Employing User Feedback for Fast, Accurate, Low-Maintenance Geolocationing*

Ezekiel S. Bhasker Steven W. Brown William G. Griswold
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114
{*ebhasker,sbrown,wgg*}*@cs.ucsd.edu*

Abstract

One way to improve inferences on sensor data is to tune the algorithms through a time-consuming offline procedure. A less expensive, and potentially more accurate method is to use an online procedure based on feedback from users, who often know best what the data means to them. We present a method for user-assisted location inference based on 802.11b wireless signal strengths. A user 'corrects' system geolocations by clicking on a map, recording a 'virtual access point' (VAP) at the selected point for future inferences. A best VAP is selected using simple criteria, including the VAP's creator. This permits using other's VAPs while getting their own if one exists, capturing user-specific behavior. The system is also self-maintaining with respect to changing access point deployments. Indoor experiments show very good accuracy for this simple method.

1. Introduction

Inferring accurate information about an entity from sensor data is fraught with problems stemming from noise in the environment or the data itself. Inferences made from idealized models often can be dramatically inaccurate. A typical solution is to survey the environment for its characteristics. This survey data can be used in at least two ways. One is to build a *de facto* model of the environment for interpreting data gathered later. Another approach is to use the survey to subtract environmental effects that deviate from an ideal mathematical model. Such methods, although effective, are labor intensive and sometimes cannot use the unique characteristics of individual entities in inferences.

We have been investigating solutions to interpreting sensor data for the problem of location inference (or *geolocationing*) of wirelessly networked devices in 802.11b environments. The UC San Diego wireless environment covers several square miles with 1000 access points (APs) and growing, motivating a solution that does not depend on surveys by the location system's administrators.

Taking advantage of the fact that our applications involve human users, we have developed an on-line, incremental, user-based collaborative survey mechanism for improving the accuracy of inferred device locations. For the user, the mechanism is simple: When a user's inferred location is displayed incorrectly, the user clicks on the correct location and selects a menu option for correcting the location. The system then takes the corrected location and the IDs of the currently visible wireless access points to construct a *virtual access point* (VAP). Future location computations in the vicinity may then be derived from this virtual access point, instead of using the default locationing method.

Similarity-based policies are used to choose a most applicable VAP, including similarity of the signal strengths and the owner of the VAP. The latter can determine a user's likely location (i.e., a location where the user bothered to apply a correction) when more than one location has similar corrections. This situation can arise when the sparseness of the wireless deployment results in many locations manifesting similar signal strength patterns. Similarity-based methods are also used when a correction is added, to determine when an older correction should be deleted.

We have implemented and deployed this simple locationing mechanism in the ActiveCampus ubiquitous computing environment on the UC San Diego campus, and measured its effects in several locales. Without the correction mechanism, the location error is an average of 33 feet indoors, and about 75 feet outdoors. Although still useful to users (e.g., they can find colleagues), the name of the reported location (e.g., a person's office) is rarely correct. Outdoor locations are often reported inside the building in which the wireless access points reside. In our indoor experiments, corrections dramatically reduce these errors, resulting in a correctly-named location about 90% of the time. The results can be improved further when the owner of the correction is used, but it strongly depends on the user's habits.

The following sections discuss related work (Section 2),

*This work is supported in part by an HP University Mobile Technology Solutions gift, support from the California Institute for Telecommunications and Information Technology (Cal-(IT)^2), and the ActiveWeb project, funded by NSF Research Infrastructure Grant 9802219.

present our method for fast, accurate, low-maintenance geolocationing (Section 3), and describe experimental results (Section 4) before a discussion and conclusion.

2. Related Work

There has been much recent research on location detection. Indoor location positioning research includes systems like Active Badge [15], ActiveBat [1], and PinPoint [12], which require the user to wear a transmitter that periodically emits a pulse picked up by a grid of receivers whose positions are known and computes the RF time of flight to determine position. SCADDS also uses acoustic ranging [7]. The Ad Hoc Positioning System makes use of the number of hops for a message to get to receiving stations whose positions are known [11]. The SmartHome uses multi-modal sensing (optical, audio, mobile, embedded, and other sensors) and data fusion to detect a person's location [6, 14]. Hightower and Boriello provide an in-depth survey of these and other approaches [8].

A classical method of geolocationing that promises extreme accuracy is time-delay measurement for discerning distance and angle. Such methods, including GPS, require special hardware or extension of existing standard wireless protocols (e.g., clock synchronization or instantaneous response to pings). Implementation of such a system would be costly to infrastructure deployers and users, and non-portable, since new access point and client card hardware or software has to be installed.

