Practical Architectures for Fused Visual and Inertial Mobile Sensing

by

Puneet Jain

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Romit Roy Choudhury, Supervisor

_____
Landon Cox, Chair

_____
Bruce Maggs

_____
Justin Manweiler

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2015

ABSTRACT

Practical Architectures for Fused Visual and Inertial Mobile Sensing

by

Puneet Jain

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Romit Roy Choudhury, Supervisor

_____
Landon Cox, Chair

_____
Bruce Maggs

_____
Justin Manweiler

An abstract of a dissertation submitted in partial fulfillment of the requirements
for
the degree of Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2015

Abstract

Crowdsourced live video streaming from users is on the rise. Several factors such as social networks, streaming applications, smartphones with high-quality cameras, and ubiquitous wireless connectivity are contributing to this phenomenon. Unlike isolated professional videos, live streams emerge at an unprecedented scale, poorly captured, unorganized, and lack user context. To utilize the full potential of this medium and enable new services on top, immediate addressing of open challenges is required. Smartphones are resource constrained – battery power is limited, bandwidth is scarce, on-board computing power and storage is insufficient to meet real-time demand. Therefore, mobile cloud computing is cited as an obvious alternative where cloud does the heavy-lifting for the smartphone. But, cloud resources are not cheap and real-time processing demands more than what the cloud can deliver.

This dissertation argues that throwing cloud resources at these problems and blindly offloading computation, while seemingly necessary, may not be sufficient. Opportunities need to be identified to streamline big-scale problems by leveraging in device capabilities, thereby making them amenable to a given cloud infrastructure. One of the key opportunities, we find, is the cross-correlation between different streams of information available in the cloud. We observe that inferences on a single information stream may often be difficult, but when viewed in conjunction with other information dimensions, the same problem often becomes tractable.

To Sharad Kumar Jain

Contents

List of Tables

List of Figures

List of Abbreviations and Symbols

Abbreviations

3D	Three dimensional
ACM	Association for Computing Machinery
APP	Smartphone Application
AR	Augmented Reality
BF	Bloom Filter
CDF	Cumulative Distribution Function
CPU	Central Processing Unit
CSL	Coordinated Science Laboratory, University of Illinois, Urbana-Champaign
CUDA	Compute Unified Device Architecture, a NVIDIA architecture GPU Programming Language
CV	Computer Vision
FLANN	Fast Library for Approximate Nearest Neighbors
GB	Gigabyte; 2^{30} bytes, roughly one billion bytes
GPS	Global Positioning System
GPU	Graphics Processing Unit
HD	High Definition
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IR	Infrared Sensor
IRB	Institutional Review Board
LP	Linear Program
LSH	Locality Sensitive Hashing

LTE	Long-Term Evolution
LoS	Line-of-Sight
MB	Megabyte; 2^{20} bytes, roughly one million bytes
ms	Millisecond; one one-thousandth of one second $(1/1000)$
mW	Milliwatt; one one-thousandth of one watt $(1/1000)$
NP	Non-deterministic Polynomial-time
OpenCV	Open Source Computer Vision Library
OPS	Object Positioning System
OS	Operating System
s	Second
SfM	Structure from Motion
SIFT	Scale-invariant feature transform
SLAM	Simultaneous Localization and Modeling
SURF	Speeded Up Robust Features
VR	Virtual Reality
Wi-Fi	Wireless Fidelity; an IEEE 802.11 WLAN

Acknowledgements

I am indebted to many:

To my advisor Romit Roy Choudhury for his endless support, patience, and encouragement. I can proudly say that I have learnt more observing him in past four years than in 22 years of education. Thank you for being a school and completely transforming my life.

To Justin Manweiler, without whom this dissertation would not be possible. I am not sure if I have enough words of thanks to express my gratitude. Thank you for helping me in every step of this journey, bearing with my impatience, spending countless hours in guiding me, and for being strongest professional companion I would ever have.

To the Duke University Computer Science department and UIUC Coordinated Science Lab at large for providing all the generous support towards my research career. I will especially thank my committee members Landon Cox and Bruce Maggs for their guidance and support. To all the faculties at Duke who helped me understand the discipline with ease.

To my past and present labmates in the SyNRG group. To my other co-authors and mentors at IBM Watson, Arup and Kirk.

To my undergraduate alma mater IIT Kharagpur, faculties in the department of computer science, it's rigorous curriculum, and illustrious culture for making me strong and preparing for this degree.

To my friends at IIT, Duke, and UIUC – Abhinav, Anchal, Animesh, Dipanjan, Gautam, Karan, Rajat, Souvik, and Yuvraj.

To Diksha, for her endless love, belief, respect, and for being there. To my family, especially my brother Priyank for his constant encouragement and support in every endeavor. This thesis is dedicated to my father for his prescient and support toward my education goals despite several hardships.

1
Introduction

Crowdsourced live videos from smartphones are gaining immense popularity. The success of Youtube, Vine, Periscope, or Meerkat bear ample testimony of it. Unlike professionally captured videos, user-generated video is captured by smart devices – there is no coordination across what is recorded, for how long, what catches a user's interest or when recording begins or ends. Moreover, these streams emerge at massive scale in a short time interval. Real-time efficient curation, analysis, and learning on such streams is challenging. In this dissertation, we propose a set of novel applications surrounding this space. The proposed applications face common real-world challenges and opportunities to solve them have a common theme. Our solution theme is "information fusion" – across different streams, generated from independent yet time-synchronized mediums. Live video and respective inertial sensors values are an example of independent yet time synchronized streams. To demonstrate the feasibility of our approach, we adopt top-down methodology. We first propose novel applications which fit this challenge-space and then we solve them considering wider generalizability in mind.

The challenges in real-time video processing can be classified in two cate-

gories: (a) Client side challenges due to smartphone's hardware limitations (b) Server side challenges due to computation and networking latencies. Smartphones have limited computing, storage, energy, and bandwidth resources – which makes real-time processing extremely hard. Since these applications demand sub-second latency, blindly offloading high-definition video frames or extracted visual features is impractical. Moreover, after the server receives a video frame for processing, the computation is expected to happen in real-time. Since computer vision algorithms are extremely slow, for every frame the same is not possible. Therefore, to achieve the real-time goal, computation latency of vision algorithms need to be reduced. This can be done by either enhancing the hardware, designing new efficient algorithms or by reducing the search space of image database. The former two are expensive and hard to achieve, but the later may be feasible given that smartphones are equipped with a variety of sensors. Hints from mobile sensors may be used to reduce the search space. We observe that mobile sensors readings can indicate which frame might be a worthy candidate of processing (e.g., if the frame is blurry) or which images in the database should be prioritized for matching. While smartphone sensors help computer vision by accelerating the processing, due to an inherent noise they are immediately usable. Therefore, a non-trivial fusion between computer vision algorithm and mobile sensing needs to be figured out.

The fusion of computer vision with mobile sensing (and vice versa) benefits client as well as server. For phones, mobile sensing is beneficial because it has fractional computation and energy requirement, and sensor values can be represented compactly and sent to servers in real-time. For servers, hints from sensors save cloud infrastructure cost and processing overhead. Since computer vision and mobile sensing achieve complementary goals of accuracy and processing latency, the fusion is an ideal match.

The different sensor-vision fusions apply to different practical problems. For

example, a surveillance camera used to recognize familiar faces can use GPS readings of one's phone to reduce potential matching face database. Similarly, videos from a football stadium may be clustered, by not analyzing the image content of the videos, but by analyzing the location and motion patterns of the video cameras while they were recording the video. A geometric overlap across multiple cameras in the stadium can act as an indicator of a common subject in them. Similarly, real-time mobile augmented reality could be enabled by learning from the walking patterns of multiple users. A mobile AR system could potential learn temporal and angular separation between a pair of annotation from multiple users as they view them. Later this unsupervised learning could be used to reduce candidate search database size for a soon to appear annotation. These examples have a common theme that inferences on a single information stream is hard, but when viewed in conjunction with other information dimensions, the same problem becomes tractable. We harness these observations across this dissertation. Next, we present precise definition of such problems. We take deep dives into each of them in subsequent chapters.

Our first application uses multi-modal sensing and computer vision as proxy metadata to enable distant object localization on the cloud. As an example use-case, while driving on a highway entering New York, we want to look at one of the skyscrapers through the smartphone camera, and compute its GPS location. While the problem would have been far more difficult five years back, the growing number of sensors on smartphones, combined with advances in computer vision, have opened up important opportunities. We harness these opportunities through a system called *Object Positioning System* (OPS) that achieves reasonable localization accuracy. Our core technique uses computer vision to create an approximate 3D structure of the object and camera and applies mobile phone sensors to scale and rotate the structure to its absolute configuration. Then, by solving (nonlinear)

optimizations on the residual (scaling and rotation) error, we ultimately estimate the object's GPS position.

Our second application uses Hadoop data analytics, multi-view stereo reconstruction, and mobile sensing based dead-reckoning to enable video clustering based on shared content. Crowdsourced videos uploaded from mobile devices, often provides engaging and diverse perspectives not captured by professional videographers. Unfortunately, such multimedia is difficult to organize due the scale of data. Video clustering services depend on manual tagging or machine-mineable viewer comments. While manual indexing can be effective for popular, well-established videos, they do not apply to newer or live content. We envisage video-sharing services for live user video streams, indexed automatically and in real-time, especially by shared content. Our implementation FOCUS is a Hadoop-on-cloud video-analytics, uniquely leverages 3D model reconstruction and sensing based dead-reckoning to decipher and continuously track a video's line-of-sight. Through spatial understanding of the relative geometry of multiple line-of-sights, FOCUS recognizes shared content even when viewed from diverse angles and distances. We use spatial overlap in multiple line-of-sights as a metric to perform real-time video clustering on the cloud.

Our third application is an attempt of taking our past experiences to a real-world deployment. High fidelity *Mobile Augment Reality* (Mobile AR) has seemed an obvious, feasible, and exciting means of exploring the physical world through digital content. Yet, an apparently feasible vision has never yet materialized into a comprehensively usable, precise, and functional system. This work takes a top-down design approach to answering the question: *What does it take to enable immersive Mobile AR today?* In building a ready-to-use Mobile AR system, we tackle several practical challenges through a design space exploration, novel heuristics, and engineering. Our converged design synthesizes smartphone sensing, computer

vision, cloud offloading, and optimization on past user behavior to achieve two primary goals: (1) sufficient precision for useful applications and (2) responsiveness acceptable for real-time Mobile AR, from a human user's perspective.

In our final work, the aim is to uniquely fingerprint an environment in the visual domain. The fingerprinting should be such that if a user takes a video swipe of the environment with the phone camera, her precise location can be immediately inferred. The problem is challenging because the user is continuously scanning the environment and the opportunity of capturing unique parts is fleeting. A typical environment may consist of repeating visual patterns such as ceiling, floor, wall texture, etc. and a painting hanging on the wall may be the only unique identifier of the location. Our approach to solve this problem is motivated by the ability of human brain in differentiating two similar looking environments by observing subtle differences in them. Our contribution in this work is twofold: (1) summarize global uniqueness of a location in few bits of information to quickly determine whether the current phone camera view contains a unique part, (2) scale this notion of uniqueness to buildings of arbitrary sizes yet delivering near real-time performance. Once fully developed, we believe our approach can apply to accelerate mobile augmented reality and correct indoor localization errors.

2

Object Positioning System using Smartphones

This paper attempts to solve the following problem: *can a distant object be localized by looking at it through a smartphone*. As an example use-case, while driving on a highway entering New York, we want to look at one of the skyscrapers through the smartphone camera, and compute its GPS location. While the problem would have been far more difficult five years back, the growing number of sensors on smartphones, combined with advances in computer vision, have opened up important opportunities. We harness these opportunities through a system called *Object Positioning System (OPS)* that achieves reasonable localization accuracy. Our core technique uses computer vision to create an approximate 3D structure of the object and camera, and applies mobile phone sensors to scale and rotate the structure to its absolute configuration. Then, by solving (nonlinear) optimizations on the residual (scaling and rotation) error, we ultimately estimate the object's GPS position.

We have developed *OPS* on Android NexusS phones and experimented with localizing 50 objects in the Duke University campus. We believe that OPS shows promising results, enabling a variety of applications.

2.1 Introduction

Imagine the following scenario in the future. While leaving for office, Alice needs to ensure that the repairman comes to her home later in the day and fixes the leakage on the roof. Of course, the leak is small and Alice must point out the location of the leak. To this end, she walks across the road in front of her house, points her camera towards the leak, takes a few photos, and types in "`leaking from here`". Later, when the repairman comes to Alice's house, he points his camera towards the roof and scans – when the leak is inside the camera's view-finder, Alice's message pops-up. The repairman repairs the leak and leaves. Alice comes back home in the evening, points her camera towards the leak, and sees the repairman's tag: "`repaired, received payment, thanks!`". Before returning into her house, she cursorily scans the neighborhood with her phone to see if there was anything new. She finds a "`pool party Saturday evening`" tag at the community swimming pool, and another on a tall crane at a nearby construction site, that read "`too noisy: 13 votes`". Alice remembers how she has been frustrated as well, so points her camera at the crane and votes. She looks at the tag again to confirm, which now reads "`too noisy: 14 votes`".

While this may be an intriguing vision of the future, the core idea of tagging objects in the environment, and viewing them through a smartphone's viewfinder, is old. A variety of augmented reality applications have already built such frameworks – Wikitude and Enkin even offer them on the app store Schall et al. (2011). However, these applications implicitly assume that objects in the environment have been annotated out-of-band – that someone visited Google Earth, and entered a tag for the swimming pool. Later, when an Enkin user looks at the same pool through her camera viewfinder, tags of all the objects in her viewfinder pops up. We believe that out-of-band tagging is one of the impediments to augmented reality (AR) be-

coming mainstream. The ability to tag the environment spontaneously will be vital if users must embrace AR applications in their daily lives.

This project – *Object Positioning Systems (OPS)* – is tasked to address this "missing piece" in today's AR applications. Our ultimate goal is to offer a service that allows a lay user to point her smartphone to any object in the environment and annotate it with comments. While this is the front-end functionality of our system, the key challenge in the back-end pertains to object localization. Our system essentially needs to compute the GPS location of the desired object, and then trivially associate the user-generated tag to that location. Another user standing at a different location should be able to look at the same object, run our system to compute its location, and retrieve all tags associated to it. Ideally, the system should operate in real time, so the user can immediately view the tag she has created.

While translating this vision to reality warrants a long-term research effort, as a first step, we narrow down its scope as follows. We sidestep indoor environments due to their stringent requirements on object positioning accuracy – a tag for a chair cannot get attached to the table. Therefore, we focus on outdoor objects and assume desktop-type CPU capability (which if unavailable on today's phone, may be available through the cloud). Even under this narrowed scope, the challenges are multiple: (1) State-of-the-art in computer vision is capable of localizing objects from hundreds of pictures of the same object Zheng et al. (2009); Takacs and et al. (2008). In the case where a *few* pictures are available – such as those taken by Alice of her rooftop – computer vision becomes inapplicable. Our intuition suggests that sensor information from mobile devices should offer opportunities to compensate for the deficiencies in vision, but the techniques for such information fusion are non-trivial. (2) The smartphone sensors, such as GPS, accelerometer, compass, and gyroscope, are themselves noisy, precluding the ability to pivot the system on some ground truth. Hence, aligning sensor information with vision will become

even more difficult, requiring us to formulate and solve a "mismatch minimization" problem. (3) OPS needs to identify the user's intention – different objects within the viewfinder may be at different depths/locations, and only the intended object's location is of interest. (4) Finally, the system needs to be reasonably lightweight in view of the eventual goal of on-phone, real-time operation.

The design of OPS has converged after many rounds of testing and modification. Our current prototype on Android NexusS phones has been used to localize 50 objects within the Duke University campus (e.g., buildings, towers, parking lot, cranes, trees). Performance evaluation shows that the system exhibits promising behavior. In some cases, however, our errors can be large, mainly stemming from excessively-high GPS errors. Nonetheless, OPS is able to identify and communicate such cases to the user – like a confidence metric – allowing them to re-attempt the operation. While not ready for real-world deployment, we believe OPS demonstrates an important first step towards a difficult problem with wide-ranging applications.

The key contributions in OPS are summarized as follows.

1. **Localization for distant objects within view:** We show opportunities in *multimodal sensing* to localize visible objects in outdoor environments, with core techniques rooted in mismatch optimization.
2. **System design and implementation on the Android NexusS platform:** Reasonably lightweight algorithms achieve promising location accuracy, with marked improvements over an optimized triangulation-based approach using GPS and compass.

