real experiments were all carried out in the financial district of Boston, which is shown in figure 9(b). In all our real experiments we used a Nikon Coolpix E8 Fisheye lens with a field of view of 183° to capture about 300 images. We implemented all our programs using C++ and OpenGL and parallelized the GPU and CPU operations for enhanced performance. Also, we use a simple multi-resolution strategy as explained in section 2.4.

Table 1 gives the values of the parameters learnt with the approach described in section 2.1. Figure 7 shows a few results for the first step of the system, detecting sky regions in fisheye images. In the following, we comment on matching and localization performance.

In all of our test images, the match converged to solutions with very low chamfer scores of about 2 pixels (average distance of points on synthesized skylines, to skylines extracted in real images).

In order to investigate how uniquely the skylines "fingerprint" their locales, are we selected 16 random locations on the financial district. For each of these locations we generated 27 points by translations along X, Y and Z directions, separated each by 25cm. For the total of 432 images, we obtained a confusion matrix by computing the chamfer distance for all possible pairs (cf. figure 9(d)). We observed that for every point, the first 27 correct matches corresponded directly to its nearby locations. We conducted the same experiment in Boston's North End where all the buildings are residential and short as shown in figure 9(c). Our experiments demonstrated that the skyline matching algorithm can be robust even if the buildings are short. Indeed, the confusion matrix of the North End locations (cf. figure 9(e)) indicates even better location fingerprinting than that of the financial district.

θ_0^p	1
θ_0^l	-1
θ_1^p	-0.03708
θ_1^l	-0.0935
θ_{00}	-0.5638
θ_{01}	0.2487
θ_{11}	0.3151
Accuracy (%)	94.2

Table 1. The learned parameters of the energy function shown in equation (1). Note that the pairwise parameters satisfy the submodularity conditions, as imposed in our learning framework. The accuracy figure in the last row is the percentage of correctly classified sky pixels in the training set.

In order to assess the benefit of omni-directional cameras, we also conducted the same experiment of computing the confusion matrix with a perspective model in the financial district. We used a model with a field of view of 90° and synthesized images at exactly the same locations in the financial district (tall buildings) that were chosen for the fisheye experiment. The confusion matrix (cf. figure 9(f)) indicates very clearly the large number of ambiguities between different skylines.

4. Conclusion

Our main goal in this work is to assess the suitability of skylines and omnidirectional image cameras for accurate geospatial localization. Our results clearly demonstrate that it is possible to outperform GPS measurements in urban scenes, which are known to be extremely problematic for commercial GPS units.

Nevertheless, we foresee potential problems due to inaccurate 3D models and loss of edges due to too much or too little sunlight or certain weather conditions. However, we expect to resolve ambiguities in such cases using other vision algorithms: interest point matching/tracking between consecutive frames, structure from motion algorithms, Kalman filtering, and other priors based on street

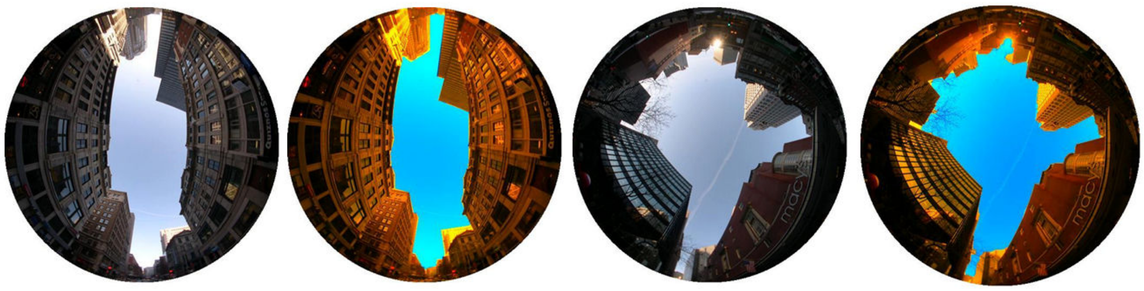


Figure 7. *Sky detection results: Original and segmented fisheye images are shown.*

map information. It is important to note that the current GPS systems have been improved and robustified for several years by addressing various complicated issues that even includes theory of relativity.

In a few years, we believe that it is possible to build an inexpensive, robust and accurate vision based GPS.

Acknowledgments: The authors would like to thank Dr. Jay Thornton for useful discussions in 2D to 3D registration and car navigation. Srikumar Ramalingam would like to thank Prof. Philip Torr for the various discussions in graph cuts and geometric labeling problems.