In an 802.11b environment, it is natural to exploit access point (AP) signal strengths as perceived by a device to infer location, as no additional hardware or battery power is required to support it. Therefore, many current algorithms for geolocation concentrate on using the current signal-strength or noise signature to discern location. However, the potential cost-effectiveness of using 802.11b signal strengths can come at the expense of accuracy, because 802.11b operates in the 2.5GHz radio band, whose signals are readily attenuated by line-of-site obstructions, and sometimes reflected [4]. The difference between algorithms is the way in which potentially unreliable signal-strength information is converted into location information, and possibly how the signal-strength information affects future calculations. The two most common forms of location detection are those that employ geometric models and those that store signal strength "maps" in a database, as well as hybrids. These are discussed below in greater detail.

Geometrical algorithms are the more adaptable and maintainable of the two. They only require the locations of wireless network access points, and a function that converts from received signal strength (or alternatively signal-to-noise ratio) to distance from the access point. Since no model of signal propagation can be accurate enough to consider all geometry possibilities, the accuracy of geometric methods is limited. Thirty feet accuracy is typical.

Map-based methods, such as RADAR [3] on the other hand, involve a labor-intensive process of creating a signal-strength map of the system's working area. The map, after being processed offline to produce nearest-neighbor interpolations of signal strengths, serves as a function to convert signal-strengths to locations. This method is costly both in human effort and time, and the movement of access points can require recomputation of the map and its nearest-neighbor interpolations. The benefit is accuracy, measuring ten feet in experiments. Several maps can be created for different times of day or different traffic levels (e.g., human bodies attenuate signals), increasing accuracy to seven feet [2]. Ekahu's proprietary commercial product also employs offline surveys, and is claimed to produce geolocation accuracy about twice that of RADAR [5].

Recently, the Location Stack method was developed as a way of combining location information from multiple sources, for example wireless signals, calendar information, and past behavior [9]. This method enables determining location from a best available source or combining information from multiple sources to disambiguate location information. By incorporating multiple single-source methods it improves upon them, and the Location Stack method improves if any of the methods it includes is improved.

3. User-Assisted Geolocation

Ideally, we would like a geolocation method with both the low cost of a geometric method and the accuracy of a map-based method. One way is to incrementally construct the map on-line, using the behavior of the numerous application users to build the signal-strength map. Our solution has two levels, one geometric, the other mapped. The first, default method is to use a geometric algorithm that incorporates basic facts about the physical environment, such as gross signal propagation characteristics and floor heights. The second, incrementally mapped method remembers users' explicit corrections on locations computed by the default method, and computes locations from them where they are present.

There are numerous challenges to making this method practical, such as developing an appropriate default geometric algorithm, making corrections easy for users, constructing the map efficiently, using the map to geolocate efficiently, accurately geolocating with a suboptimal wireless deployment, and coping with misleading user behavior as well as the movement of access points. We address each of these in turn.

Users are motivated to make corrections by seeing inaccurate information being displayed for them (e.g., miscentering on a map) or about them (e.g., wrong location conveyed to colleagues). In our system, virtually every display conveys the user's location in some fashion, enhancing one's awareness of inaccuracies.

3.1. Baseline Geometric Method

Because the system may go into operation with no corrections, it is the baseline algorithm must deliver acceptable accuracy. For one, users should not have to constantly correct locations. Two, a poor estimated location can complicate the user's process of correction: If the wrong map is displayed, additional clicks are required to navigate to the correct map before the correction can be issued. Third, the user's location should not inexplicably jump as the system switches between using the baseline and mapped methods.

There are two critical elements to accurate geometric geolocationing via signal strengths. The first is the accurate computation of distances from a user device to each visible access point. The second is combining those distances into a single location.

3.1.1. Estimating Distances

Theoretical and empirical models can achieve similar results in idealized situations, and both can behave badly in real environments due to obstructions, reflections, and signal fading. After attempting to work with theoretical models, we fit several curves to empirical measurements in our environment. Both quadratic and linear formulae (relative to dBm[1]) fit our measurements equally well, so we chose the linear formula for its computational simplicity, code simplicity, and stability under the wild swings in signal strength that we would see. Its main weakness is underestimating distances when both the user and the access points are outdoors, where few obstructions are present.

3.1.2. Synthesizing Location

Our attempts to use direct methods of geometric location inference failed. They were either unstable or too mathematically complex. The details of our results revealed two things. First, the placement of the APs in our environment are optimized for coverage and simplicity of administration. Most APs were arranged in a plane, meaning that accurate three-dimensional estimates were essentially impossible, and the power of the direct methods was at least going to waste. Second, as observed above, the weaker a signal was, the less we could depend on the accuracy of the computed distance. This meant that methods that worked directly with distances and treated them all as equally valid would be prone to error.

These problems led us to consider alternate methods that would (a) scale gracefully to a large number of APs and (b) let us readily incorporate physical and logical properties of the environment in the geolocation computation. A simplified variant of hillclimbing[2] geolocation [10], which generates a sequence of location guesses that are checked against a measure of quality (e.g., minimal error), seemed natural.

