Gnome: A Practical Approach to NLOS Mitigation for GPS Positioning in Smartphones

Xiaochen Liu
liu851@usc.edu
University of Southern California

Suman Nath
suman.nath@microsoft.com
Microsoft Research

Ramesh Govindan
ramesh@usc.edu
University of Southern California

ABSTRACT

Accurate positioning in urban areas is important for personal navigation, geolocation apps, and ride-sharing. Smartphones localize themselves using GPS position estimates, and augment these with a variety of techniques including dead reckoning, map matching, and WiFi localization. However, GPS signals suffer significant impairment in urban canyons because of limited line-of-sight to satellites and signal reflections. In this paper, we focus on scalable and deployable techniques to reduce the impact of one specific impairment: reflected GPS signals from non-line-of-sight (NLOS) satellites. Specifically, we show how, using publicly available street-level imagery and off-the-shelf computer vision techniques, we can estimate the path inflation incurred by (the extra distance traveled by) a reflected signal from a satellite. Using these path inflation estimates we develop techniques to estimate the most likely actual position given a set of satellite readings at some position. Finally, we develop optimizations for fast position estimation on modern smartphones. Using extensive experiments in the downtown area of several large cities, we find that our techniques can reduce positioning error by up to 55% on average.

CCS CONCEPTS

• **Information systems** → **Global positioning systems**; • **Networks** → **Mobile networks**; *Location based services*; • **Human-centered computing** → **Mobile computing**; *Ubiquitous and mobile computing systems and tools*;

Research reported in this paper was sponsored in part by the Army Research Laboratory under Cooperative Agreement W911NF-17-2-0196. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.
This work was supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

MobiSys '18, June 10–15, 2018, Munich, Germany

ACM ISBN 978-1-4503-5720-3/18/06…$15.00
https://doi.org/10.1145/3210240.3210343

KEYWORDS

GPS, Localization, NLOS Mitigation, Mobile Computing

ACM Reference Format:
Xiaochen Liu, Suman Nath, and Ramesh Govindan. 2018. Gnome: A Practical Approach to NLOS Mitigation for GPS Positioning in Smartphones. In *MobiSys '18: The 16th Annual International Conference on Mobile Systems, Applications, and Services, June 10–15, 2018, Munich, Germany.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3210240.3210343

1 INTRODUCTION

Accurate positioning has proven to be an important driver for novel applications, including navigation, advertisement delivery, ride-sharing, and geolocation apps. While positioning systems generally work well in many places, positioning in urban canyons remains a significant challenge. Yet it is precisely in urban canyons in megacities that accurate positioning is most necessary. In these areas, smartphone usage is high, as is the density of places (storefronts, restaurants etc.), motivating the need for high positioning accuracy.

Over the last decade, several techniques have been used to improve positioning accuracy, many of which are applicable to urban canyons. Cell tower and Wi-Fi based localization [56, 57] enable smartphones to estimate their positions based on signals received from these wireless communication base stations. Map matching enables positioning systems to filter off-road location estimates of cars [28, 43]. Dead-reckoning uses inertial sensors (accelerometers, and gyroscopes) to estimate travel distance, and thereby correct position estimates [43]. Crowd-sourcing GPS readings [24] or using differential GPS systems [38, 39] can also help improve GPS accuracy. Despite these improvements, positioning errors in urban canyons can average 15m.

That is because these techniques do not tackle the *fundamental* source of positioning error in urban canyons [8, 45]: non-line-of-sight (NLOS) satellite signals at GPS receivers. GPS receivers use signals from four or more satellites to triangulate their positions (§2). Specifically, each GPS receiver estimates the distance traveled by the signal from each visible satellite: this distance is called the satellite's *pseudorange*. In an urban canyon, signals from some satellites can reach the receiver after being reflected from one or more buildings. This can *inflate* the pseudorange: a satellite may appear farther from the GPS receiver than it actually is. This *path inflation* can be tens or hundreds of meters, and can increase positioning error.

Contributions. This paper describes the design of techniques, and an associated system called Gnome, that revises

GPS position estimates by *compensating for path inflation due to NLOS satellite signals*. Gnome can be used in many large cities in the world, and requires a few tens of milliseconds on a modern smartphone to compute revised position estimates. It does not require specialized hardware, nor does it require a phone to be rooted. In these senses, it is immediately deployable.

This paper makes three contributions (§3) corresponding to three design challenges: (a) How to compute satellite path inflation? (b) How to revise position estimates? (c) How to perform these computations fast on a smartphone?

Gnome estimates path inflation using 3D models of the environment surrounding the GPS receiver's position. While prior work on NLOS mitigation [23, 34, 36, 40, 46, 47, 51, 54, 62] (§5) has used proprietary sources of 3D models, we use a little known feature in Google Street View [21] that provides depth information for *planes* (intuitively, each street-facing side of a building corresponds to a plane) surrounding the receiver's position. This source of data makes Gnome widely usable, since these planes are available for many cities in North America, Europe and Asia. Unfortunately, these plane descriptions lack a crucial piece of information necessary for estimating path inflation: the *height* of building planes. Gnome's first contribution is a novel algorithm for estimating building height from panoramic images provided by Street View. Compared to prior work that determines building height from public data [37, 55], or uses remote sensing radar data [29–31, 60], our approach achieves higher coverage by virtue of using Street View data.

To compute the path inflation correctly, Gnome needs to know the ground truth position. However, GPS receivers don't, of course, provide this: they only provide satellite pseudoranges, and an estimated position. Gnome must therefore *infer* the position most likely to correspond to the observed satellite pseudoranges. To do this, Gnome's second contribution is a technique to search *candidate positions* near the GPS location estimate, revise the candidate's position by compensating for path inflation, and then determine the revised candidate position likely to be closest to the ground truth. This contribution is inspired by, but different from the prior work that attempts to infer actual positions by simulating the satellite signal path [40, 47, 54], or by determining satellite visibility [23, 36, 62].

Gnome's third contribution is to enable these computations to scale to smartphones, a capability that, to our knowledge, has not been demonstrated before. To this end, it leverages the observation that 3D models of an environment are relatively static, so Gnome aggressively pre-computes, in the cloud, *path inflation maps* at each candidate position. These maps indicate the path inflation for each possible satellite position, and are loaded onto a smartphone. At runtime, Gnome simply needs to look up these maps, given the known positions of each satellite, to perform its pseudorange corrections. Gnome also scopes the search of candidate positions and hierarchically refines the search to reduce computation overhead.

Gnome differs from [40, 47, 54] in two ways. First, these approaches use proprietary 3D models for ray-tracing, which are not accessible in many cities. In contrast, Gnome leverages highly available Street View data for satellite signal tracing. Second, these approaches are offline while Gnome can compute location estimates in real-time on Android devices.

Our evaluations (§4) of Gnome in four major cities (Frankfurt, Hong Kong, Los Angeles and New York) reveal that Gnome can improve position accuracy in some scenarios by up to 55% on average (or up to 8m on average). Gnome can process a position estimate on a smartphone in less than 80ms. It uses minimal additional battery capacity, and has modest storage requirements. Gnome' cloud-based path inflation map pre-computation takes several hours for the downtown area of a major city, but these maps need only be computed once for areas with urban canyons in major cities. Finally, Gnome components each contribute significantly to its accuracy: height adjustment accounts for about 3m in error, and sparser candidate position selections also increase error by the same amount.

2 BACKGROUND, MOTIVATION, AND APPROACH

How GPS works. GPS is an instance of a Global Navigation Satellite System. It consists of 32 medium earth orbit satellites, and each satellite continuously broadcasts its position information and other metadata at an orbit of about $2x10^7$ meters above the earth. The metadata specify various attributes of the signal such as the satellite position, timestamp, etc. Using these, the receiver computes, for each received signal, its *pseudorange* or the signal's travel distance, by multiplying light of speed with the signal's propagation delay. With these pseudorange estimates, GPS uses three satellites' position to trilaterate the receiver's position in 3D coordinates. In practice, the receiver's local clock is not accurate compared with satellite's atomic clock, so GPS needs another satellite's signal to estimate the receiving time. Thus, a GPS receiver must be able to receive signals from at least four satellites in order to fix its own position.

GPS Signal Impairments. GPS signals undergo four[1] different types of impairments [8] that introduce errors in position estimates. The earth's rotation between when the signal was transmitted and received can impact travel time, as can the Doppler effect due to the satellite's velocity. Ionospheric and tropospheric delays caused by the earth's atmosphere can inflate pseudoranges. Multipath transmissions, where the same signal is received directly from a satellite and via reflection, can cause constructive or destructive interference and introduce errors in the position fix. Finally, a receiver may receive a signal, via reflection, from an NLOS satellite.

Many modern receivers compensate for, either in hardware or software, the first two classes of errors. Specifically, GPS receivers can compensate for earth's rotation and satellite

[1]We have simplified this discussion. Additional sources of error can come from clock skews, receiver calibration errors and so forth [8].

Doppler effects. GPS signals also contain metadata that specify *approximate* corrections for atmospheric delays. Higher accuracy applications that need to eliminate such *correlated* errors (the atmospheric delay is correlated in the sense that two receivers within a few kilometers of each other are likely to see the same coordinated delays) can use either Differential GPS or Real-Time Kinematic GPS. Both of these approaches use base stations whose precise position is known *a priori*. Each base station can estimate correlated errors based on the difference between its position calculated from GPS, and its actual (known) position. It can then broadcast these corrections to nearby receivers, who can use these to update their position estimates.