References

[1] T. Brodsky, C. Fermüller, and Y. Aloimonos. Directions of motion fields are hardly ever ambiquous. In *ECCV*, 1996.

[2] N. Cornelis, K. Cornelis, and L. V. Gool. Fast compact city modeling for navigation pre-visualization. In *CVPR*, 2006.

[3] A. Dunne, J. Mallon, and P. Whelan. A comparison of new generic camera calibration with the standard parametric approach. *MVA*, 2007.

[4] D. Gallup, J.-M. Frahm, P. Mordohai, Q. Yang, and M. Pollefeys. Real-time plane-sweeping stereo with multiple sweeping directions. In *CVPR*, 2007.

[5] M. Grossberg and S. Nayar. The raxel imaging model and ray-based calibration. *IJCV*, 61(2):119–137, 2005.

[6] J. Hays and A. Efros. Im2gps: estimating geographic images from single images. In *CVPR*, 2008.

[7] D. Hoiem, A. A. Efros, and M. Hebert. Recovering surface layout from an image. *IJCV*, 75(1):151–172, 2007.

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

[9] K.Daniilidis and H.H.Nagel. The coupling of rotation and translation in motion estimation of planar surfaces. In *CVPR*, 1993.

[10] O. Koch and S. Teller. Wide-area egomotion estimation from known 3d structure. In *CVPR*, 2007.

[11] J. Meguro, T. Murata, H. Nishimura, y. Amano, T. Hasizume, and J. Takiguchi. Development of positioning technique using omni-directional ir camera and aerial survey data. In *Advanced Intelligent Mechatronics*, 2007.

[12] J. Neumann, C. Fermüller, and Y. Aloimonos. Polydioptric camera design and 3d motion estimation. In *CVPR*, volume 2, pages 294–301, 2003.

[13] T. Pajdla. Stereo with oblique cameras. *IJCV*, 47(1), 2002.

[14] R. Pless. Using many cameras as one. In *CVPR*, pages 587–594, 2003.

[15] S. Ramalingam. Generic imaging models: Calibration and 3d reconstruction algorithms. In *INRIA Rhone Alpes, Grenoble, France*, 2006.

[16] S. Ramalingam, P. Kohli, K. Alahari, and P. Torr. Exact inference in multi-label crfs with higher order cliques. In *CVPR*, 2008.

[17] S. Ramalingam, P. Sturm, and S. Lodha. Towards complete generic camera calibration. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[18] D. Robertson and R. Cipolla. An image-based system for urban navigation. In *BMVC*, 2004.

[19] F. Stein and G. Medioni. Map-based localization using the panoramic horizon. In *IEEE Transactions on Robotics and Automation*, 1995.

[20] R. Swaminathan, M. Grossberg, and S. Nayar. A perspective on distortions. In *CVPR*, volume 2, pages 594–601, 2003.

[21] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. F. Tappen, and C. Rother. A comparative study of energy minimization methods for Markov random fields. In *ECCV*, volume 2, pages 16–29, 2006.

[22] M. Szummer, P. Kohli, and D. Hoiem. Learning crfs using graph cuts. In *ECCV*, 2008.

[23] S. Teller, M. Antone, Z. Bodnar, M. Bosse, S. Coorg, M. Jethwa, and N. Master. Calibrated, registered images of an extended urban area. *IJCV*, 2003.

[24] T. Yeh, K. Tollmar, and T. Darrell. Searching the web with mobile images for location recognition. In *CVPR*, 2004.

[25] W. Zhang and J. Kosecka. Image based localization in urban environments. In *3DPVT*, 2006.