The rest of the paper expands on these contributions, beginning with motivation and overview in Section 2.2 and primitives of OPS localization in Section 2.3. Next, in Section 3.3, we present the design of OPS. In Section 3.5, we address additional practical challenges for translating the core design into a complete system.

We provide results from our testing experiences in Section 3.4 and our ongoing work to improve OPS in Section 2.7. We compare OPS with the state of the art in Section 3.6. Section 3.7 concludes with a brief summary.

2.2 Motivation and Overview

This section visits the motivation of the paper, with a generalization of OPS to other applications, and then presenting a functional overview of the system. The subsequent sections elaborate on the core technical challenges and solutions.

2.2.1 Applications beyond Tagging

An Object Positioning System (OPS) has natural applications in tagging the environment. While this was our initial motivation, we observed that the core capability to localize a distant object is probably a more general primitive. In contemplating on the possibilities, we envisioned a number of other applications that can overlay on OPS:

(1) Location-based queries have been generally interpreted as queries on the user's current location (e.g., "restaurants around me," "driving directions from here to the airport"). However, queries based on a distant object can be entirely natural, such as "how expensive are rooms in that nice hotel far away," or "is that cell tower I can see from my house too close for radiation effects?" While walking or driving up to the object location is one way to resolve the query, the ability to immediately look up the hotel price based on the hotel's location, is naturally easier. OPS could enable such "object-oriented queries."

(2) OPS could potentially be used to improve GPS, particularly where the GPS errors are large or erratic. This is true even though OPS actually depends on GPS. The intuition is that combination of multi-modal information – vision and GPS in this case – can together improve each of the individual dimensions. Thus, knowing

the location of the object can help improve the location of the camera.

(3) High-end cars entering the market are embedded with a variety of safety features Moon et al. (2009), such as adaptive cruise control, lane change detection, blind spot alerts, etc. Existing cars remain deprived of the capabilities since upgrades may be expensive, even if feasible. High accuracy OPS technologies on mobile smartphones may enable services that approximate these capabilities. Smartphones mounted near the car's windshield could estimate location of other objects in the surroundings, and trigger appropriate reactions.

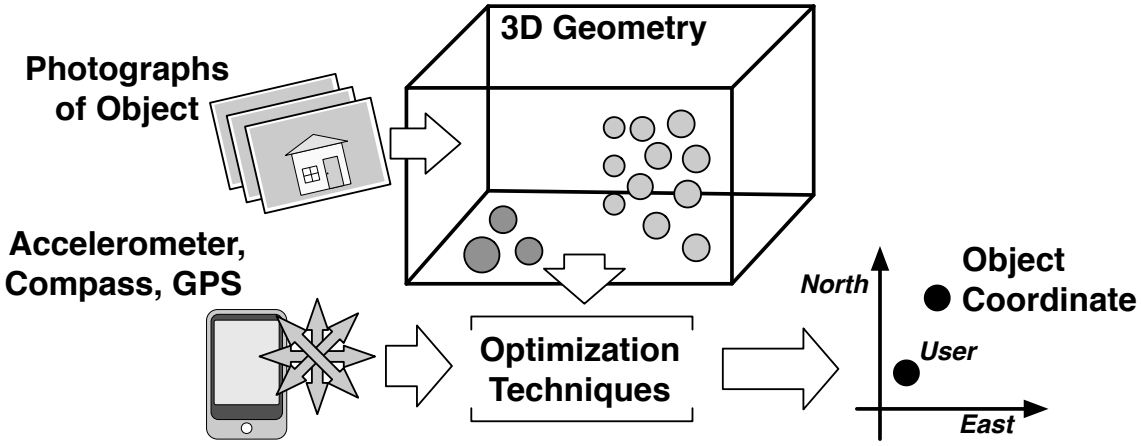


FIGURE 2.1: An architectural overview of the OPS system – inputs from computer vision combined with multi-modal sensor readings from the smartphone yield the object location.

2.2.2 System Overview

We present an overview of OPS with the goal of introducing the functional components in the system, and their interactions. We expect it to help the transition to technical details.

When a user activates OPS on her smartphone, the camera is automatically turned on, along with the GPS, accelerometer, compass, and gyroscope. The user is expected to bring the object of interest near the center of her viewfinder, and take a few pictures from different positions. These positions can be separated by a few

steps from each other in any direction – the goal is to get multiple views/angles of the same object. As few as 4 photos are adequate, however, more the better. Once completed, OPS displays the object's GPS coordinate.

While this is a simple front-end, Figure 2.1 shows the flow of operations at the back-end. The pictures taken by the user are accepted as inputs to the computer vision module, which implements a technique called *structure from motion* (SfM) Koenderink et al. (1991). Briefly, SfM is the process of extracting a 3D structure of an object from diverse views of a moving camera. As a part of this process, SfM first identifies *keypoints* in each picture – keypoints may be viewed as a set of points that together capture the defining aspects of the picture. The keypoints are matched across all the other pictures, and those that match offer insights into how the camera moved (or its angle changed) while the user clicked the different pictures. The final output of this process is a 3D structure, composed of the object and the camera locations.

Importantly, the 3D structure – also called the *point cloud* – is not in absolute scale. Rather, the point cloud offers information about the *relative* camera positions, as well as the *relative* distances between the cameras and the object. To be able to obtain the GPS location of the object, the point cloud needs to be "grounded" on the physical coordinate system. In an ideal scenario, where the point cloud and the GPS locations are both precise, it would be easy to scale the relative camera locations to match the GPS points. This will scale the object-distance as well, eventually yielding the absolute object location. Unfortunately, errors in the point cloud, and particularly in GPS readings, introduce a mismatch. Therefore, OPS uses the configuration of the camera-locations in the point cloud to first adjust the GPS positions. To this end, OPS formulates and solves an optimization problem to minimize the total adjustments.

The next goal is to use the compass readings from these (corrected) locations

to triangulate the object of interest. Again, if all compass readings were accurate, any pairwise triangulation from the GPS points should yield the same object location. Unsurprisingly, compasses are noisy as well – therefore OPS executes another optimization that minimizes the total adjustments on all compasses, under the constraint that all triangulations result in the same object location. This corrects the compass readings, and also offers a rough estimate of the object's distance from the GPS locations. By applying the compass readings back on the 3D point cloud, and again solving an optimization problem (detailed later), OPS finally converges on the object location.

OPS also extracts the height of the object, by incorporating the *angular pitch* of the phone while taking the picture. Thus, the final output is a location in 3D space, represented as a GPS coordinate and a height above the ground. The following sections zoom into the details of each of these components, beginning with the primitives of object localization.

2.3 Primitives for Object Localization

Inferences of a distant location from a known point-of-origin is an old problem. Historically, the principles of *triangulation* date to Greek philosophers of the 6th Century BC. Land surveying applies the same basic techniques today at great precision. The related technique of *trilateration* (location determination through known distances, rather than angles) is the technical basis of GPS. As a starting point, we investigate the applicability of these techniques to object localization.

Why not use GPS/compass to triangulate?

Smartphones have embedded GPS and compass (magnetometer) sensors. The precise location and compass bearings from any two points determines a pair of

lines[1]. The object-of-interest should fall at their unique intersection. We illustrate compass-based triangulation in Figure 2.2.

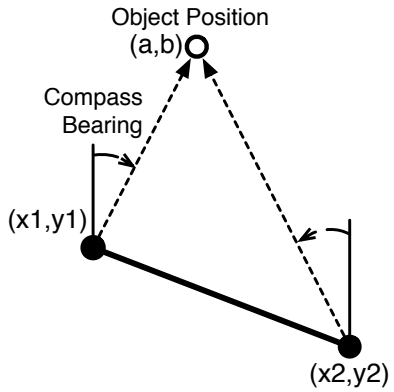


FIGURE 2.2: Compass-based triangulation from GPS locations (x_1, y_1), (x_2, y_2) to object position (a, b).

In principle, if a user points her phone at an object-of-interest from two distinct locations, we should be able to easily infer the object's location. Of course, to obtain these distinct locations, we cannot ask the user to walk too far, or using the system would be impractical. Instead, we can imagine the user walking just a few steps to infer the location of the object, say 40 meters away. When the scenario is such (i.e., distance between camera views is much smaller than the distance from the camera to the object), compass precision becomes crucial. A few degrees of compass error can dramatically reduce the accuracy of triangulation. Similarly, if the distance between the camera views are slightly erroneous, the result can also be error-prone. Smartphone sensors are not nearly designed to support such a level of precision. GPS can be impacted by weather (due to atmospheric delay), clock errors, errors in estimated satellite ephemeris, multipath, and internal noise sources from receiver hardware. Compass magnetometer readings are imprecise and subject to bias, due to variation in the Earth's magnetic field and nearby ferromagnetic material. Triangulation, at least under these extreme conditions, does

[1] The two points must not be collinear to the remote location.

not apply immediately.

Can smartphones apply trilateration?

Trilateration requires estimating the distance to the object-of-interest (range), from two vantage points. GPS is a popular implementation of trilateration – the distances from multiple satellites are computed from the propagation delay of the corresponding signals. Unfortunately, a GPS-like scheme is inapplicable for object positioning, since the objects are not collocated with a wireless radio. The phone camera, however, may partially emulate this functionality without any requirement of infrastructure at the object. This can naturally be an appealing alternative.

So long as the object-of-interest remains clearly in the camera view, the size of an object in the picture is a function of the camera's distance to that picture. The size can be estimated by the *visual angle* needed for that object (Figure 2.3), which can be computed as $v = 2\arctan(s/d)$, where v is the visual angle, s is the size (or height) of the object, and d is the distance to the object. Since we do not know object size s, we cannot compute d. However, knowing two different visual angles from two distinct locations, it is possible to eliminate s and obtain a ratio of the distances to the object from these locations. Let σ denote this ratio; then σ can be computed as

$$\sigma := \frac{d'}{d} = \frac{\tan(v/2)}{\tan(v'/2)}$$

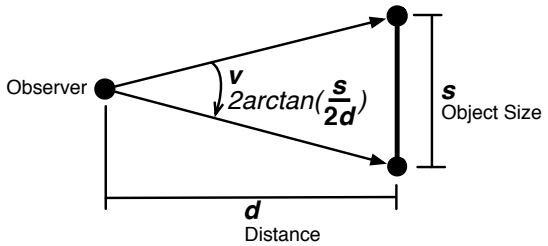


FIGURE 2.3: The visual angle v relates the apparent size s of an object to distance d from the observer.

Thus, although visual trilateration cannot precisely localize the object, the value of σ can certainly offer hints about the object's position. If one plots all points in space that are away from two camera locations in the ratio of σ, one gets a curve as shown in Figure 2.4. The object will naturally lie at some location on this curve.

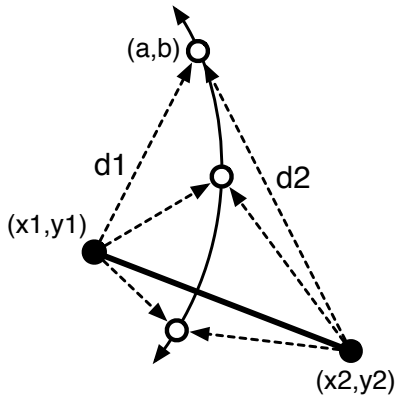


FIGURE 2.4: Visual Trilateration: unknown distances from GPS locations (x_1, y_1) and (x_2, y_2) to object position (a, b) are in a fixed ratio d_2/d_1.

Can phone cameras also triangulate?

Land surveying systems typically use optical sensing for precise triangulation. Possibly, the camera could be exploited to improve the accuracy of compass-based triangulation as well. Multiple views of an object from different angles, even if only slightly different, produce visual distortions, due to the phenomenon of *parallax*. Points in the foreground appear to change in relative position to points in the background. The properties of parallax, and visual perception in general, are well-understood. For example, stereo vision leverages parallax effects to invoke a three-dimensional perception from two-dimensional images. Thus, with a careful analysis of images taken from multiple nearby locations, it should be possible to invert these effects. In particular, it would be possible to infer the interior angle between a pair of GPS locations and the object position. However, knowing the interior angle is again not adequate to pinpoint the object location – instead it

offers a curve and the object can be at any location on this cure. Figure 2.5 captures this efficacy of visual triangulation.

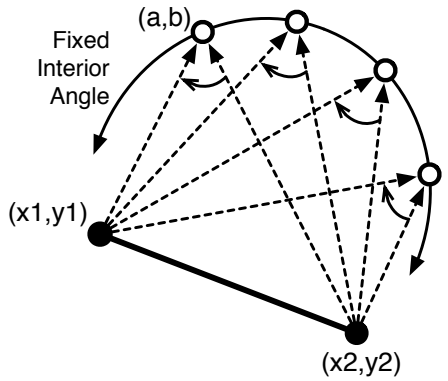


FIGURE 2.5: Visual Triangulation: fixed interior angle from known GPS location (x_1, y_1) to unknown object position (a, b) to known GPS position (x_2, y_2) .

Combining Triangulation and Trilateration

While neither triangulation nor trilateration can pinpoint object location, observe that computing the intersection of the two curves (in Figure 2.4 and Figure 2.5) yields a small number of intersection points. Moreover, if compass triangulation is added, there is more than adequate information to uniquely identify the object position (a, b). Figure 2.6 shows the superimposition of all four curves – observe that this is an over-constrained system, meaning that there is more than sufficient information to compute an unique solution.

This excess of information will later form the basis for noise correction on imperfect sensors. This is necessary because, with errors from GPS, compass, and inaccurate parameter estimation from the visual dimensions, we do not obtain a single point of intersection across all curves. While increasing the number of camera views will help, it will also increase the number of curves (each with some error). Thus, ultimately, we are left with many points of intersection, many of

which can be far away from the true object position. To find a single point of convergence, we will rely on optimization techniques, finding the most-likely true object point by minimizing estimates of sensor error.

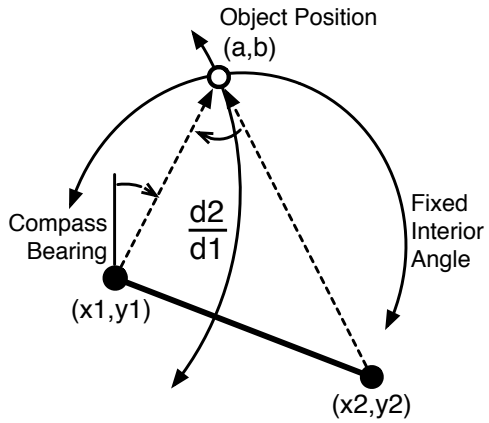


FIGURE 2.6: Intersection of the four triangulation curves for known points $(0, 0)$ and $(10, -4)$, localized point $(4, 8)$, distance ratio $\sigma = 6\sqrt{(5)}/4\sqrt{(5)} = 1.5$, and internal angle $\gamma = 2 \cdot \arctan(1/2) \approx 53^{\circ}$.

Next, we describe the OPS system design, focusing mainly on how advanced computer vision techniques can be applied to implement visual trilateration and triangulation. In particular, vision will quantify relative distance and invert the effects of parallax to find the interior angle between a pair of photographs, which will guide the other sensors to ultimately yield object location.

2.4 OPS: System Design

In an ideal world, visual information should not be necessary – noise-free sensors should be able to triangulate the object position. Since real-world sensors are noisy, OPS uses visual information to combat the impact. However, visual information relies partly on sensors, and thus, the overall system needs to be optimized jointly, to marginalize the noise. For ease of explanation, we first describe the individual techniques in isolation (i.e., without considering the effect of noise). Then,

we explain how noise forced many of our designs to fail, motivating our ultimate methods of "mismatch optimization." Figure 2.7 captures this flow of operations.

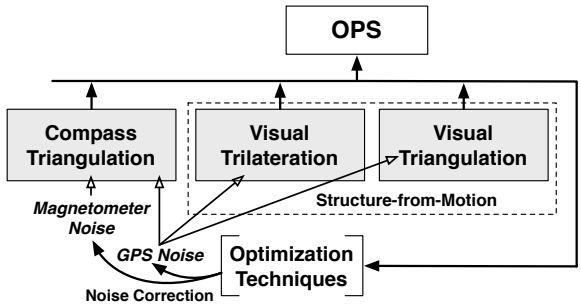


FIGURE 2.7: OPS builds on triangulation and trilateration, each underpinned by computer vision techniques, and multi-modal sensor information. Sensor noise affects the different techniques, and makes merging difficult.