Two other sources of error, multipath and NLOS reflections are *not* correlated, so different techniques must be used to overcome them. These sources of error are particularly severe in urban canyons [36, 47, 62].

Urban Canyons. To understand how NLOS reflections can impact positioning accuracy, consider Figure 1(a) in which a satellite is within line-of-sight (LOS) of a receiver, so the latter receives a direct signal. If a tall building blocks the LOS path, the satellite signal may still be received after being reflected, and causes NLOS reception (Figure 1(b)). Thus, depending on the environment and the receiver's position, a satellite's *primary* received signal can either be direct or reflected. In addition, the *primary* signal can itself be reflected (more than once), resulting in multipath receptions (Figure 1(c)).

To mitigate the impact of multipath, GPS receivers use multipath correctors ([32, 66, 67]) that use signal phase to distinguish (and filter out) the reflected signal from the primary signal. Modern receivers can reduce the impact of multipath errors to a few meters.

However, when the primary signal is reflected (*i.e.*, the signal is from an NLOS satellite), the additional distance traveled by the signal due to the reflection can inflate the pseudorange estimate. The yellow lines in Figure 1 represent reflected signal paths, which are longer than the primary paths shown in green. The difference in path length between these two signals can often be 100s of meters. Unfortunately, GPS receivers *cannot reliably distinguish between reflected and direct signals*, and this is the primary cause of positioning error in urban areas. Our paper focuses on NLOS mitigation for GPS positioning.

Alternative approaches. To mitigate NLOS reception errors in urban canyons, smartphones use several techniques to augment position fixes. First, they use proximity to cellular base stations [33] or Wi-Fi access points [22] to refine their position estimates. In this approach, smartphones use multilateration of signals from nearby cell towers or Wi-Fi access points whose position is known a priori in order to estimate their own position. Despite this advance, positioning errors in urban areas can be upwards of 15m, as our experiments demonstrate in Figure 2.

Second, for positioning vehicles accurately, smartphones use map matching and dead-reckoning [43] to augment position fixes. Map matching restricts candidate vehicle positions

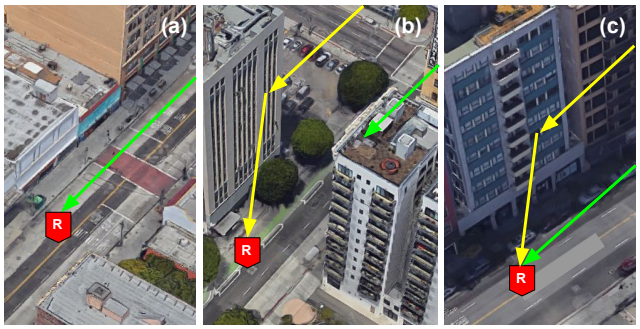

Figure 1: ***(a) A line-of-sight (LOS) signal path. (b) A non-line-of-sitgh (NLOS) signal path. (c) Multipath***

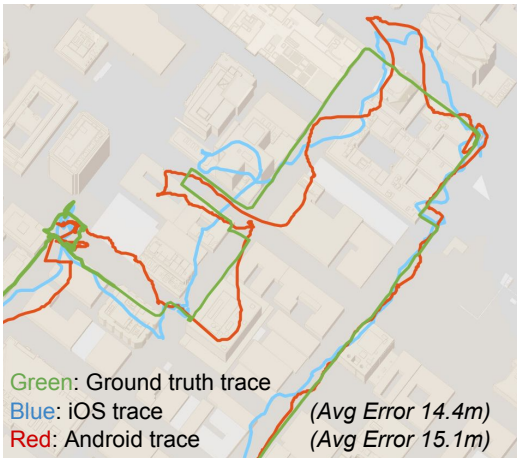

Figure 2: ***An example of localization results in urban canyon on today's smartphone platforms***

to street surfaces, and dead-reckoning uses vehicle speed estimates to update positions when GPS is unavailable or erroneous. While these techniques achieve good accuracy (as we show in §4), they are not applicable to localizing pedestrians in urban settings.

Finally, as we have discussed above, approaches like Differential GPS and Real-time Kinematics assume correlated error within a radius of several hundred meters or several kilometers. Errors due to NLOS receptions are not correlated over these large spatial scales, so these techniques cannot be applied in urban canyons.

Goal, Approach and Challenges. The goal of this paper is to develop a *practical* and *deployable* system for NLOS mitigation on smartphones. To be practical, such a system must not require proprietary sources of information. To be deployable, it must, in addition, be capable of correcting GPS readings efficiently on the smartphone itself.

Our approach is motivated by the following key insight:

> If we can *determine the extra distance* traveled by an NLOS signal, we can *compensate for* this

extra distance, and *recalculate* the GPS location *on the smartphone.*

This insight poses three distinct challenges. The first is *how to determine NLOS satellites and compute the extra travel distance*? While satellite trajectories are known in advance, whether a satellite is within line-of-sight at a given location L depends upon the portion of the sky visible at L, which in turn depends on the position and height of buildings around L. This latter information, also called the *surface geometry* at L can be used to derive the set of surfaces that can possibly reflect satellite signals so that they are incident at L.

The second challenge is *how to compensate for the extra distance traveled by an NLOS signal* (we use the term *path inflation* to denote this extra distance) incident at a location L. This is a challenge because a smartphone cannot, in general, know the location L: it only has a potentially inaccurate estimate of L. To correctly compensate for path inflation, the smartphone has to determine that the location whose predicted reflected signal best explains the GPS signals observed at L.

The final challenge is to be able to perform these corrections on a smartphone. Determining the visibility mask and the surface geometry are significantly challenging both in terms of computing and storage, particularly at the scale of large downtown area of several square kilometers, and especially because these are functions of L (*i.e.*, each distinct location in an urban area has a distinct visibility mask and surface geometry). These computing and storage requirements are well beyond the capability of today's smartphones.

In the next section, we describe the design of a system called Gnome that addresses all of these challenges, while providing significant performance improvements in positioning accuracy over today's smartphones.

Prior work in this area has fallen into two categories: those that filter out the NLOS signal [41, 46, 61], and those that compensate for the extra distance traveled [14, 26, 40, 48]. Gnome falls into the latter class, but is unique in addressing our deployability goals (§5).

3 GNOME DESIGN

In this section, we describe the design of Gnome. We begin by describing how Gnome addresses the challenges identified in §2, then describe the individual components of Gnome.

3.1 Overview

As described above, Gnome detects whether a satellite is within line-of-sight or not, and for NLOS satellites, it estimates and compensates for the extra travel distance for the NLOS signal. Gnome is designed to perform all of these calculations entirely on a smartphone.

To determine whether, at a given location L, a satellite is NLOS or not, and to compute the path inflation, Gnome uses the satellite's current position in the sky as well as the surface geometry around L. Specifically, with a 3-D model of the buildings surrounding L, Gnome can determine whether a satellite's signal might have been reflected from any building

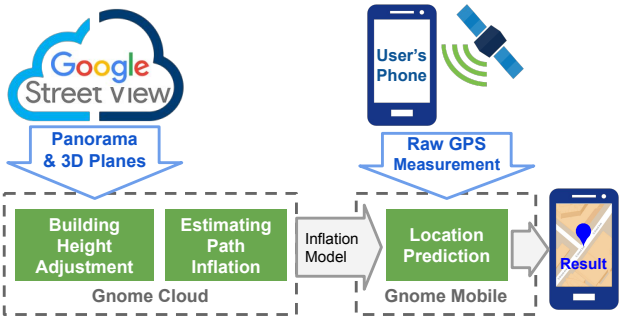

Figure 3: *Gnome workflow.*

by tracing signal paths from the satellite to L, and use that to compute the path inflation. There exist public services to precisely determine the position for satellite at a given time. Less well known is the fact that there also exist *public* sources for *approximate* surface geometry: specifically, Google Street View [21] provides both 2-D imagery of streets as well as 3-D models (as an undocumented feature) of streets for the downtown area of most large cities in the US, Europe, and Asia. These 3-D models are, however, incomplete: they lack building height information, which is crucial to trace reflected signals from satellites. In the section below, we describe how we use computer vision techniques to estimate a building's height. The availability of public datasets with 3-D information makes Gnome widely applicable: prior work has relied on proprietary datasets, and so has not seen significant adoption.

To compensate for the NLOS path inflation on a smartphone, Gnome leverages the fact that modern mobile OSs expose important satellite signal metadata such as what satellite signals were received and their relative strength [10]. Gnome uses this information. The metadata, however, does not inform Gnome whether the satellite was LOS or NLOS. To determine whether a satellite is NLOS at L, Gnome can use the derived surface geometry. Unfortunately, the GPS signal only gives Gnome an estimate of the true location L, so Gnome cannot know the exact path inflation. To address this challenge, Gnome searches within a neighborhood of the GPS-provided location estimate L_{est} to find a *candidate* for the ground truth L_c whose positioning error after path inflation adjustment is minimized.

Our third challenge is to enable Gnome to run entirely on a smartphone. Clearly, it is unrealistic to load models of surface geometry for every point in areas with urban canyons. We observe that, while satellite positions in the sky are time varying, the surface geometry at a given location L is relatively static. So, we *precompute the path inflation*, on the cloud, of every point on the street or sidewalk from every possible location in the sky. As we show later, this scales well in downtown areas of large cities in the world because in those areas tall buildings limit the portion of the sky visible.

