Single View Pose Estimation of Mobile Devices in Urban Environments

Aaron Hallquist
Avideh Zakhor, Ed.

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2012-122
http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-122.html

May 25, 2012

Acknowledgement

We acknowledge the image and 3D dataset provided to us by Earthmine, as well as the help received from Jimmy Wang and John Ristevski.

Single View Pose Estimation of Mobile Devices in Urban Environments

Aaron Hallquist

EECS Department

University of California, Berkeley

aaronh@eecs.berkeley.edu

ABSTRACT

Pose estimation of mobile devices is useful in a wide variety of applications, including augmented reality and geo-tagging. Even though most of today's cell phones are equipped with sensors such as GPS, accelerometers, and gyros, the pose estimated via these is often inaccurate. This is particularly true in urban environments where tall buildings block satellite view for GPS, and distortions in Earth's magnetic field from power lines adversely affect compass readings. In this thesis, we describe an image based localization algorithm for estimating the pose of cell phones in urban environments. This is motivated by the fact that most of today's cell phones are equipped with cameras whose imagery can be matched against an image database for localization purposes. We use the sensors available on the cell phone, an image taken with its camera, and a database of geo-tagged images and associated 3D depth to estimate the position and orientation of the cell phone. Our proposed approach consists of two steps. The first step, based on existing work, matches the query image from the cell phone against the image database in order to retrieve a database image of the same scene. The second step, which is the focus of this thesis, begins by using pitch and roll estimates from the cell phone to recover plane normals and yaw via vanishing points. These are then used to solve for a constrained homography matrix for the detected planes to recover translation via matching point feature correspondences between the query and database images. We characterize the performance of this approach for a

dataset in Oakland, California and show that for a query set of 92 images, our computed yaw is within 10 degrees for 96% of queries as compared to 26% for the compass on the cell phone; similarly, our estimated position is within 10 meters for 92% of queries as compared to 31% for GPS on the cell phone.

1. INTRODUCTION

Position and orientation, or pose of a mobile device is useful in many online applications such as gaming, entertainment and augmented reality as well as offline applications such as geo-tagging. In order to determine pose, most modern smart-phones are equipped with full sensor packages, including an accelerometer, gyros, GPS, and a compass. These tools are used in applications such as Yelp Monocle [11], which uses the phone's GPS and compass to overlay nearby entities such as shops and restaurants on the viewfinder. Unfortunately, in urban environments, tall buildings block satellite view and introduce multi-path effects, while power lines and trains add significant magnetic interference, making both GPS and compass readings error prone. Furthermore, the barometer, the only source of altitude information, is extremely unreliable. Since most of today's mobile devices are equipped with camera sensors, it is conceivable to use camera imagery to compensate for the above deficiencies in recovering full position and orientation of the mobile device.

In this thesis, we propose a sensor fusion approach for determining the pose of a mobile device equipped with typical sensors in a smart-phone. Our approach uses readings from the GPS, accelerometer, and compass to guide a two-step image based localization system. The first step, discussed extensively in [12], uses a city-scale image database to find an image that matches the scene in the query image captured in the cell phone's viewfinder. This is done by searching a local area of the database using the cell phone's coarse position reading either via GPS or cell tower triangulation, despite the fact that they can be off by hundreds of meters. It then matches SIFT features [6] extracted from the query image to those extracted from the database. The approach in [12] achieves 90% image retrieval accuracy for datasets in Berkeley and Oakland, California, made of tens of thousands of images.

The second step, which is the focus of this thesis, uses the matching database image recovered in the first step and its associated pose in the global world coordinates to recover the pose of the query image by taking into account the input from the accelerometer and compass on the mobile device. Our city-scale database contains three dimensional (3D) information, including the full pose of each database image and 3D point clouds corresponding to the depth map of each image. For certain regions such as the city of Oakland, California, our image database also contains a plane map associated with each image, which fits clusters in the 3D point clouds to large planes such as building faces and the ground. Such databases are becoming increasingly more common as witnessed in Street View by Google and Bing's Streetside by Microsoft.

Our proposed algorithm uses pitch and roll from the cell phone within a vanishing point framework to detect planes corresponding to building faces in both the query and the matched database image. The building faces are then aligned between the query and the matched database image in order to solve for the cell phone's yaw. Using the full orientation and the planes detected, we then estimate a homography matrix using point feature matches between the query and database image. The typical algorithm for such a homography solution is modified to use the 3D information from our database, allowing us to compute the full translation vector including the scale factor, rather than the direction of translation only.

In Section 2, we review related work in this area and the connections to our proposed method. We go over our approach in detail in Section 3. Section 4 shows the results applied to datasets in Berkeley and Oakland California; we present our conclusions and discuss future work in Section 5.

2. RELATED WORK

A number of papers deal with pose estimation for urban scenes, with a wide variety of motivations and tools at their disposal. The two major related categories are (a) those solving the single view pose recovery problem, but not necessarily targeting a mobile device, and (b) those targeting mobile platforms, but tracking pose over a video sequence rather than single view pose estimation.

In the former category, the most related work is by Zhang and Kosecka [13], where pose is recovered for a single query image using a similar database matching step and homography estimation. A key

difference is that Zhang and Kosecka work with a database containing only the absolute pose of images, while our database contains 3D point cloud associated with imagery as well as the absolute pose of the imagery. In order to compute the scale of translation, [13] must retrieve two images from the database and use the absolute distance between them to triangulate the query position. A similar triangulation approach is taken in [5], estimating the rotation and translation between three views, two of which have known locations. However, they use an essential matrix and also optimize for speed on a mobile device, both of which sacrifice accuracy. Using two database views is potentially unreliable for city-scale databases because it might not be possible to retrieve two matching images in the database. Since we have access to 3D depth information in the database imagery, we opt to use it to compute translation scale without the use of multiple database matches.

Rather than using point features to recover pose, other papers have experimented with matching line features [3] or 2D patterns from a 3D database [7]. The latter is particularly interesting because it takes advantage of the fact that many urban scenes contain repetitive patterns, even though for point features such repetitions are problematic [7].

The majority of work targeting mobile platforms focuses on the tracking problem attempting to create an online algorithm to retrieve images. This is demonstrated in [8], which deals with the accuracy of retrieving database images within X meters of the query image. Others solve the tracking problem, but provide no quantitative performance [9,10]. In contrast, we focus on recovering the pose for a scene from a single camera image captured by a mobile device and evaluate the system performance relative to ground truth. In this thesis we do not deal with real time aspects of our approach leaving it as future work.

3. SYSTEM OVERVIEW

As described in the introduction, our proposed approach estimates the pose of the cell phone by retrieving a matching image from a city-scale database and combining the camera imagery with cell phone sensors.

We assume image retrieval has already been performed and that the input to our algorithm is a correct matching database image. In addition, we assume a specific type of query image taken, one which (a) is

mostly upright, i.e. with small roll close to zero, (b) views an urban scene from a reasonable distance and reasonable angles, generally across the street rather than down the street, (c) is taken at roughly human height e.g. 1 to 3 meters above ground, and (d) stays steady for the duration of the capture. Each of these assumptions play their roles in various parts of our algorithm.

The database we work with contains the following information:

- A dense set of black and white image panoramas covering the city.
- The position and orientation information for each panorama.
- A depth map for each panorama.
- (Optional) A plane map for each panorama, representing a set of fit planes and their association with panorama pixels. Plane equations however are not known.
- A set of six "street-facing" rectilinear images, depth and plane maps are extracted from each panorama.

The rectilinear images extracted from each panorama are used for image retrieval and pose recovery, and their camera parameters are assumed to be known. Similarly, the camera parameters extracted from the cell phone are assumed to be known. We perform no camera calibration and work in the 3D world rather than with pixel coordinates. That is, for every pixel location (p_x, p_y) in an image, we pre-multiply by the inverse of the camera matrix K to attain its in-world ray w, as in

$$w = K^{-1}[p_x, p_y, 1]^T$$

The skew parameter for K is assumed to be $\gamma = 0$, and its aspect ratio is assumed to be $\alpha = 1$. These conventions and assumptions about image coordinates and camera parameters apply to both the query and database images.

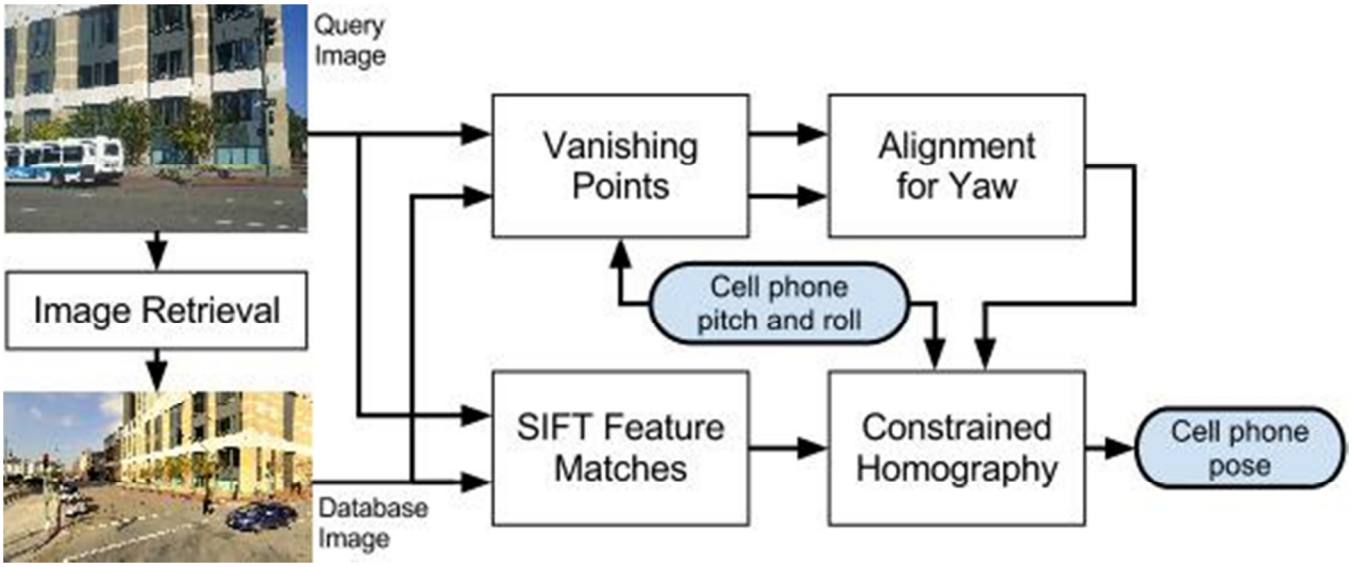


Figure 1: Layout of the pose estimation system.

There are two main parts to our approach: vanishing point alignment and constrained homography estimation via point feature correspondences. In the system layout in Figure 1, this is visually shown by separating them into a top and bottom flow. The respective goals of the two subsystems are as follows. Using vanishing points we find plane normals in the scenes and align them between the query and database image in order to find the yaw orientation of the query view. Using feature matches between the query and database image, we solve for the query position by estimating a constrained homography transformation. Vanishing points and their application are described in Section 3.2, while point features and their application are described in Sections 3.3 and 3.4. At various stages in each

(a) (b)

Figure 2: (a) Sample query image with (b) its retrieved database image. This query will be used to demonstrate the system outputs at various stages of the algorithm.

subsection, we present the system output on the sample query and database image shown in Figure 2. This is intended to help illustrate the system process visually and with a concrete example.

Note that we do not explicitly solve for pitch and roll. Rather, assuming a steady cell phone, they are estimated by using the direction of gravity as measured by a low pass filter of the accelerometer.

3.1 Notation

We use multiple coordinate systems and types of variables throughout the thesis. We refer to the query image taken from the cell phone as Q and to its matched image retrieved from the database as D. Furthermore we distinguish between a "full 3D vector" and a "3D ray": a full 3D vector has a known scale and 3 independent parameters, while a 3D ray is always normalized and has an unknown scale, i.e. 2 independent parameters. In an effort to maintain clarity, the following notation is used:

- Capital letters denote matrices, sets, and objects while lowercase letters denote vectors and scalars:
    - Hat on top to denote a normalized vector or a 3D ray, e.g. $\hat{t}$ as the direction of translation.
    - Vector symbol to denote a full 3D vector with scale, e.g. $\vec{t}$ to denote 3D translation.
    - Nothing on top of lower-case letters to denote a scalar.
    - For capital letters, scripts denote a set, bolds denote a matrix, normal capitals denote an object.
- When coordinate frames are specified, it is done with superscripts and subscripts, as in the examples below:
    - $\vec{x}^d$ is a vector in the database coordinate frame.
    - $\boldsymbol{R}_d^q$ represents the rotation matrix which maps the database coordinate system to the query coordinate system, e.g. $\vec{x}^q = \boldsymbol{R}_d^q \vec{x}^d$.

For clarification, a full list of all variables is presented in Table 1.