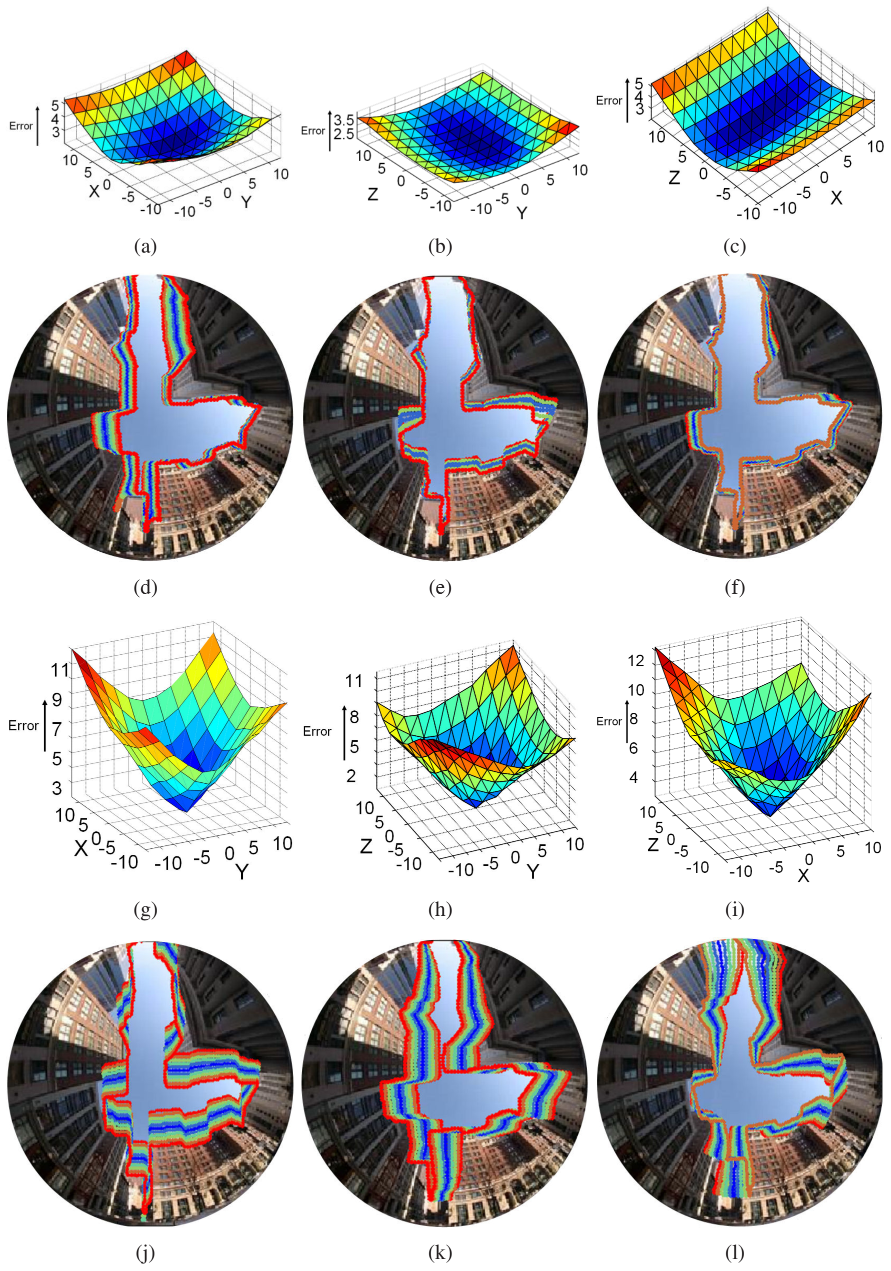
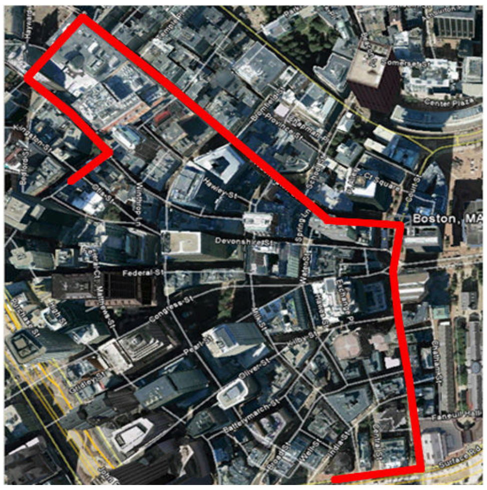
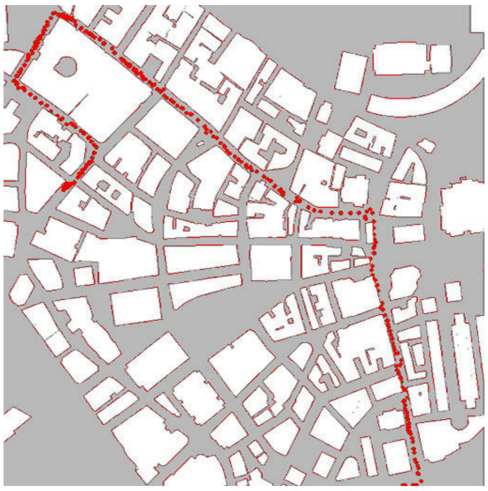