Gnome is implemented (Figure 3) as a library on smartphones (our current implementation runs on Android) which, given a GPS estimate and satellite visibility information,

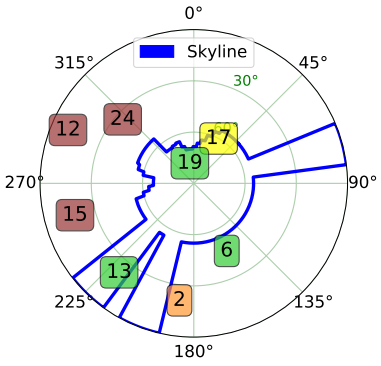

Figure 4: ***The skyline and satellites locations seen by a receiver. The receiver is at the center of the circle and the squares represent satellites. The numbers represent satellite IDs and color signifies signal strength.***

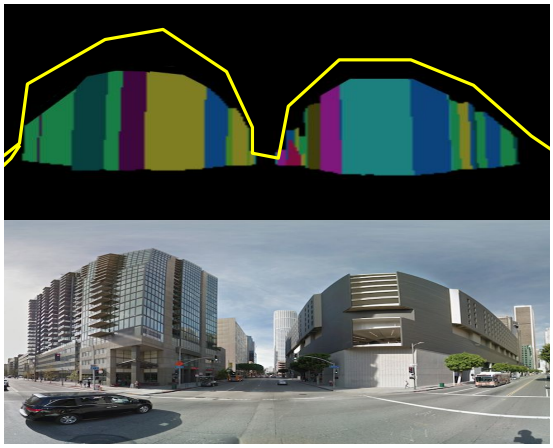

Figure 5: ***Upper: the original depth information (each colored plane represents one surface of a building), together with the missing height information in yellow. Lower: the corresponding panoramic image***

outputs a corrected location estimate. The library includes other optimizations that permit it to process GPS estimates within tens of milliseconds.

3.2 Data sources

Gnome uses three distinct sources of data. First, whenever Gnome needs a position fix, it uses the smartphone GPS API to obtain the following pieces of information:

1. *Latitude, longitude, and error*: The latitude and longitude specify the estimated position, and the error specifies the position uncertainty (the actual position is within a circle centered at the estimated position, and with radius equal to the error).
2. *Satellite metadata*: This information (often called NMEA data [15]) includes each satellite's azimuth and elevation, as well as the signal strength represented as the carrier-to-noise density, denoted C/N_0 [44]. Figure 4 shows an example of satellite metadata obtained from a satellite during our experiments. Each square represents a satellite with the number as satellite ID. The color of the square indicates the satellite's signal strength: green is very good ($C/N_0 > 35$), yellow is fair ($25 < C/N_0 < 35$), and red is bad ($C/N_0 < 25$). The blue line denotes the skyline for a particular street in our data. Notice that satellite number 6 which is NLOS with respect to the center still has good carrier-to-noise density, so this metric is not a good discriminator for NLOS satellites.
3. *Propagation delay and pseudorange*: This contains, for each satellite, the estimated propagation delay and pseudorange for the received signal. This data is read from phone's GPS module and has become accessible since a recent release of Android. This information is crucial to Gnome, as we shall describe later.

Second, Gnome uses street-level imagery data available through Google Street View. This cloud service, when provided with a location L, returns an panoramic *image* around L.

Third, but most important, Gnome uses an approximate 3D model available through a separate Street View cloud service [7]. This service, given a location L returns a 3-D model of all buildings or other structures around that point. This model is encoded (Figure 5 top) as a collection of *planes*, together with their *depth* (distance from L). Intuitively, each plane represents one surface of a building. The depth information is at the resolution of 0.7° in both azimuth and elevation, and has a maximum range of about 120m [25].

Effectively, the 3D model describes the *surface geometry* around L, but it has one important limitation. The maximum height of a plane is 16m, a limitation arising from the range of the Street View scanning device. This limitation is critical for Gnome, because many buildings in urban canyons are an order of magnitude or more taller, and a good estimate of plane height is important for accurately determining satellite visibility.

Google Earth [2] also provides a 3D map with models of buildings. As of this writing, extracting these 3D models is labor-intensive [5] and does not scale to large cities. Moreover, its 3D models do not cover most countries in Asia, Europe, and South America, whereas Street View coverage is available in these continents. We leave it to future work to include 3D models extracted from Google Earth.

3.3 Estimating Building Height

Figure 5 shows how the 3D model's height differs from the actual height of a building: the yellow line above the planes is the actual height. To understand why obtaining height information for planes is crucial, consider a GPS receiver that is 15m away from a 30m building. All satellites behind the building with an elevation less than 63° will be blocked from

the view of the receiver. However, with the planes we have from Street View, all satellites above 45° will still be LOS. Thus, our model may wrongly estimate an NLOS satellite as LOS, and may compensate for the path inflation where no such inflation exists.

Gnome leverages Street View's *panoramic images* to solve the issue. The basic idea is to estimate the building's height by detecting skylines in the Street View images, and then extending each 3D plane to the estimated height. After extracting the surface geometry for a given location L, Gnome selects all planes whose reported height in the surface geometry is more than 13m: with high likelihood, such planes are likely to correspond to buildings higher than 15m. Next, for each such tall plane, Gnome computes its latitude and longitude (denoted by L_{plane}) on the map. It does this using the location L and the relative location of the plane from L (available as depth information in the 3D model).

Gnome then selects a viewpoint L_v near the plane and computes the vector D from L_v to L_{plane} which is the centroid of the plane. This vector contains two pieces of information: (a) it specifies the heading of the plane relative to L_v, and (b) it specifies the distance from L_v to the plane. Gnome then downloads the Street View *image* at L_v and *identifies the plane in the image* using the heading information in H. Figure 6 shows an example of this calculation. Figure 6(a) shows the satellite view at a given location L and the heading vector for the plane (the blue box). The heading vector has azimuth ϕ. Figure 6(b) shows how Gnome uses ϕ to find the plane's horizontal location x in the corresponding panoramic image from Street View.

Now, Gnome runs a skyline detection algorithm on the image. Skyline detection demarcates the sky from other structures in an image. Figure 7 shows the output of skyline detection on some images. Intuitively, the part of the skyline that intersects with the plane delineates the actual height of the plane. Thus, in the three images on the right of Figure 7, the intersection between the blue skyline and the red plane signifies the top of the building. Unfortunately, this intersection is visible in a two-dimensional *image*, whereas we need to augment the height of a plane in a 3D map.

We use simple geometry to solve this (Figure 6(b)). Recall that the vector D also encodes the distance d from L_v to the plane L_{plane}. To estimate the height, we need to estimate the angle of elevation of the intersection θ. We estimate this using a property of Street View's panorama images. Specifically, for a $W \times H$ Street View panorama image, a pixel at (x, y) corresponds to a ray with azimuth $\frac{x}{W} \times 360°$ and elevation $\frac{H/2-y}{H} \times 180°$. So, to estimate θ, we first find the pixel(s) in the panoramic image corresponding to the intersection between the skyline and the plane. θ is then the elevation of that pixel, and we estimate the height of the building as $d \times tan(\theta)$. Figure 8 shows the corrected height of all the planes in the surface geometry of one viewpoint.

In practice, because the target plane may be occluded by trees or other obstructions, we run the above procedure on three viewpoints (Figure 7) near L_{plane} and estimate the height of the building using the average of these three estimates.

At the end of this procedure, for a given location L, we will have an accurate surface geometry with better height estimates than those available with Google Street View.

3.4 Estimating Path Inflation

To estimate the path inflation, Gnome uses ray tracing [19]. Consider a satellite S_i and a reflection plane P_j. As Figure 9 shows, if the receiver is at position R, Gnome first computes its mirror point with respect to P_j, denoted by R'. Then, it initializes a ray to S_i starting from R' and intersects with P_j at point $N_{i,j}$. For this ray to represent a valid reflection of a signal from S_i to R on plane P_j, three properties must hold. First, $N_{i,j}$ must be within the convex hull (or the boundary) of the plane P_j. Second, the ray $N_{i,j}$ to S_i must not intersect any other plane. Finally, the ray from $N_{i,j}$ to R must not intersect any other plane.

For each ray that represents a valid reflection, Gnome employs geometry to calculate the path inflation (shown by the solid red arrows in Figure 9). For a given plane, a given satellite and a given receiver position, there can be at most one reflected ray. For a given receiver R, Gnome repeats this computation for every pair of S_i and P_j, and obtains a path inflation in each case. It also computes whether S_i is within line-of-sight of R: this is true if the ray from S_i to R does not intersect any other plane.

These calculations can result in two possibilities for S_i, with respect to R. If the S_i is within LOS of R, it will likely be the case that there may be one or more planes which provide valid reflections of the signal from S_i to R. In this case, Gnome *ignores* these reflected signals, since they constitute multipath, and modern GPS receivers have multipath rejection capabilities. Thus, when there is a LOS path, path inflation is always assumed to be zero.

The second possibility is that S_i is *not* within line-of-sight of R. In this case, R can, in theory, receive multiple NLOS reflections from different planes. In practice, however, because of the geometry of buildings and streets, a receiver R will often receive only *one* reflected signal. This is because building surfaces are either parallel or perpendicular to the street. If a street runs north-south, and a satellite is on the western sky, then, if the receiver is on the street, it will receive a reflection only from a plane on the east side of the street. However, in some cases, it is possible to receive more than one signal. In our example, if there is a gap between two tall buildings on the west side of the street, then it is possible for that signal to be reflected by a plane on one of those buildings perpendicular to the street (in addition to the reflection from the east side). In cases like these, Gnome computes path inflation using the plane nearer to R.