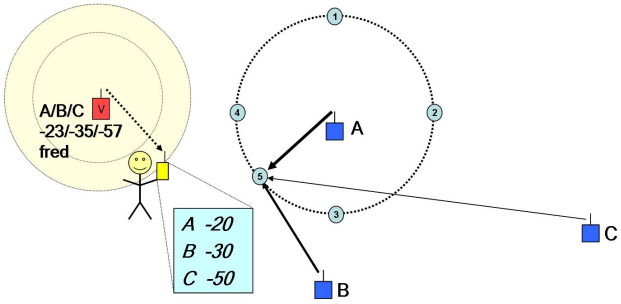

Figure 1. Diagram of how the baseline geometric and correction-based mapping geolocators operate. On the right appear three access points: A, B, and C. A user's PDA observes them with the signal strengths of -23, -35, and -57dBm respectively. The baseline algorithm would choose to geolocate around AP A, as it has the strongest signal. A search around the circle above the floor on which AP A is installed (guesses 1 through 5) converges on guess 5 as having the least error. The user notes that this is the wrong location, and enters a correction, recorded as virtual AP V on the left with its requisite signal-strength signature and owner. A subsequent geolocation based on signal strengths of -20, -30, and -50dBm is computed as an adjustment off of virtual AP V. The circles around virtual AP V connote the applicable range of signal strengths that will be recognized as matching V, with the owner (inner) and without (outer).

Against our criterion of scaling, hillclimbing is advantageous because computing the location is fast (a guess), and computing its error is linear in the number of APs. It is a computation of the aggregate difference between the distances estimated from the observed signal strengths and the distances of the APs from the guessed location ($\sqrt{\sum(observed_i - actual_i)^2}$).

Against our second criterion, the search-based nature of hillclimbing permits us to exploit characteristics of the environment by eliminating some guesses before searching for the best guess. Constraining the search space also reduces the length of the search, further improving scalability. We currently employ three constraints, resulting in a simple and fast algorithm (diagrammed on the right side of Figure 1, with pseudo-code appearing in Figure 2):

1. We restrict guesses to the sphere defined by the estimated distance from the AP with the strongest received signal strength. This generally prevents high errors due to strongly attenuated signals: the signals can only affect the choice of location on the sphere, bounding the error to the size of the sphere.

2. The floor of the location search is chosen before

[1] dBm is a logarithmic measure of signal strength, with 0 dBm $\equiv 1mW$. For 802.11b, values from -90 to -20dBm are typical.

[2] Hillclimbing is a classic AI search-based problem-solving technique that narrows the search by constraining the search's next solution guess to the vicinity of the best results so far [13, ch. 4.3].

```
class Location extends Vector3D;

class APSignalSamples
      HashMap<integer /* MAC Address */,
              integer /* signal str. */>;

record APRelLocation extends Location {
  float aboveFloor;   // Adds loc. info relative
  float userDistance; // to floor and user
}

record CandidateLocation extends Location {
  float error;
}

function locateGeometric(APSignalSamples apSignals)
    returns Location {

  vector<APRelLocation> apRelLocations;
  foreach currentMAC in keys(apSignals) do
    apRelLocations.insert(
      new APRelLocation(
        apLocation(currentMAC),
        apFloorHeight(currentMAC),
        dBmToDistance(apSignals.
                      getValue(currentMAC))));
  end

  // Find closest (best) AP for circle to search.
  APRelLocation bestAP := closest(apRelLocations);
  integer bestZ  :=
            (bestAP.z - bestAP.aboveFloor) +
            DEVICE_HEIGHT;
  integer radius := sqrt(bestAP.userDistance^2 +
                         (bestAP.z - bestZ)^2);

  // Candidate locs around circle, store error.
  vector<CandidateLocation> candidateLocations;
  float angleInc := 2*PI/DIVISIONS_OF_CIRCLE;
  foreach angle in angleInc to 2*PI by angleInc do
    candidate := new CandidateLocation(
                 bestAP.x + radius*cos(angle),
                 bestAP.y + radius*sin(angle);
                 bestZ);
    float errorSquared := 0;
    foreach apInfo in apRelLocations do
      float actualDistance :=
          sqrt((candidate.x - apInfo.x)^2 +
               (candidate.y - apInfo.y)^2 +
               (candidate.z - apInfo.z)^2);
      errorSquared +:=
      (actualDistance - apInfo.userDistance)^2;
    end

    candidate.error := sqrt(errorSquared);
    candidateLocations.insert(candidate);
  end

  // Pick candidates with least error, avg locs.
  return bestXYMatch(candidateLocations);
}

function dBmToDistance(float dbm) returns float {
  return -0.4*(dbm+10); // linear -> conservative
}
```

Figure 2. Pseudo-code for the baseline algorithm.

proper hillclimbing takes place. Our current algorithm chooses the floor on which the AP with the strongest received signal strength resides. This limits "floor jumping" that can occur when locations on two floors have similar errors. It tends to work well because floors are thick and hence strongly attenuate AP signals, typically eliminating APs from other floors from consideration.