2.4.1 Extracting a Visual Model

We begin with a discussion of the kind of information deducible from multiple photographs of the same object. Figure 2.8 shows how two observers each experience a different *perspective transformation* of the same object – a building. Note that the apparent *size* of the same building is different in each, as is the *shape*. The differences in scale[2] can be used to determine relative distances to the building. The shape of the transformation can reveal something about the difference in angle from each view to the building. As evident from our discussion of visual trilateration and triangulation, relative distances and angles from the cameras to the object can be valuable for estimating its location. OPS relies on a state-of-the-art computer vision technique, called *Structure from Motion* (SfM), to extract these distances and angles Koenderink et al. (1991); Snavely et al. (2006). As we will see next, SfM derives a 3D model of objects in the visible environment, including the object-of-interest and the camera, and computes these relations from them.

[2] After accounting for differences in camera focal length and lens distortions.

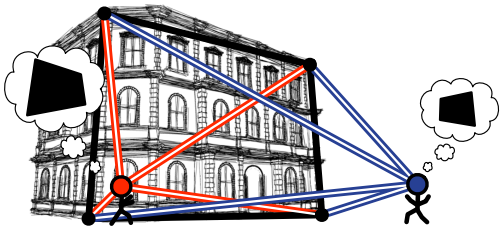


FIGURE 2.8: Two vantage points of the same object of interest. The "thought-bubbles" show the two different perspective transformations, each observing the same four feature corner points.

Structure from Motion

The SfM module functions as a multi-step process as follows. First, SfM accepts multiple photos from the user, and on each photo, runs an algorithm known as a *feature detector* Lowe (1999); Bay et al. (2006); Rosten and Drummond (2006). The algorithm identifies various "interesting" points of the photo, called *keypoints*. Ideally, the keypoints should be those that are likely to be robust (stable) across different perspectives of the same object. Put differently, a robust keypoint is one that consistently reflects the same physical point on the object. For example, the peak of a pitched roof may be taken as a keypoint in each image of the same house. Of course, the keypoint detection algorithm is a heuristic, and is prone to inconsistencies across multiple photos. However, with a large number of keypoints per image, there is likely to be a substantial number of keypoints that derive from the same physical point in all photos.

The keypoints from each photo are updated to a server, which then executes a keypoint matching algorithm. The matching process entails comparison of *feature descriptors* associated with each keypoint. A feature descriptor can be thought of as a unique "fingerprint" of a photograph, taken from the pixels around the keypoint. If a pair of feature descriptors are a strong match (numerically), the corresponding pair of keypoints can be assumed to likely capture the same physical point in the real world. Once keypoints are linked across multiple photos, SfM now prepares

to analyze the spatial relationship between the locations at which the photographs were taken.

For spatial reasoning, SfM applies algorithms that bear similarity to stereo vision. Perspective differences from multiple views of the same object (arising from parallax) can be used to reconstruct depth information. However, unlike stereo vision, SfM does not require a (known) fixed distance and relative orientation between a pair of views. Instead, SfM takes multiple sets of matched keypoints and attempts to reconstruct (1) a sparse 3D *point cloud* of the geometry captured by those keypoints, and (2) the relative positions and orientation of the camera when the original photographs were taken, known as *pose*. Figure 2.9 shows an example point-cloud for a building – observe that the points in the cloud are located on the surface of the building and other visible objects in the environment, as well as at the location of the camera.

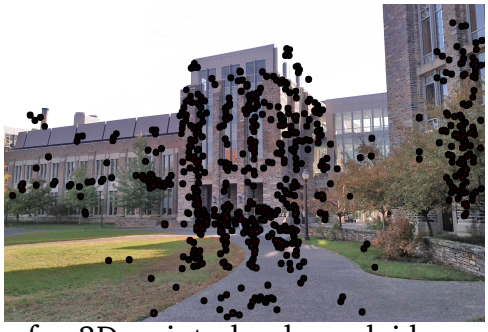


FIGURE 2.9: Example of a 3D point cloud overlaid on one of the images from which it was created.

SfM relies on *Bundle Adjustment* to perform a simultaneous refinement on the estimated 3D point cloud and parameters for each camera view (including, camera pose and lens distortions). Popular implementations of Bundle Adjustment use the Levenberg-Marquardt algorithm to perform a numerical nonlinear optimization on *reprojection error* between what is seen in each image (as described by the keypoints) and what is predicted by different parameterizations of camera pose

and 3D geometry. Thus, in summary, the final output from SfM is a considerably-accurate 3D point cloud.

2.4.2 From 3D Point-Cloud to Physical Location

For OPS, we utilize SfM as a "black box" utility. As input, SfM takes the matched keypoints of the user's images. As output OPS receives a 3D *point cloud* of estimated $< X, Y, Z >$ coordinates for each keypoint that was successfully matched across a sufficient number of images. We also have estimated $< X, Y, Z >$ camera pose coordinates from where each photo was originally taken. This coordinate system, however, even if completely precise, exists at an unknown relative scaling, translation, roll, pitch, and tilt from the corresponding locations in the real-world. To compute the physical location of the object, the camera locations and orientations in the 3D model needs to be "aligned" with the GPS and compass readings from the smartphone. However, since the GPS/compass readings themselves will be noisy, this alignment will be non-trivial – the GPS/compass values will need to be adjusted to minimize the mismatch. Moving forward, OPS will focus on addressing these challenges.

Questions

Before we continue further into the challenges of mismatch minimization, we briefly discuss a few natural issues related to the system build-up.

(1) Capturing User Intent

The use of computer vision entails a second practical challenge – many objects may appear in the camera view. OPS must be able to infer, automatically, which object in view the user is most-likely interested in localizing. For example, a building may be partially occluded by trees. Thus, the point cloud may contain many keypoints

that are not reflective of the specific object-of-interest. In general, we assume that the user positions the object-of-interest roughly at the center of the camera's viewfinder. Across multiple photographs, the intended object will become a "visual pivot." Near-foreground and distant-background points appear to shift away from this central point, due to parallax. More sophisticated techniques based on the computer vision techniques of *segmentation* are also relevant here. For example, in Section 3.5, we will consider an alternative approach for cases where we can assume the user is focused on a building. In our evaluation, however, we will avoid such assumptions.

(2) Privacy

We note that while OPS may offload computational tasks to a central server/cloud, users need not ever upload actual photographs of objects-of-interest (in fact, they can be discarded from the smartphone as well). Instead, they can only upload the keypoints and feature descriptors, that contain all the information needed by SfM. This serves to address any privacy concerns that a user may have with OPS.

2.4.3 Point Cloud to Location: Failed Attempts

In our original design, we expected that once a Structure-from-Motion point cloud is extracted, estimation of the real-world coordinate of the object would be reasonably straightforward. This would only require mapping vision $< X, Y, Z >$ coordinates to real-world ¡latitude, longitude, altitude¿. In practice, substantial sensor noise makes this mapping more difficult and error-prone than we had anticipated.

The point cloud reflects the structure of the object location relative to the user's locations, when the original photographs were taken, but at an unknown relative scaling, translation, roll, pitch, and tilt from the real-world. Importantly, the locations at which the photographs have been taken are known in both real-world

coordinates (through GPS) as well as in the SfM-derived coordinate system. In principle, some *affine transformation* should exist to convert from one coordinate system to the other. We sought to apply state-of-the-art computer vision optimization techniques to estimate this affine transformation, and we describe three of our failed attempts, followed by the actual proposal.

Attempts using Point Cloud Registration

We applied the computer vision technique of Iterative Closest Point (ICP) to find the mapping. ICP is commonly used in the *registration* of one point cloud to another. Before applying ICP, we first eliminated what is typically a key challenge; we pre-defined the association of points from one point cloud to the other. We also eliminated as many degrees-of-freedom in the transformation as possible: we normalized the $< X, Y, Z >$ coordinates to eliminate translation from the search space and constrained scaling to be uniform in all dimensions. We attempted three mechanisms for estimation of the affine transformation: first, an approach based on the Singular Value Decomposition (SVD); next, a nonlinear minimization of transformation error based on the Levenberg-Marquardt algorithm; third, we exploited knowledge of surface normals of camera pose (the 3D direction at which the camera points) in the SfM point cloud and attempted to match with a 3D phone rotation matrix from compass and accelerometer (to find orientation relative to the vector of gravity). In all cases, sensor noise (in GPS, compass, and accelerometer) resulted in a nonsensical transformation. With only a few camera locations to connect one coordinate system to the other, and a still-large number of remaining degrees-of-freedom in the transformation, there is simply insufficient data to overcome sensor noise.

Attempts Intersecting Triangulation/Trilateration

After many unsuccessful attempts at applying computer vision techniques to estimate the affine transformation between coordinate systems, we attempted to sim-

plify the localization problem. We tried to directly apply our intuitions for (1) compass triangulation; (2) visual trilateration; and (3) visual triangulation. The parameters of relative distance and interior angles can be trivially estimated from a SfM point cloud. If all sensors and vision techniques were fully-precise, the true (a, b) object location should fall at the intersection of those equations. In practice, after numerically solving the roots of all equation pairs, sensor noise and bias create many intersection points. We applied a 2D hierarchical clustering on these intersection points, hoping that one of the clusters would be distinctly dense, and the centroid of that cluster would be the estimated object location. In many cases, this proved correct. However, in more cases, sensor error (especially GPS) was large. Intersection points became diffused, and no longer indicative of the true location. To be practical, OPS would need to apply a more robust approach.

Attempts Optimizing Across Error Sources

We were encouraged by the partial-success of our second approach, directly applying our equations of triangulation and trilateration. Next, we attempted to push this technique further, this time integrating an optimization approach. We formulated a minimization problem on the error terms for each sensor value and vision-derived parameter, and with the optimization constrained to find a single object location (a, b). This led to a complex nonlinear optimization with many error terms and constraints. While this would occasionally converge to the correct object location, more often it found a trivial and nonsensical solution. For example, GPS error terms would "correct" all GPS locations to the same point. Further, the complexity of the optimization led to an impractically-long running time.

2.4.4 The Converged Design of OPS

From our early attempts to build a real-world object location model, we learned two important lessons that influenced the final design of OPS. First, any opti-

mization would need to be constrained to be limited in the number of degrees-of-freedom and avoid degenerate cases. Second, we would need a mechanism to reduce the impact of GPS error. The final design of OPS consists of two optimization steps, each designed to limit the potential for degenerate solutions.

Before continuing, it is helpful the simplify our notion of location. We do not consider latitude and longitude directly, as angular values are inconvenient for measuring distances. Instead, we apply a precise Mercator projection to the square UTM coordinate system. Therefore, we can now refer to a latitude/longitude position as a simple 2D (x, y) coordinate. Recovery of the final ¡latitude, longitude¿ coordinate of the object is a straightforward inversion of this projection.

"Triangulation" via Minimization on Compass Error

Before explaining the optimizations underlying OPS, it is instructive to consider a reasonable baseline comparison. Since we assume that the user will take more than two photographs when using OPS, it would be unfair to compare OPS to a triangulation with only two GPS readings $(G_1^x, G_1^y), (G_2^x, G_2^y)$ and two compass bearings θ_1, θ_2. Instead, we must generalize the notion of triangulation to support as many measurements as will be available for OPS.

In Table 2.1, we present a nonlinear optimization that represents a triangulation-like approach to object localization. Unlike standard triangulation, this scales to support an arbitrary number of GPS (G_i^x, G_i^y) and compass heading θ_i pairs. In noise-free conditions, all lines of compass bearing originating at the corresponding GPS point would converge to a single point (a, b). Of course, due to sensor error, we can expect that all pairs $\binom{n}{2}$ of compass lines will result in $\binom{n}{2}$ different intersection points. The optimization that follows seeks to find the most-likely single point of intersection by rotating each compass bearing as little as possible until all converge at the same fixed point (a, b). We experimented with a number of other approaches for "generalized triangulation." For one, we

considered joint optimizations on GPS and compass. For another, we considered the 2D median of all $\binom{n}{2}$ intersections points. After experimentation, we believe this is the most-effective technique, and thus the fairest baseline comparison method to OPS.

Table 2.1: Optimization for Triangulation

$$
\begin{aligned}
\textbf{Minimize} \quad & \sum_{\forall i} |E_i^\theta| \\
\textbf{Subject to} & \\
\forall i \quad : \quad & b - G_i^y = (a - G_i^x) \cdot \cot(\theta_i + E_i^\theta) \\
\textbf{Solving for} \quad & a, b \\
\forall i \quad : \quad & E_i^\theta \\
\textbf{With parameters} & \\
\forall i \quad : \quad & G_i^x, G_i^y, \theta_i
\end{aligned}
$$

Name	**Parameter Sources**
G_i^x, G_i^y	GPS position (of user at each photograph)
θ_i	Compass bearing (at each photograph)
Name	**Solved Variable Interpretation**
a, b	Estimated object location at (a, b)
E_i^θ	Estimated error for compass bearing θ_i

Now, we turn our attention to the two-step process employed in OPS. First, we apply the output of computer vision to correct for noise in GPS measurements. Second, we extend this baseline sensor-only optimization for triangulation to (1) use our corrected GPS points; and (2) exploit our intuitions for visual trilateration and triangulation.

Minimization of GPS Noise, Relative to Vision

From our earlier attempts, we realized that a direct application of our intuitions for visual trilateration and triangulation would be insufficient. Instead, we need a mechanism to reduce sensor noise before a object-positioning step can be effectively applied. Here, we rely on the output of structure from motion to correct for

random GPS noise. Bias across all GPS measurements will remain. However, a consistent bias will be less damaging to the final localization result than imprecision in the relative GPS positions across multiple photographs. Structure from motion can help eliminate this noise between relative positions, as it tends to capture this relative structure with far greater precision than GPS. We design a nonlinear programing optimization that seeks to move the GPS points as little as possible, such that they match the corresponding relative structure known from vision.

We design an optimization that maps the original GPS points where photographs were taken $\{\forall i : (G_i^x, G_i^y)\}$ to a set of fixed GPS points $\{\forall i : (F_i^x, F_i^y) = (G_i^x + E_i^x, G_i^y + E_i^y)\}$. The optimization will also solve a scaling factor λ that proportionally shrinks or expands the point-to-point distances in the structure-from-motion point cloud to match the equivalent real-world distances measured in meters. The constraints simply enforce that the distance between any pair of GPS points, $\sqrt{(G_i^x - G_j^x)^2 + (G_i^y - G_j^y)^2}$, is equal to the distance between those same points in vision coordinates, $\sqrt{(V_i^h - V_j^h)^2 + (V_i^d - V_j^d)^2}$, after multiplying the vision distance by a constant factor λ.[3] Since we expect the GPS points to have some noise relative to the same points in vision, we introduce error terms for the GPS distance, $\sqrt{(E_i^x - E_j^x)^2 + (E_i^y - E_j^y)^2}$. With these error terms, the optimization is simply a minimization on the sum of squared error. Table 2.2 presents the complete optimization.

OPS Optimization on Object Location

From the GPS-correction optimization (Table 2.2), we are left with a set of fixed GPS points, $\{\forall i : (F_i^x, F_i^y)\}$, and a scaling factor λ from vision to GPS coordinates.

[3] To avoid confusion with GPS x and y dimensions, we use h and d to represent the relevant two dimensions of the vision coordinate system. The h, or horizontal, dimension runs left/right of the object from the perspective of the user. The d, or depth, dimension runs towards/away from the object.

Table 2.2: OPS Optimization on GPS Error

$$
\begin{aligned}
\textbf{Minimize} \quad & \sum_{\forall i} \left(E_i^{x2} + E_i^{y2} \right) \\
\textbf{Subject to} & \\
\forall i, j \quad : \quad & \left[(G_i^x + E_i^x) - (G_j^x + E_j^x) \right]^2 + \\
& \left[(G_i^y + E_i^y) - (G_j^y + E_j^y) \right]^2 = \\
& \lambda^2 \cdot \left[(V_i^h - V_j^h)^2 + (V_i^d - V_j^d)^2 \right] \\
\textbf{Solving for} \quad & \sigma \\
\forall i \quad : \quad & E_i^x, E_i^y \\
\textbf{With parameters} & \\
\forall i \quad : \quad & G_i^x, G_i^y, V_i^h, V_i^d
\end{aligned}
$$

Name	**Parameter Sources**
G_i^x, G_i^y	GPS position (of user at each photograph)
V_i^h, V_i^d	Vision position (arbitrary units)
Name	**Solved Variable Interpretation**
λ	Scaling factor, vision coordinates to GPS
E_i^x, E_i^y	Estimated camera GPS error

Now, we take these parameters to extend the baseline sensor-only optimization for triangulation (Table 2.1), along with additional context from visual trilateration and triangulation. We present this final optimization as Table 2.3.