A signal from S_i can, of course, be reflected off multiple planes before reaching R. In our example, the signal may first be reflected from a building on the east side, and then again from a building on the west side of the street. If a signal can reach the receiver after a single reflection *and*

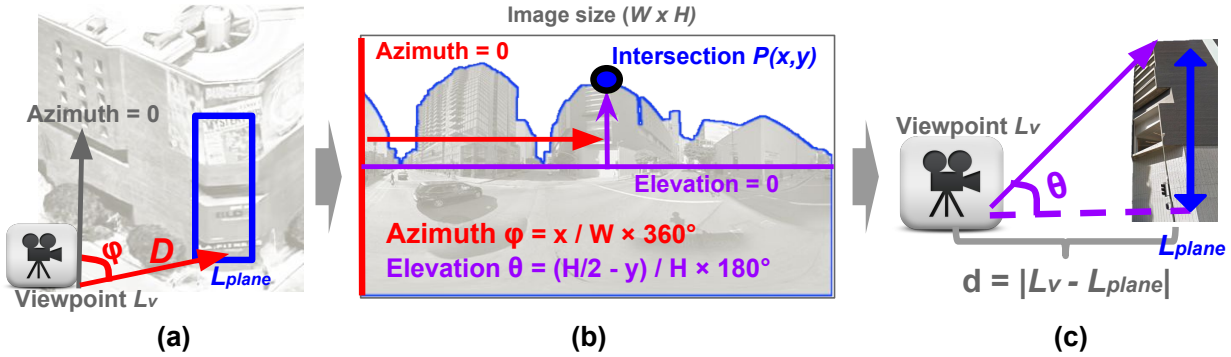

Figure 6: ***Building height estimation. (a) shows how the vector D is estimated from the surface geometry. (b) shows how the panoramic image can be used for skyline detection and θ estimation. (c) shows how building height is estimated using D and θ.***

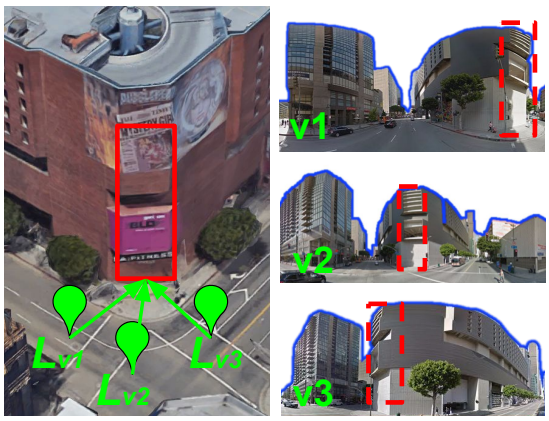

Figure 7: ***For robust height estimation, Gnome uses three viewpoints to estimate height, then averages these estimates.***

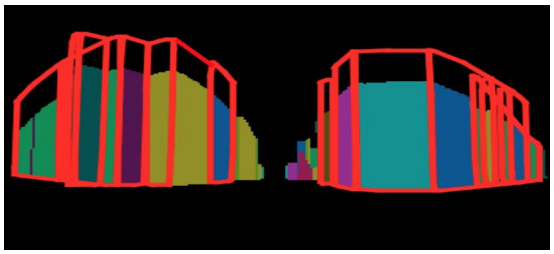

Figure 8: ***The adjusted depth planes, augmented with the estimated height.***

after a double reflection, the calculated pseudorange will be very close to the single-reflection trace because of multipath mitigation. If the signal can only reach the receiver after two reflections, the signal strength will be low ($<$ 20dB) and will always be ignored in position computation [27]. For this reason, Gnome only models single reflections to reduce computational complexity.

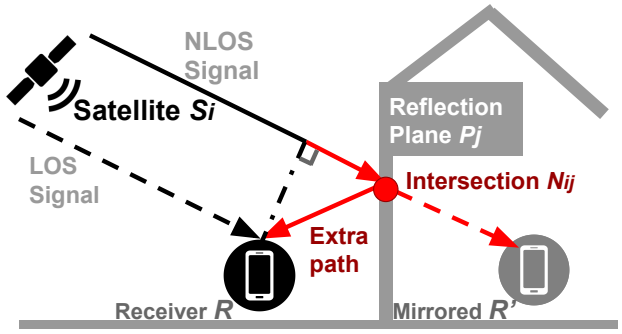

Figure 9: ***An example of ray-tracing and path inflation calculation.***

3.5 Location Prediction

In this section, we describe how Gnome uses the path inflation estimates to improve GPS positioning accuracy. Recall that the GPS receiver computes a position estimate from satellite metadata including pseudoranges for satellites. At a high level, Gnome *subtracts the path inflation* from the pseudorange for each satellite, then computes the new GPS position estimate.

However, the reported satellite pseudoranges correspond to the *ground truth* position of the receiver, which is not known! More precisely, let the estimated position be L_e and the ground truth be L_g. The path inflation of satellite S_i for these two points can be different. If we use the path inflation from L_e, but apply it to pseudoranges calculated at L_g, we will obtain incorrect position estimates.

Searching Ground Truth Candidates. To overcome this, Gnome selects several *candidate positions* within the vicinity of L_e (we describe below how these candidate positions are chosen) and effectively tests whether a candidate location could be a viable candidate for L_g, the ground truth position. At each candidate L_c, it (a) reduces the pseudorange of each NLOS satellite by the computed path elevation and (b) recomputes the GPS position estimate with the revised pseudoranges. This gives a new position estimate L'_c. Gnome chooses that L'_c (as the estimate for L_g) whose distance to

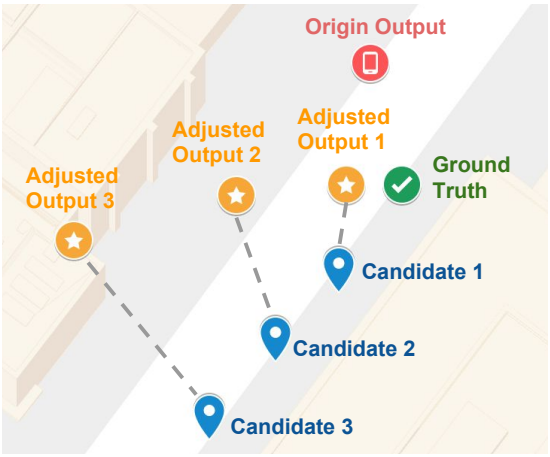

Figure 10: ***After adjusting pseudoranges, candidate positions nearer the ground truth will have estimates that converge to the ground truth, while other candidate positions will have random corrections.***

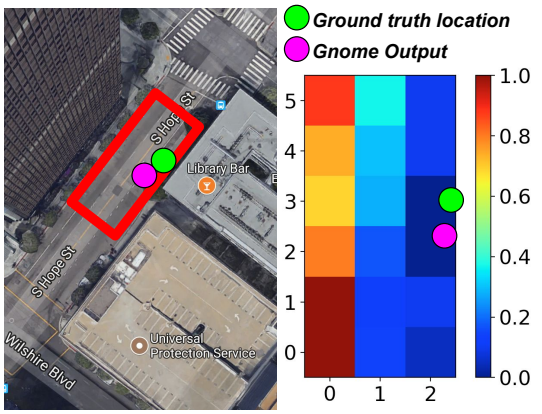

Figure 11: ***Heatmap of the relative distance between candidate positions and the revised candidate positions. Candidates close to ground truth have small relative distances.***

its corresponding L_c is least. Figure 11 shows the heatmap of the relative distances between L_c and L'_c for candidates in a downtown area: notice how candidates closest to the ground truth have the low relative distances. In practice, for a reason described below, Gnome actually uses a *voting* strategy: it picks the five candidate positions with the lowest relative distance, clusters them, and uses the centroid of the cluster as the estimated position.

The intuition for this approach is as follows. When a candidate L_c is close to (within a few meters of) the ground truth, its reflections are most likely to be correlated with the ground truth location L_g. In this case, the candidate's estimated position L'_c will likely converge to the true ground truth position. Because L_c is close to L_g, the distance between L_c and L'_c will be small (*e.g.*, candidate 1 in Figure 10). However, when L_c is far away from L_g, Gnome corrects the pseudoranges observed at L_g with *corrections appropriate for reflections observed at* L_c. In this case, Gnome is likely to go astray since a LOS signal at L_g may actually be an NLOS signal at the candidate position, or vice versa. In these cases, Gnome is likely to apply random corrections at the candidate positions (*e.g.*, candidates 2 and 3 in Figure 10). Because these corrections are random, the relative distances for distant candidate positions are unpredictable: they might range from small to large values. To filter randomly obtained small relative distances, Gnome uses the voting strategy described above. For example, in Figure 11, the grid point at the bottom right of the heatmap shows up among the top three candidate positions with the lowest distance, for exactly this reason.

Position Tracking. As described until now, each predicted location is independent from the previous. However, most smartphone positioning applications require continuously tracking a user's location. So, many navigation services, and, more generally, most localization algorithms for robots, drones or other mobile targets, smooth successive position estimates by using a Kalman filter. Gnome uses a similar technique to improve accuracy in tracking the smartphone user. Specifically, a Kalman filter treats the input state as an inaccurate observation and produces a statistically optimal estimate of the real system state. When Gnome computes a new location, its Kalman filter takes that location as input and outputs a revised estimate of the actual position. We refer the reader to [9] for details on Kalman filtering.

3.6 Scaling Gnome

As described, algorithms in Gnome to estimate building height and path inflation, as well as to predict location can be both compute and data intensive. 3D models can run into several tens of gigabytes, and ray-tracing is a computationally expensive operation. In this section, we describe how we architect Gnome to enable it to run on a smartphone. We use two techniques for this: pre-processing in the cloud, and scoped refinement for candidate search.