3. We further restrict the guesses to a single horizontal circle on the sphere. In particular we expect that a mobile device is generally a few feet or so above the floor in a user's hand or pocket–neither on the floor nor near the ceiling. Thus, the algorithm searches a circle on the sphere that is a three feet above the chosen floor. This avoids deriving locations that are physically implausible.

With these constraints, a few guesses on a single circle are sufficient to produce a plausible set of locations, permitting us to do a simple exhaustive search, stopping after several guesses, rather than a directed one. If there are multiple guesses on a floor with nearly the same minimal error, their average is taken to compute a "best" location for the floor.

This algorithm produces indoor accuracy of about 30 feet in our environment, with larger errors outdoors, where fewer APs are available to contribute to the estimate. The correct floor is chosen about 95% of the time. Thus, the system typically produces a useful map—simplifying correction—but often the wrong detailed location name (e.g., "Rm 4218 at APM" instead of the nearby "Griswold's Office at APM"). Indeed, the planar deployment of APs virtually guarantees such errors.

3.2. Correction-Based Mapped Method

When the baseline method produces an unsatisfactory location, the user is permitted to enter a correction that both changes the user's recorded location and incrementally updates a signal-strength map of the environment for future use. User corrections are supported by a clickable map interface (Figure 3). The user corrects the location by clicking on the correct location, which produces a menu of actions, including a "correct my location" option. Selecting this action captures two pieces of information: the asserted location, and the observed *signal-strength signature* (composed of the IDs and signal strengths of the access points), as seen by the user's network device. Collectively called a *correction*, this data is stored in a database along with the user ID of who created it, the time, and the model of network device that captured the data.

When a user enters an area where corrections have been previously entered, the map-based method geolocates using

Figure 3. The user's perspective on correcting location. The user, looking down at their display, notices either their placement on the display (center) or the named location (upper right) is incorrect (first image). Perhaps the thick walls are attenuating the signal. Clicking the correct location on the display results in displaying an action screen, from which the user selects "correct my location" (second image). This results in creating a virtual access point at the place of correction, and redisplays the user at that location (third image). The next time this user arrives at this location and the same signal-strength pattern is observed, the VAP will be used in computing the user's location.

these corrections. There are three issues that need to be addressed to achieve an accurate, stable, efficient, and scalable solution. How are corrections selected for use in geolocationing? How are locations computed using corrections? How are corrections added and maintained in the database?

3.2.1. Policies for Choosing a Correction

Our mapped method geolocates based on a single, most appropriate correction, if one exists. A best correction is one that is most similar to the current situation with respect to signal-strength signature and owner of the correction. The similarity of signal-strength signature abstractly captures the similarity of the current location and the location of the correction. If there is low similarity in the signatures, then the correction-based method is unlikely to be able to improve on the baseline method. Matching the owner captures the owner's idiomatic behaviors, such as spending a majority of time in a given office versus an adjacent one that might contain a similar correction. As discussed above, this is likely because of both symmetric deployments of access points and signal fading.

To identify a best correction, the geolocation algorithm works as follows (pseudo-code in Figure 4):

1. Match the current signal-strength signature against correction signatures in the database, sorting them by their distance (in signal-strength space) from the current signature. All corrections matching at least one access point in the current signature are selected. A missing access point is treated as having negligible signal strength (e.g., -92dBm).

2. Select the correction with the least distance.

3. Select the user-owned correction with the least distance.

4. If the user-owned correction's standard deviation is less than 30dB, it is returned, otherwise if the absolute best is less than 50dB, it is returned, otherwise, the baseline value is returned.

The thresholds in step 4 were determined through trial and error. A higher value would widen the applicable range of corrections, but perhaps lower the accuracy of corrections matched in the upper part of the range.

In establishing a policy for selecting one's own correction over someone else's, we are attempting to determine whether one's current location is "within range" of the correction—a certain number of feet—and hence capable of capturing idiomatic behavior. Not knowing the user's location yet, we use a correction's distance in signal space from the currently observed signature as a proxy for physical distance.