The additional context for visual trilateration and triangulation is encoded as parameters $\{\forall i, j : \gamma_{ij}\}$, (C_x, C_y), and D. Each value γ_{ij} represents the angle from (F_i^x, F_i^y) to (a, b) to (F_j^x, F_j^y), estimated from vision. As represented in the notation, this is the positive acute angle between vectors $\vec{V_i}$ and $\vec{V_j}$. Thus, $\{\forall i, j : \gamma_{ij}\}$ directly encodes our original interpretation of visual triangulation. To avoid redundancies in the constraints (since triangulation and trilateration parameters are directly measured from the same vision point cloud), we only need to partially encode visual trilateration. Instead of encoding each of the relative distance from

each camera point, we can simply enforce that the distance from the user's position to the object as a known, fixed value. We compute (C_x, C_y) as the 2D median across $\{\forall i : (F_i^x, F_i^y)\}$, by applying convex hull peeling. Next we enforce that the distance from (C_x, C_y) to the object at (a, b), $\sqrt{(a - C_x)^2 + (b - C_y)^2}$, is equal to the distance D. We can compute D from the vision point cloud along with the vision-to-GPS scaling factor λ.

The minimization function must change to accommodate context from triangulation (visual trilateration is fully incorporated as hard constraints). The addition of γ_{ij} error terms $\{\forall i, j : E_{ij}^{\gamma}\}$ allow angular error to be traded between magnetometer-derived compass headings and vision-derived angles. The compass error scaling factor, $(n - 1)/2$, balances for the lessor quantity of compass error terms relative to pairwise vision angles.

2.5 Discussion

Extending the Location Model to 3D

In Section 2.4.4, we described how OPS estimates the object-of-interest location in two dimensions, namely as the point (a, b). In some contexts, the 3D location of the object can also be useful. For example, we might want to localize a particular window of a multistory building. Ultimately, OPS should provide a $<$ latitude, longitude, altitude $>$ location tuple. However, the height dimension adds additional challenges not faced on the ground, following a plane tangential to the Earth's surface. First, GPS-estimated altitude is prone to greater inaccuracy than latitude and longitude. Second, while it is natural for a user to take multiple photos by walking a few steps in-between, a requirement to take photographs at multiple heights would become awkward. Thus, while vision provides three-dimensional geometry, GPS locations for photographs are roughly planar. Further,

since the object localization task is already challenging in two dimensions, it is desirable to avoid integrating 3D into our location optimizations.

Instead, OPS finds the two-dimensional object location first. Next, it uses the now-known distance to the object, along with accelerometer and vision inputs, to estimate height. For each photograph, OPS records the raw three-axis accelerometer output. Since we can expect the phone to be roughly still while the photograph is taken, the accelerometer is anticipated to measure only gravitational force. This gravitational vector defines a unique orientation in terms of phone roll (rotational movement on the plane of the phone screen, relative to the ground) and pitch (rotational movement orthogonal to the plane of the phone screen, relative to the horizon). Pitch provides a rough estimate of how much higher (or lower) the user is focusing, relative to a plane parallel to the ground and intersecting the user at eye-level. To improve the accuracy of this measurement, we can "average" pitch measurements from every photograph. Importantly, the user might not align the object in every photo at exactly the same pitch. For example, the window might appear higher or lower on the screen. We can correct for this by leveraging vision once again. From our 3D point cloud, there is a unique mapping of every 3D point back to each original 2D image. We can now compute an adjustment value, measured in pixels, from the horizontal center line of the screen. We can convert this pixel value to an angle, given the known camera field-of-view. Next, the angular sum of pitch and adjustment, averaged across all photographs, can be used to estimate height when combined with the known two-dimensional distance.

$$h_{\text{object}} = h_{\text{observer}} + \frac{\text{2D distance}}{2 \cdot \tan\left(\frac{\text{pitch}+\text{adjustment}}{2}\right)}$$

Alternatives for Capturing User Intent

If we can make assumptions regarding the structure of the object-of-interest, com-

puter vision techniques of segmentation can assist in isolation of the object from the structure-from-motion point cloud. For example, consider a typical multistory building with large flat sides. Images of an exterior wall will tend to yield many keypoints along a flat plane, roughly perpendicular to the ground plane. These points on the wall plane are often clearly distinct from points on the ground, smaller clusters of points in the nearest-foreground from occlusions, or sparse points in the distant background. To determine where the object-of-interest lies within a point cloud, we attempt to segment the point cloud and find such a predominate plane. We apply Random Sample Consensus (RANSAC) Fischler and Bolles (1981), an iterative method to estimate a model for a plane, under an angular constraint that it must be roughly-perpendicular to the ground and parallel to the field-of-view. All points in the point cloud are then classified as either inliers or outliers to the plane. Next, we find the spatial centroid among inliers to the plane. This point is considered to be the object-of-interest.

2.6 Evaluation

We take a systems-oriented approach in evaluating OPS, so as to capture real-world performance. Phone sensors are subject to noise and biases. GPS can be impacted by weather (due to atmospheric delay), clock errors, errors in estimated satellite ephemeris, multipath, and internal noise sources from receiver hardware. Compass magnetometers are affected by variations in the Earth's magnetic field and nearby ferromagnetic material. Computer vision techniques, such as structure from motion, can break down in a variety of scenarios. For example, keypoint extraction may fail if photographs have insufficient overlap, are blurred, are under or over-exposed, or are taken with too dark or bright conditions (such as when the sun is in the user's eyes). The primary goal of our evaluation is to consider how

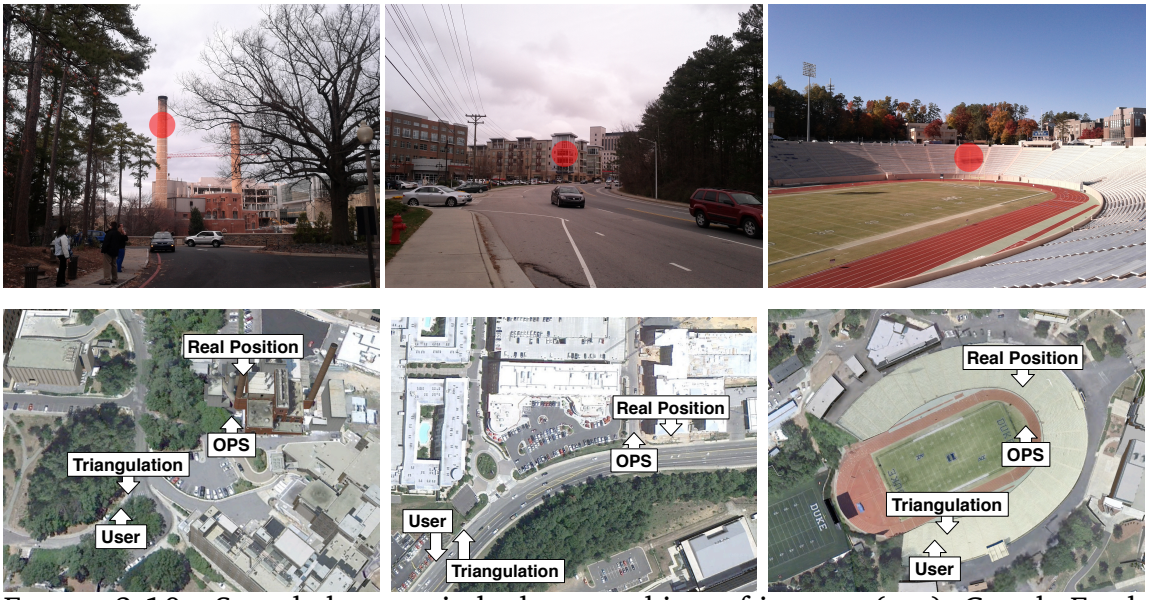


FIGURE 2.10: Sampled tests; circle denotes object-of-interest (top), Google Earth view (bottom): (a) smokestack of a coal power plant; (b) distant building with vehicles in foreground; (c) stadium seats near goal post.

well OPS overcomes this naturally-challenging operational context.

2.6.1 Implementation

OPS is implemented in two parts, an OPS smartphone client and a back-end server application. We built and tested the OPS client on the Google NexusS phone, as a Java extension to the standard Android 2.4 camera program. Photographs may be pre-processed locally on the phone to extract keypoints and feature descriptors (reducing the required data transfer), or simply uploaded to our server for processing (faster with a high-performance WiFi connection). Along with photographs (or keypoints and descriptors), the phone uploads all available sensor data from when each photograph was taken, to include GPS, compass, and accelerometer. Our server is a Lenovo desktop running Ubuntu Linux 11.04.

Computer Vision

Both the client and server applications support the basic computer vision tasks of keypoint detection and extraction of feature descriptors. We use the SURF

(Speeded Up Robust Feature) algorithm Bay et al. (2006). We choose SURF over the related SIFT (Scale Invariant Feature Transform) Lowe (1999) as it is known to be considerably faster to compute and provide a greater robustness against image transformations. SURF detection and extraction are performed using OpenCV. For both the client and server-side applications, JavaCV provides java wrappers of native C++ calls into OpenCV.

For server-side structure from motion, we use Bundler, an open-source project written by Noah Snavely and the basis for the Microsoft Photosynth project Snavely et al. (2006). Bundler operates on an unordered set of images to incrementally-build a 3D reconstruction of camera pose and scene geometry. As input, Bundler expects keypoints matched across multiple photos (on the bases of the corresponding feature descriptors). Bundler can operate on any keypoint type, expecting SIFT by default. We adapt the output of the OpenCV SURF detector to match the SIFT-based input expected by Bundler, substantially decreasing processing time per photo on the phone. As output, Bundler provides a sparse point cloud representation of the scene in view. OPS builds its real-world localization model on top of this point cloud, which exists at an unknown relative scaling, translation, roll, pitch, and tilt from the corresponding locations in the real-world.

Server-side Nonlinear Optimization

The OPS server is implemented primarily in Java. Mathematica is used through a Java API for solving nonlinear optimizations. We use differential evolution as the optimization metaheuristic for its strong robustness to local minima (simulated annealing proved to be similarly effective, with both outperforming the default Nelder-Mead) Storn and Price (1997).

2.6.2 Accuracy of Object Localization

We tested OPS at more than 50 locations on or near the Duke University campus. We attempted to use OPS in the most natural way possible, focusing on localization tests that mirror how we would expect a real user would want to use the system. Primarily, we considered objects at distances between 30m and 150m away, for two reasons. First, the object should be far enough away that it makes sense to use the system, despite the hassle of taking photographs. Though it only takes about a minute to take the required photographs, a user should not be more willing to simply walk over to the object to get a GPS lock. Second, distances are limited by the user's ability to clearly see the object and focus a photograph. Building densities, building heights, and presence of significant occlusions (such as trees), constrain the distances at which photographs can be easily taken.

We compare OPS accuracy to "Optimization (Opt.) Triangulation." To provide the fairest comparison, Triangulation reflects the triangulation-like optimization described in Section 2.4.4, designed to scale to an arbitrary number of GPS, compass-heading pairs. For some graphs, we also show the distance from the position at which photographs were taken (centroid across all photographs) to the true object location. This is an important consideration; as sensor values are projected into the distance, noise and bias can be magnified.

Testing Parameters

OPS performance is dependent on the way OPS is used. For our evaluation, we took four photographs for each localization (the minimum required for structure from motion). Each successful photograph was taken between 0.75m and 1.5m from the location of the preceding photograph, in no specified direction. This flexibility is important, as it allows the user the compose the shot as is natural, given obstacles in the vicinity and visual occlusions. The distance (about one or two paces) is

sufficiently far that GPS is able to estimate distance with sufficient (albeit still poor) accuracy. The distance is sufficiently small as to ensure a relatively small angular separation to the object of interest, enabling structure from motion to work well. Further, the distances are not too burdensome for the user; the required photographs can be taken in less than one minute. Once the user takes the required photos, processing requires approximately an additional 30-60 seconds, primarily attributable to structure from motion computation and depending on the number of keypoints detected in each photo. With additional photographs, both accuracy and processing time increase.

Structure from motion is sensitive to the quality of photographs. In poor lighting, keypoint detection becomes less robust. Thus, OPS is sensitive to the quality of lighting conditions. At night, there is unlikely to be sufficient light for high-quality photographs. Further, a camera flash is useless for the distances under consideration. At dawn and dusk, direct sun into the camera lens can also ruin photographs. We test OPS in daylight conditions, and avoid early morning and late evening. We are considering extensions to OPS to enhance performance in poor lighting, likely avoiding structure from motion in these cases.

Example Usage Scenarios

In Figure 2.10, we show three example photos, taken while using OPS. Below each photo, we show a screenshot taken from Google Earth with four annotations, (1) the location at which the user photographed the object-of-interest; (2) the true location of the intended object (positioned at the center of the screen in each photo; (3) the object position inferred by Opt. Triangulation; and (4) the object position inferred by OPS. We show these particular examples to highlight that OPS is a general approach, and can therefore localize for a wide variety of distant objects. Further, while the final result may still be a substantial distance from the precise true location, OPS is typically able to find a position that is far more representative

of the object-in-view than triangulation.

Overall Performance

Figures 2.11 and 2.12 present overall performance of OPS across all locations. Figure 2.11 shows three CDF curves, (1) error in meters from the OPS-localized position to the true object position; (2) error in meters from the Opt. Triangulation position to the true object position; and (3) distance in meters from the position at which photographs were taken (centroid across all photographs) to the true object location. Figure 2.12 shows individual localization results, sorted by user-to-object distance. Overall, OPS provides a substantial performance improvement over triangulation.

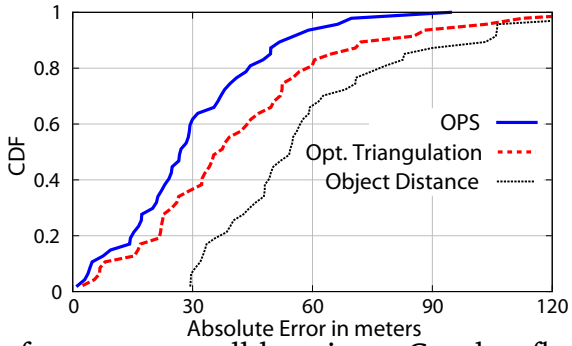


FIGURE 2.11: CDF of error across all locations. Graph reflects four photos taken per location. 50 locations.

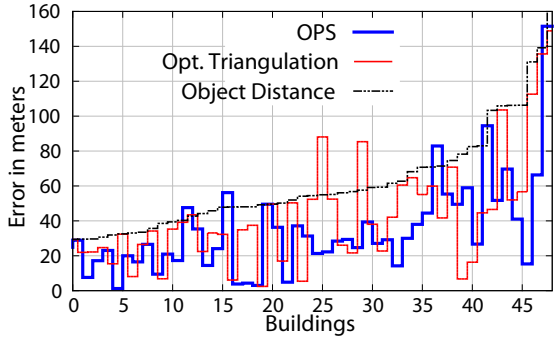


FIGURE 2.12: OPS and triangulation error at 50 locations. Graph reflects four photos taken per location.

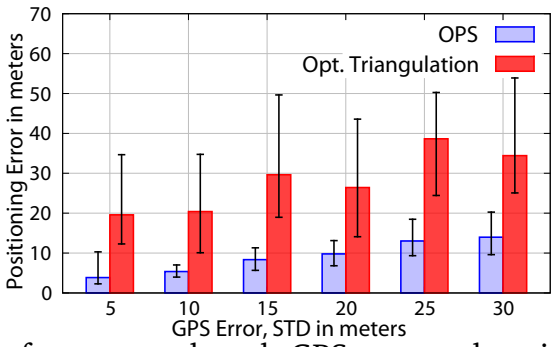


FIGURE 2.13: Error from ground-truth GPS camera locations. X-axis shows the standard deviation of introduced Gaussian GPS errors. Bars show median error; whiskers show first and third quartiles.