Table 1: List of all variables present in this thesis and a short description of each.

c	Confidence parameter for vanishing points
conf_align	Confidence computed for vanishing point alignment
conf_hom	Confidence computed for homography estimation
e	Error metric used in homography computation
f_{inlier}	Minimum number of feature inliers for valid homography
g	Function assigning lines to seeds
iter	Number of iterations required by RANSAC loop
length	Lengths of the lines L in the image plane
l	Minimum number of feature correspondences needed to compute solution set; used to compute iter
$\hat{n}$	Normal vector for homography estimation
p	Distance from center of camera for D to the scene's plane
r	Scale factor in homography estimation $\left\|\vec{t}\right\|/p$
$\hat{s}$	Seeds for computing vanishing points
$\hat{t}, \vec{t}$	Translation for homography estimation
$\hat{v}$	Vanishing points found in Q and D
$\Delta vert$	Vertical component of estimated $\vec{t}$ in the world frame
w	Weight for feature correspondence $\mathcal{F}$
$\hat{x}$	3D ray of point feature in D

$\lvert\vec{x}\rvert$	Distance of database point feature from D
$\hat{y}$	3D ray of point feature in Q
Δyaw	Estimated compass error
$\vec{z}$	Homography-transformed $\hat{x}$ used in error metric e
α	Importance factor for depth error constraint
ε	Error threshold for inliers in homography estimation
θ	Seed angle error for line L and seed $\hat{s}$
ϕ	Plane yaws computed from vanishing points or from database planes
ρ	Desired probability of solution found from RANSAC; used to compute the expected iterations iter
ω	Fraction of all correspondences which are inliers; used to compute iter
D	Database image (object)
$\boldsymbol{H}$	Canonical homography matrix
$\boldsymbol{I}$	Identity matrix
$\boldsymbol{J}$	Constrained homography matrix
L	Lines from images (object)
M	Line connecting midpoint of line L to a seed $\hat{s}$ (object)
Q	Query image (object)
$\boldsymbol{R}$	Rotation matrix
$\mathcal{A}$	Alignment sets for planar alignment

$\mathcal{C}$	Set of feature correspondences $\mathcal{F}$
$\mathcal{F}$	Feature correspondence set consisting of $(\hat{x}, \hat{y}, \lvert\vec{x}\rvert, w)$
$\mathcal{L}$	Set of contributing vanishing lines
$\mathcal{M}$	Inlier set of feature correspondences $\mathcal{F}$ from RANSAC
$\mathcal{S}$	Solution set from RANSAC consisting of $(r, p, \hat{t}, \hat{n})$

3.2 Plane Estimation and Yaw Recovery

Since urban scenes are dominated by building faces, we assume that many point features lie in a plane and compute the cell phone pose by solving for a homography transformation from the database image D to the query image Q. However, the simultaneous solution of position, orientation, and plane normal is ill-conditioned in a noisy environment with respect to the minimization of reprojection error associated with a homography matrix [4]. To address this, we strive to solve for as few of these parameters as possible in a given homography estimation. This section describes the way we estimate the orientation and plane normals before computing a homography.

To obtain the orientation of Q, we use the cell phone accelerometer and magnetometer to extract yaw, pitch, and roll. This extraction is trivial and relies on the direction of gravity from the accelerometer to solve for pitch, roll, and the direction of north from the compass to solve for yaw. To ensure accuracy in these measurements, the accelerometer requires the phone to be steady during image capture, and the compass requires an environment free from magnetic interference. The first condition can be satisfied by holding the camera steady, which is also required for capturing crisp images. The second condition is impossible to ensure in urban environments, as sources of magnetic interference are all around, from power lines to trains. Therefore, we anticipate the reported yaw by the mobile device, yaw_{cell} ,to be error prone.

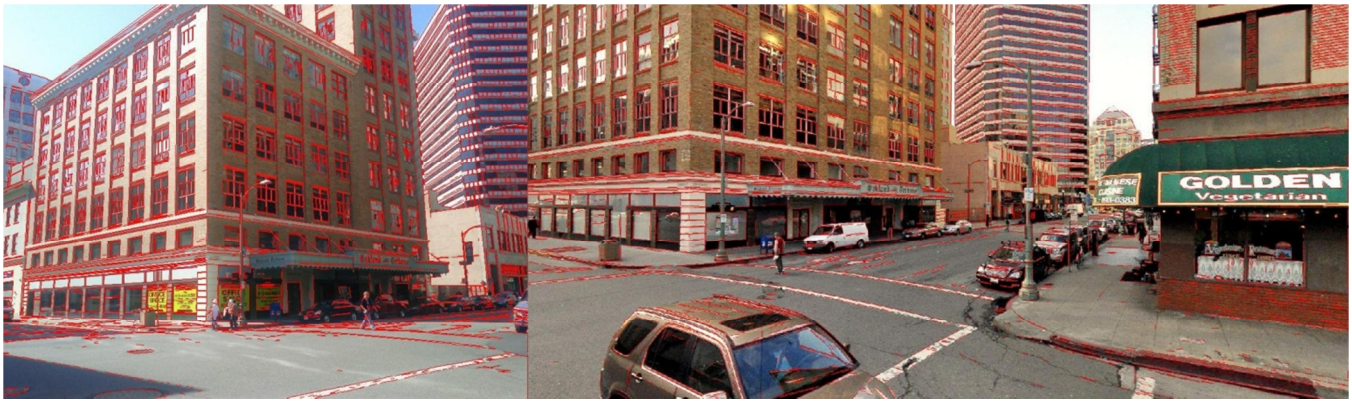
In order to recover a more accurate yaw value for the cell phone and also to compute plane normals for our scene, we turn to vanishing points. Under the assumption that our scenes have planar and

upright building faces, we obtain the plane normals of these building faces in Section 3.2.1 by computing horizontal vanishing points for both Q and D. This provides a set of planes in each image which we then align in Section 3.2.2 to compute a more accurate yaw value for the cell phone.

3.2.1 Vanishing Point Extraction

In this section we describe our approach to computing the horizontal vanishing points for Q and D. We extract lines using the algorithm proposed in [2]. This is shown on our example query in Figure 3. Since we only need horizontal vanishing points and we assume our images are near upright, we remove lines within 10 degrees of vertical on the image plane. After multiplying by the inverse camera matrix, we obtain a set of lines $\mathcal{L} = \{L_i\}$ with associated lengths, $\text{length}(L_i)$, in the image plane.

In order to organize the detected lines into vanishing points, we create "seeds" representing potential vanishing points to which we assign vanishing lines. The seeds sample the complete set of all vanishing points perpendicular to gravity, are spaced 3 degrees apart, and are computed using yaw, pitch, and roll for the image. That is, we create a set of sixty potential vanishing point seeds $\{\hat{s}_k\}$ which have no vertical component when represented in the world frame, with yaws ranging from 0 degrees to 177 degrees, and we use the image orientation to transform these seeds to the image frame.



(a) (b)

Figure 3: The set of lines extracted, marked in red, for (a) the example query image and (b) its corresponding database image.

From this dense set of seeds, we find a subset which lie near many of the lines detected in the image, indicating the possibility that the seed is a vanishing point. To do this we find the maxima of the

function $g(\hat{s}_k)$ constructed by assigning image lines to their nearest seeds. Specifically, we define the weight for the kth seed to be:

$$g(\hat{s}_k) = \frac{\sum_{i \in \mathcal{L}_k} \text{length}\,(L_i)}{\sum_{j \in \mathcal{L}} \text{length}(L_j)}$$

where $\mathcal{L}_k$ is the set of indices i corresponding to every line L_i which is assigned to vanishing point $\hat{s}_k$. The assignment process is as follows. As shown in Figure 4, for every line L_i, we compute the seed angle error θ_{ik} for each $\hat{s}_k$. This is defined as the angle in the image plane between the lines L_i and M_{ik}, where M_{ik} is the line connecting the midpoint of L_i to s_k. The index i is then added to the three sets $\mathcal{L}_k$ corresponding to the three values of k minimizing θ_{ik}. Assigning weights to three seeds rather than one, smoothes the function $g(\hat{s}_k)$ and accounts for the fact that the true vanishing points may not be perpendicular to gravity.

Figure 4: An example image line, vanishing point seed, and the pair's associated seed angle error.

We select a subset of the seeds $\{\hat{s}_k\}$, satisfying these two conditions:

$$g(\hat{s}_j) \geq g(\hat{s}_k) \qquad j-3 \leq k \leq j+3 \tag{1a}$$

$$g(\hat{s}_j) \geq 1.5 * g_{unif} \quad \text{where} \quad g_{unif} = 3/60 \tag{1b}$$

as a set of vanishing points $\{\hat{v}\}$. g_{unif} represents the weight assigned to each seed if all seeds were to receive equal weight, i.e. 3/60. The first condition is a local maximum condition which requires the vanishing point to have a larger weight than all those neighboring seeds within 10 degrees. Note that in our case, $\hat{s}_{j-1}$ and $\hat{s}_{j+1}$ are the seeds with yaw orientations at a -3 and 3 degree difference from $\hat{s}_j$. The second condition requires the weight of the vanishing point to be large enough so that it cannot be an accidental local maximum in a region of sparse contributions. The graph of this function and its chosen seeds for both the query and database image example is in Figure 5. The minimum cutoff line of $1.5 * g_{unif}$ is plotted, and the chosen seeds are marked. Since the two functions are similar to each other with an offset, this plot hints at the topic of Section 3.2.2, which involves aligning the chosen seeds to improve upon the error prone yaw from our cell phone.

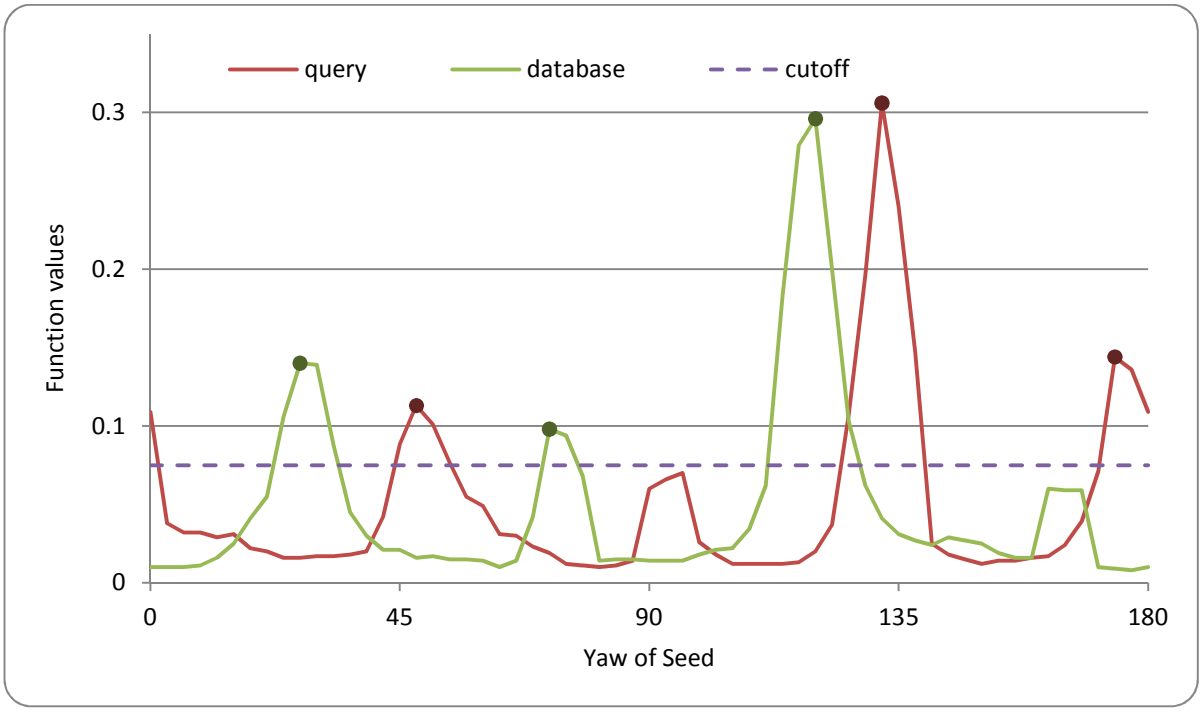


Figure 5: Plot of $\boldsymbol{g}(\hat{\boldsymbol{s}}_{\boldsymbol{k}})$ for both the query and database image from our example query. The cutoff value is 1.5$\boldsymbol{g}_{\boldsymbol{unif}}$. Points which satisfy the conditions in Equation 1 are candidate vanishing points and are marked with dark circles in the plot.*

In practice, not all vanishing points are perfectly perpendicular to gravity. In fact, in the most extreme case of the steepest street in the world, the angle between gravity and the hill's vanishing point is only

70 degrees. In our datasets however, the angle between gravity and vanishing points is always greater than 85 degrees; therefore, we account for that small difference by applying a guided matching algorithm [4] to the previously computed set of vanishing points obtained from seeds found via in Equation 1. This steers each vanishing point toward a nearby location, not necessarily perpendicular to gravity, which simultaneously maximizes the number of contributing vanishing lines and minimizes the vanishing point deviation from all contributing vanishing lines.

The output of guided matching yields a refined set of vanishing points in the world frame and a set of contributing lines $\mathcal{L}_k$. In practice, we output between one and four horizontal vanishing points per image. For the kth vanishing point $\hat{v}_k$, we compute a confidence value based on the total length of all contributing lines. This is defined as

$$c_k = \frac{\sum_{i \in \mathcal{L}_k} \text{length}(L_i)}{\sum_{j \in \mathcal{L}} \text{length}(L_j)} \tag{2}$$

where $\mathcal{L}$ is the set of all lines detected in the image. This confidence is used in Section 3.2.2 for alignment.