Preprocessing in the cloud. Gnome preprocesses surface geometries in the cloud to produce *path inflation maps*. Because surface geometries are relatively static, these path-inflation maps can be computed once and reused by all users.

The input to this preprocessing step is a geographic area containing urban canyons (*e.g.*, the downtown area of a large city). Given this input, Gnome first builds the 3D model of the entire area. To do this, it first downloads street positions and widths from OpenStreetMaps [17], and then retrieves Street View images and 3D models every 5m or so along every street in the area, similar to the technique used in [42]. At each retrieval location, it augments its 3D model with the adjusted building height. At the end of this process, Gnome has a database of 3D models for each retrieval point.

In the second step of pre-processing, Gnome covers every street and sidewalk with a fine grid of *candidate positions*. These candidate positions are used in the location prediction algorithm. Our implementation uses a grid size of 2m × 2m, and we explore the sensitivity of this choice in §4. For each

candidate position, Gnome pre-computes the path inflation *for every possible satellite position*. Specifically, it does this by using the pre-computed 3D models in the previous step and does ray tracing for every point (at a resolution of 1° in azimuth and elevation) in a hemisphere centered at the candidate position to determine the path inflation.

The output of these two steps is a path inflation map: for each candidate position, this map contains the path inflation from every possible satellite position. This map is pre-loaded onto the smartphone. The map captures the static reflective environment around the candidate position. This greatly simplifies the processing on the smartphone: when Gnome needs to adjust the pseudorange for satellite S_i at candidate position L_c, it determines S_i's location from the satellite metadata sent as part of the GPS, and uses that to determine the path inflation from L_c's path inflation map.

Scoped refinement for candidate search. Even with path inflation maps, Gnome can require significant overhead on a smartphone because it has to search a potentially large number of candidate positions in its location prediction phase. Gnome first scopes the candidate positions to be within the error range reported by the GPS device — recall that GPS receivers report a position estimate and an error radius. In urban canyons, however, the radius can be large and include hundreds or thousands of candidate positions. To further optimize search efficiency, we use a coarse-to-fine refinement strategy. We first consider candidate positions at a coarser granularity (*e.g.*, one candidate position in every 8m × 8m grid) and select the best candidate. We then repeat the search on the finer spatial scale around the best candidate selected in the previous step. This reduces computational overhead by 20×.

4 EVALUATION

Using a full-fledged implementation of Gnome, we evaluate its accuracy improvement in 4 major cities in North America, Europe and Asia. We also quantify the impact of individual design choices, and the overhead incurred in estimating building height, path inflation, and in location prediction.

4.1 Methodology

Implementation. Our implementation of Gnome has two components, one on Android and the other on the cloud and requires 2100 lines of code in total. The cloud-side component performs data retrieval (for both depth information and Street View images), height adjustment, and path inflation computations. The smartphone component pre-loads path inflation maps and performs pseudorange adjustments at each candidate position, recomputes the revised location estimate using the Ublox API, and performs voting to obtain the estimated location. Currently, Gnome directly outputs its readings to a file. Without being rooted, Android does not permit Gnome to be run in the background by another app. Apps would have to incorporate its source code [6] in order to use Gnome.

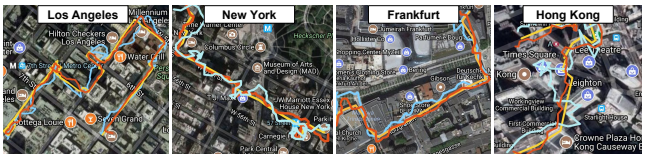

Figure 12: ***The ground truth traces (yellow), Android output (blue), and Gnome output (red) in the four cities.***

Metrics. We measured several aspects of Gnome including: positioning *accuracy* measured as the average distance between estimated position and ground truth; processing *latency* of various components, both on the cloud and mobile; *power consumption* on the smartphone; and *storage* usage on the smartphone for the path inflation maps.

Scenarios and Ground Truth collection. To evaluate accuracy, we take measurements using Gnome in the downtown areas of four major cities in three continents: Los Angeles, New York, Frankfurt, and Hong Kong. In most of these cities, we have measurements from a smartphone carried by a pedestrian. These devices include Huawei Mate10, Google Pixel, Samsung S8, and Samsung Note 8. In Los Angeles, we also have measurements from a smartphone on a vehicle, and from a smartphone on a stationary user. In the same city, we have measurements both on an Android device and an iPhone. Across our four cities, the total walking distance is 4.7km and the total driving distance is 9.3km. Figure 12 shows the pedestrian traces collected in the four cities.

For the stationary user, we placed the phone at ten fixed known locations for 1 minute in each and recorded the locations output by Gnome app. For the pedestrian experiment, tester A walks while holding the phone running the Gnome app. Meanwhile, another tester B follows A and takes a video of A. We use the video to manually determine the ground truth position of A: we pinpoint this location manually on a map to determine the ground truth. For the driving experiment, we place the phone in car's cup holder and drive in the target area. To collect the ground truth location of the car, we use a somewhat unusual technique: we attach a stereo camera [20] to the car and use the 3D car localization algorithm described in [52]. The algorithm can estimate ground truth positions with sub-meter accuracy, sufficient for our purposes.

4.2 Results

Before we describe our results, we describe some statistics about satellite visibility and path inflation in our dataset. These results quantitatively motivate the need for Gnome, and also give some context for our results.

In our data, only 14.4% of the observation points can see more than four LOS satellites, and only 27.5% of all the received satellites are LOS satellites (Figure 13). Thus, urban canyons in large cities contain significant dead spots for GPS signal reception. This also motivates our design decision to not omit NLOS satellites as other work has proposed: since

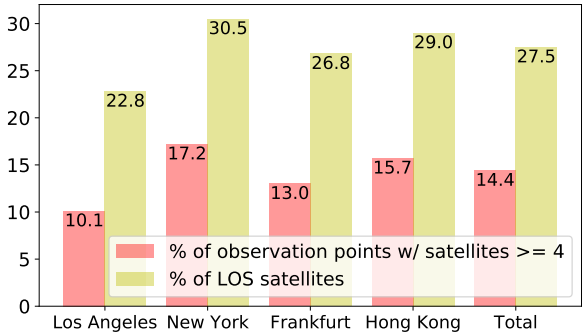

Figure 13: ***Statistics of NLOS signals in the dataset***

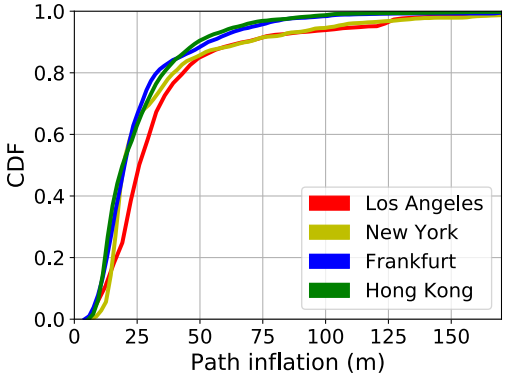

Figure 14: ***Extra NLOS signal path distribution.***

four satellites are required for position fixes, omitting an NLOS satellite would render unusable nearly 85% of our readings.

Figure 14 shows the CDF of path inflation across the 4 cities in our dataset. Depending on the city, between 10% and 15% of the observations incur a path inflation of more than 50m. For two of the cities, 7-8% incur a path inflation of over 100m because the signal is reflected from a building plane that is far from the receiver and therefore leads to a large increase in pseudorange. This suggests that correcting these inflated paths can improve accuracy significantly, as we describe next.

Accuracy. Figure 15 compares the average positioning error of Gnome, Android, and iPhone in four settings (stationary, walking, cycling, driving) for Los Angeles. We use the track recording app for Android [1] and iPhone [3] as location recorders. In the stationary setting, Gnome incurs an average error of less than 5m while Android and iOS incur more than twice that error. This setting directly quantifies the benefits of compensating for path inflation. More important, this shows how much of an improvement is still left on the table after all the optimizations that smartphones incorporate, include cell tower and Wi-Fi positioning, and map matching [13]. The gains in the pedestrian setting are relatively high: Gnome incurs only a 6.4m error, while the

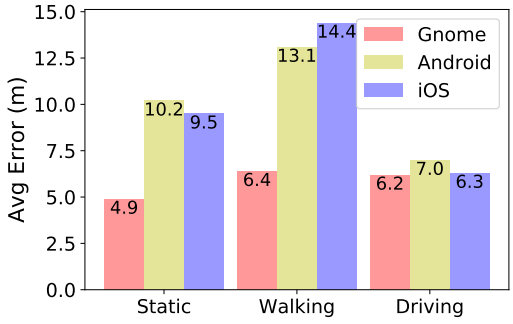

Figure 15: ***Average positioning accuracy in different scenarios.***

iPhone's error is the highest at 14.4m. In this setting, in addition to compensating for path inflation, Gnome also benefits from trajectory smoothing using Kalman filters.

Gnome performance while driving in Los Angeles is comparable to that of Android and iOS. We conjecture that this is because today's smartphones use dead-reckoning based on inertial sensors [11], and this appears to largely mask inaccuracy due to NLOS satellites. These benefits aren't evident in our walking experiments, where the phone movement likely makes it difficult to dead-reckon using accelerometers and gyroscopes.

Across our other cities also (Figure 16), Gnome consistently shows better performance than Android localization. In New York, Gnome obtains a 30% reduction in error, in Frankfurt, a more than 38% error reduction, and in Hong Kong, a more than 40% reduction. Equally important, this shows that the methodology of Gnome is generalizable. In obtaining these results, we did not have to modify the Gnome processing pipeline in any way. As described before, the Street View 3D models are available for many major cities across the world, which is the most crucial source of data for Gnome, so Gnome is applicable across a large part of the globe.