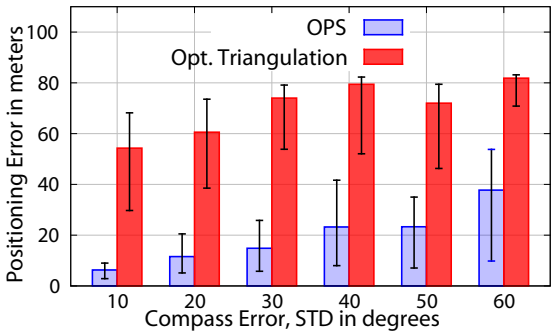


FIGURE 2.14: Error from ground-truth GPS camera locations. X-axis shows the standard deviation of introduced Gaussian compass errors. Bars show median error; whiskers show first and third quartiles.

Sensitivity to GPS and Compass Error

To better understand OPS's robustness to sensor noise, we wanted to carefully evaluate the impact of GPS and compass noise, in isolation. We took a set of four photographs of a Duke University Science Center building, from a mean distance of 87m. We ensured that this particular set of photographs provided a robust model from vision, through a manual inspection of the point cloud. For each photograph, we used Google Earth to mark the exact location at which it was taken, with less than one meter error. From these carefully-marked points, we mathematically computed the compass bearing to the object, assuring less than 5° error (attributable to error in marking the points). Next, we randomly perturbed each GPS point or

compass angle according to random deviates of a Gaussian (Normal) distribution with mean 0 and a varied standard deviation. Figure 2.13 shows performance as a function of the standard deviation of injected GPS error (in meters). OPS is able to leverage vision to remove much of this GPS noise, providing strong robustness. Figure 2.14 shows performance as a function of the standard deviation of injected compass error (in degrees). Superior performance of OPS is especially reflective of how visual triangulation enforces the true relative angle between a pair camera locations and the object position. Bars reflect median of 50 trials with first and third quartiles on whiskers.

Sensitivity to Photograph Detail

Figure 2.15 shows OPS performance with varied levels of photographic detail. For this experiment, four photographs were used, along with ground-truth GPS and compass data. Full-resolution photos were downsampled, reducing the efficacy of keypoint detectors. With fewer keypoints, especially at resolutions below 1024x768 pixels, structure from motion suffers and can impact the object position.

Overall Processing Latency

Figure 2.16 shows various processing latencies incured by OPS. The feature extraction latency is the total time taken by SURF algorithm in extracting features for 4 images. The bundle adjustment latency is the time taken in constructing 3D model using those images. Finally, optimization latency is the time taken by Mathematica is solving non-linear optimization. The median processing time to determine object location using OPS is ≈ 30 seconds.

2.7 Room for Improvement

OPS remains an ongoing research project. We are exploring a variety of mechanisms to improve accuracy.

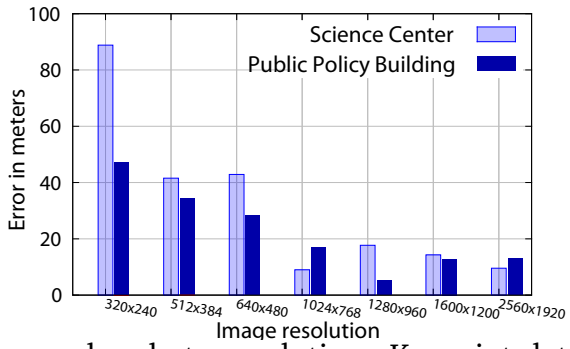


FIGURE 2.15: OPS error by photo resolution. Keypoint detection is less reliable below 1024x768 pixels.

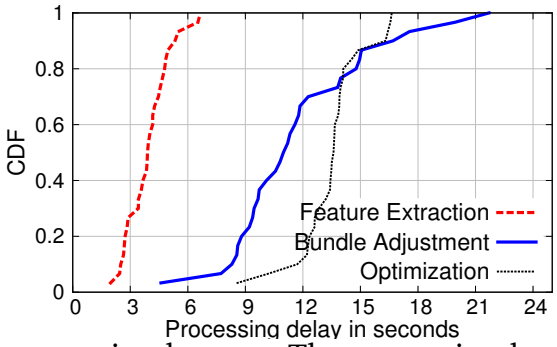


FIGURE 2.16: OPS processing latency. The processing latency of bundle adjustment and non-linear optimizations could be a possible bottleneck in the real-world deployment.

Live Feedback to Improve Photograph Quality

OPS is impacted by the quality of user photographs. Poor angular separation between two photographs (too small or large) can reduce the efficacy of structure from motion. We imagine a system of continuous feedback to the user, aiding in shot selection and framing. By combining an especially-lightweight keypoint detection heuristic, such as FAST Rosten and Drummond (2006), on the camera video with information from the phone gyroscope, it would be feasible to suggest when a "good" photograph can be taken. Otherwise, we would inform the user to take a corrective step (to the left or right).

Improving GPS Precision with Dead Reckoning

From Figure 2.13 and 2.14, it is clear that OPS is already relatively insensitive to

GPS and magnetometer noise. Although OPS explicitly uses the output of structure from motion to cancel GPS noise, large GPS errors can still become disruptive – this is quite often the cause of poor performance. One possibility is to apply additional sensors to improve GPS precision. In particular, a constant bias is less damaging than imprecise relative positions across camera locations. We are considering mechanisms to leverage gyroscope and accelerometer for dead reckoning between camera locations.

Continual Estimation of Relative Positions with Video Continuous video could potentially be used to substantially augment or even replace the structure-from-motion point cloud. A frame-by-frame comparison of these keypoints, in fine-grained comparison with accelerometer, gyroscope, compass, and GPS, could provide a highly-detailed trace of how the smartphone (1) moved in 3D space and (2) viewed the object-of-interest in terms of relative angles/distances.

2.8 Related Work

To the best of our knowledge, OPS is a first-of-kind system to find the precise position for objects of significant distance away, despite noisy sensors and without requiring a database of pre-existing photographs or maps. Nonetheless, there is a substantial body of related work, especially leveraging computer vision for recognition of well-known landmarks.

Localization through Large-Scale Visual Clustering

OPS is related to the problem of worldwide localization on the basis of visual classification Zheng et al. (2009); Li et al. (2009); Hays and Efros (2008); Cuellar et al. (2008); Takacs and et al. (2008). Most closely related to OPS, Zheng et al. (2009) is the back-end system underlying Google Goggles: (1) online travel guides are mined for possible tourist landmarks; (2) photo-sharing databases of millions

of GPS-tagged photos are searched using keywords from these travel guides; (3) unsupervised visual clustering on these search results provide a visual model of the landmark. From 20 million GPS-tagged photos, 5312 landmarks can be recognized by visual comparison of a photograph with these landmark models. OPS is designed to be more generic, able to localize objects which cannot be considered landmarks (without a pre-exiting photo library of these objects). We believe that a hybrid approach can be valuable, which leverages a photo database to enhance accuracy (where useful photos are available), but can also provide OPS functionality in the general case.

Aligning Structure from Motion to the Real World

Substantial computer vision literature has considered improvements and applications of structure from motion (SfM) Snavely et al. (2006); pro (2014). However, our notion of "object positioning" should not be confused with the computer vision concept of object localization, which seeks to find the relative location of objects within an image or point cloud. For mobile systems, closely related to OPS, Hile et al. (2010) uses SfM for a "landmark-based" directional navigation system. SfM, along with a preexisting database containing many photographs of the area in view, enable a augmented reality view with highlighted annotations for known landmarks. More-directly related to the goals of OPS, Kaminsky et al. (2009) seeks to align the output of structure from motion to real-world coordinates, using GPS coordinates to improve the scalability of structure from motion when using hundreds of photos. However, the techniques require the availability of overhead maps and are not suitable to the extremely small number of photos (typically four) expected for OPS.

Real-time 3D reconstruction and distance estimation Real-time 3D reconstruction has been gaining increasing attention over the last few years Izadi et al. (2011). The first commerical prototype of in-phone 3D reconstruction is now avail-

able in pro (2014). With the advances in these areas, we feel OPS can be further strengthened and realized into a true product. The other way of approaching to distant object localization could be by estimating absolute distance of the object using depth cameras such as Kinect. Unfortunately, range of such cameras is often limited to a few meters, hence can not be used in OPS like applications. Nevertheless, many AR applications use tricks to estimate distances of near by object as discussed in Debnath and Borcea (2013).

Building Object Inventories

Our use of multimodal sensing inputs for estimating object position is related to the techniques in Chen et al. (2010) for building inventories of books in a library. The authors project a rough indoor location for a book through WiFi and compass, then apply vision techniques to visually detect book spines.

Applying Computer Vision to Infer Context

CrowdSearch Yan et al. (2010) combines human validation with location-specific image search (for objects such as buildings) through computer vision. By leveraging Amazon Mechanical Turk for human-in-the-loop operation, CrowdSearch enables precision in cases of poor image quality. TagSense Qin et al. (2011) infers "tags" of human context for photographs, by combining computer vision with multimodal sensing. OPS is complementary to CrowdSearch and TagSense, by enabling awareness of precise location for objects in photographs.

Object Recognition and Location Inference

In Magnor (2002), the authors try to estimate the 3D-orientation and location of an object in the real world scene using a polygonal 3D-model of a depicted object. The accuracy of OPS could be enhanced opportunistically by utilizing the size of known, recognizable objects when they happen to appear in the view.

2.9 Conclusion

Today's augmented reality applications are less useful than their potential. We believe this is due, at least in part, to an unfortunate usage asymmetry. Users can retrieve available localized content in a natural way, viewing pop-up annotations for the world through their smartphone, but there is no means to equivalently introduce new annotations of objects in the vicinity. By providing localization for objects in a user's view, this paper seeks to enable convenient content creation for augmented reality. Beyond augmented reality, we believe that a precise *Object Positioning System* can be widely enabling for a variety of applications, and is worthy of a significant research endeavor. In this light, we believe that our approach, OPS, takes a substantial first step.

Table 2.3: OPS Final Object Localization

$$
\begin{aligned}
\textbf{Minimize} \quad & \frac{n-1}{2} \cdot \sum_{\forall i} |E_i^\theta| + \sum_{\forall i,j} |E_{ij}^\gamma| \\
\textbf{Subject to} \quad & (a - C_x)^2 + (b - C_y)^2 = D^2 \\
\forall i \quad : \quad & b - F_i^y = (a - F_i^x) \cdot \cot(\theta_i + E_i^\theta) \\
\forall i \quad : \quad & \vec{V_i} = (a - F_i^x)\hat{\imath} + (b - F_i^y)\hat{\jmath} \\
\forall i,j \quad : \quad & \gamma_{ij} + E_{ij}^\gamma = \\
& \arccos\left(\vec{V_i} \cdot \vec{V_j} / |\vec{V_i}||\vec{V_j}|\right) \\
\textbf{Solving for} \quad & a, b \\
\forall i \quad : \quad & E_i^\theta \\
\forall i,j \quad : \quad & E_{ij}^\gamma \\
\textbf{With parameters} \quad & C_x, C_y, D \\
\forall i \quad : \quad & F_i^x, F_i^y, \theta_i \\
\forall i,j \quad : \quad & \gamma_{ij}
\end{aligned}
$$

Name	**Parameter Sources**
F_i^x, F_i^y	Fixed GPS position (at each photograph)
θ_i	Compass bearing (at each photograph)
γ_{ij}	Vision estimate for vector angle $\angle \vec{V_i}\vec{V_j}$
C_x, C_y	2D Median of $\{\forall i : (F_i^x, F_i^y)\}$
D	Estimated distance from (C_x, C_y) to (a, b)
Name	**Intermediate Results**
$\vec{V_i}$	Vector from (F_i^x, F_i^y) to (a, b)
Name	**Solved Variable Interpretation**
a, b	Estimated object location at (a, b)
E_i^θ	Estimated error for compass bearing θ_i
E_{ij}^γ	Estimated error in vector angle $\angle \vec{V_i}\vec{V_j}$

3

Line-of-Sight: A New Paradigm for Video Analytics

Crowdsourced video often provides engaging and diverse perspectives not captured by professional videographers. Broad appeal of user-uploaded video has been widely confirmed: freely distributed on YouTube, by subscription on Vimeo, and to peers on Facebook/Google+. Unfortunately, user-generated multimedia can be difficult to organize; these services depend on manual "tagging" or machine-mineable viewer comments. While manual indexing can be effective for popular, well-established videos, newer content may be poorly searchable; live video need not apply. We envisage video-sharing services for live user video streams, indexed automatically and in realtime, especially by shared content. We propose *FOCUS*, for Hadoop-on-cloud video-analytics. FOCUS uniquely leverages visual, 3D model reconstruction and multimodal sensing to decipher and continuously track a video's line-of-sight. Through spatial reasoning on the relative geometry of multiple video streams, FOCUS recognizes shared content even when viewed from diverse angles and distances. In a 70-volunteer user study, *FOCUS' clustering correctness is roughly comparable to humans*.

3.1 Introduction

With the ubiquity of modern smartphones, photo and video journalism is no longer limited to professionals. By virtue of having a Internet-connected videocamera always at arms' reach, the average person is always ready to capture and share exciting or unexpected events. With the advent of Google Glass, the effort required to record and share will soon disappear altogether. The popularity of YouTube and video sharing on social media (e.g., Facebook and Google+) is evidence enough that many already enjoy creating and distributing their own video content, and that such video content is valued by peers. Indeed, major news organizations have also embraced so-called "citizen journalism," such as CNN *iReport*, mixing amateur-sourced content with that of professionals, and TV broadcasting this content worldwide.

Amateur video need not be immediately newsworthy to be popular or valuable. Consider a sporting event in a crowded stadium. Often, spectators will film on their smartphones, later posting these videos to YouTube, Facebook, etc. Even if their content is generally mundane, these videos capture the unique perspective of the observer, and views potentially missed by professional videographers, even if present. Unfortunately, given a multitude of sources, such video content is difficult to browse and search. Despite the attempts of several innovative startups, the value can be lost due to a "needle in a haystack" effect Str (2013b); Swi (2013); Vyc (2013); Str (2013a). To provide some organization, websites like YouTube rely on an haphazard index of user-provided tags and comments. While useful, tags and comments require manual effort, may not be descriptive enough, are subject to human error, and may not be provided in realtime — thus, not amenable to live video streams. In contrast, we envisage a realtime system to extract content-specific metadata for live video streams. This metadata is sufficiently precise to immedi-

ately identify and form "clusters" of synchronized streams with related content, especially, a precise subject in shared "focus."

In this paper, we propose *FOCUS*, a system for realtime analysis and clustering of user-uploaded video streams, especially when captured in nearby physical locations (e.g., in the same stadium, plaza, shopping mall, or theater). Importantly, FOCUS is able to deduce content similarity even when videos are taken from dramatically different perspectives. For example, two spectators in a soccer stadium may film a goal from the East and West stands, respectively. With up to 180 degrees of angular separation in their views, each spectator may capture a distinct (*uncorrelated*) background. Even the shared foreground subject, the goalkeeper, will appear substantially different when observed over her left or right shoulder. Without a human understanding of the game, it would be difficult to correlate the East and West views of the goalkeeper, while distinguishing from other players on the field. Novelly, FOCUS' analysis reasons about the relative camera location and orientation of two or more video streams. The geometric intersection of line-of-sight from multiple camera views is indicative of shared content. Thus, FOCUS is able to infer logical content similarity even when video streams contain little or no visual similarity. record and upload video using our Android app, which pairs the video content with precise GPS-derived timestamps and contextual data from sensors, including GPS, compass, accelerometer, and gyroscope. Each video stream arrives at a scalable service, built on the IBM SmartCloud, designed for deployment on an infrastructure-as-a-service cloud, where a Hadoop-based pipeline performs a multi-sensory analysis. This analysis blends smartphone sensory inputs along with *structure from motion*, a state-of-the-art technique from computer vision. For each stream, FOCUS develops a model of the user's line-of-sight across time, understanding the geometry of the camera's view — position and orientation, or *pose*.

Across multiple user streams, FOCUS considers video pairs, frame by frame. For each pair, for each frame, FOCUS determines commonality in their respective lines-of-sight, and assigns a "similarity" score. Across multiple feeds, across multiple frames, these scores feed a pairwise *spatiotemporal* matrix of content similarity. FOCUS applies a form of clustering on this matrix, invoking ideas from community identification in complex networks, returning groups with a shared subject.