3.2.2 Planar Alignment to Compute Yaw

In this section we describe the way vanishing points computed for Q and D are used to extract planes for the scene and to recover yaw orientation for Q. To retrieve planes from the vanishing points, we note that the vanishing points represent the directions of a scene's streets and building faces. Using the assumption that all building faces are upright, we compute the plane normals from the vanishing points by taking the cross product of each vanishing point with the gravity vector. Since the sign of the resulting plane normal cannot be determined by the vanishing point, we use scene geometry to ensure the dot product between the plane normal and the camera vector[1] is negative, thus specifying the sign of the plane normal. With the determined sign and the upright plane constraint, we reduce each plane

[1] The camera vector is defined as the world vector for the direction the viewfinder of the cell phone is facing.

normal to an associated plane yaw between 0 and 360 degrees, indicating the direction each plane faces.

So far, we have estimated vanishing points, plane yaws, and their associated confidences as computed in Equation 2 for Q and D. We compute plane yaws for D using vanishing points regardless of whether or not we have access to database planes. There are three reasons for this: first, we wish our algorithm to be applicable to situations where database planes are not available; second, the confidence parameter is computed as a part of vanishing points and is otherwise not available from the database; third, since in the later steps we align planes between Q and D, it is advantageous to have computed both sets of planes using the same approach. We label the vanishing points for Q and D as $\hat{v}^Q$ and $\hat{v}^D$, respectively, where the superscripts here denote that the vanishing point source is from D and Q, not that they are represented in the query and database coordinate frames. Similarly the plane yaws are labeled ϕ^Q and ϕ^D, while the confidences are labeled c^Q and c^D.

Note that the plane yaws and vanishing points for D are considerably more accurate than those for Q since the latter are based on the yaw reported by the cell phone. It is easy to show that, for a known pitch and roll, and a yaw error of $\Delta yaw = yaw_{actual} - yaw_{cell}$, the transformations to be applied to the query vanishing points and plane yaws to correct for this error are given by:

$$\bar{v}^Q = \boldsymbol{R}(\Delta yaw)\ \hat{v}^Q \tag{3a}$$

$$\bar{\phi}^Q = \phi^Q + \Delta yaw \tag{3b}$$

where $\boldsymbol{R}(\Delta yaw)$ is the rotation matrix associated with a pitch and roll of zero and a yaw of Δyaw. The bars mark a corrected vanishing point or plane yaw after applying an error Δyaw.

Our objective is to find the optimal alignment of the vanishing points from Q and D in the world frame in order to estimate the yaw error from the cell phone. To proceed, we exhaustively go through every alignment pair (i, j) corresponding to aligning the ith plane yaw from Q, namely ϕ_i^Q, and the jth plane yaw from D, namely ϕ_j^D. This is equivalent to making the assertion $\bar{\phi}_i^Q = \phi_j^D$, which in turn represents a cell phone yaw error $\Delta yaw_{ij} = \phi_j^D - \phi_i^Q$. The reported yaw from the compass guides this search by

discarding all pairs (i,j) with resulting compass error $\left|\Delta yaw_{ij}\right| > 50$ degrees, in order to avoid aligning the wrong planes.

For each candidate Δyaw_{ij} within the above range, we compute an alignment confidence representing how strong the alignment is. To do this for a given (i,j), we correct the query vanishing points by applying the rotation $\boldsymbol{R}\left(\Delta yaw_{ij}\right)$ as in Equation 3(a) and search for a larger set of aligned pairs, defined as

$$\mathcal{A}_{ij} = \left\{ (m,k) \mid \measuredangle\left[\boldsymbol{R}\left(\Delta yaw_{ij}\right)\hat{v}_m^Q , \hat{v}_k^D \right] < 5^\circ \right\}$$

This alignment set contains every pair of vanishing points in Q and D which is aligned when correcting query vanishing points by a yaw error of Δyaw_{ij}. With the alignment set for each pair (i,j), we compute a confidence for the alignment, conf_align_{ij}, which resembles a dot product over every alignment, using the confidence of the aligned vanishing points:

$$\text{conf\_align}_{ij} = \sum\nolimits_{(m,k)\in\mathcal{A}_{ij}} c_m^Q \, c_k^D$$

where c_m^Q and c_k^D denote the confidence values of the mth query and kth database vanishing points as shown in Equation 2. The best alignment pair (i,j) is declared to be the pair which maximizes this confidence parameter conf_align_{ij}. With this alignment pair, we refine the estimate of cell phone yaw error, using its alignment set $\mathcal{A}_{ij}$, as a weighted average of each aligned plane yaw. That is,

$$\Delta yaw = \frac{\sum_{(m,k)\in\mathcal{A}_{ij}} c_m^Q \, c_k^D \Delta yaw_{mk}}{\sum_{(m,k)\in\mathcal{A}_{ij}} c_m^Q \, c_k^D}$$

where, as before, $\Delta yaw_{mk} = \phi_k^D - \phi_m^Q$. From this yaw error, we estimate the yaw for the cell phone as $yaw = yaw_{cell} + \Delta yaw$. Similarly, we estimate the yaws for our scene's planar normals as $\phi_m = \phi_m^Q + \Delta yaw \;\; \forall \;\; m \in \mathcal{A}_{ij}$, generating one plane for every alignment in the optimal set. In practice, we output one or multiple planes and their normals after the alignment process.

The plane yaws for the example query computed from vanishing points are plotted in Figure 6, with the best alignment pair indicated by its yaw error of 26 degrees. After the weighted average to

compute the final yaw error, we conclude that the query image has a yaw orientation of 252 degrees, which has a 4 degree error from the ground truth, improved upon a 22 degree error of the cell phone's reported yaw. The plane yaws for the query were found to be $\varphi_1 = 118$ and $\varphi_2 = 24$.

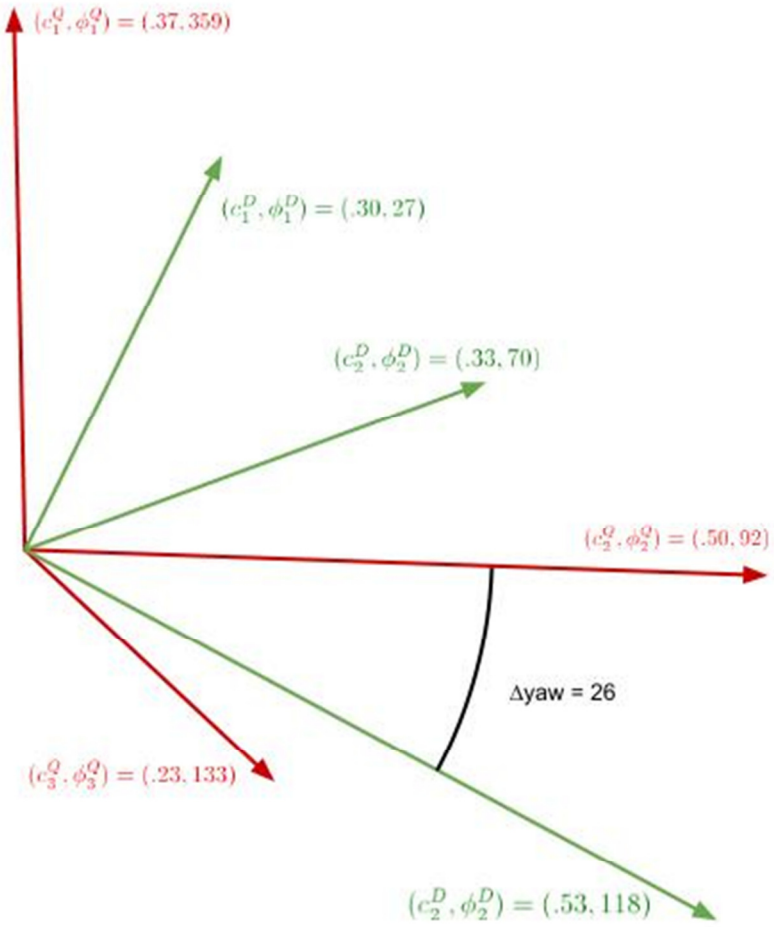


Figure 6: A plot of the plane yaws for the example database (green) and query (red) images. The direction of each line corresponds to the yaw value $\boldsymbol{\phi}$*, where up is 0 degrees and right is 90 degrees, while the length is proportional to each plane's confidence parameter* $\boldsymbol{c}$*.*

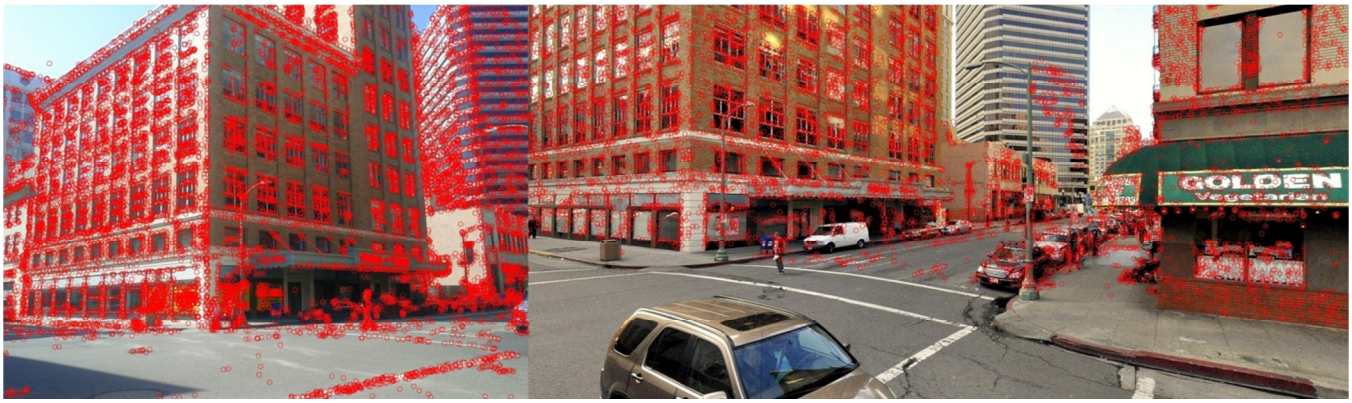
3.3 Feature Correspondences

In order to estimate a homography matrix, we need point correspondences between Q and D. To do so, we extract high resolution SIFT features from the two images and perform a nearest neighbor search between the two sets of features to find matches [6]. These features are shown in Figure 7(a). We apply a series of pre-homography filters to be described shortly to reduce the number of

correspondences, and in particular reduce the number of incorrect feature matches, i.e. outliers. Those features which remain after filtering are showing in Figure 7(b).

3.3.1 SIFT Filters

Since we expect both Q and D to be upright, i.e. roll around 0, we can filter out feature correspondences by their gradient orientation, a parameter we have access to from the SIFT decomposition. The first filter we apply requires feature correspondence to have orientations within 60 degrees of each other. Note that this threshold is deliberately chosen to be fairly large, as the SIFT orientations are computed on the two dimensional image plane, and therefore a difference in three dimensional viewing angle, e.g. yaw, between Q and D can lead to a large difference in the SIFT feature orientation. We apply a relative distance filter, or ratio test, to keep only feature correspondences that are unique [6].



(a)

(b)

Figure 7: The image in (a) shows the extracted features from the example query and database images before filters, while (b) shows those feature correspondences which remain after applying the filters.

3.3.2 Database Filters

We use the 3D database extensively in the homography computation, and as such, it also plays a large part in filtering out correspondences pre-homography. The first such filter removes all database correspondences that lack depth data. This is because all such features are either too near to the database camera or are part of the sky, neither of which is desirable to keep. In practice this removes very few features.

The next filter only applies to databases which contain 3D plane information. When plane data is present, we are able to isolate and remove the ground plane present in D. In practice, we have found that feature correspondences that include the database features on the ground rarely help and primarily hinder the homography solution, and therefore all such features are removed.

3.4 Homography and Scale Estimation

Given yaw, plane normals, and feature matches, we now estimate a homography matrix to solve for translation. To do so, we must take into account our known parameters, utilize 3D information to recover the scale of translation, and account for outliers with robust estimation. These three processes are addressed in Sections 3.4.1, 3.4.2, and 3.4.3 respectively. In the remainder of the thesis, all vectors are assumed to be represented in the coordinate frame for Q, except when the vector has a superscript d, which denotes that the vector is in the coordinate frame for D.

3.4.1 Constrained Homography

This section outlines homography estimation using the constraints imposed by the known planes and orientations. The homography transformation maps a plane from a starting view D , which in our set up has a known pose, to a new view, Q, containing the same plane [4]. The homography transformation is defined as the mapping from any point feature $\hat{x}^d$ on D to its corresponding feature $\hat{y}$ on Q using the following relation:

$$\hat{y} \propto \boldsymbol{H}\hat{x}^d \qquad \text{where} \qquad \boldsymbol{H} = \boldsymbol{R}_d^q + r\,\hat{t}\,(\hat{n}^d)^T \tag{4}$$

The symbol $\propto$ represents proportionality and the superscript T denotes a vector transpose. $\boldsymbol{H}$ is the canonical homography matrix, representing a rotation and translation of $\hat{x}^d$ to map to the feature location $\hat{y}$. $\boldsymbol{R}_d^q$ and $\vec{t}$ are the rotation and translation parameters that map a 3D point from D to Q, i.e. $\vec{y} = \boldsymbol{R}_d^q \vec{x}^d + \vec{t}$, where $\vec{x}^d$ and $\vec{y}$ represent the true 3D locations, including depth, of the feature points $\hat{x}^d$ and $\hat{y}$ respectively. In this way, $\hat{t}$ is the direction of translation. The parameter $\hat{n}^d$ is the normal vector of the scene plane, while r is a unitless scale factor defined as the ratio $r = |\vec{t}|/p$, where p is the projected distance from the plane to the center of the camera for D, i.e. $p = (\hat{n}^d)^T \vec{x}^d$ for all points $\vec{x}^d$ on the plane. Note that the homography matrix traditionally requires solving for eight parameters.