The maximum localization errors for Gnome for the four cities are 37m, 35m, 41m, and 29m respectively. In comparison, the maximum errors for Android are 51m, 42m, 45m, and 47m. These occur either because (a) the ground truth location is not within the error radius reported by the GPS receiver, so Gnome cannot generate candidate points near the ground truth location (§3.6), or (b) because the error radius is too large and includes some non-ground-truth candidates where the distance from the candidate point to the pseudorange-adjusted position is short. The latter confuses Gnome's voting mechanism of candidate selection (§3.5). In future work, we plan to explore mitigations for these corner cases.

Processing Latency. Gnome performs some computationally expensive vision and graphics algorithms. Fortunately, as we now show below, the latency of these computations is not on the critical path and much of the expensive computation happens on the cloud. (In the paper, we have used the term "cloud" as a proxy for server-class computing resources. In

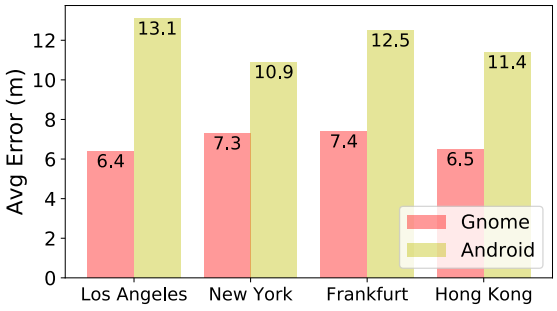

Figure 16: ***Average accuracy in different cities.***

Module	Latency
Mobile: Client-side positioning	77 ms/estimate
Cloud: Street View data retrieval	1.9 s/viewpoint
Cloud: Height adjustment	2.1 s/viewpoint
Cloud: Path inflation calculation	28 s/candidate

Table 1: Processing latency measurement

fact, our experiments were carried out on a 12-core 2.4Ghz Xeon desktop with 32GB of memory and running Ubuntu 16.04).

Table 1 summarizes the processing latency of several components of Gnome. The critical path of position estimation (which involves scoped refinement based candidate search) on the smartphone incurs only 77ms. Thus, Gnome can support up to a 13Hz GPS sampling rate, which is faster than default location update rates (10 Hz) on both Android and iOS. Retrieval of depth data (a few KBs) and a panoramic image (about 450KB) for each viewpoint (recall that Gnome samples these at the granularity of 5m) takes a little under 2s, while adjusting the height of planes at each viewpoint takes an additional 2s. By far the most expensive operation (29s) is computing the path inflation for each candidate position: this requires ray tracing for all points in a hemisphere around the candidate position. This also explains why we only compute single reflections: considering multiple reflections would significantly increase the computational cost. The candidate positions are arranged in a 2m × 2m grid, and it takes about 39 minutes to compute the path inflation maps for a 1-km street, or about 17 hours for the downtown area of Los Angeles which has 26.8km of roads. It is important to remember that Gnome's path inflation maps are compute-once-use-often: a path inflation map for an urban canyon in a major city need only be computed once. We choose the 2×2 candidate scale because it achieves the best balance between the accuracy and runtime latency.

Power consumption. In Android 7.0, the battery option in "Settings" provides detailed per-hour power reports for the top-5 highest power usage apps. Gnome is implemented as an app, so we obtain its power consumption by running it for three hours. While doing so, we run the app in the foreground

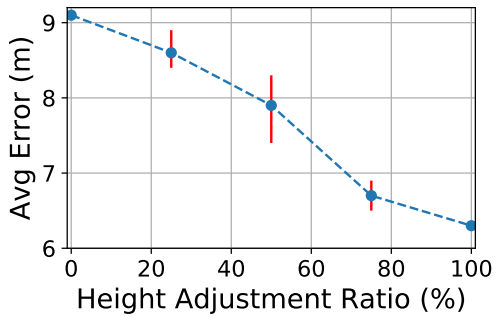

Figure 17: ***Accuracy with different level of height adjustment.***

with screen brightness set to the lowest. We compare Gnome's power consumption with that of the default Android location API (implemented as a simple app), and our results are averaged over multiple runs. During our experiment, Gnome app consumes 151 mAh and is 31% higher than the default Android location API, which consumes 112 mAh. Most of the additional energy usage is attributable to the UBlox library that computes adjusted locations. All the four phones we tested have batteries larger than 2700mAh, so Gnome would deplete the battery by about 5.5% every hour if used continuously. Our implementation is relatively unoptimized (Gnome is implemented in Python), and we plan to improve Gnome's energy efficiency in future work.

Storage Usage. We have generated path inflation maps for the downtown area of Los Angeles, a $3.9km^2$ area. The total length of road is 26.8km and the average road width (road width is used for sampling candidate positions) is 21.3m. In this area, there are 2531 Street View viewpoints and Gnome generates 16390 candidate positions at a 2m × 2m granularity. The total size of the path inflation maps for Los Angeles is 340 MB in compressed binary format. While this is significant, smartphone storage has been increasing in recent years, and we envision most users loading path inflation maps only for downtown areas of the city they live in.

4.3 Evaluating Gnome components

Each component of Gnome is crucial to its accuracy. We evaluate several components for Los Angeles.

Height adjustment. In Los Angeles, there are nearly 15,000 planes of which about 30% need height adjustment. To understand how the height adjustment affects the final localization accuracy, we compute the accuracy when Gnome selects different random subsets of planes for which to perform height adjustment. In Figure 17, the x-axis represents the fraction of planes for which height adjustment is performed. In our experiment, for each data point, we repeated the random selection five times, and the figure shows the maximum and minimum values for each data point. Height adjustment is responsible for up to a 3m reduction in error.

Finally, our height estimates themselves can be erroneous. We compared our estimated height with the actual height

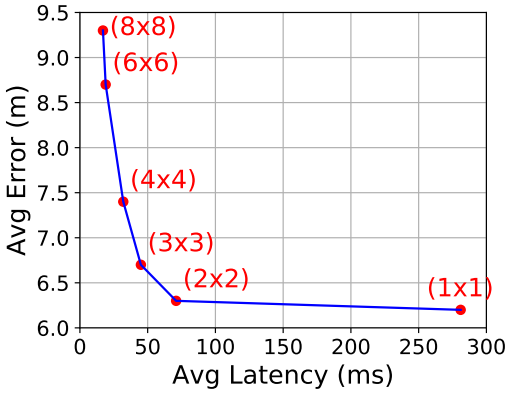

Figure 18: ***Latency vs Accuracy with different grid sizes.***

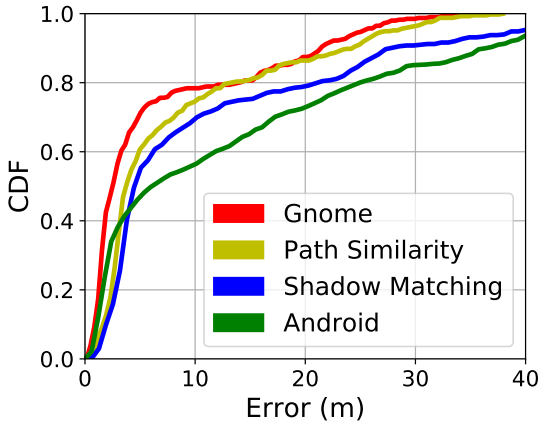

Figure 19: ***Accuracy of different candidate ranking approaches.***

for 50 randomly selected buildings in Los Angeles, whose heights are publicly available. Our height estimates are correct to within 5% at the median and within 14% at the 90th percentile. Our position estimation accuracy is largely due to the fact that we are able to estimate heights of buildings quite accurately. There are two causes for the height estimation error: (1) the inaccurate sky detection could recognize the building's top as part of the sky, which makes the estimated height lower than the actual value, and (2) massive obstacles like trees and taller buildings behind the target plane will cause larger height measurement. The largest errors in both cases are 16% and 23%.

Candidate selection and ranking. Candidate position granularity can also impact the error. Gnome uses a 2m × 2m grid. Using a coarser 8m × 8m grid would reduce storage requirements by a factor of 16 and could reduce the processing latency on the smartphone. Figure 18 captures the tradeoff between candidate granularity, accuracy and processing latency. As the figure shows, candidate selection granularity can significantly impact accuracy: an 8m × 8m grid would add almost 3m error to Gnome while reducing processing latency by about 50ms. Our choice of granularity is at the point of diminishing returns: a finer grid of 1m × 1m would more than triple processing latency while reducing the error by about 10cm.

Position accuracy is also a function of the candidate search strategy. Prior work has considered two different approaches. The first [40, 47, 54] is based on path similarity. This line of work uses ray-tracing to simulate the signal path and calculate the difference between the simulated one and the actual travel distance calculated by GPS module. The candidate whose path difference is least is selected as the output. The second approach is called shadow matching [23, 36, 62]. It uses satellite visibility as ranking indicator and assumes that NLOS signal always have worse carrier-to-noise density C/N_0 than LOS signal. It uses C/N_0 to classify each satellite's visibility at the ground truth point and compares that with simulated (from 3D models) satellite visibility at each candidate point. The point with the highest similarity is the estimated position. Figure 19 shows the error distribution in Los Angeles: we include the Android error distribution for calibration. While all approaches improve upon Android, Gnome is distributionally better than the other two approaches. The crucial difference between Gnome and path similarity is that Gnome revises the candidate positions using the path inflation maps, and this appears to give it some advantage. Gnome is better than shadow matching because carrier-to-noise-density is not a good predictor of NLOS signals (§3). In addition to better localization, Gnome is more practical than these prior approaches in three ways. First, these approaches use proprietary 3D models for ray-tracing, which may not be widely available for many cities. In Gnome, we use Google Street View which is available for most cities (as shown in our evaluation). Second, these approaches work offline and do not explore efficient online implementations. For instance, path simulation in [40, 47, 54] can take up to 2 sec on a desktop, rendering them impractical for mobile devices. Finally, these approaches rely on an external GPS receiver (UBlox) while Gnome is implemented on Android phones.