FOCUS remains an active research project, as we endeavor to further harden its accuracy and responsiveness. However, despite ample room for further research, *we believe that FOCUS today makes the following substantial contributions:*

1. **Novel Line-of-Sight Video Content Analysis:** FOCUS uses models derived from multi-view stereo reconstruction to reason about the *relative position and orientation of two or more videos*, inferring shared content, and providing robustness against visual differences caused by distance or large angular separations between views.
2. **Inertial Sensing for Realtime Tracking:** Despite an optimized Hadoop-on-cloud architecture, the computational latency of visual analysis remains substantial. FOCUS uses lightweight techniques based on smartphone sensors, as a form of dead reckoning, to provide *continuous realtime video tracking* at sub-second timescales.
3. **Clustering Efficacy Comparable to Humans:** In a 70-volunteer study, clustering by modularity maximization on a "spatiotemporal" matrix of content similarity yields a *grouping correctness comparable to humans*, when measured against the ground truth of videographer intent.

3.2 Intuition

User-generated, *crowdsourced*, multimedia has clear value. YouTube and other sharing sites are immensely popular, as is community-sourced video on Facebook and Google+. The value of particular shared content, however, can be lost in volume, due to the difficulty of indexing multimedia. Today, user-generated "tags" or mineable text comments aid peers while browsing rich content. Unfortunately, newer and real-time content cannot benefit from this metadata.

We envisage large-scale sharing of live user video from smartphones. Various factors are enabling: the pervasiveness of smartphones and increasing use of social apps; improving cellular data speeds (e.g., 4G LTE); ubiquity of Wi-Fi deployments; increased availability and adoption of scalable, cloud-based computation, useful for low-cost video processing and distribution; enhanced battery capacity, computational capabilities, sensing, and video quality of smartphones; and the advent of wearable, wirelessly-linked videocameras for smartphones, such as in Google Glass.

The dissemination of live, mobile, crowdsourced multimedia is relevant in a variety of scenarios (e.g., sports). However, to extract this value, its presentation must not be haphazard. It must be reasonably straightforward to find live streams of interest, even at scales of hundreds or thousands of simultaneous video streams. In this paper, we propose *FOCUS*, a system to enable an organized presentation, especially designed for live user-uploaded video. FOCUS automatically extracts contextual metadata, especially relating to the line-of-sight and subject in "focus," captured by a video feed. While this metadata can be used in various ways, our primary interest is to classify or *cluster* streams according to *similarity*, a notion of shared content. In this section, we will consider what it means for a pair of video

streams to be judged as "similar," consider approaches and metrics for quantifying this understanding of similarity, and identify challenges and opportunities for extracting metric data on commodity smartphones.

3.2.1 Characterizing Video Content Similarity

While there can be several understandings of "video similarity," we will consider two videos streams to be more similar if, over a given period of time, a synchronized comparison of their constituent frames demonstrates greater "subject similarity." We judge two frames (images) to be similar depending on how exactly each captures the same physical object. Specifically, that object must be the subject, the focal *intent* of the videographer. By this definition, subject-similar clusters of live video streams can have several applications, depending on the domain. In sporting events, multiple videos from the same cluster could be used to capture disparate views of a contentious referee call, allowing viewers to choose the most amenable angle of view — enabling a crowdsourced "instant replay." For physical security, multiple views of the same subject can aid tracking of a suspicious person or lost child. For journalism, multiple views can be compared, vetting the integrity of an "iReport."

It is important to note that this definition of similarity says nothing of the perspective of the video (i.e., the location from where the video is captured), so long as the foreground subject is the same. We believe our definition of similarity, where the angle of view is not considered significant, is especially relevant in cases of a human subject. Naturally, two videos of a particular athlete share "similar" content, regardless of from which grandstand she is filmed. However, if these videos are captured from a wide angular separation, they may "look" quite distinct. Contingent on the angle of separation, the visual structure and color of an object or person, lighting conditions (especially due to the position of the sun early or

late in the day), as well as the background, may vary considerably. Perhaps counterintuitively, two "similar" views might actually look quite different (Figure 3.1). Our techniques must accommodate this diversity of view. Using the techniques we describe in Section 3.3, it would also be possible to judge a pair of videos shot from a more-nearby location as more similar. However, we see fewer immediate applications of this definition, and henceforth exclude it.

By our definition, videos which look heterogeneous may be judged similar, if they share the same subject. Further, videos which look homogenous may be judged dissimilar, if their subjects are physically different. For example, videos that capture different buildings, but look homogenous due to repetitive architectural style, should not be considered similar. Thus, a system to judge similarity must demonstrate a high certainty in deciding whether the object in a video's focus is truly the same precise subject in some other video.

Visual Metrics for Content Similarity

Understanding that two videos that "look" quite different might be judged "similar," and *vice versa*, several otherwise-reasonable techniques from vision are rendered less useful. Histograms of color content, spatiograms Birchfield and Rangarajan (2005), and feature matching Rosten and Drummond (2006), are valuable for tracking an object across frames of a video though a "superficial" visual similarity. However, they are not intended to find similarity when comparing images that, fundamentally, may share little in common, visually. Though complementary to our approach, visual comparison is insufficient for our notion of subject-based similarity.

FIGURE 3.1: Time-synchronized frames from four videos of an athlete on a stadium running track. Note that these frames are considered "similar," capturing the same athlete, but "look" heterogeneous.

Leveraging Line-of-Sight

Our definition of similarity requires a precise identification of a shared subject (difficult). One possible proxy is to recognize that a pair of videos capture *some subject at the same physical location.* If we know that a pair of videos are looking towards the same location, at the same time, this strongly indicates that they are observing the same content. Precisely, we can consider the line-of-sight of a video, geometrically, a vector from the camera to the subject. More practically, we can consider the collinear infinite ray from the same point-of-origin and in the same direction. The geometric relationship of a pair of these rays reflects the similarity of the corresponding videos, at the corresponding precise instant in time. A maximally-similar pair of views will have line-of-sight rays that perfectly intersect in 3D — the intersection point will be within the volume of their mutual subject (e.g., person or building). Line-of-sight rays which do not nearly intersect will not be similar. Figure 3.2 illustrates that, with multiple videos, intersecting line-of-sight rays suggest shared video content. Consistency across time reinforces the indication.

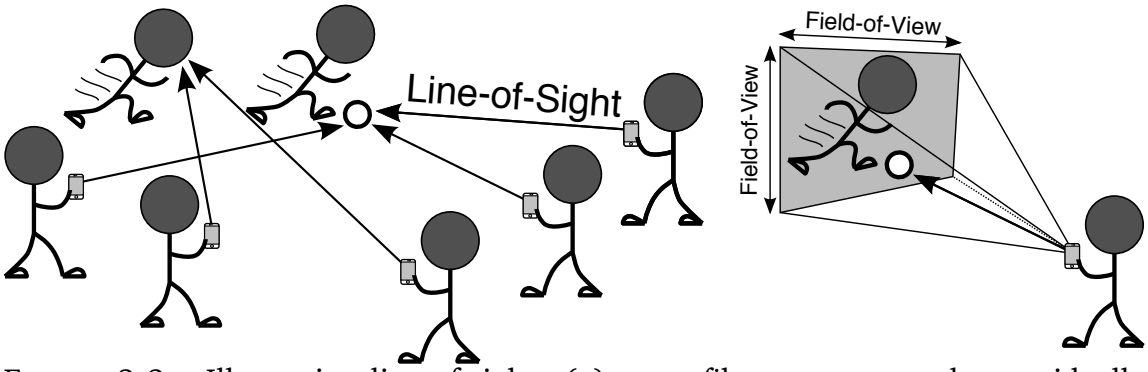


FIGURE 3.2: Illustrating line-of-sight: (a) users film two soccer players, ideally defining two clusters, one for each; (b) line-of-sight can be better understood as a 3D pyramid-shaped region, capturing the camera's horizontal and vertical *field-of-view* angles.

Strictly, and as illustrated by the left athlete in Figure 3.2(a), intersecting line-of-sight rays does not guarantee a shared subject. The principle subject of each video may appear in the foreground of (or behind) the intersection point. Thus, an inference of similarity by line-of-sight must be applied judiciously. In Section 3.3, we explain how FOCUS's similarity metric leverages vision to roughly estimate the termination point of a line-of-sight vector, substantially reducing the potential for *false positive* similarity judgments.

Our system, FOCUS, leverages multi-view stereo reconstructions and gyroscope-based dead-reckoning to construct 3D geometric equations for videos' lines-of-sight, in a shared coordinate system. More precisely, we will define four planes per video frame to bound an infinite, pyramid-shaped volume of space, illustrated in Figure 3.2(b). The angular separation between these planes corresponds to the camera's *field-of-view*, horizontally and vertically, and is distributed symmetrically across the line-of-sight ray. Geometric and computer vision calculations, considering how thoroughly and consistently these pyramid-shaped volumes intersect the same physical space, will form the basis for a content similarity metric.

To simplify, FOCUS understands the geometric properties of the content observed in a video frame, according to line-of-sight and field-of-view. FOCUS compares the geometric relationship between the content one video observes with that of others. If a pair of videos have a strong geometric overlap, indicating that they both capture the same subject, their content is judged to be "similar." Ultimately, groups of videos, sharing a common content, will be placed in self-similar groups, called *clusters*. Clusters are found through a technique called *weighted modularity maximization*, borrowed from *community identification* in complex networks. FOCUS finds "communities" of similar videos, derived from their geometric, or "spatial" relationship with time. Thus, uniquely, we say FOCUS groups live user videos streams based on a *spatiotemporal* metric of content similarity.

3.2.2 Opportunities to Estimate Line-of-Sight

With the availability of sensors on a modern smartphone, it may seem straightforward to estimate a video's line-of-sight: GPS gives the initial position; compass provides orientation. Unfortunately, limited sensor quality and disruption from the environment (e.g., difficulty obtaining a high-precision GPS lock due to rain or cloud cover, presence of ferromagnetic material for compass) may make line-of-sight inferences too error-prone and unsuitable for video similarity analysis.

Figure 3.3 illustrates this imprecision; lines of the same color should converge. Further, GPS is only useful outdoors — applications in indoor sporting arenas, shopping malls, and auditoriums would be excluded. Of course, a sensing-only approach can be valuable in some scenarios: outdoors when a reduced precision is tolerable. In Section 3.3.5, we use GPS, compass, and gyroscope for a lightweight clustering, formulated to minimize the impact of compass imprecision.

Smartphone sensing is, in general, insufficient for estimating a video's line-of-

sight. The content of video itself, however, presents unique opportunities to extract detailed line-of-sight context. Using the well-understood *geometry of multiple views* from computer vision, it is possible to estimate the perspective from which an image has been captured. In principle, if some known reference content in the image is found, it is possible to compare the reference to how it appears in the image, deducing the perspective at which the reference has been observed. At the most basic level, how large or small the reference appears is suggestive of from how far away it has been captured. In the next section, we describe how our solution leverages *structure from motion*, a technique enabling analysis and inference of visual perspective, to reconstruct the geometry of a video line-of-sight.

Both smartphone sensing and computer vision provide complimentary and orthogonal approaches for estimating video line-of-sight. This paper seeks to blend the best aspects of both, providing high accuracy and indoor operation (by leveraging computer vision), taking practical efforts to reduce computational burden (exploiting gyroscope), and providing failover when video-based analysis is undesirable (using GPS/compass/gyroscope). We actualize our design next, incorporating this hybrid of vision/multimodal sensing.

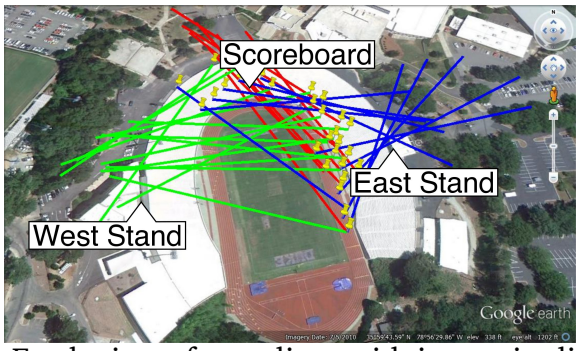


FIGURE 3.3: Google Earth view of a stadium with imprecise line-of-sight estimates from GPS/compass: (green) towards West Stands, left; (red) towards Scoreboard, top; (blue) towards East Stands, right.

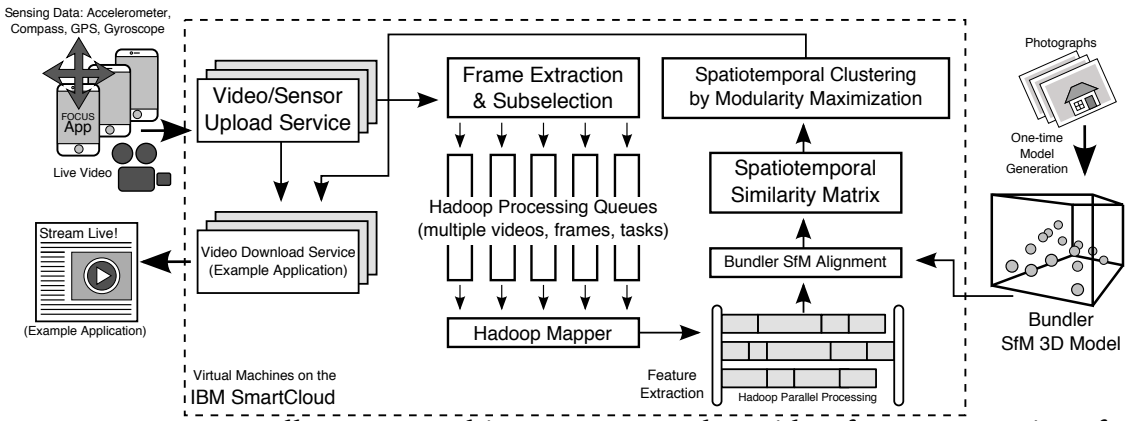


FIGURE 3.4: Overall FOCUS architecture. Note that video frame extraction, feature analysis, and alignment to an SfM model are parallelizable, enabling high analysis throughput from our Hadoop-on-cloud prototype.

3.3 Architecture and Design

Fundamentally, an accurate analysis of content similarity across video streams must consider the video content itself — it most directly captures the intent of the videographer. Accordingly, reflecting the importance of visual inputs, we name our system *FOCUS* (*F*ast *O*ptical *C*lustering of *U*ser *S*treams). FOCUS leverages those visual inputs to precisely estimate a video stream's line-of-sight. Geometric and sensory metadata provides context to inform a spatiotemporal clustering, derived from community identification, to find groups of subject-similar videos. The FOCUS design, while willing to exercise substantial computation, is considered with a view towards real-world deployability, emphasizing (1) scalability, leveraging a cloud-based elastic architecture, and (2) computational shortcuts, blending computer vision with inertial sensing inputs into a hybrid analysis pipeline.

Architectural Overview

The FOCUS architecture includes: (1) a mobile app (prototyped for Android) and (2) a distributed service, deployed on an infrastructure-as-a-service cloud using Hadoop. The app lets users record and upload live video streams annotated with

time-synchronized sensor data, from GPS, compass, accelerometer, and gyroscope. The cloud service receives many annotated streams, analyzing each, leveraging computer vision and sensing to continuously model and track the video's line-of-sight. Across multiple streams, FOCUS reasons about relative line-of-sight/field-of-view, assigns pairwise similarity scores to inform clustering, and ultimately identifies groups of video streams with a common subject.

Figure 3.4 illustrates the overall flow of operations in the FOCUS architecture. We describe the key components in this section. We (1) describe computer vision and inertial sensing techniques for extracting an image's line-of-sight context; (2) consider metrics on that context for assigning a similarity score for a single pair of images; (3) present a clustering-based technique to operate on a two-dimensional matrix of similarity scores, incorporating spatial and temporal similarity, to identify self-similar groups of videos; (4) explain how computations on the Hadoop-on-cloud FOCUS prototype have been optimized for realtime operation; and (5) describe a lightweight, reduced accuracy sensing-only technique for line-of-sight estimation, for use in cases when computer vision analysis is undesirable or impractical.

3.3.1 Line-of-Sight from Vision and Gyroscope

By leveraging an advanced technique from computer vision, *Structure from Motion* (SfM), it is possible to *reconstruct* a 3D representation, or model, of a physical space. The model consists of many points, a *point cloud*, in 3D Euclidean space. It is possible to *align* an image (or video frame) to this model and deduce the image's *camera pose*, the point-of-origin location and angular orientation of line-of-sight, relative to the model. Multiple alignments to the same model infer line-of-sight rays in a single coordinate space, enabling an analysis of their relative geometry. As noted in Figure 3.4, FOCUS leverages Bundler Snavely et al. (2006), an open

source software package for SfM, both for the initial model construction and later video frame-to-model alignment.