Since we know the absolute orientation of D, and have computed the same for Q using vanishing points, we know $\boldsymbol{R}_d^q$. Therefore we can remove orientation from Equation 4 by mapping $\hat{x}^d$, $\vec{x}^d$, and $\hat{n}^d$ to the query coordinate frame, i.e. $\hat{x} = \boldsymbol{R}_d^q \hat{x}^d$, $\vec{x} = \boldsymbol{R}_d^q \vec{x}^d$, and $\hat{n} = \boldsymbol{R}_d^q \hat{n}^d$. In doing so, we obtain the simplified homography relation with all variables in the query coordinate frame:

$$\hat{y} \propto \boldsymbol{J}\hat{x} \qquad \text{where} \qquad \boldsymbol{J} = \boldsymbol{I} + r\,\hat{t}\,\hat{n}^T \tag{5}$$

We now have a simple expression containing a transformed homography matrix with only five unknown parameters: two in each of $\hat{t}$ and $\hat{n}$ and one in r. Using the assumption that all viewed building faces are upright, we eliminate one parameter from the plane normal leaving a scalar plane yaw as the only parameter in $\hat{n}$. This reduces the number of unknown parameters to four in Equation 5. It is possible to further reduce the number of unknown parameters to three by using known plane yaws from vanishing points as described in Section 3.2 or from database planes. Then, in practice, the total number of unknown parameters for this constrained homography could be 3 or 4 depending on the reliability of plane yaw estimates from other sources, as described in Section 3.4.4.

In order to find an optimal solution minimizing error, we define an error metric on Equation 5. We note that Equation 5 contains three proportionality constraints using the same proportionality factor, and so we can divide by one of them to obtain two equality constraints. The two equality constraints can be formulated as a reprojection error metric for a given feature correspondence pair $(\hat{x}, \hat{y})$ and parameter solution set $(r, \hat{t}, \hat{n})$

$$e[0] = \hat{y}[0]/\hat{y}[2] - \vec{z}[0]/\vec{z}[2] \tag{6a}$$

$$e[1] = \hat{y}[1]/\hat{y}[2] - \vec{z}[1]/\vec{z}[2] \tag{6b}$$

$$\text{with} \quad \vec{z} \triangleq \hat{x} + r\,\hat{t}\,\hat{n}^T\hat{x} \tag{6c}$$

The first terms $\hat{y}[0]/\hat{y}[2]$ and $\hat{y}[1]/\hat{y}[2]$ of Equations 6(a) and 6(b) respectively represent the left side of Equation 5, while the second terms $\vec{z}[0]/\vec{z}[2]$ and $\vec{z}[1]/\vec{z}[2]$ represent the right side. Equation 6 is the canonical homography reprojection error under the constraints we have imposed. Since the error constraint is two-dimensional and we solve for at most four parameters, we can find a unique solution which minimizes error using only two independent correspondence pairs, rather than the four that would have been required had we not constrained the problem as above. This reduced complexity better conditions the homography estimation and improves runtime dramatically for our robust estimation algorithm, which is outlined in Section 3.4.3.

3.4.2 Scale Recovery

This section describes how we solve for the scale of translation. In the error metrics described in Section 3.4.1, we found that solving for the homography parameters does not provide the absolute distance $\left|\vec{t}\right|$ between the query and database views, but only the direction $\hat{t}$. To recover the distance, note that the homography solution obtained does solve for $r = \left|\vec{t}\right|/p$. Using a database feature's depth $|\vec{x}|$, we can compute p with only one database feature as

$$p = \hat{n}^T\vec{x} = |\vec{x}|\,\hat{n}^T\hat{x} \tag{7}$$

Once we have the distance p from the center of the camera for D to the plane, we also have the translation scale as $\left|\vec{t}\right| = rp$, resulting in the absolute position of Q.

We incorporate Equation 7 into the error metric of Equation 6 to find an optimal solution by defining an expanded three-dimensional error metric for a given feature pair and depth $(\hat{x}, \hat{y}, |\vec{x}|)$ and expanded solution set $(r, p, \hat{t}, \hat{n})$

$$e[0] = \hat{y}[0]/\hat{y}[2] - \vec{z}[0]/\vec{z}[2] \tag{8a}$$

$$e[1] = \hat{y}[1]/\hat{y}[2] - \vec{z}[1]/\vec{z}[2] \tag{8b}$$

$$e[2] = \alpha\ (\ 1 - |\vec{x}|\ \hat{n}^T \hat{x}\ /\ p\ ) \tag{8c}$$

We have deliberately chosen the error metric $e[2]$ in Equation 8(c) not to be $p - |\vec{x}|\ \hat{n}^T \hat{x}$, because this would result in an error metric with units of meters. Rather, we choose the form in Equation 8(c) to match the unitless reprojection errors $e[0]$ and $e[1]$. The constant α in $e[2]$ is a factor which determines the relative importance of the plane depth error $e[2]$ with respect to lateral reprojection errors $e[0]$ and $e[1]$. Since our 3D database tends to have greater depth error than lateral error, we set $\alpha = 0.2$, which equates 5% depth error to 1% lateral deviation error. The addition of scale to the error metric has introduced one more unknown parameter, namely p, and one more constraint, namely Equation 8(c). With this change, we still require two correspondence pairs to find a unique solution minimizing the error function e in Equation 8.

3.4.3 Robust Estimation

We estimate the homography parameters using RANSAC [4]. Since we have constrained our homography estimation problem, we only need two random correspondences in each iteration to find an initial solution set to test. The number of samples l which is required to compute a hypothesis solution is one of three parameters dictating the approximate number of iterations that our RANSAC loop requires. The other two parameters are ω, the approximate fraction of all samples that are inliers to the solution, and ρ, the probability that we attain a solution. The number of iterations is given by [4]:

$$\mathrm{iter}(\rho, \omega, l) \cong \frac{\log(1-\rho)}{\log(1-\omega^l)}$$

Based on the dependence of the function iter on l, the required number of iterations is considerably lower for $l = 2$ than for $l = 4$. However, this is offset by a decrease in ω, since we add an additional depth constraint to the error metric in Equation 8(c), possibly rejecting feature correspondences which could have been inliers in a standard homography. In practice, the percentage of inliers for our system is between 2% and 20%, and as such, we generally run between 1000 and 100,000 iterations, with the higher number of iterations occurring when a resulting solution appears to be physically infeasible.

For the RANSAC loop, we compute a weight w for each feature correspondence in order to find the "best" inlier set of features. This weight places a larger emphasis on database features which in 3D are spatially closer to the center of the camera for D. We place this weight on each feature for two reasons: first, the error metric of Equation 6 tends to have lower reprojection errors for features in a plane far away from the center of the camera; second, closer features tend to be more accurate and therefore produce more accurate pose estimates. For these two reasons, we also target the system for a user taking images across the street rather than down the street or at a further distance. The weight we apply to each feature correspondence is based on the database feature depth and is defined as

$$w(|\vec{x}|) = \begin{cases} 20/|\vec{x}|, & |\vec{x}| \geq 20 \text{ meters} \\ 1, & |\vec{x}| < 20 \text{ meters} \end{cases}$$

For each potential RANSAC solution, we apply three filters which weed out poor solutions and allow us to detect when RANSAC has failed. These three validity checks are:

$$|\mathcal{M}| \geq f_{inlier} \tag{9a}$$

$$\left|\vec{t}\right| < 75 \text{ meters} \tag{9b}$$

$$|2 + \Delta vert| < 3 \text{ meters} \tag{9c}$$

where $\mathcal{M}$ is the inlier subset of all features correspondences and $\Delta vert$ is the computed vertical displacement between the center of the cameras from D to Q. Equation 9(a) implies the number of inliers must be greater than or equal to a fixed parameter f_{inlier} chosen ahead of time; Equation 9(b) implies the distance between Q and D must be less than 75 meters; Equation 9(c) implies vertical displacement in the world frame must lie in a bounded region. The third validity check in Equation 9(c) uses an assumption about the constraints of our problem. In practice we have found that query images taken by a human user are consistently around 2 meters below their respective database images taken by a van. We therefore add this check to reject solutions that do not fit in a bounded region around that vertical displacement. When all solutions fail these validity checks, the homography estimation has failed and we output a null solution. This case allows for fail-safe runs and conditions, laid out in Section 3.4.4.

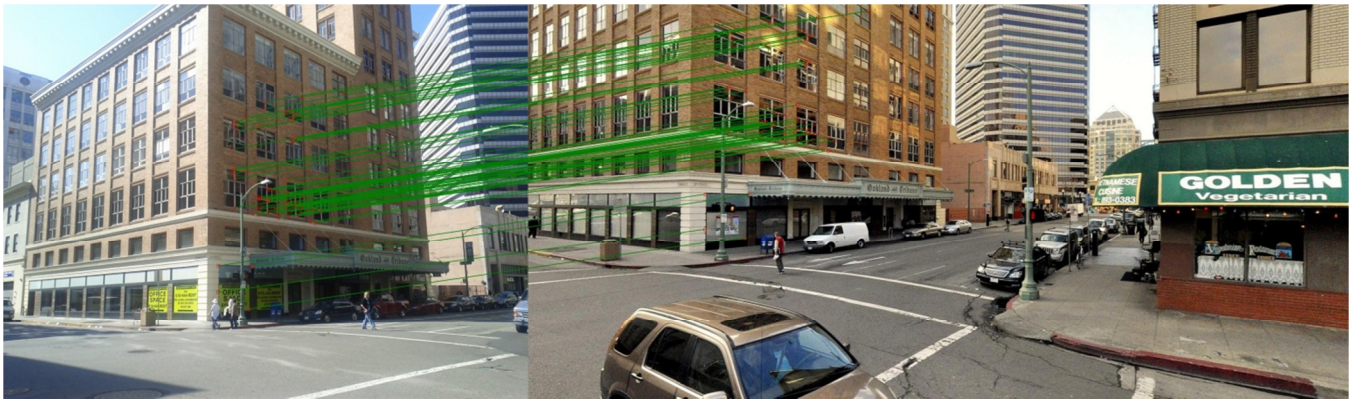
For each valid solution, we compute the confidence of its corresponding inlier set $\mathcal{M}$, defined as the ratio of the sum of the inlier weights to the sum of all correspondence weights:

$$\text{conf\_hom}(\mathcal{M}) = \frac{\sum_{m \in \mathcal{M}} w_m}{\sum_k w_k} \tag{10}$$

Over all iterations, the valid solution set with the largest inlier confidence is chosen as the optimal solution. Following the RANSAC loop, we apply guided matching to the optimal solution set [4].

We also use Equation 10 to choose among multiple plane candidates in a given scene. Unlike the case where the plane yaw for $\hat{n}$ is computed from vanishing points, whenever it is derived from database planes, the spatial extent of the plane is also known. This helps to filter out features that do not lie on the plane resulting in a smaller input correspondence set to the homography algorithm.

For the example query, the plane yaw from vanishing points $\phi_1 = 24$, when used as the fixed plane parameter, finds the best homography solution, resulting in the inlier feature correspondences and homography mapping shown in Figure 8. The solution finds a total translation of 10 meters between the query and database image, resulting in a 1 meter error from the ground truth query location, improving upon the 17 meter error from the GPS reading.



(a) (b)

Figure 8: An image showing the feature correspondence inliers for the solution set, with lines connecting the point features on (a) the query image which match with those on (b) the database image.

3.4.4 Setup and Fail-safe Conditions

The validity checks outlined in Section 3.4.3 allow for failure cases, where no solutions are found. We opt to take advantage of this by being stringent on our definition of a valid solution, then relaxing the validity checks and inputs for queries that need it. As such, we run up to three execution cases, one primary and two fail-safe. Three parameters are varied over these cases: the error threshold ε for defining an inlier, the minimum number of feature inliers f_{inlier}, and the approach taken with the plane normal $\hat{n}$. The number of RANSAC iterations does not change between cases.

In the first execution case, we set $\varepsilon = 0.01$ and $f_{inlier} = 15$. In addition, we fix the plane normal $\hat{n}$ as known, drawing from a set of planar candidates from both vanishing points and database planes. If the result of this execution case returns no valid solution, then the parameters ε and f_{inlier} are relaxed. The second execution case handles $\hat{n}$ identically to the first, but sets $\varepsilon = 0.03$ and $f = 10$.

If there is no solution after the second case, we relax the restriction on $\hat{n}$ for a final execution. The other parameters remain the same, i.e. $\varepsilon = 0.03$ and $f = 10$, but rather than drawing from a set of planar candidates, we solve for the plane yaw of $\hat{n}$ within the homography estimation of Section 3.4.3.

If the final execution case fails to retrieve a valid solution, then we naively set the 3D translation vector $\vec{t} = 0$, implying that Q and D are taken from the same spot. This mirrors the approach taken by [13] when homography solutions fail. In practice, most queries find a valid solution in the first execution case, while only a few fail in all execution cases.

4. EXPERIMENTAL RESULTS

We have tested our algorithm on city-scale databases of Berkeley and Oakland, California provided by Earthmine Inc. The database images provided are geo-tagged panoramas taken from a van with a corresponding depth map and, in the case of the Oakland database, a plane map. From the panoramas, we extract 6 rectilinear building-facing views, corresponding to 60, 90, 120, 240, 270, and 300 degrees relative yaw, defining 0 degrees relative yaw as the direction the van is facing and 90 degrees as the street-side view to the right of the van. Table 2 contains more of the specifics of the Berkeley and Oakland databases.