Kalman filter. We also disabled the Kalman filter in Gnome to evaluate its contribution to the final accuracy. As Figure **??** shows, Kalman filter improves accuracy by 1.2m for the stationary case and about 0.5m in the other two scenarios. The image on the right shows how the Kalman filter results in a smoother trace closer to the ground truth.

5 RELATED WORK

NLOS Mitigation. Prior work has explored NLOS mitigation. These all differ from Gnome along one or more of these dimensions: they either require specialized hardware, use simplistic or proprietary 3D models, or have not demonstrated scalability to smartphones. NLOS signals are known

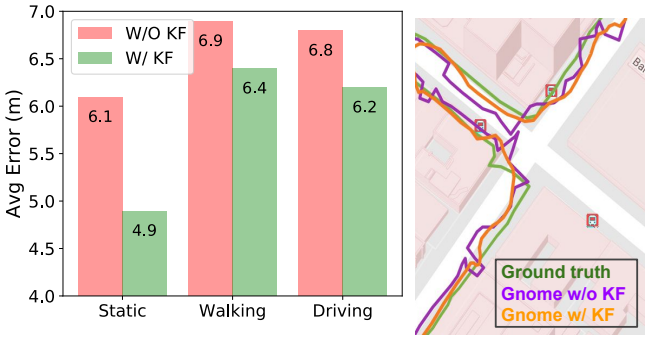

Figure 20: *Effectiveness of Kalman filter.*

to be the major cause of GPS errors in urban canyons [8, 45]. Other work [35] has shown that ray-tracing a single reflection generally works as well or better than ray-tracing multiple reflections. Early work [26] uses the width of a street and the height of its buildings to build a simple model of an entire street as consisting of two reflective surfaces. This model deviates from reality in most modern downtown areas, where building heights vary significantly. To overcome this drawback, other work proposes specialized hardware to build models of reflective surfaces, including stereo fish-eye cameras [48], LiDAR [14], or panoramic cameras [61].

A long line of work has explored using proprietary 3D models to compute the path inflation, or simply to determine whether a satellite is within line of sight or not. One branch of this research uses 3D models to determine and filter out NLOS satellites [34, 46, 51]. However, as we show, in our dataset, nearly 90% of the GPS readings see fewer than 4 satellites, and removing NLOS satellites would render those readings unusable. A complementary approach has explored, as Gnome does, correcting the NLOS path inflation. Closest to our work is the line of work [40, 47, 54] that uses candidate positions like Gnome does, but estimates the ground truth position using path similarity (§4). As we have shown earlier, this approach performs worse than ours. A second line of work [23, 36, 62] assumes that NLOS satellites generally have lower carrier-to-noise density than LOS satellites: Figure 4 shows that this may not hold in general and §4 shows that this approach also does not perform well.

Building height computation. Several pieces of work have used techniques to build 3D models of buildings from a series of 2D images, using a technique called structure-from-motion (SfM, [50]). Our approach uses 3D models made available from LiDAR devices mounted on Street View scanning vehicles. Other work has used complementary methods to obtain the height of buildings. Building height information is publicly available from government websites [12, 16] or 3rd party services [17, 18], and some work has explored building 3D models using images and building height information [37, 55]. However, these datasets have spotty coverage across the globe. Recognizing this, other work [29–31, 60] estimates building heights using Synthetic Aperture Radar (SAR) data generated with remote sensing technologies. This data also has uneven coverage. Finally, one line of work [58, 59] analyzes the shape and size of building shadow to estimate building height. This approach needs the entire shadow to be visible on the ground with few obstructions. In downtown areas, the shadow of tall buildings fall on other buildings, and this approach cannot be used.

Complementary approaches to improving localization accuracy. In recent years, the mobile computing community has explored several complementary ways to improve location accuracy: using the phone's internal sensors to track the trajectory of a user [53]; using cameras [4] or fusing GPS readings with sensors, dead-reckoning, map matching, and landmarks to position vehicles [28, 43]; using Wi-Fi access point based localization [56, 57] as well as camera-based localization [65]; and crowdsourcing GPS readings [24] to estimate the position of a target. Other work [38, 39] has explored accurate differential GPS systems which require satellite signal correlation across large areas and don't work well in downtown areas. GPS signals have also been used for indoor localization [49], and other work has explored improving trajectory estimation [63, 64] by using map-matching to correct GPS readings. While map matching works well for streets, it is harder to use for pedestrians. In contrast to this body of work, Gnome attacks the fundamental problem in urban canyons: GPS error due to satellite signal reflections.

6 CONCLUSION

In this paper, we have described a practical and deployable method for correcting GPS errors resulting from non-line-of-sight satellites. Our approach uses publicly available 3D models, but augments them with the height of buildings estimated from panoramic images. We also develop a robust method to estimate the ground truth location from candidate positions, and an aggressive precomputation strategy and efficient search methods to enable our system to run efficiently entirely on a smartphone. Results from cities in North America, Europe, and Asia show 6-8m positioning error reductions over today's highly optimized smartphone positioning systems. In the future, we plan to further optimize the energy usage and storage requirements of our implementation, and test it more extensively in urban canyons in other major cities of the world.

Acknowledgements. We thank our shepherd Junehwa Song and the anonymous referees for their feedback, which greatly improved the quality of the paper's presentation.

REFERENCES

[1] Gpx Logger for Android. https://play.google.com/store/apps/details?id=com.eartoearoak.gpxlogger&hl=en.
[2] Google Earth Pro. https://www.google.com/earth/desktop/.
[3] Kinetic Lite Gps for Ios. https://itunes.apple.com/us/app/kinetic-lite-gps/id390946616?mt=8.
[4] Mobileye Auto Mapping Technology. http://gpsworld.com/gm-volkswagen-to-use-mobileye-auto-mapping-technology/.
[5] Site Modeling in Sketchup From Google Earth. https://youtu.be/nVhM3IYMF8o.
[6] Gnome's Code. http://github.com/USC-NSL/Gnome.

[7] Extract Depth Maps From Google Street View. https://0xef.wordpress.com/2013/05/01/extract-depth-maps-from-google-street-view/, 2017.

[8] Gps and Gnss for Geospatial Professionals. https://www.e-education.psu.edu/geog862/, 2017.

[9] Kalman Filter. https://en.wikipedia.org/wiki/Kalman_filter, 2017.

[10] Raw Gnss Measurements on Android. https://developer.android.com/guide/topics/sensors/gnss.html, 2017.

[11] Gps Module with Dead Reckoning. http://gpsworld.com/skytraq-gnss-receiver-module-provides-indooroutdoor-positioning/, 2017.

[12] Building Information of Los Angeles. http://geohub.lacity.org/datasets/, 2017.

[13] Map Matching. https://blog.mapbox.com/matching-gps-traces-to-a-map-73730197d0e2, 2017.

[14] Position Estimation using Non-line-of-sight Gps Signals. http://gpsworld.com/innovation-position-estimation-using-non-line-of-sight-gps-signals, 2017.

[15] Nmea Protocol. https://en.wikipedia.org/wiki/NMEA_0183, 2017.

[16] Building Information of New York. http://www1.nyc.gov/nyc-resources/service/2266/property-deeds-and-other-documents, 2017.

[17] Open Street Map. https://www.openstreetmap.org/, 2017.

[18] Propertyshark: Real-estate Data Source. https://www.propertyshark.com/mason/, 2017.

[19] Ray Tracing. https://www.cs.unc.edu/~rademach/xroads-RT/RTarticle.html, 2017.

[20] Zed Stereo Camera. https://www.stereolabs.com/zed/, 2017.

[21] Google Streetview. https://www.google.com/streetview/, 2017.

[22] Wifi Based Localization on Android. https://gigaom.com/2012/11/29/android-app-toggles-wi-fi-based-on-location-no-gps-needed, 2017.

[23] Mounir Adjrad and Paul D Groves. Enhancing Conventional Gnss Positioning with 3d Mapping Without Accurate Prior Knowledge. The Institute of Navigation, 2015.

[24] Ioannis Agadakos, Jason Polakis, and Georgios Portokalidis. Techu: Open and Privacy-preserving Crowdsourced Gps for the Masses. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 475–487. ACM, 2017.

[25] Dragomir Anguelov, Carole Dulong, Daniel Filip, Christian Frueh, Stéphane Lafon, Richard Lyon, Abhijit Ogale, Luc Vincent, and Josh Weaver. Google Street View: Capturing the World At Street Level. *Computer*, 43(6):32–38, 2010.

[26] David Bétaille, François Peyret, and Miguel Ortiz. How to Enhance Accuracy and Integrity of Satellite Positioning for Mobility Pricing in Cities: The Urban Trench Method. In *Transport Research Arena TRA 2014*, page 8p, 2014.

[27] Andria Bilich and Kristine M Larson. Mapping the Gps Multipath Environment using the Signal-to-noise Ratio (snr). *Radio Science*, 42(6), 2007.