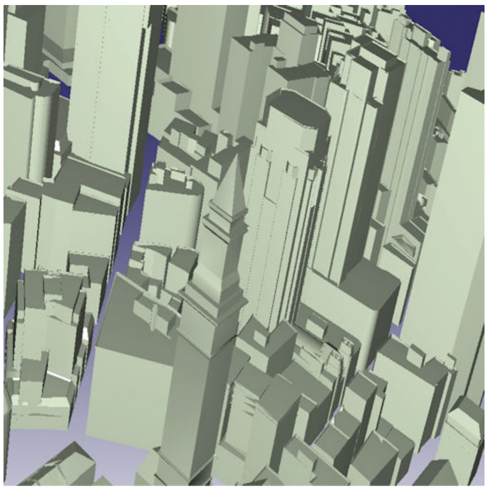
Figure 8. *Chamfer distance between the skylines from real and synthetic images, for changes in translation and rotation parameters in (a-c) and (g-i) respectively. The change in shape of the skylines is also studied for small perturbations in translation (X in (d),Y in (e), Z in (f)) and rotation parameters ($\mathbf{R}_x$ in (j), $\mathbf{R}_y$ in (k) and $\mathbf{R}_z$ in (l)). We use color-coding to display the change in shape of the skylines. The skylines drawn in blue, green and red show the initial, intermediate and final shapes. One unit in the model space corresponds to 25cms for translation and 1 degree for rotation.* [Best viewed in color]



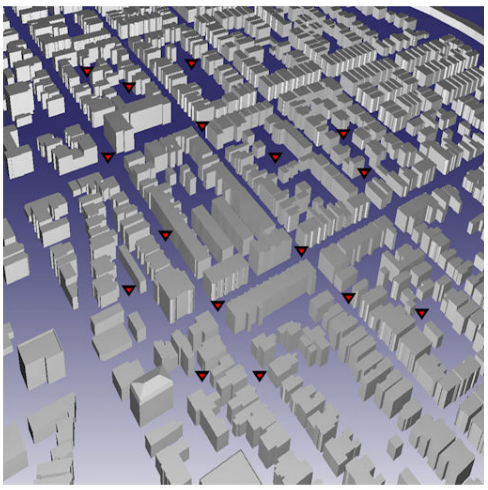
(a)



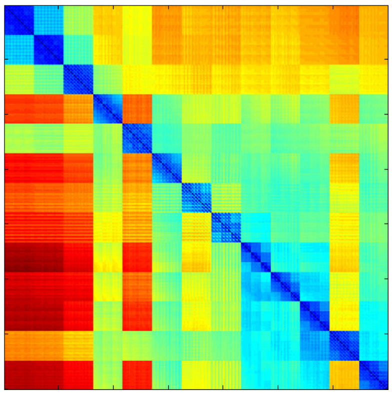
(b)



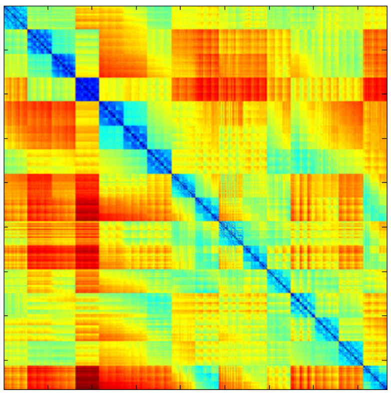
(c)



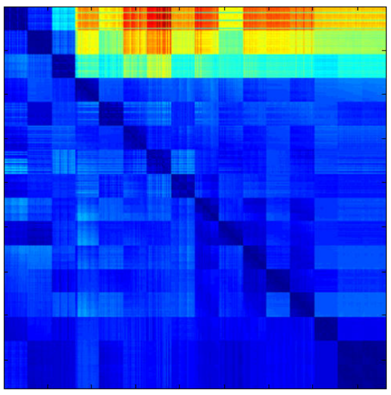
(d)



(e)



(f)



(g)

Figure 9. *(a) Aerial image of the financial district of Boston showing the trajectory of our data collection. (b) The computed locations are shown in the aerial image. (c) 3D model of the financial district, which mainly consists of tall buildings (d) 3D model of Boston's North End, where houses are mostly of the same height and residential. Confusion matrices in (e) and (f) are computed using 432 fisheye images synthesized at various locations in (c) and (d) respectively. (g) Confusion matrix computed for synthesized perspective views with a field of view of 90°* at exactly the same locations where the one shown in (d) was computed.* `[Best viewed in color]`