The database images are used as potential matching views to the query dataset, where the matched views are obtained from the image retrieval algorithm presented in [12]. Since the algorithm does not have perfect performance, we eliminate those matched views which cannot be usefully applied to our pose estimation system. Those eliminated include the following cases:

(a) The matched view is of the wrong building, shown in Figure 9(a).
(b) The focal scene is too far from either camera, shown in Figure 9(b).
(c) The matched view is of the wrong building face on the correct building, shown in Figure 9(c).
(d) The matched view has too little in common, i.e. the overlap between both views is less than 5%, shown in Figure 9(d).
(e) The matched view is at too oblique an angle with respect to the relevant building face; generally < 15 degrees, shown in Figure 9(e).

Table 2: Dataset properties.

Dataset	Berkeley	Oakland
Database area	~1 km^2	~4 km^2
Database image size	2048x1371	2500x1200
Database size	~12000 images	~29000 images
Plane fitting	no	yes
Cell phone	HTC Droid Incredible	Samsung Nexus S
Query image size	2592x1952	2560x1920
Query size	91 images	112 images
Correct retrievals	83 images	102 images
Useful retrievals	73 images	92 images

Those query images not removed by case (a) represent the “Correct retrieval” query images in Table 2, while those not removed by any case above are called the “Useful retrievals”. Case (b) is removed because our target application currently intends the user to take an image from across the street and no further, which is motivated by the limitations of accurate pose recovery at a distance. Case (c) is removed because the planes appear to be visually similar, but it is not possible to recover pose with a homography transformation when the planes between scenes are not the same. Case (d) is removed because scenes with minimal overlap cannot be expected to recover relative pose. Case (e) is removed because the image retrieval stage has recovered a database image which matches, except that the geometry is difficult to resolve. This particular case is often an overlap with case (c), where image retrieval has matched the wrong sides of a building, but the correct side is still visible, though at an extremely oblique angle.

In order to evaluate the performance of our system, we use Google Earth, query imagery, and memory knowledge of the query path to record a ground truth latitude, longitude, and yaw orientation for each

(a)

(b)

(c)

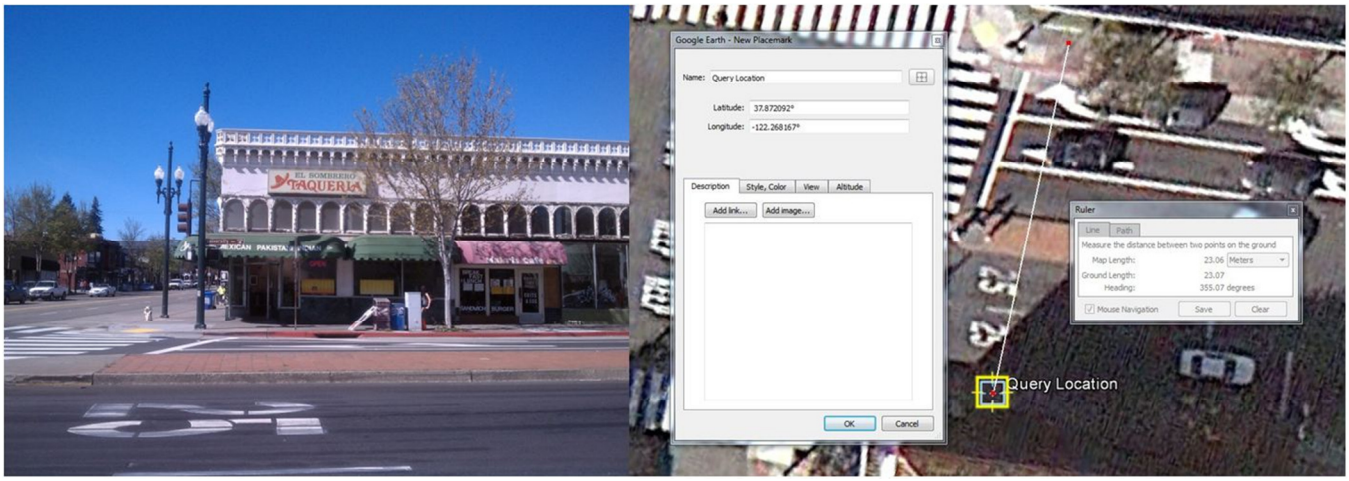
(d)

(e)

Figure 9: The above images show Oakland query images (left) and their retrieved database images (right). (a) through (e) show five cases of eliminated queries from analysis.

query image in both the Berkeley and Oakland query sets. To do this, as shown in Figure 10, in Google Earth we go to the general area where the image was taken and estimate with the query image the

location of its camera center, recording its latitude and longitude. Then, using the Ruler tool, we estimate the line from the origin of the camera to the content in the center of the image. This gives us a bearing for that line, which we record as the ground truth yaw orientation. These tools are displayed on a query image in Berkeley in Figure 10. We acknowledge that these ground truth values may be error prone, both because they were estimated by hand and eye and because there may be a discrepancy between Google Earth coordinates and Earthmine coordinates. However, as discussed in Appendix A, our performance plots to be shown later indicate that the errors in our ground truth locations and yaw are within 5 meters and 5 degrees respectively. For this reason, we focus on 5 and 10 meter performance values, and we do not attempt to analyze performance within a couple of meters.



(a) (b)

Figure 10: (a) query image in Berkeley; (b) satellite view in Google Earth. Shown in (b) are the Placemark tool (left), for recovering latitude and longitude, and the Ruler tool (right), for recovering bearing. The yellow box indicates the approximated ground truth location, while the white line from the Ruler tool indicates the approximate yaw bearing.

4.1 Yaw Performance

Figure 11 shows the performance of the vanishing point method discussed in Section 3.2, as compared to the compass on the mobile device for the Oakland and Berkeley datasets. In Oakland, we obtain a yaw within 5 degrees of ground truth 76% of the time, compared to 10% for the compass, and within 10 degrees of ground truth 96% of the time, compared to 26% for the compass. Similar gains are found

in Berkeley, where 62% of queries are within 5 degrees, compared to 13% for the compass, and 90% of queries are within 10 degrees, compared to 36% for the compass.

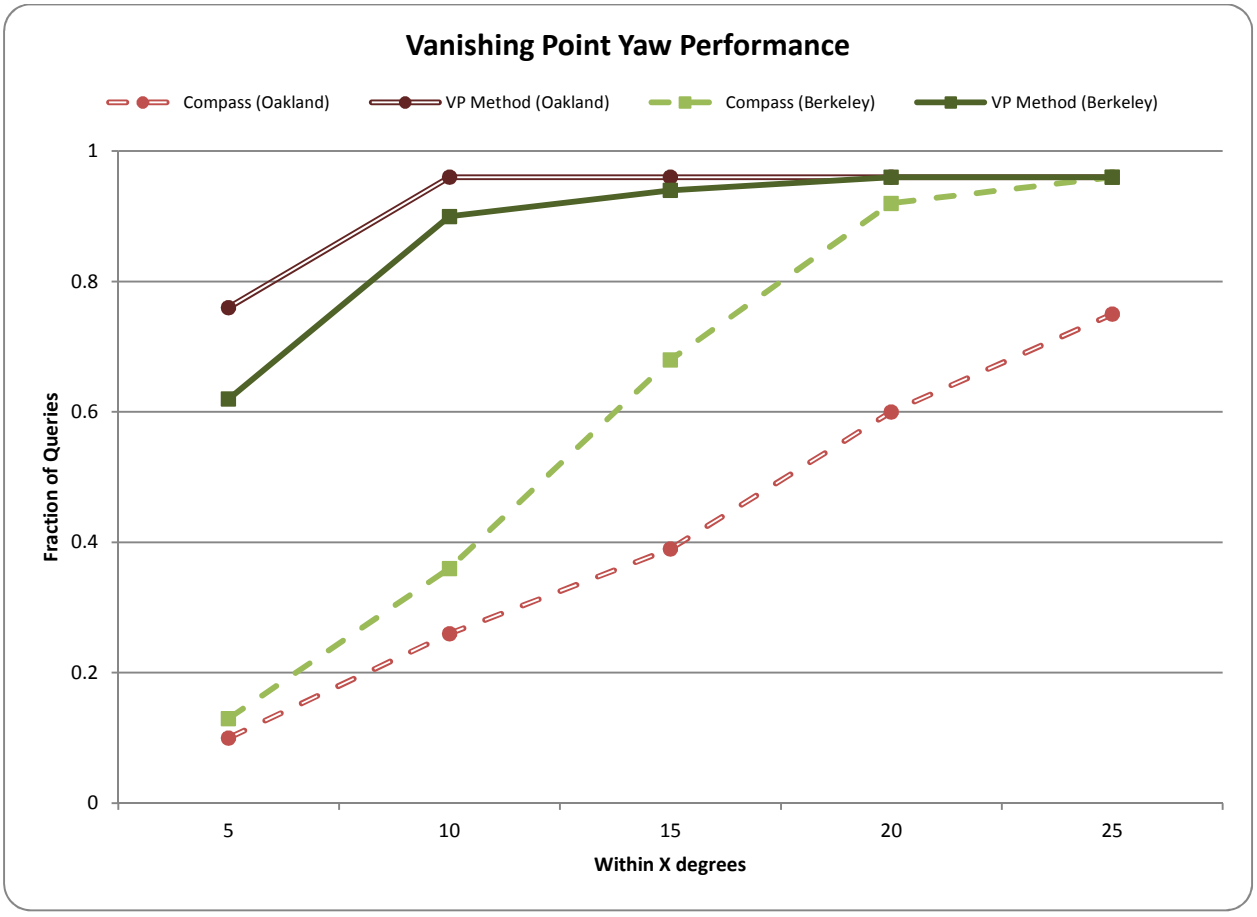
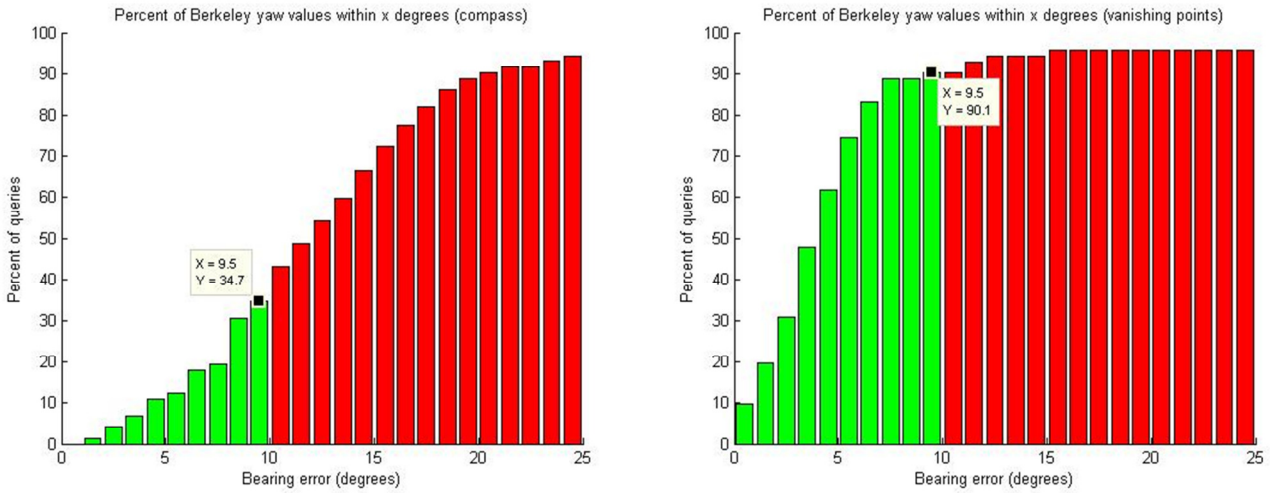


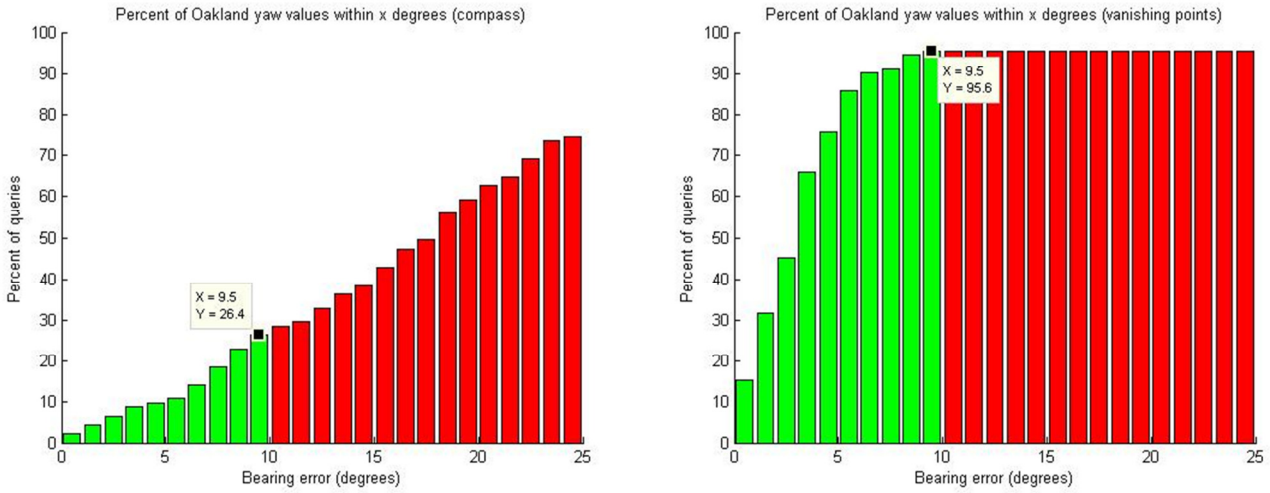
Figure 11: Yaw orientation error in degrees using manual ground truth yaw value from Google Earth for both the Berkeley and Oakland datasets. The plots compare our vanishing point method to compass values from the cell phone.