3.2.2. Computing Location Using a Correction

The device's location is computed as an adjustment from the selected correction's location, based on the difference between the device's current signal-strength signature and the selected correction's signature. As this is akin to how we geolocate using access points, a correction can be conceptualized as a *virtual access point* that acts as an (improved) locationing proxy for the access points that comprise it (conceptually diagrammed on the left side of Figure 1, pseudo-code in Figure 4).

```
record Correction {
  APSignalSamples apSignatures;
  Location location;
}

function locateMapped(integer userID,
                      APSignalSamples apSignals)
     returns Location {

  Vector3D totalLocationAdjustment;
  Correction bestCorrection :=
    getBestCorrection(userID, apSignals);

  if (bestCorrection = null) then
    return locateGeometric(apSignals);

  // Compute adjustment off the chosen correction.
  foreach currentMAC in keys(apSignals) {
    Vector3D locationAdjustment :=
        vectorTowardsAP(
          apLocation(currentMAC),
          apSignals.getValue(currentMAC),
          bestCorrection.apSignature);
    totalLocationAdjustment +:= locationAdjustment;
  }

  Location userLocation :=
      bestCorrection.location +
      totalLocationAdjustment/size(apSignals);
  return userLocation;
}


function getBestCorrection(integer userID,
                           APSignals apSignals)
    returns Correction {

  Array<Correction> corrections[] :=
      getCorrectionsWithSignatureAP(apSignature);

  Correction bestUserCorrection :=
      selectMostSimilarCorrection(corrections,
                                  apSignature,
                                  30dB,
                                  userID);

  if (bestUserCorrection ~= null) then
    return bestUserCorrection;

  Correction bestPublicCorrection :=
      selectMostSimilarCorrection(corrections,
                                  apSignature,
                                  50dB,
                                  NONUSERID);

  return bestPublicCorrection;  // may be null
}
```

Figure 4. Pseudo-code for the correction-based algorithm.

As with our baseline geometric method, the adjustment is based on distances estimated from the signal strengths. In particular, the signal strengths for both the current input and the correction are converted to distances from their access points, and then subtracted to produce differences in distance. These are then converted into difference vectors from the correction towards each access point.

These vectors need to get combined into a single vector representing the adjustment. As discussed earlier, a direct geometric solution is complex, computationally expensive (quadratic in the number of access points), and often doesn't have a solution (because signal fading is not accounted for). A hillclimbing search would have to be more general than our simple baseline geolocation case, resulting in many more iterations, more complexity in choosing the next guess, and a more complex termination case.

Given that the choice of the correction has likely pegged the user's location quite accurately, we simply sum the vectors and divide the result by the number of vectors (i.e., compute the average vector). The resulting approximate difference vector is then added into the correction's location to produce a final adjusted location. This approximation likely *underestimates* the resulting distance—it's the average distance, which is often less than the net distance—and so is essentially *conservative* by keeping all adjustments in the vicinity of the correction. This bias avoids "bad" adjustments as a result of from signal fading.

3.2.3. Policies for Adding & Maintaining Corrections

Due to an incorrect correction or symmetries in the environment's access point deployment, two corrections could have similar signal-strength signatures but rather different locations. Thus, a device's location could "bounce" between their locations in adjacent geolocation events, causing confusion amongst users. In many cases, use of a correction's owner in selecting a best correction will avoid these problems. They remain, however, if neither correction is owned by the user. Therefore, to protect against such instability in location computation, we have developed the following policies when adding a new correction to the correction database:

1. If the new correction is similar to existing corrections added by other users, these similar corrections are marked *private*. Private corrections are only visible to the users who added them. Thus, the new correction essentially replaces these private ones for non-owners. An important benefit of this policy is that malicious behavior is addressed relatively easily–only one user has to 'correct' the bad correction to to hide it, although the malicious user is still geolocated by the bad correction.

2. If the new correction is similar to existing corrections added by the same user, these existing corrections are

deleted. This policy permits users to remediate a miscorrection or obsolete correction (e.g., the user's office has moved, coincidentally to a location with a similar signal-strength signature). This policy has the added benefit of regulating the size of the correction database.

3. The new correction is added to the correction database and marked *public*.

The preference given to new corrections over old ones builds in a natural implementation of obsolescence for corrections.

These policies require a threshold for determining the similarity of correction signal-strength signatures. Our experiments have shown that a standard deviation of 10dB permits sufficiently tight packing of corrections without introducing too much ambiguity into which corrections might apply. Because of the wide variance in received signal strengths a stationary wireless device can see over time, we initially thought this value should be higher. However, it turns out that the widest variances are also the least frequent to occur. If a correction is made on one of these outlying corrections, it will still be deleted or privatized if those using the system provide corrections within the 10dB threshold.

An access point may be moved from time to time. When such a change is recorded in the access point database, any corrections that depend on that access point's old location are deleted. This is trivial to implement since corrections are already looked up by an access point's unique "MAC" address. A more user-friendly policy would be to determine how to update the affected corrections, if the move is small.

3.2.4. Discussion

The algorithms and retrieval methods described here are conceptually and computationally simple, and are governed by just a few parameterizable policies. This yields an implementation that is easy to implement correctly, fast, portable, and easy to customize.