[28] Cheng Bo, Xiang-Yang Li, Taeho Jung, Xufei Mao, Yue Tao, and Lan Yao. Smartloc: Push the Limit of the Inertial Sensor Based Metropolitan Localization using Smartphone. In *Proceedings of the 19th annual international conference on Mobile computing & networking*, pages 195–198. ACM, 2013.

[29] Dominik Brunner, Guido Lemoine, and Lorenzo Bruzzone. Extraction of Building Heights From Vhr Sar Imagery using an Iterative Simulation and Match Procedure. In *Geoscience and Remote Sensing Symposium, 2008. IGARSS 2008. IEEE International*, volume 4, pages IV–141. IEEE, 2008.

[30] Dominik Brunnera, Guido Lemoinea, and Lorenzo Bruzzoneb. Building Height Retrieval From Airborne Vhr Sar Imagery Based on an Iterative Simulation and Matching Procedure. In *Proc. of SPIE Vol*, volume 7110, pages 71100F–1, 2008.

[31] François Cellier and Elise Colin. Building Height Estimation using Fine Analysis of Altimetric Mixtures in Layover Areas on Polarimetric Interferometric X-band Sar Images. In *Geoscience and Remote Sensing Symposium, 2006. IGARSS 2006. IEEE International Conference on*, pages 4004–4007. IEEE, 2006.

[32] Christopher J Comp and Penina Axelrad. Adaptive Snr-based Carrier Phase Multipath Mitigation Technique. *IEEE Transactions on Aerospace and Electronic Systems*, 34(1):264–276, 1998.

[33] Christopher Drane, Malcolm Macnaughtan, and Craig Scott. Positioning Gsm Telephones. *IEEE Communications magazine*, 36 (4):46–54, 1998.

[34] Vincent Drevelle and Philippe Bonnifait. Igps: Global Positioning in Urban Canyons with Road Surface Maps. *IEEE Intelligent Transportation Systems Magazine*, 4(3):6–18, 2012.

[35] Rudy Ercek, Philippe De Doncker, and Francis Grenez. Nlos-multipath Effects on Pseudo-range Estimation in Urban Canyons for Gnss Applications. In *Antennas and Propagation, 2006. EuCAP 2006. First European Conference on*, pages 1–6. IEEE, 2006.

[36] Paul D Groves. Shadow Matching: A New Gnss Positioning Technique for Urban Canyons. *The journal of Navigation*, 64(3): 417–430, 2011.

[37] Tao Guo and Yoshifumi Yasuoka. Snake-based Approach for Building Extraction From High-resolution Satellite Images and Height Data in Urban Areas. In *Proceedings of the 23rd Asian Conference on Remote Sensing*, pages 25–29, 2002.

[38] Will Hedgecock, Miklos Maroti, Janos Sallai, Peter Volgyesi, and Akos Ledeczi. High-accuracy Differential Tracking of Low-cost Gps Receivers. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 221–234. ACM, 2013.

[39] Will Hedgecock, Miklos Maroti, Akos Ledeczi, Peter Volgyesi, and Rueben Banalagay. Accurate Real-time Relative Localization using Single-frequency Gps. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 206–220. ACM, 2014.

[40] Li-Ta Hsu, Yanlei Gu, and Shunsuke Kamijo. 3d Building Model-based Pedestrian Positioning Method using Gps/glonass/qzss and Its Reliability Calculation. *GPS solutions*, 20(3):413–428, 2016.

[41] LT Hsu and S Kamijo. Nlos Exclusion using Consistency Check and City Building Model in Deep Urban Canyons. In *ION GNSS*, pages 2390–2396, 2015.

[42] Yitao Hu, Xiaochen Liu, Suman Nath, and Ramesh Govindan. Alps: Accurate Landmark Positioning At City Scales. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 1147–1158. ACM, 2016.

[43] Yurong Jiang, Hang Qiu, Matthew McCartney, Gaurav Sukhatme, Marco Gruteser, Fan Bai, Donald Grimm, and Ramesh Govindan. Carloc: Precise Positioning of Automobiles. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 253–265. ACM, 2015.

[44] Angelo Joseph. Measuring Gnss Signal Strength. *Inside GNSS*, 5 (8):20–25, 2010.

[45] Elliott Kaplan and Christopher Hegarty. *Understanding GPS: Principles and Applications*. Artech house, 2005.

[46] Ashwani Kumar, Yoshihiro Sato, Takeshi Oishi, and Katsushi Ikeuchi. Identifying Reflected Gps Signals and Improving Position Estimation using 3d Map Simultaneously Built with Laser Range Scanner. *Rapport technique, Computer Vision Laboratory, Institute of Industrial Science, The University of Tokyo*, 2014.

[47] Shunsuke Miura, Li-Ta Hsu, Feiyu Chen, and Shunsuke Kamijo. Gps Error Correction with Pseudorange Evaluation using Three-dimensional Maps. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):3104–3115, 2015.

[48] Julien Moreau, Sébastien Ambellouis, and Yassine Ruichek. Fisheye-based Method for Gps Localization Improvement in Unknown Semi-obstructed Areas. *Sensors*, 17(1):119, 2017.

[49] Shahriar Nirjon, Jie Liu, Gerald DeJean, Bodhi Priyantha, Yuzhe Jin, and Ted Hart. Coin-gps: Indoor Localization From Direct Gps Receiving. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 301–314. ACM, 2014.

[50] Onur Özyeşil, Vladislav Voroninski, Ronen Basri, and Amit Singer. A Survey of Structure From Motion. *Acta Numerica*, 26:305–364, 2017.

[51] Sébastien Peyraud, David Bétaille, Stéphane Renault, Miguel Ortiz, Florian Mougel, Dominique Meizel, and François Peyret. About Non-line-of-sight Satellite Detection and Exclusion in a 3d Map-aided Localization Algorithm. *Sensors*, 13(1):829–847, 2013.

[52] Hang Qiu, Fawad Ahmad, Ramesh Govindan, Marco Gruteser, Fan Bai, and Gorkem Kar. Augmented Vehicular Reality: Enabling Extended Vision for Future Vehicles. In *Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications*, pages 67–72. ACM, 2017.

[53] Nirupam Roy, He Wang, and Romit Roy Choudhury. I Am a Smartphone and I Can Tell My User's Walking Direction. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 329–342. ACM, 2014.

[54] Mohamed Sahmoudi, Aude Bourdeau, and Jean-Yves Tourneret. Deep Fusion of Vector Tracking Gnss Receivers and a 3d City Model for Robust Positioning in Urban Canyons with Nlos Signals. In *Satellite Navigation Technologies and European Workshop on GNSS Signals and Signal Processing (NAVITEC), 2014 7th ESA Workshop on*, pages 1–7. IEEE, 2014.

[55] Shunta Saito, Takayoshi Yamashita, and Yoshimitsu Aoki. Multiple Object Extraction From Aerial Imagery with Convolutional Neural Networks. volume 2016, pages 1–9. Society for Imaging Science and Technology, 2016.

[56] Souvik Sen, Božidar Radunovic, Romit Roy Choudhury, and Tom Minka. You Are Facing the Mona Lisa: Spot Localization using Phy Layer Information. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 183–196. ACM, 2012.

[57] Souvik Sen, Jeongkeun Lee, Kyu-Han Kim, and Paul Congdon. Avoiding Multipath to Revive Inbuilding Wifi Localization. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 249–262. ACM, 2013.

[58] Yang Shao, Gregory N Taff, and Stephen J Walsh. Shadow Detection and Building-height Estimation using Ikonos Data. *International journal of remote sensing*, 32(22):6929–6944, 2011.

[59] VK Shettigara and GM Sumerling. Height Determination of Extended Objects using Shadows in Spot Images. *Photogrammetric Engineering and Remote Sensing*, 64(1):35–43, 1998.

[60] Uwe Soergel, Eckart Michaelsen, Antje Thiele, Erich Cadario, and Ulrich Thoennessen. Stereo Analysis of High-resolution Sar Images for Building Height Estimation in Cases of Orthogonal Aspect Directions. *ISPRS Journal of Photogrammetry and Remote Sensing*, 64(5):490–500, 2009.

[61] Sarab Tay and Juliette Marais. Weighting Models for Gps Pseudorange Observations for Land Transportation in Urban Canyons. In *6th European Workshop on GNSS Signals and Signal Processing*, page 4p, 2013.

[62] Lei Wang, Paul D Groves, and Marek K Ziebart. Smartphone Shadow Matching for Better Cross-street Gnss Positioning in Urban Environments. *The Journal of Navigation*, 68(3):411–433, 2015.

[63] Hao Wu, Weiwei Sun, and Baihua Zheng. Is Only One Gps Position Sufficient to Locate You to the Road Network Accurately? In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 740–751. ACM, 2016.

[64] Hao Wu, Weiwei Sun, Baihua Zheng, Li Yang, and Wei Zhou. Clsters: A General System for Reducing Errors of Trajectories Under Challenging Localization Situations. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):115, 2017.

[65] Danfei Xu, Hernán Badino, and Daniel Huber. Topometric Localization on a Road Network. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 3448–3455. IEEE, 2014.

[66] Cheng Zhang, Wangdong Qi, Li Wei, Jiang Chang, and Yuexin Zhao. Multipath Error Correction in Radio Interferometric Positioning Systems. *arXiv preprint arXiv:1702.07624*, 2017.

[67] Yujie Zhang and Chris Bartone. Multipath Mitigation in the Frequency Domain. In *Position Location and Navigation Symposium, 2004. PLANS 2004*, pages 486–495. IEEE, 2004.