We now show performance plots with 1 degree resolution, even though the ground truth values may not be accurate enough to make such resolution meaningful. These are shown in Figure 12 and Figure 13, and use a green and red color scheme, where 10 degree errors have arbitrarily been chosen as a cutoff for adequate performance errors. The plots mirror a probability cumulative distribution function (CDF), where an 80% level at 10 degrees means that the system estimates the query yaw orientation within 10 degrees of its ground truth location 80% of the time.



(a) (b)

Figure 12: (a) Berkeley yaw performance using (a) cell phone compass readings and (b) the vanishing point method outlined in Section 3.2.



(a) (b)

Figure 13: (a) Oakland yaw performance using (a) cell phone compass readings and (b) the vanishing point method outlined in Section 3.2.

One limitation of our approach is the reliance on the compass reading in Section 3.2.2 to ensure that two different planes are not accidentally aligned. Since compass error is generally within 50 degrees and most planes in the city are separated by 90 degrees, there are not too many misalignments.

Figure 14: An example query demonstrating improper planar alignment to compute yaw, reporting a yaw with 55 degree error as measured by the manual ground truth.

However, not all city planes lie on a rectangular grid. As such, the algorithm struggles when plane normals are separated by less than 50 degrees, as the compass cannot distinguish between multiple distinct plane alignments when they are all within the bounds of compass error. When there are multiple plane alignments, we depend on the plane weights and alignment confidence to choose the right alignment, which is more susceptible to error.

4.2 Location Performance

The results for location from homography estimation, discussed in Sections 3.4, are presented in Figure 15, which plots location performance for the Berkeley and Oakland datasets using both our system and the raw GPS reading from the cell phone. Our algorithm improves location estimates substantially, especially with the Oakland dataset, where it is within 5 meters of ground truth 84% of the time compared to only 16% for GPS, and within 10 meters 92% of the time compared to only 31% of the time for GPS. Berkeley performance does not improve as much over GPS, both because our system performs more poorly and GPS performs better. Using the homography location estimate in Berkeley, 64% of queries are localized within 5 meters, while 82% are localized within 10 meters. This is still better than GPS, which performs at 34% within 5 meters and 66% within 10 meters.

As before, we show the high resolution plots with 1 meter resolution, in Figure 16 and Figure 17. The plots again use the red and green color scheme where a 10 meter location error has been arbitrarily declared as an adequate performance.

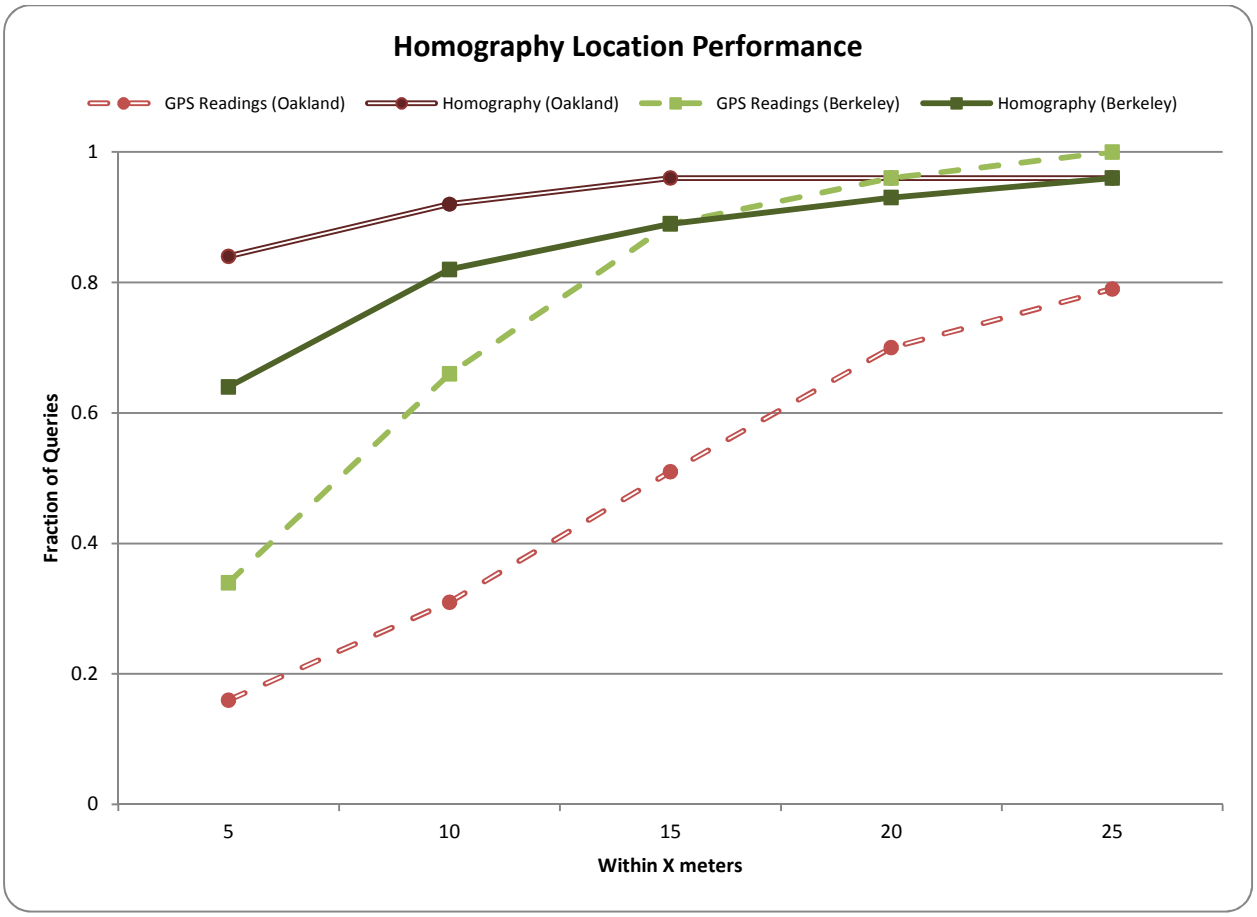
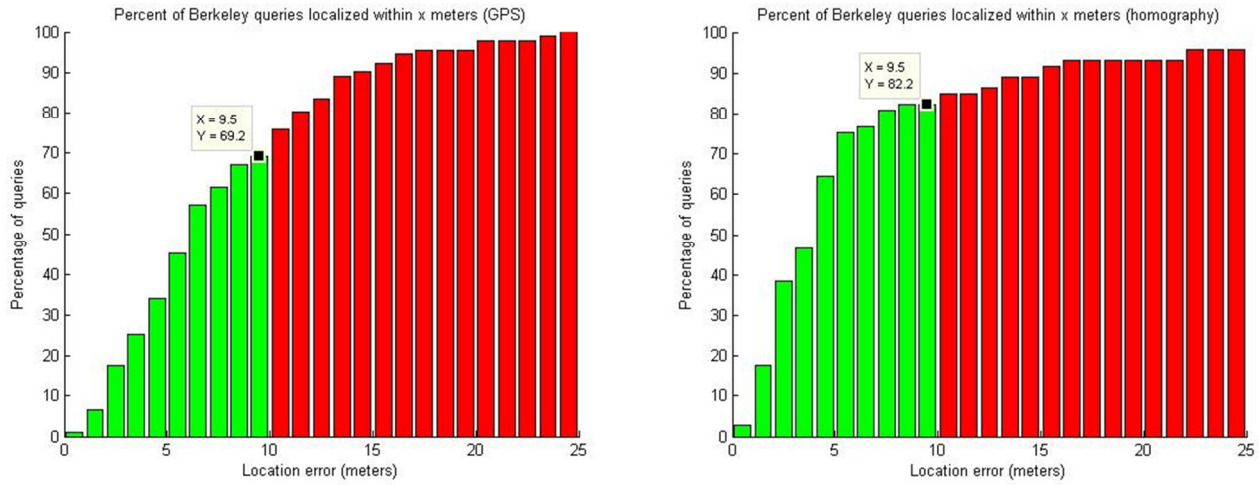


Figure 15: Location performance within X meters of the ground truth location for both the Berkeley and Oakland datasets. The plots compare our homography estimation method to GPS readings from the cell phone.

We speculate there are two reasons Berkeley does not perform as well as Oakland. The first is that the Berkeley database has no plane data, making it impossible to remove ground features and also to restrict features to planes. The second is that the accuracy of the Berkeley 3D database from Earthmine is lower than that of Oakland, and improvements in the 3D point clouds result in more accurate pose estimation.



(a) (b)

Figure 16: (a) Berkeley location performance using (a) GPS readings from the cell phone and (b) the constrained homography estimation outlined in Section 3.4.

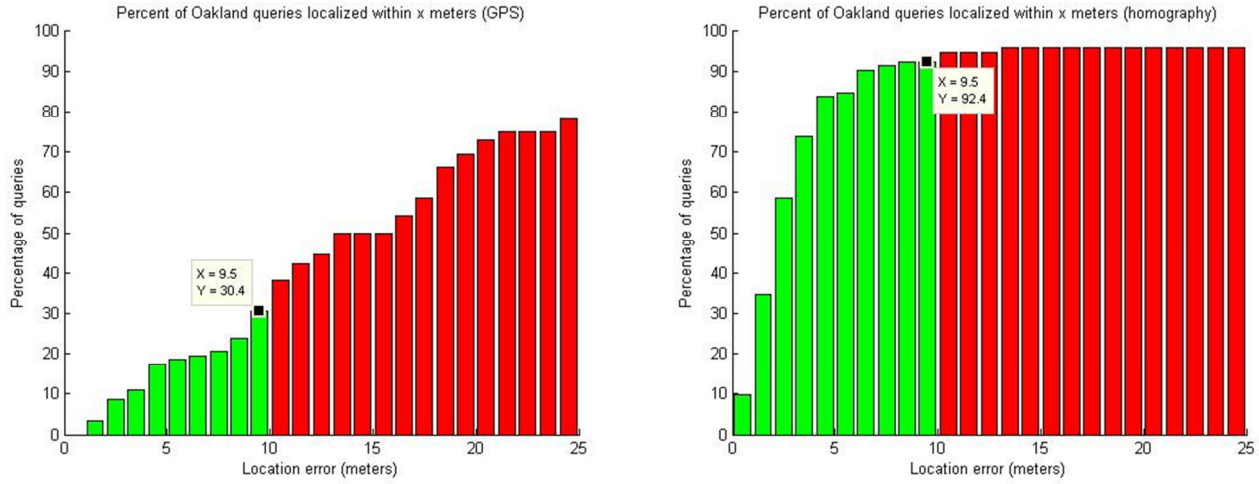


Figure 17: (a) Oakland location performance using (a) GPS readings from the cell phone and (b) the constrained homography estimation outlined in Section 3.4.

Table 3 shows the breakdown of various execution cases for Oakland queries with less than 10 meter location error. As seen, the majority of queries with less than 10 meters of error find a valid estimate of pose from the first execution case. Recall that each homography solution is chosen from a set of solutions determined by a pool of known plane normals. The best is chosen from the set using their confidence values from Equation 10. The pool of known plane normals has two sources: those from vanishing point alignment from Section 3.2 and those from the Oakland database. The final column of

Table 3 enumerates how many solutions used known plane normals from vanishing point alignment, revealing that these planes are chosen more often than those from the database.

Table 3: Oakland dataset run characteristics, including execution cases and the use of planes from vanishing points.

Execution Case	*1*	*2*	*3*	*T=0*	*Total*
Total number of queries	83	7	1	1	92
Number within 10 meters	79	4	1	1	85
Within 10 m using VP plane	63	1	0	0	64

As shown in the examples of Figure 18, homography fail cases generally can be explained by problems with the scenery. For example, when the database and query images are separated by a large distance or angle, the homography estimate struggles to find inlying correspondences. Additionally, feature repetition can cause homography matches to exhibit a displacement that produces an incorrect position estimate. More generally, the algorithm can fail because of scenes that make feature matching difficult, such as extreme occlusion, drastic lighting differences, or changes in scenery in the time between query and database snapshots. Since we work with a homography transformation, the algorithm also struggles in the absence of dominant planes.

(a)

(b)

(c)

Figure 18: The homography estimation error for the above three images fall outside 10 meters and exhibit some of the common problems with the point feature based homography estimation, including (a) repetitive features, (b) lack of dominant planes, and (c) occlusion and lighting differences.

Since the homography estimation algorithm leaves yaw orientation fixed, its performance is inherently linked to the performance of our yaw estimate. In our datasets, this accounts for a portion of the incorrect homography estimates. Also, we have implicitly assumed that our overall system has correctly retrieved a usable database image to match the query. Thus the overall system performance depends on three subsystems: retrieval, vanishing points, and homography estimation. The following three location performance cases are plotted in Figure 19 for both Oakland and Berkeley: dependency on both image retrieval and yaw recovery, dependency on yaw recovery but assuming a successful retrieval, and assuming both a successful retrieval and a perfect yaw using the ground truth value. This analysis shows we perform extremely well given the correct yaw, localizing within 5 meters 86% of the time and within 10 meters 96% of the time in Oakland. However, if we incorporate image retrieval

performance into our system, we only localize within 10 meters 81% of the time in Oakland and 66% of the time in Berkeley. This is identical to the performance obtained with GPS in Berkeley but still greatly outperforms that of Oakland. Located in Figure 20 and Figure 21 are the plots with 1 meter resolution.