The entire geolocator system is under 700 lines of code. The baseline geometric geolocator is implemented in 339 lines of C++, and about 50 lines of PHP code. The correction-based locator brings the C++ count up to 399 lines and the PHP count up to 276 lines.

The computational complexity of the baseline algorithm is linear: the product of the number of visible access points (usually a few and never greater than a dozen) and the number of guesses, no greater than 8 in our current implementation. The computational complexity of the correction-based method is on the order of the number of corrections (virtual access points) within the selectivity threshold (i.e., 50dB) and the number of access points that comprise the correction signature. In the current implementation, the practical bottleneck is the database retrieval of the observed or applicable access points. A mechanism for caching previous (virtual) access point query results would dramatically reduce the number of required database queries, especially for relatively stationary users.

Different environments or applications might benefit from different thresholds. Most policies for selection, prioritization, privatization and deletion can be easily customized by changing the threshold or priority levels. Some policy changes could require writing new code, such as using location proximity in addition to signal similarity in deleting corrections. Section 5 discusses a range of possible extensions.

4. Experimental Results

Our method is error-driven in that users are prompted to enter corrections by the display of inaccurate information on their device. In this sense, the method can be no better than the users, but also needn't be: if a location is not sufficiently inaccurate, from the user's perspective, to motivate a correction, then it is not really inaccurate. In our application, the threshold is being able to assess whether someone is nearby, and if so, find them easily by walking around the indicated area. As a proxy for these hard-to-assess qualitative measures, we use the correctness of the reported *location name*. We also report rather typical distance error measurements, but these should not be taken seriously, as both our corrections and sample set are derived from scenarios of use rather than exhaustive measurements in the environment.

Consequently, we performed several experiments mimicking a variety of user-like scenarios. Corrections were assessed against data representing both revisits to the offices and nearby, by both visitors and those who own the corrections. Several of these experiments use the same raw data, but change the owners of the corrections. Our experiments involved corrections placed in advance in each of 17 offices on the fourth floor of our building. The building has wireless on all floors, predominantly older Lucent access points. Our measurements were made with a Windows 2000 laptop with a Lucent Orinoco wireless card. In this space, the mean error of the *baseline* method is 33 feet, with the maximum error being about double that distance. Qualitatively, the correct location name is rarely reported, but is often still useful in tracking someone down, especially if the user knows that person (Figure 5, white bar, 0 value in the "correct location category").

In these experiments, we expected there to be occasionally large errors due to the selection of the "wrong" correction, caused either by signal-fading effects on the data sample or the correction itself. We expected correction ownership to rectify many of the poor selections. Finally, we expected our conservative method of adjusting off of corrections to underadjust, but be adequate.

For the experiment ignoring ownership, 17 corrections were created by randomly selecting one of 10 samples taken at each office location. All 17 corrections were made pub-

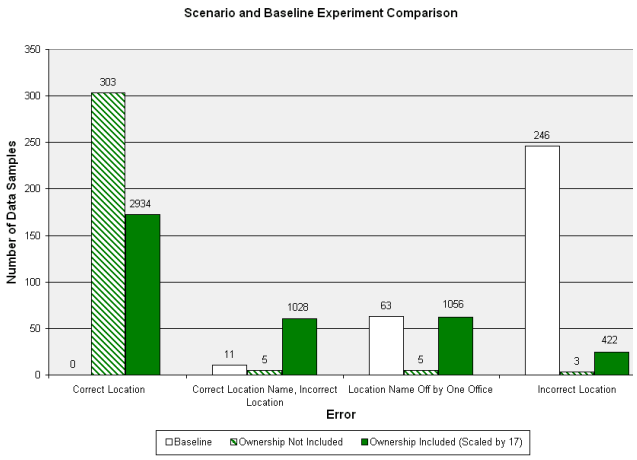

Figure 5. Bar chart comparing our baseline geometric algorithm's accuracy to correction algorithm scenarios that do not and do use correction ownership.

lic regardless of their similarity. These same 10 samples at each office were then used exhaustively as user inputs, with the ownership not used in selecting corrections. This configuration covers two scenarios. One is the preconfiguration by a system administrator (without the benefit of averaging several samples); the other is a zealous user who makes corrections wherever they go, for their own purpose.

Of the 170 office data points, 157 (92%) had no error (correct location name, less than one foot from the actual location), and the remaining 13 data points (8%) had an error sufficient to report the wrong office location, with a mean error of 18.4 feet (Figure 5, hatched bar). Three of those 13 data points matched with a correction diametrically opposite to the correct location relative to the column of access points in the vicinity, with an average error of 51.4 feet. Against the 15 locations in the hallway outside the offices, 146 (97%) matched the correct correction, with a mean error of 7.6 feet. For completeness, replaying the 170 office data points resulted in a mean error of 1.4 feet; the average of the two is 4.5 feet.