While the SfM technique is complex (though powerful and accurate), its usage is straightforward. Simply, one must take several photographs of a physical space (while a minimum of four is sufficient, efficacy tends to improve with a much larger number of photos). With these images as input, SfM operates in a pipeline: (1) extraction of the salient characteristics of a single image, (2) comparison of these characteristics across images to find shared points of reference, and (3) an optimization on these reference points, constrained by the well-understood geometry of multiple views, into a reconstructed 3D point cloud. We assume that the SfM model will be generated and available in advance (perhaps by the operator of a sporting arena), to be used for analysis as live video feeds arrive at the FOCUS cloud service. In Section 3.5, we consider relaxing this assumption, using the content of incoming video feeds themselves for model generation.

FIGURE 3.5: 3D reconstruction using Bundler SfM. 33K points from 47 photos of a university plaza, model post-processed for enhanced density/visual clarity.

Reconstructing 3D from 2D Images

For each image, a set of *keypoints* is found by computing a *feature extractor* heuristic Rosten and Drummond (2006). Each keypoint is a 2D $< x, y >$ coordinate that locates a clear point of reference within an image — for example, the peak of a pitched roof or corner of a window. Ideally, the keypoint should be robust, appearing consistently in similar (but not necessarily identical) images. For each keypoint, a *feature descriptor* is also computed. A feature descriptor may be viewed as a "thumbprint" of the image, capturing its salient characteristics, located at a particular keypoint. We use the SIFT Rosten and Drummond (2006) extractor/descriptor.

Across multiple images of the same physical object, there should be shared keypoints with similar feature descriptor values. Thus, for the next stage of the SfM pipeline, we can perform a N^2 pairwise matching across images, by comparing the feature descriptors of their keypoints. Finally, the true SfM step can be run, performing a nonlinear optimization on these matched keypoints, according to the known properties of *perspective transformation* in a 3D Euclidean space. Once complete, the output is the 3D model in the form of a *point cloud*, consisting of a large number of $< x, y, z >$ points. Each 3D point corresponds to 2D keypoints extracted and matched from the original images.

Figure 3.5 shows the construction of a 33K-point model of a campus plaza from 47 high resolution photos. Figure 3.6 shows an (overhead view) 190K-point cloud generated from 412 photos of a 34K-seat collegiate football stadium. Note that model generation is feasible in both outdoor and indoor spaces, given sufficient light (not shown, we generated a 200- photo, 142K-point model of an indoor basketball arena).

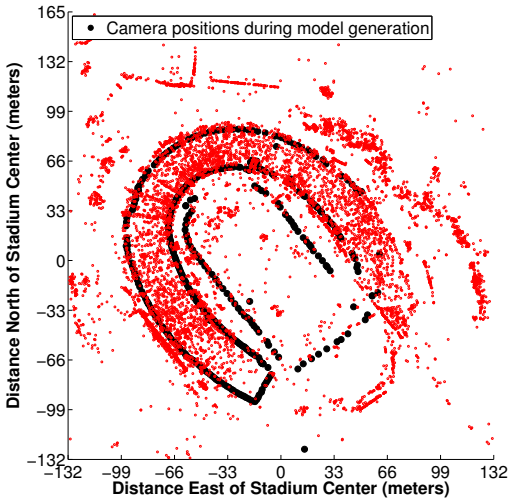


FIGURE 3.6: 3D reconstruction of a collegiate football stadium. Red dots show an overhead 2D projection of the 3D model. Black dots show locations from which photographs of the stadium were captured, systematically, around the top and bottom of the horseshoe-shaped grandstands and edge of the field.

Aligning a Frame to the Model: Estimating Pose

Once a model is constructed, it is possible to *align* a image (or video frame) taken in the same physical space. The alignment results in an estimate of its relative camera *pose*, a 3×1 *translation vector* and a 3×3 *rotational matrix* of orientation. The resulting 4×4 *rotation and translation matrix* can be used to construct the equation of a ray, with a point of origin at the camera, through the subject in the center of the

FIGURE 3.7: Challenging example photo that aligns accurately to our SfM model (Figure 3.6), despite capacity attendance (vs. empty during model capture).

view. This ray follows the line-of-sight from the camera, enabling similarity metrics based on view geometry. In Section 3.4, we evaluate SfM alignment performance against the stadium model construction shown in Figure 3.6. However, even prior to pursuing this technique, it was important to validate that SfM-derived models are robust to transient environmental changes. For example, we generated our stadium model for photos captured during off hours. In Figure 3.7, we present an example image that aligns accurately to our model, taken during a well-attended football game. Despite occluded bleachers, alignment is still feasible as much of the core "structure" of the stadium (i.e., the rigid stands, buildings, and boundaries captured in the model) remains visible.

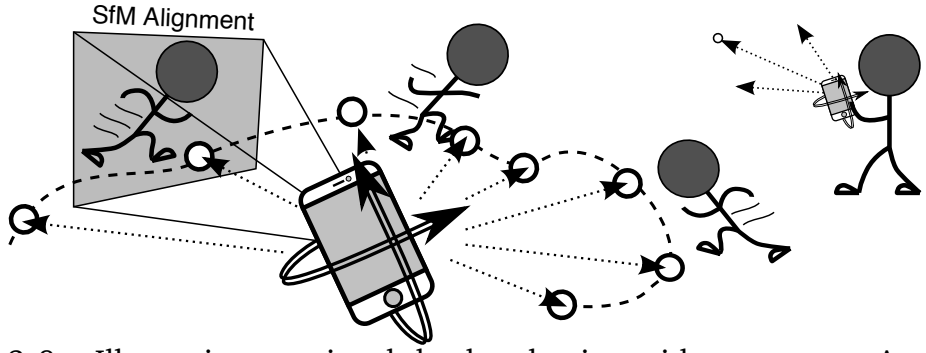


FIGURE 3.8: Illustrating rotational dead reckoning with gyroscope. As the user follows a moving target, gyroscope tracks a rotational matrix "diff" (forward or in reverse in time) from the closest SfM alignment.

Augmenting Vision with Smartphone Sensing

Video frame-to-model alignment, while quick in FOCUS' scalable Hadoop-on-cloud pipeline (Section 3.3.4), is a heavyweight process. To reduce the computational burden, it is useful to combine SfM alignment with inputs from smartphone sensing. The inertial gyroscope, present on most new smartphones today, can provide a rotational "diff" across time, in the form of a rotation matrix. By matrix multiplication, FOCUS combines this gyroscope-derived rotational matrix with that of an

SfM-estimated camera pose. We illustrate this process, akin to a rotational "dead reckoning," in Figure 3.8. Of course, errors will accumulate with time, due to inherent noise in the gyroscope sensor. FOCUS periodically re-runs SfM alignment, resetting this noise, and maintaining a bounded inaccuracy (relative to the last frame-to-model alignment). Moreover, since SfM alignment itself is prone to some error, this input from gyroscope can be used to inform hysteresis across multiple alignment attempts.

Unsurprisingly, video frame-to-model alignment can fail for several reasons: if the frame is blurred, poorly lit (too dark), captures sun glare (too bright), the extracted keypoints or feature descriptors have low correspondence with the model, or if the model is too sparse, self-similar, or does not capture the content of the to-be-aligned frame. In a video stream across time, these failures result in alignment "cavities" between successful alignments. To "fill" the cavities, and achieve a continuous alignment, gyroscope-based dead reckoning is especially useful. Note that dead reckoning is possible in either direction, forward or backward with time, from the nearest successful alignment. To dead reckon forward with time, the SfM-derived rotational orientation matrix is multiplied with a gyroscope-derived rotational matrix, accounting for the relative rotational motion accumulated over the time interval from the last alignment. To dead reckon in reverse, the gyroscope rotational matrix must first be inverted. Luckily, this inversion is trivial: as an invariant, the inverse of a rotation matrix is its transpose.

Other smartphone sensors are also valuable during alignment. GPS, compass, and accelerometer, can be used to estimate a rough camera pose. While these estimates are prone to error, due substantial sources of noise in each sensor, they are valuable to "sanity check" outputs from SfM — immediately rejecting otherwise-silent alignment failures. In these cases, dead reckoning can be applied to over-

write what, otherwise, would be an erroneous alignment result.

3.3.2 Quantifying Spatial Content Similarity

To cluster video feeds into self-similar groups, FOCUS will assume a metric to quantify the logical content "similarity." Pairs of videos with a high mutual similarity are likely to be placed into the same cluster. As an invariant, each video will be placed in the cluster with which it has the greatest spatial (from line-of-sight) content similarity, averaged across time, averaged across all other cluster members. In this subsection, we will present the design of FOCUS' spatial similarity metric for a pair of video frames. FOCUS' metric was not successfully designed "at once;" instead its techniques were evolved and refined through system experimentation.

By Line-of-Sight Intersection (Failed Attempt)

FOCUS leverages SfM and gyroscopic tracking to estimate a 3D rays of camera pose — originating from the camera and along the line-of-sight. For a pair of frames capturing the same object of interest, the corresponding rays should intersect, or nearly intersect, through the mutual object in view. One possible similarity metric is to consider the shortest distance between these two rays. The resulting 3D line segment must be either (1) between the two points of origin, (2) from the point of origin of one ray to a perpendicular intersection on the other, (3) perpendicular to both rays, or (4) of zero length. We may treat cases (1) and (2) as having no view similarity; line-of-sight rays diverge. In cases (3) and (4), shorter line segments reflect a nearer intersection, and suggest a greater view similarity. Assuming that the constructed rays are accurate, this metric is not subject to *false negatives*; for any pair of videos sharing the same content, the length of the line segment between the rays must be small. Unfortunately, this simple metric is not foolproof. *False positive* indications of similarity may result; the intended subject may fall in front

or behind the point of ray intersection. For example, if stadium spectators in the grandstand focus on different players on the field, this metric will return that the views are similar if the corresponding rays intersect "underground."

By SfM "Point Cloud" Volumetric Overlap

During early experimentation, we found that the above ray-intersection metric, while intuitively appealing in its simplicity, is overly susceptible to false positives. We could eliminate the potential for false positives by replacing each camera pose ray with a vector, terminating at the object in view. While this is difficult to estimate, we can leverage context from the 3D model structure to terminate the vector roughly "on" the model, for example, capturing the ground below the subject. Looking down from a stadium grandstand, subterranean intersections would be eliminated. Similarity, intersections in the air, above the field, can be ignored.

Recall that the SfM model is a point cloud of $< x, y, z >$ coordinates, capturing rigid structures. Instead of only considering the geometry of a line-of-sight ray, we may identify structures captured by a video frame. For a pair of videos, we can compare if both capture the same structures. Several techniques from vision apply here. For example, 3D *point cloud registration* heuristics exist for estimating boundaries of a mesh surface, and approximating structures. However, as a simpler, computationally-tractable alternative, we may count the model points mutually visible in a pair of video frames. More shared points suggest greater similarity in their views. In the next subsection, we discuss how high similarity values, filling an $N \times N$ spatial similarity matrix, encourage placement of these videos in the same cluster.

To count the number of common points in the intersecting field-of-views of two videos, we must first isolate the set of points visible in each. As we describe

next, the set can be found by considering the pyramid-shaped field-of-view volume, originating from the camera and expanding with distance into the model. Later, we can quickly count the number of shared points across multiple video frames by applying a quick set intersection. Simply, we construct a bitmap with each bit representing the presence of one point in the view.[1] The number of shared points can be found by counting the number of bits set to 1 in the bitwise AND of two bitmaps.

Finding Model Points in a Video View

Let a single estimated line-of-sight be expressed as $L(R, t)$. R represents a 3×3 rotation matrix, the 3D angle of orientation. t represents a 3×1 vector of translation. $-R^{-1}t$ defines the $< x, y, z >$ camera position coordinate, the location in the model from where the video frame was captured. R can be further decomposed as three row vectors, known respectively as RIGHT, UP, and OUT, from the perspective of the camera. To capture the camera's view of the model, we form a pyramid emerging from the camera position ($-R^{-1}t$) and extending in the direction of OUT vector. The four triangular sides of the pyramid are separated, horizontally and vertically, according to the camera's field-of-view (Figure 3.2).

The pyramid-shaped camera view can be abstracted as four planes, all intersecting at the camera position coordinate. Now, to fully describe equations for these planes, we must only find a plane normal vector for each. In order to find four plane normals, we rotate the OUT vector along the RIGHT and UP vectors, so that the transformed OUT vector becomes perpendicular to one of these planes. Rotation of any 3D vector, along a unit-length 3D vector, is given by Rodrigues' rotation formula. Using this equation, we rotate the OUT vector along the RIGHT vector by

[1] Alternatively, Bloom Filters are suitable as probabilistic substitutes for direct point-to-bit maps, for space-saving bitmaps.

angle $\pm(\pi/2 - vAngle/2)$ to estimate normals for two planes (top/bottom). Similarly, rotations along the UP vector with angle $\pm(\pi/2 - hAngle/2)$ results in normals to left and right planes. Here, $vAngle$ and $hAngles$ are taken as parameters for the smartphone camera's field-of-view angle, horizontally and vertically. We test the signs from these four planar equations for each point in the model, determining the set of points potentially visible from a particular video frame. Later, we perform set intersections to estimate similarity between the N^2 pairs of time-synchronized frames of N videos. This $N \times N$ value table completes our notion of a *spatial similarity matrix*.

3.3.3 Clustering by Modularity Maximization

So far, we have discussed what it means for a pair of video frames to be judged "similar," especially by the intersection of their respective line-of-sight and field-of-view with an SfM-derived 3D model. This notion of "similarity" is a static judgment, based on an instantaneous point in time. In reality, we are interested in the similarity of a pair of videos, across multiple frames, for some synchronized time interval. This requires a further understanding of what it means for a pair of videos to be "similar," above and beyond the similarity of their constituent frames. We will assume that a pair of "similar" video streams need not both track the same spot consistently. Instead, it is only required that they should both move in a correlated way, consistently capturing the same physical subject, at the same time. Simply, both streams should maintain (instantaneous) similarity with each other across time, but not necessarily have self-similarity from beginning to end. This seems reasonable in the case of a soccer game: some videos will follow the ball, some will follow a favored player, and others will capture the excitement of the crowd or changes to the scoreboard. These "logical clusters" should map as neatly as possible to FOCUS' groupings.

Spatiotemporal Similarity

To capture the mutual correspondence in a set of N videos with time, we apply our notion of an $N \times N$ spatial similarity matrix across T points in time. For every instant $t \in T$ in a synchronized time interval, we find the corresponding spatial matrix S_t and apply clustering, finding some set of groupings G_t from line-of-sight and field-of-view at time t. Next, we aggregate these spatial results into an $M = N \times N$ *spatialtemporal* similarity matrix. Let $\delta_g(i, j) = 1$ if streams i and j are both placed into the same spatial cluster $g \in G_t$. $\delta_g(i, j) = 0$, otherwise.

$$M_{ij} = \sum_{\forall t \in T} \sum_{\forall g \in G_t} \delta_g(i, j)$$

Finally, we apply clustering again, on M, providing groups of videos matching our notion of *spatiotemporal* similarity. Next, we elaborate on our choice of clustering heuristics.

The "Right" (and Right Number) of Clusters

Several clustering approaches require some parameterization of how many clusters are desired (e.g., the k value in k-means clustering). By comparison, community identification via *modularity maximization* has the appealing property that community boundaries are a function of their *modularity*, that is, a mathematical measure of network division. A network with high modularity implies that it has high correlation among the members of a cluster and minor correlation with the members of other clusters. For FOCUS, we apply a weighted modularity maximization algorithm Clauset et al. (2004). As input, FOCUS provides an $N \times N$ matrix of "similarity" weights — either that of spatial or spatiotemporal similarity values. Modularity maximization returns a set of clusters, each a group of videos, matching our notions of content similarity.

3.3.4 Optimizing for Realtime Operation

A key motivation for FOCUS is to provide content analysis for streaming realtime content *in realtime*. Excess computational latency cannot be tolerated as with it increases (1) the delay before content consumers may be presented with clustered videos and (2) the monetary costs of deploying the FOCUS Hadoop-on-cloud prototype. Here, FOCUS makes two contributions. First, as previously discussed, FOCUS leverages gyroscope-based dead reckoning as a lightweight proxy to fill gaps between SfM camera pose reconstruction, reducing the frequency of heavyweight computer vision (alignment of an image to an SfM-derived model) to once in 30 seconds. Second, FOCUS applies *application checkpointing* to combat startup latency for SfM alignment tasks.