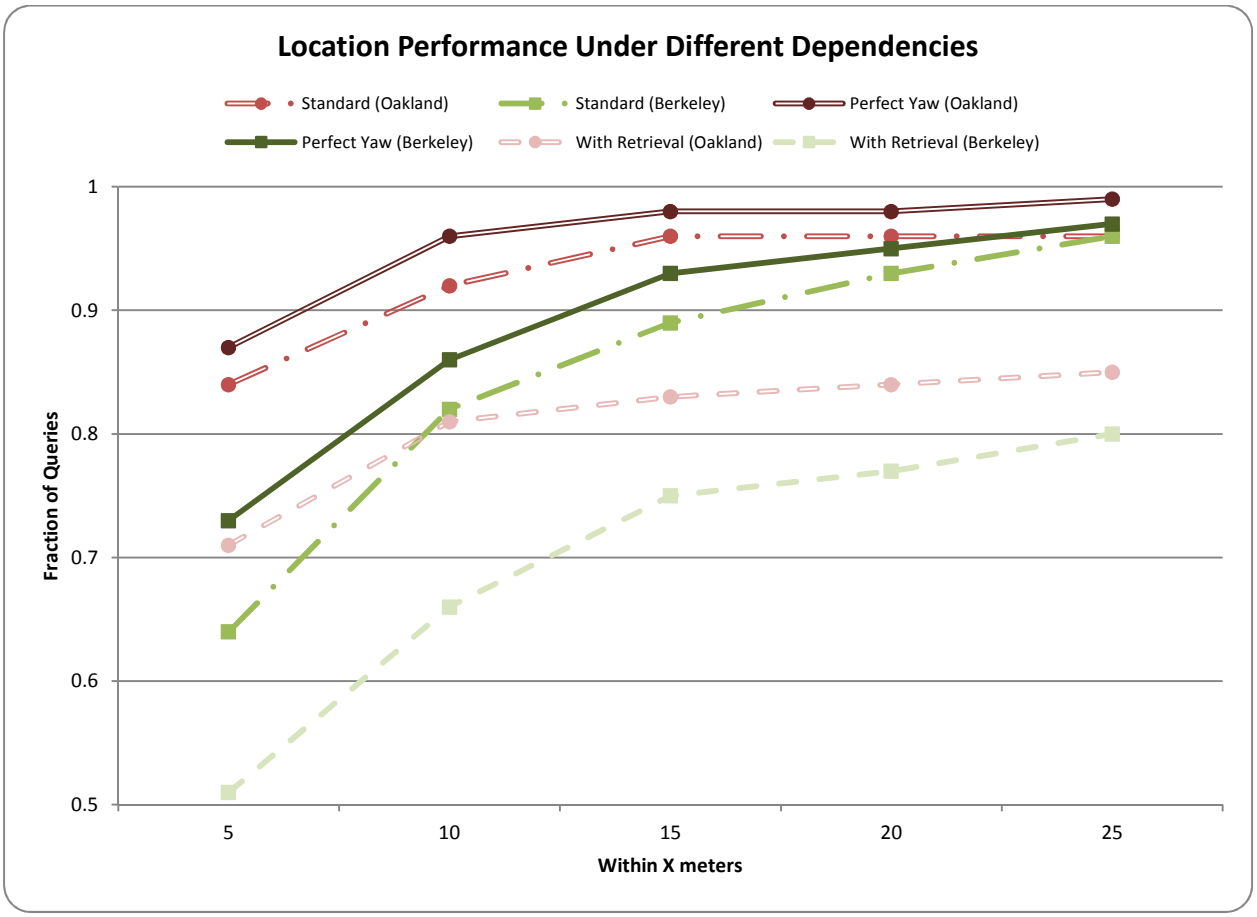
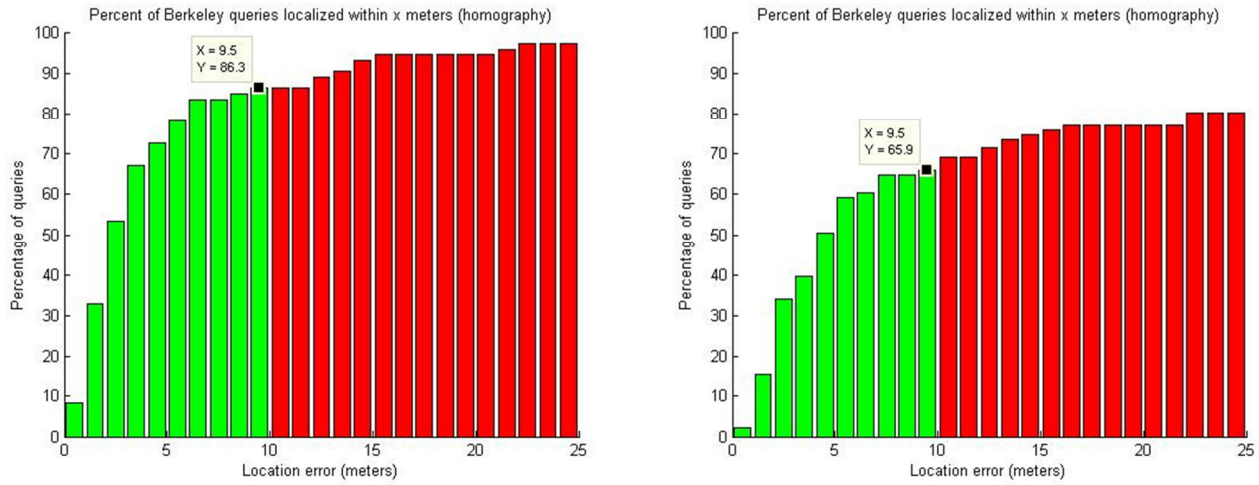
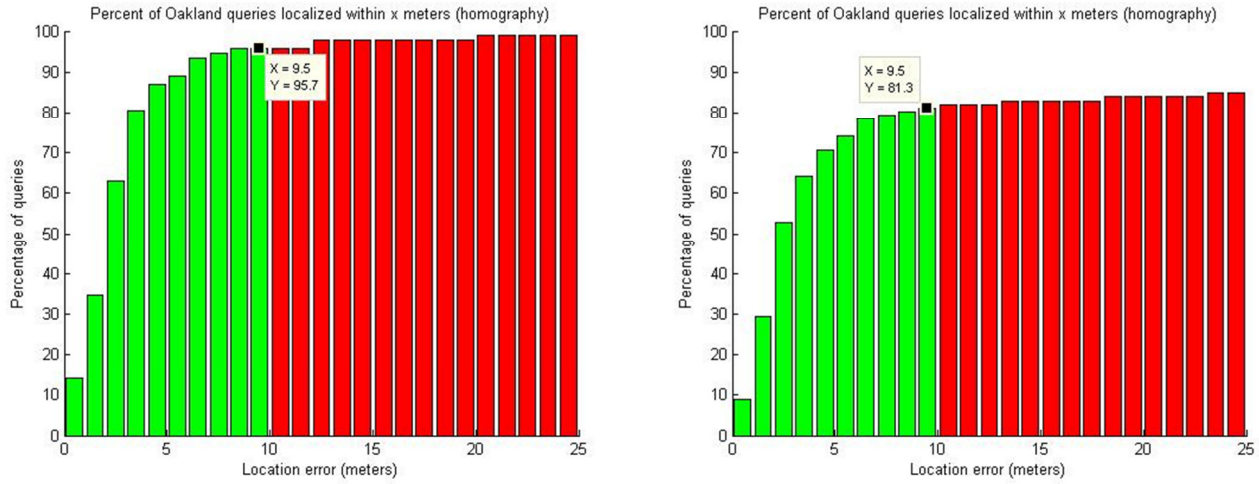


Figure 19: Location performance within X meters of ground truth for both the Berkeley and Oakland datasets. The plots compare how the homography estimation performs when incorporating the results of image retrieval as well as when removing the dependence on a yaw estimate from vanishing points to the standard performance obtained from Figure 15.



(a) (b)

Figure 20: Berkeley location performance (a) when using the ground truth yaw orientation rather than the result from vanishing points, and (b) when incorporating the performance of image retrieval into the system.



(a) (b)

Figure 21: Oakland location performance (a) when using the ground truth yaw orientation rather than the result from vanishing points, and (b) when incorporating the performance of image retrieval into the system.

4.3 Oakland Failure Cases

This section discusses failure cases in Oakland, defined as queries which estimate location more than 10 meters away from ground truth. In total there are seven such queries, each with its own subsection

and ordered by increasing location error. Each subsection is its own query, titled by the query id used in the file system. Immediately following the heading is a figure showing the query and database image, with the location error in its caption. For each image, we discuss the reasons this query has failed in localization and what can be done to address these problems.

4.3.1 Query 2011-10-28_12-01-26_932

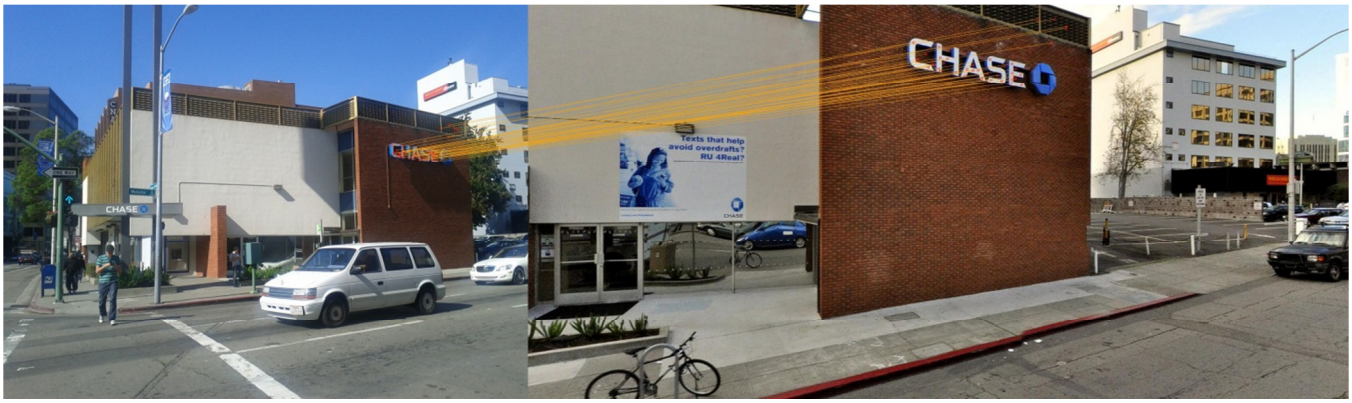
(a) (b)

Figure 22: Error = 10.3 meters. (a) Query image; (b) Database image.

This query shown in Figure 22 fails because it matches feature on the wrong plane. There are two planes for the building in the scene. Both are shown in 22(b), but only one appears in 22(a). The feature correspondences in the two images match the only plane in 22(a) to the wrong plane in 22(b). Since the wrong plane is matched, we cannot expect this to perform well, and in fact it is surprising that the error is so low. A more root cause for this query's problem is image retrieval and repeated features. Since SIFT features are not invariant under rotation, it is most natural to match features with the same skew in the image, and so matching the correct plane between these two images is difficult, and a different retrieved image would make pose estimation easier. Along this line, repeated features is an issue as well, because image retrieval and pose estimation would not associate incorrect planes if the two planes did not share very common features.

To address these problems, one could work with line feature matching [3], which could more easily notify the algorithm that the lines being matches are not oriented the same in the world frame. The same could be done with 3D pattern features [7].

4.3.2 Query 2011-10-28_12-32-21_902



(a) (b)

Figure 23: Error = 10.6 meters. (a) Query image; (b) Database image.

In contrast to the previous query, it is surprising that the error is so large in this query shown in Figure 23. Every indicator would seem to predict that location error will be low, e.g. good yaw estimate and features matched correctly. However, we obtain a location error of 11 meters. This is most likely due to two reasons. First, most of the features are located in a small section of both the query and database image, making the relation between pose and reprojection error more ill-conditioned. In addition, it is hard to notice, but the off-white wall has a different plane normal than the brick wall with the CHASE lettering. The difference is about 10 degrees, and matching features in a small area with the wrong plane normal could explain a poor estimate.

This problem would be hard to fix, as it requires distinguishing between those two planes. Such resolution on plane normals is hard to attain, but could be attempted. In the current vanishing point detection algorithm, we look for vanishing points separated by more than 10 degrees in order to eliminate false detections. Removing or reducing this threshold might help with this query image, but would also have negative consequences for other queries.

4.3.3 Query 2011-10-28_12-56-39_060

(a) (b)

Figure 24: Error = 13.7 meters. (a) Query image; (b) Database image.

This query shown in Figure 24 has a few different issues that likely compound upon each other. First, there is temporal change and occlusion in reference to the trees. Since the database image was taken during winter and the query image during summer, the leaves make for a very different image, primarily with regards to occlusion. Second, the building is plagued by repeated features, and this is exhibited by the feature matches used in homography estimation, which have an offset from their true matches. Last, the reflection in the "mirror-like" building face are at different locations depending on the angle we view the building from. These are features that can be matched but will not be correctly matched for the plane the building represents.