To get a sense of how correction ownership affects geolocationing, in particular its ability to exploit idiomatic behavior, we assigned each correction to the owner of the office and privatized corrections within the threshold of similarity (Figure 5, dark bar). We assessed two user scenarios, combined unweighted in the figure. In the first, the office datasets were re-run, with the assumption that each office occupant generated the 10 datapoints in their office. This captures the typical scenario of office occupants spending the vast majority of their time in their own offices. In this scenario, all 170 points selected the right correction and the correct location. In the second scenario, each office occupant wandered among all 17 offices and 15 hallway locations. In this scenario, with 320 samples per user, 5440 samples all told, 2934 samples (54%) produced the correct location, while the remaining 2506 (46%) had a mean error of 49.7 feet, all choosing the "wrong" correction. For completeness, the mean error for all 5440 samples is 23 feet.

In looking more closely at the data from this last scenario, we found that, for each user, an average of 7 corrections in other offices that had been previously been available for geolocation had been privatized, forcing the selection of a less appropriate correction.

5. Discussion

5.1. Data interpretation

What we observe from these experiments is that the basic correction-based mapping method can dramatically improve the accuracy of a simple geometric geolocation method. Computing corrections from an average of several samples does not increase accuracy, thus the extra effort does not seem warranted, at least in our context.

In about 10% of our samples, using correction-based mapping produced anomalous results, caused by geolocating from the "wrong" correction. Our method for overcoming these anomalies, correction ownership, can improve geolocation accuracy in spaces frequently occupied by a user, but dramatically decreases accuracy in other areas. Thus, if a user has a highly constrained routine, then our method of using correction ownership is effective, otherwise it is highly uneffective.

Improvements to the correction-ownership policy are possible. For example, we could drop the privatization/deletion threshold to 5dB. We could also restrict to privatizing/deleting corrections whose locations did not correlate with the owner's correction location. In looking at our data set, reducing the threshold to 5dB would drop the number of privatizations from 7 to 2 on average. Including correction location in privatization decisions would typically prevent 2 nearby corrections from being privatized. We have yet to assess the effects of these alternate policies.

We have yet to formally assess the effectiveness of our method outdoors. In one preliminary experiment, we applied corrections along one wall outside our building where students typically hang out. These corrections were effective in pulling the reported location outside the building and showing an outdoor map rather than the building maps.

In our day-to-day use of our geolocation system, we have had very satisfactory experiences. Our idiomatic patterns typically take us between a laboratory, office, and a conference room, but not among many offices in the same area. Outdoor locationing has proven quite satisfactory, for example correctly placing us in cafe areas when we go out for lunch or coffee. However, we note that sometimes a few corrections have to be entered at a location to achieve stable, predictable geolocationing. This is no doubt due to

the variations incurred by signal fading. We anticipate that the necessity to enter several corrections at a location would be confusing to end users unfamiliar with the properties of wireless communications. An interesting extension of our approach would be to recognize the patterns of variation as belonging to a location, thus helping to better disambiguate two locations that sometimes have the same signal strength.

Finally, we observe that our distance computation algorithm and correction adjustment algorithms, although effective in our experiments and daily use, are quite conservative and could be improved. They might need to be improved in other environments with different application requirements. The direct trigonometric solution to this problem is computationally expensive, quadratic in the number of access points.

5.2. Extensions

Our experiments make some implicit assumptions that do not hold in a production environment. With the concepts of similarity and correction maintenance that we introduced, several improvements on our basic mechanisms and policies are possible.

- *Correction similarity.* The similarity of corrections can be extended to include the model of network device used, accounting for properties in device implementation (e.g., antenna) that would cause it to perceive different signal strengths than other device models at the same location.[3]

  Similarity could also include the time of day, which could account for variance in an area's resident population, which can affect perceived signal strengths. This is akin to RADAR's use of different signal-strength maps at different times of the day [2]. These properties are currently captured but not used in our implementation. Introducing these properties into correction selection raises the question of prioritizing the properties and defining what it means to match. Because ownership of the correction provides very valuable information about a user's typical locations, these would remain primary in our scheme. Second would be device, since the sensitivity amongst network devices varies widely. Third would be time, for although time effects could be significant, they are also the most unreliable and difficult for the system to use effectively. For example, class schedules are different on Monday/Wednesday/Friday than Tuesday/Thursday, holidays and exam schedules vary year to year, etc.

- *Rejection of bad corrections.* As a guard against user malice and mistakes, a correction could be rejected outright if it was physically impossible, for example if the specified location of the correction is beyond the range of any or all the APs seen by the network device at the time.

- *Correction averaging.* Another possible extension is to use multiple corrections in computing one's location. This could be useful when multiple equally plausible VAPs are retrieved from different locations. Placing the user at the weighted center of these locations could reduce the maximum error the user sees. To preserve the other benefits of our method, this averaging should not violate our similarity criteria. To make averaging more effective, the threshold for deleting older corrections could be raised to provide more averaging options.