This query would be hard to fix in the system. It suffers from problems that are not easily remedied except to take another query or database image at a different time. The query reveals a strong weakness to image based localization, and would require a much "smarter" algorithm to correct. For example, a large 3D pattern representing the building face [7] might help, except that it would not be perfectly planar and so it would be harder to match.

4.3.4 Query 2011-10-28_12-56-42_448

(a) (b)

Figure 25: Error = 31.1 meters. (a) Query image; (b) Database image.

All queries from here on suffer from major problems which result in very large location errors, all of which are over 30 meters. The image shown in Figure 25 has an improper yaw estimate, and therefore has a query yaw orientation error of 45 degrees. With such a large error, we cannot even analyze the homography estimation process, we can only look at vanishing point alignment. Because the planes presented are all separated by 45 degrees, there will be multiple valid alignments, based on the cell phone yaw reading. The algorithm chooses the one which maximizes a confidence function, but the confidence function is based on how many lines contribute to the vanishing points in the imagery. Since the views for the query and database image can be so different, this is not always dependable, and in this instance, the wrong alignment is chosen.

To fix this, we would need more advanced techniques to prevent improper alignment. The lines we assign to vanishing points do not tell us about the colors, shapes, patterns, or point features present, but we could imagine developing an algorithm which visually matches planes, then aligns the correct ones to attain a yaw orientation estimate. This would be very complex compared to just using vanishing points.

4.3.5 Query 2011-10-28_11-59-28_617

(a) (b)

Figure 26: Error = 33.1 meters. (a) Query image; (b) Database image.

The cause for the poor location estimate in this query shown in Figure 26 is repeated features, exacerbated by the far viewing distance of the retrieved image. It is easy to see that the features matched between the query and database images are offset in the real world by about 30 meters, primarily because the scenery contains repeated features that make the offset area match well.

One way to help with this query would be to use pattern features [7] to get around the limitations of point features. While this would help, it would be best introduced in image retrieval, because the far view distance and repeated feature problem affects retrieval such that we do not get the best database image out, which would be one that is closer to the actual scene.

4.3.6 Query 2011-10-28_12-02-44_400

(a) (b)

Figure 27: Error = 43.8 meters. (a) Query image; (b) Database image.

The image shown in Figure 27 has a poor yaw estimate, with an error of 49 degrees. The reason for the poor yaw estimate is not as easy to determine as it was in Section 4.3.4, where there are easily defined planes that are separated by 45 degrees. However, it is likely due to the same problem, namely misalignment, because there are so many planar candidates in these two images. Recall that the planes are detected with vanishing points, and notice how that line directions are scattered about, so that it is hard to find one or two dominant vanishing points, but rather many weaker planes are found. The two major planes, the building on the right and the far street's buildings, do not dominate the image as planes usually do. Therefore, many planes are detected and misalignments are made more probable.

In general, this is the type of query image that our system does not perform well on, because the major planes are either far away or at a very oblique angle. It is unclear what can be done to improve the yaw estimate, except to not support such query images for pose estimation.

4.3.7 Query 2011-10-28_12-56-32_511

(a) (b)

Figure 28: Error = 62.1 meters. (a) Query image; (b) Database image.

The query shown in Figure 28 has the same problem as in Section 4.3.4, with very regular planes that are separated by less than 90 degrees. Consequently this query misaligns vanishing points as well and estimates yaw with 55 degree error.

To fix the issue, we would need the advanced techniques described in Section 4.3.4 to prevent misalignment.

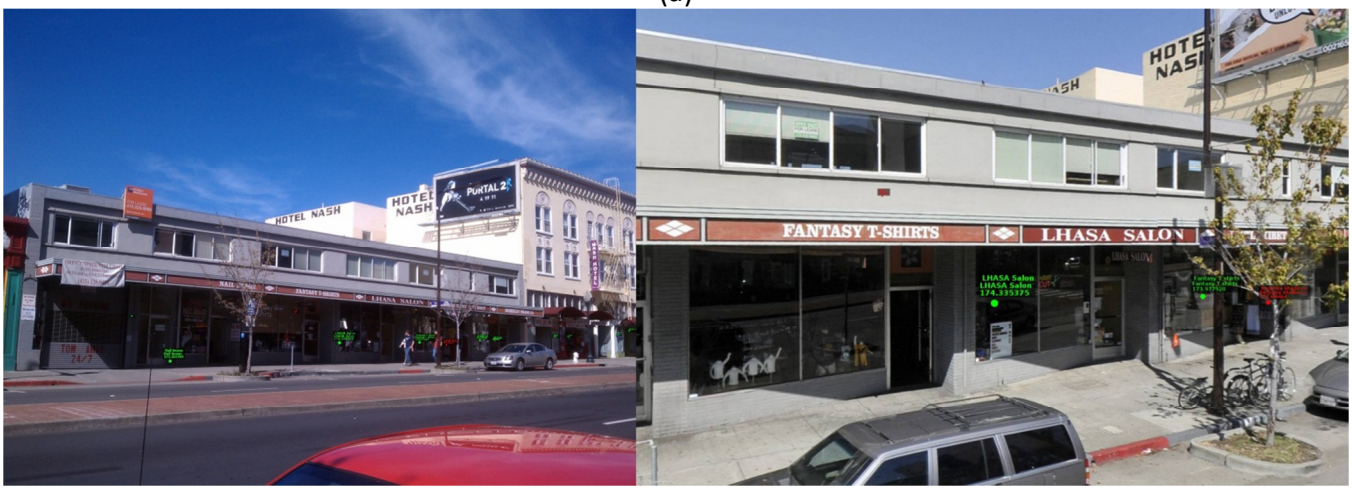
4.4 Tag Transfer in Berkeley

In the Berkeley dataset, we have created sample tags with specific 3D locations in the Earthmine database which we can use to give an example of tag transfer from database to cell phone viewfinder. The homography matrix allows us to do this, and two examples are shown in Figure 29.

With the orientation of the query image and the plane normal, we can also align the text to display the plane geometry, though that would make the text unreadable for highly oblique planes in the scene. In general, however, tag transfer reprojection accuracy has been sacrificed for pose accuracy, resulting in many erroneous tag transfers, as shown in Figure 30.

In addition, the distance of the plane to the camera center affects the tag transfer accuracy, specifically relating to the error in yaw. For a given yaw error, the tag transfer error will be greater if the plane is further from the center of the database image camera.

(a)



(b)

Figure 29: Two example of tag transfer using the rotation and translation drawn from the pose estimation system. Text skew is not performed.

Figure 30: An example of a erroneous tag transfer.

5. CONCLUSIONS AND FUTURE WORK

We have presented a system for accurate pose estimation in urban environments. Using a rich 3D database, cell phone sensors, and imagery, we have applied vanishing points to improve upon compass values and homography estimation to improve upon GPS readings. To best condition the system, we have used scene assumptions highly relevant to urban environments.

In order to improve errors for applications such as geo-tagging, we can examine matching line features [3] or patterns [7] in addition to point features. Also we can expand upon and improve the effectiveness of our fail-safe runs by better detecting failures, for example by generating a confidence parameter for both the yaw and location results. For situations with multiple matching database images for a single query image, we can apply the triangulation techniques of [13] to improve our pose estimate and better condition the results. This requires knowing whether our retrieved images have been correctly matched or not, but preliminary results using an image retrieval confidence parameter have already been established in [12]. Similarly, we can attempt to detect when image retrieval has failed so that we can run a separate method or merely default to GPS and compass. Finally, we currently convert our images to black and white prior to detecting line segments and point features. Using color as a distinguishing feature would increase the complexity of our system, but may improve performance.

For real time application areas such as augmented reality, computational complexity is an important factor. In this thesis, we have mostly ignored all real time issues as we deal with single image pose recovery. One can envision systems in which single image pose recovery is used either as initial condition, or as periodic update of a tracking system for augmented reality applications. The most time consuming aspects of our current system are line detection, feature detection and matching, and homography estimation. To reduce the processing time associated with line detection, we can use a more speed optimized line detector even though performance may take a hit. For feature detection, we can use a feature such as SURF [1] rather than SIFT. We can also match these features more intelligently than with a linear nearest neighbor search, which guarantees the closest match, but does not scale well with number of features. To improve the homography estimation runtime, it is possible to examine robust estimation schemes which scale better when the fraction of inliers goes down.

Finally to improve all three, we can analyze the performance of the system with lower resolution images, which would reduce the number of lines, number of features, and RANSAC runtime.

6. ACKNOWLEDGMENTS

We acknowledge the image and 3D dataset provided to us by Earthmine, as well as the help received from Jimmy Wang and John Ristevski.

7. REFERENCES

[1] Bay, H., Tuytelaars, T., Van Gool, L. 2006. SURF: Speeded Up Robust Features. In *European Conference on Computer Vision* (Graz, Austria, May 7-13, 2006).

[2] Grompone, R., Jakubowicz, J., Morel, J., Randall, G. 2010. LSD: A Fast Line Segment Detector with a False Detection Control. *Pattern Analysis and Machine Intelligence,* 32, 4 (Apr. 2010), 722-732.

[3] Guan, W., Wang, L., Mooser, J., You, S., Neumann, U. 2011. Robust Pose Estimation in Untextured Environments for Augmented Reality Applications. In *Mixed and Augmented Reality* (Orlando, Florida, USA, October 19-22, 2011).

[4] Hartley, R., Zisserman, A. 2004. *Multiple View Geometry*. Cambridge University Press.

[5] Hile, H., Grzeszczuk, R., Liu, A., Vedantham, R., Kosecka, J., Borriello, G. 2009. Landmark-Based Pedestrian Navigation with Enhanced Spatial Reasoning. In *Proceedings of Pervasive* (Nara, Japan, May 11-14, 2009).

[6] Lowe, D. G. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision,* 60, 2 (Jan. 2004), 91-110.

[7] Schindler, G., Krishnamurthy, P., Lublinerman, R., Liu, Y., Dellaert, F. 2008. Detecting and Matching Repeated Patterns for Automatic Geo-tagging in Urban Environments. In *Computer Vision and Pattern Recognition* (Anchorage, AK, June 23-28, 2008).

[8] Schroth, G., Huitl, R., Chen, D. 2011. Mobile Visual Location Recognition. *Signal Processing Magazine,* 28, 4 (Jul. 2011), 77-89.

[9] Takacs, G., Choubassi, M., Wu, Y., Kozintsev, I. 2011. 3D Mobile Augmented Reality in Urban Scenes. In *International Conference on Multimedia and Expo* (Barcelona, Spain, July 11-15, 2011).

[10] Wu, Y., Choubassi, M., Kozintsev, I. 2011. Augmenting 3D Urban Environment Using Mobile Devices. In *Mixed and Augmented Reality* (Basel, Switzerland, October 26-29, 2011).

[11] www.yelp.com

[12] Zhang, J., Hallquist, A., Liang, E., Zakhor, A. 2011. Location-based Image Retrieval for Urban Environments. In *International Conference on Image Processing* (Brussells, Belgium, September 11-14, 2011).

[13] Zhang, W., Kosecka, J. 2006. Image Based Localization in Urban Environments. In *3D Data Processing, Visualization, and Transmission* (Chapel Hill, NC, June 14-16, 2006).

APPENDIX

A. Ground truth error

Since our performance plots depend on the accuracy of the ground truth, we should question its accuracy. For any value we are concerned with, whether it be yaw or location values, let T represent the true value, G represent the recorded ground truth, and C represent the calculated value based on our algorithm. Then we can ask, how accurate is the recorded ground truth? To answer this question, first assume that the method of determining the manual ground truth will not result in large errors. At worst, it will be "in the ballpark" of the correct value, which we define as a 25 meter or 25 degree ceiling on the errors. If the error between G and C is greater than this, then we are confident that the error is due to poor system error between T and C rather than poor ground truth error between T and G. Such queries are removed for the analysis so that we can analyze the statistical expectation of ground truth errors without worrying about poor system performance adversely affecting the expectation. Consider, then, the standard deviation

$$\boldsymbol{E}[(C-G)^2]^{1/2}$$

With the outlier queries removed, this results in a 3.9 meter error for location and 3.5 degree error for yaw orientation, using Oakland system performance here. We can say for both values, in their respective units, that $\boldsymbol{E}[(C-G)^2] < 16$. Since the methods for arriving at G and C are not related, we assume that the error of the ground truth, $G-T$, and the error of the computed value, $C-T$, are independent of each other. Using this assumption, we can argue that

$$\begin{aligned}
\boldsymbol{E}[(C-G)^2] &= \boldsymbol{E}[((C-T)-(G-T))^2] \\
\boldsymbol{E}[(C-G)^2] &= \boldsymbol{E}[(C-T)^2] + \boldsymbol{E}[(G-T)^2] - 2\boldsymbol{E}[(C-T)(G-T)] \\
\boldsymbol{E}[(C-G)^2] &= \boldsymbol{E}[(C-T)^2] + \boldsymbol{E}[(G-T)^2] \\
\boldsymbol{E}[(C-G)^2] &\geq \boldsymbol{E}[(G-T)^2] \\
16 &> \boldsymbol{E}[(G-T)^2] \\
4 &> \sqrt{\boldsymbol{E}[(G-T)^2]} \text{ (Standard deviation of ground truth error)}
\end{aligned}$$

Therefore we conclude that our performance level gives us an upper bound of 4 meters or 4 degrees error in the ground truth process. In fact, the above analysis indicates that our measured error presents a sum of the error of our ground truth and the error of our system, and our system error and ground truth error must both be better than the plots, under the independence assumption. However, we do not try to make claims about the performance of our system under 5 meters or 5 degrees.