[3] Another approach for dealing with differing device implementations that would include access points is to maintain a database of device characteristics, and then adjust their captured signals to represent a common reference.

6. Conclusion

Inferring accurate information about an entity from sensor data is fraught with problems stemming from noise in the environment or the data itself. Offline survey methods can be effective in correcting for this noise, but they are costly, become obsolete as the environment changes, and do not capture user-specific characteristics.

In our investigations of this problem, we have developed an online, incremental, user-based collaborative survey technique for accurate location inference in 802.11b wireless environments. For the user, the mechanism is straightforward. When a user's location is displayed incorrectly, the user clicks on the correct location and selects a menu option for correcting the location. The system then takes the corrected location and the IDs of the currently visible wireless access points to construct a "virtual access point". Future location computations in the vicinity may then be derived from this virtual AP, instead of using the default locationing method.

Similarity-based policies are used to choose a most applicable correction, including not only similarity of the signal strengths, but also the owner of the correction. This is useful because a sparse wireless deployment may result in very different locations manifesting similar signal signatures. If two users make corrections with the similar signatures at different locations, the system will locate each at their likeliest location—where they entered their corrections. This permits a user's corrections to reflect their idiomatic movements, while still benefiting from others' corrections when one's own is lacking. When two corrections greatly overlap or conflict, the older correction is either made private to its creator or deleted altogether if the same user created both corrections. In this way, the system is self-maintaining with respect to both changes in the environment and data bloat.

The method is simple to implement, requiring only 700 lines of C++ and PHP code. Our measurements show that

the quality of locationing in our environment compared to our baseline geometric method is increased qualitatively from almost never reporting a correct location name (e.g., a person's office) to reporting it correctly about 90% of the time. The use of correction ownership, while capable of improving location accuracy in one's typical locations, can decrease quality at nearby locations. Further work is required in tuning the ownership policies. The method has worked quite well for us, as we have fairly structured routines.

Acknowledgments. We thank Tan Minh Truong, David Hutches, Robert Boyer, Anand Balachandran, Marvin McNett, Geoff Voelker, and Stefan Savage for their assistance and ideas. We also thank UCSD's Facilities office, Network Operations, and Academic Computing and Telecommunications for their their time, expertise, and resources. In particular we thank Roger Andersen, Robert Clossin, Kirk Belles, Jim Madden, Don McLaughlin, Lou Forbis, and Elazar Harel. Finally, we thank Intel's Network Equipment Division for donating network processors and Symbol Technologies for their software technical support.

References

[1] M. Addlesee, R. Curwen, S. Hodges, J. Newman, P. Steggles, A. Ward, and A. Hopper. Implementing a sentient computing system. *IEEE Computer*, 34(8):50–56, 2001.

[2] P. Bahl, A. Balachandran, and V. Padmanabhan. Enhancements to the RADAR user location and tracking system. Technical Report MSR-TR-2000-12, Microsoft Research, February 2000.

[3] P. Bahl and V. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *Proceedings of IEEE INFOCOM*, volume 2, pages 775–784, March 2000.

[4] J. Beutel. Geolocation in a picoradio environment. Masters Thesis, Dept. of Electrical Engineering, ETH Zurich, and Dept. of Electrical Engineering and Computer Science, UC Berkeley, 2000.

[5] Ekahu web site. http://www.ekahau.com/.

[6] I. A. Essa. Ubiquitous sensing for smart and aware environments. *IEEE Personal Communications*, 2000.

[7] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building efficient wireless sensor networks with low-level naming. In *Proceedings of the Symposium on Operating Systems Principles*, pages 146–159, Chateau Lake Louise, Banff, Alberta, Canada, October 2001. ACM.

[8] J. Hightower and G. Borriello. Location systems for ubiquitous computing. *IEEE Computer*, 34(8):57–66, 2001.

[9] J. Hightower, B. Brumitt, and G. Borriello. The Location Stack: A layered model for location in ubiquitous computing. In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2002)*, pages 22–28, Callicoon, NY, June 2002. IEEE Computer Society Press.

[10] J. Hightower, R. Want, and G. Borriello. SpotON: An indoor 3D location sensing technology based on RF signal strength. Technical Report UW CSE 00-02-02, University of Washington, Department of Computer Science and Engineering, Seattle, WA, February 2000.

[11] D. Niculescu and B. Nath. Ad-hoc positioning system. In *Proceedings of IEEE GLOBECOM*, November 2001.

[12] RF Technologies. PinPoint local positioning systems. URL, 2002. http://www.rftechnologies.com/pinpoint/.

[13] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, 2002.

[14] J. M. Sanders. Sensing the subtleties of everyday life. *Research Horizons*, Winter, 2000.

[15] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The Active Badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, January 1992.