As described in Section 3.3.1, FOCUS uses Bundler to align an image (estimate camera pose) relative to a precomputed 3D model. Bundler takes approximately 10 minutes to load the original model into memory before it can initiate the relatively quick alignment optimization process. To avoid this latency, FOCUS uses BLCR Duell et al. (2005) to *checkpoint* the Bundler application (process) state to disk, just prior to image alignment. The process is later *restarted* with almost zero latency, each time substituting the appropriate image for alignment.

FOCUS Hadoop Prototype Cluster

FOCUS exists as a set of cloud virtual machine instances configured with Apache Hadoop for MapReduce processing. FOCUS informs virtual machine elastic cloud scale-up/down behavior using the Hadoop queue size (prototype FOCUS elasticity manager currently in development). For FOCUS, there are several types of MapReduce task: (1) base video processing, to include decoding a live video feed and sampling frames for further image-based processing; (2) image feature extraction,

computation of feature descriptors for each keypoint, alignment to an SfM model, and output of a bitmap enumerating the set of visible model points; (3) pairwise image feature matching, used when building an initial 3D SfM model; and (4) clustering of similar video feeds. Tasks of multiple types may be active simultaneously.

3.3.5 Failover to Sensing-only Analysis

In certain circumstances, it may be undesirable or infeasible to use SfM-based line-of-sight estimation. For example, in an "iReport" scenario, video may be captured and shared from locations where no SfM model has been previously built. Further, users may choose to upload video only if very few peers are capturing the same video subject — saving battery life and bandwidth for the user. A lightweight clustering technique, without requiring upload of the video stream, could be used to pre-filter uploads of redundant streams. FOCUS provides a sensing-only alternative (using GPS and compass), clustering streams without requiring computer vision processing or even access to video sources. Through iterative design and testing, we have refined our technique to be relatively insensitive to compass error. By considering the wide camera field-of-view angle in the direction of line-of-sight, our metric is not substantially impacted by compass errors of comparable angular size.

For each latitude/longitude/compass tuple, FOCUS converts the latitude/longitude coordinates to the rectangular Universal Transverse Mercator (UTM) coordinate system, taking the EASTING and NORTHING value as an $< x, y >$ camera coordinate. From the camera, the compass angle is projected to find a 2D line-of-sight ray. Next, two additional rays are constructed, symmetric to and in the same direction as the line of sight ray, and separated by the horizontal camera field-of-view ($hAngle$). This construction can be visualized as a triangle emerging from the GPS location of a camera and expanding outward to infinity (with an

angle equal to the camera's horizontal field-of-view). A metric for view similarity is computed as the area bounded by intersecting two such regions. Since this area can be infinite, we impose an additional bounding box constraint. The resulting metric values are used to populate the spatiotemporal similarity matrix. Clustering proceeds as for SfM-based similarity. To reduce the potential for compass error, gyroscope informs a hysteresis across multiple compass line-of-sight estimates.

To compute the area of intersection (and thus our metric), we find the intersection of the constraining rays with each other and with the bounding box, forming the vertices of a *simple* (not-self-intersecting) polygon. We may order the vertices according to *positive orientation* (clockwise) by conversion to polar coordinates and sorting by angle. Next, the polygon area is found by applying the "Surveyor's Formula."

3.4 Evaluation

Through controlled experimentation[2] *in two realistic scenarios and comparison of FOCUS' output with human efforts,*[3] *we endeavor to answer the following key questions:*

1. How accurate is line-of-sight for identifying unique subject locations? *(Figs. 3.10, 3.12)* Indoors? *(Fig. 3.14)* For objects only a few meters apart? *(Figs. 3.11b, 3.14b)*
2. How does GPS/compass-based line-of-sight estimation compare with SfM/gyroscope? *(Figs. 3.3, 3.11, 3.18)*
3. When video streams are misclassified, are incorrectly clustered videos placed in a reasonable alternative? Are SfM processing errors silent or overt (en-

[2] Constrained by (a) privacy for video-recording human subjects and (b) copyright ownership for NCAA athletic events.

[3] The procedures for this study were vetted and approved in advance by our institution's ethics and legal review board.

abling our gyroscope-based hysteresis/failover)? *(Figs. 3.12, 3.13)*

4. It our spatiotemporal similarity matrix construction robust to videos with dynamic, moving content, tracking spatially-diverse subjects with time? *(Fig. 3.15)*
5. What is the latency of vision-based analysis? *(Figs. 3.16, 3.17)*
6. Is FOCUS' sensing-only failover approach tolerant to large compass errors? Can GPS/compass provide a reasonable accuracy when SfM/gyroscope is undesirable, infeasible, or as temporary failover? *(Figs. 3.3, 3.18)*
7. How do FOCUS's clusters compare with those created by human volunteers? *(Fig. 3.20)*

Methodology

Our FOCUS evaluation takes the perspective of a likely use case: user video streams in a collegiate football stadium and an indoor basketball arena. With 33K seats and a maximum attendance of 56K, by sheer numbers, it is likely that multiple (even many) visitors to our stadium would choose to stream video simultaneously. To exercise FOCUS across dynamic lighting by time of day, a range of cloud cover, variations in attendance, a span of videographer skill, and transient occlusions,[4] we collected short video clips from the stadium over a period of two months.

To build the stadium SfM model, containing 190K points and shown in Figure 3.6, we took 412 2896x1944 resolution photographs with a Nikon D3000 SLR camera. All videos were taken in 1920x1088 resolution at 15 FPS, using our Android app, on a Samsung Galaxy Nexus or Galaxy S3. In line-of-sight estimation results, short clips (20-30 seconds each) were used. For longer motion tracking results, videographers were asked to keep the running "athlete" centered in view

[4] For example, an opaque protective tarp not captured by our SfM model was typically present, covering the entire field.

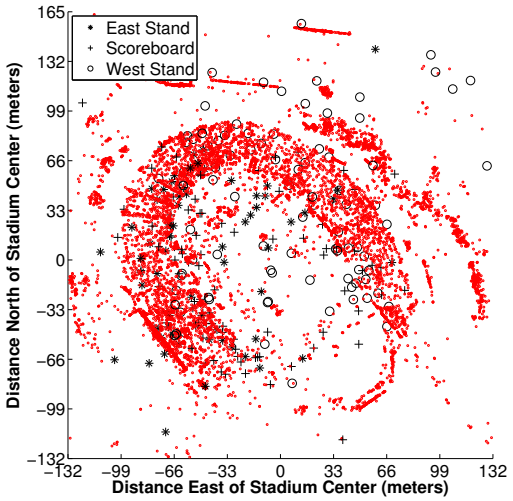


FIGURE 3.9: Experimental locations in/around the stadium. Symbols denote each video's focal subject: (∗) East Stand; (+) Scoreboard; and (○) West Stand.

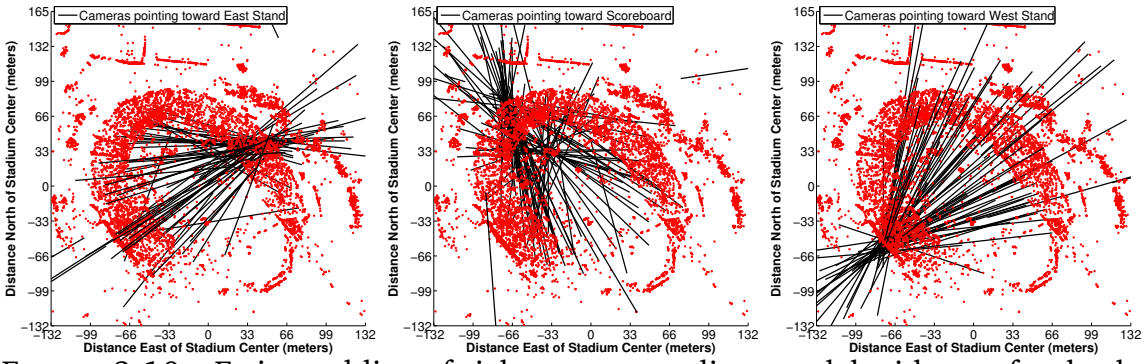


FIGURE 3.10: Estimated line-of-sight rays on stadium model with true focal subject: (a) in the East Stand; (b) on the Scoreboard; and (c) in the West Stand. Converging lines reflect precision of SfM line-of-sight estimation.

as consistently as possible, over periods of minutes.

Line-of-sight Estimation Accuracy

Using our app, three large, preselected points-of-interest in the stadium were filmed: (1) bleachers where the "pep" band sits, centered in the the EAST STAND; (2) the press box above mid-field in the WEST STAND; and (3) the SCOREBOARD.[5]

[5] Large subject areas chosen to reduce impact of human filming error. *Figure 3.11(b) confirms applicability to small subjects.*

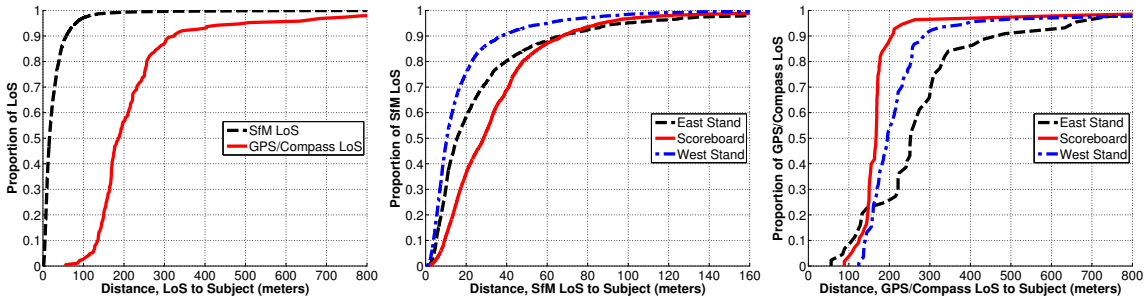


FIGURE 3.11: CDFs of alignment "error." Y-axis shows the shortest distance from the estimated line-of-sight ray to the intended subject: (a) SfM/gyroscope versus GPS/compass overall; (b) SfM/gyroscope by subject; (c) GPS/compass by subject. Note that a nontrivial error proportion is attributable to videographer imprecision.

Videographers walked around arbitrarily, capturing each of the intended subjects from 325 locations (shown in Figure 3.9, with locations from GPS, marked by intended subject). Figures 3.10 (a,b,c) show visualizations of SfM/gyroscope-based line-of-sight estimation accuracy, for each subject. Dark lines show estimated line-of-sight. Rough convergence through the designated subject in the EAST STAND, SCOREBOARD, or WEST STAND, respectively, visually suggests that typically SfM frame-to-model alignment is highly accurate. Figure 3.11 (a,b) plot CDFs, confirming this accuracy. Further, 3.11 (a,c) confirm the inferior accuracy of line-of-sight estimation with GPS/compass (only). Note that in both Figures 3.10 and 3.11, a substantial portion of angular "error" is attributable to inaccuracy in filming a designated subjects. Visible outliers in Figure 3.10 (b) are attributable to poor SfM alignment, typically due to difficult viewing angles (e.g,. closeups).

Spatial Clustering Accuracy

Figure 3.12 summarizes the accuracy of FOCUS spatial clustering using SfM/gyroscope in a challenging scenario: clustering on 325 video streams simultaneously (using sequentially-collected 20-30 second video clips as a proxy for a large deployment), from the diverse set of locations shown in Figure 3.9. FOCUS placed each "stream" (video clip) in one of three spatial clusters, or marked

the stream as a processing failure (no SfM alignment, e.g., due to blur). For every assigned member of every spatial cluster, we compared the video's intended subject to the geographic centroid of each output cluster. If the assigned cluster was the closest (geographically) to the intended subject, it was considered a "true positive" result (placed in the most correct cluster). Otherwise, it was considered both a "false negative" for the intended subject and "false positive" for its actual placement. Note that we consider false positives/negatives less desirable than

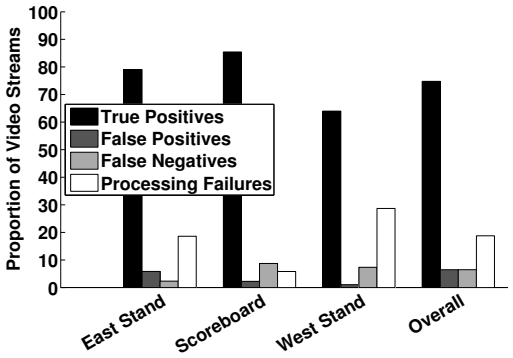


FIGURE 3.12: Stadium spatial clustering accuracy. For each True Positive, FOCUS correctly aligned a video clip and placed it into a cluster with others predominantly capturing the same intended subject.

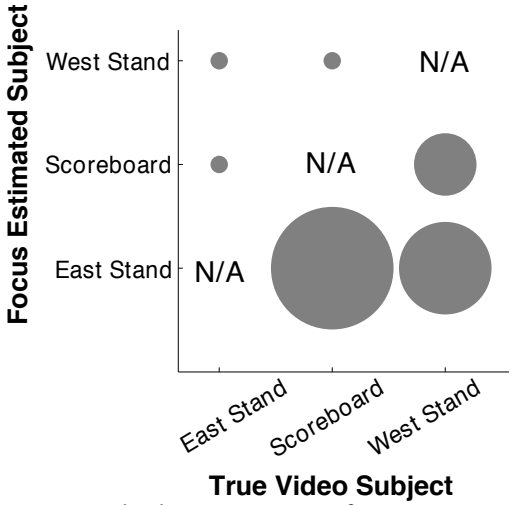


FIGURE 3.13: Stadium spatial clustering confusion matrix. For each intended subject along the X axis, the size of the circle reflects the proportion of video clips misplaced in the corresponding Y-axis cluster.

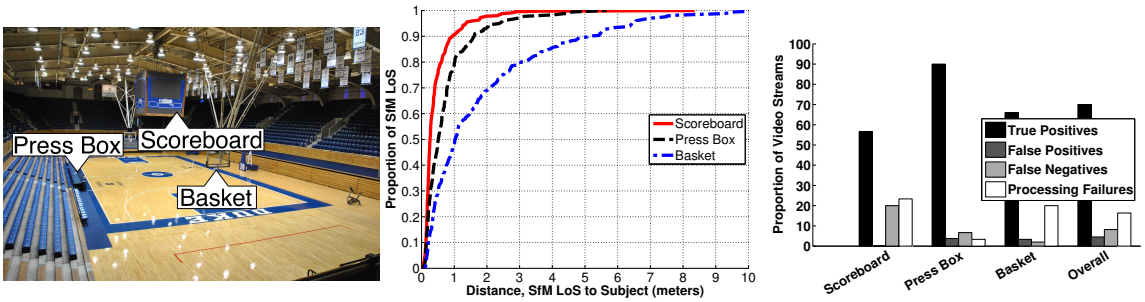


FIGURE 3.14: Indoor performance in a collegiate basketball arena: (a) example image from SfM model generation; (b) SfM/gyroscope line-of-sight estimation accuracy (110 locations); (c) spatial clustering accuracy.

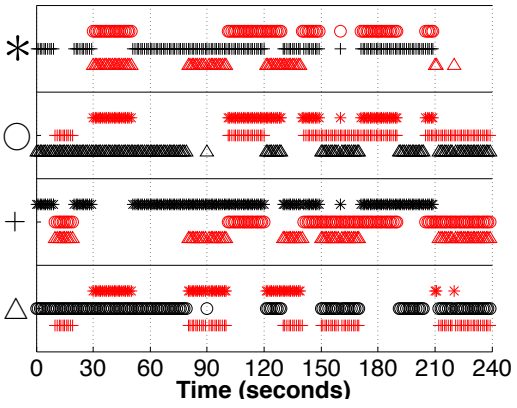
processing failures, as they represent a silent failure. Sanity checking (by GPS, compass, and accelerometer) was not applied. Further, though gyroscope-based hysteresis was used in this experiment, correcting several poor SfM alignments, the short video clip time interval limited opportunities for dead reckoning. Figure 3.13 presents a confusion matrix showing how false positive/negative cluster assignments were distributed. As expected, nearer subjects were more frequently confused. Overall results achieve 75% "true positive" success. 19% are processing failures, in which case FOCUS would invoke GPS/compass failover (we disabled FOCUS' failover for this experiment to restrict results to SfM/gyroscope exclusively).

Indoor, Low-light Performance

We tested in a 10K-seat basketball arena. Strong performance in this challenging indoor environment, where GPS/compass-based techniques do not apply and under relatively low light, demonstrates generic applicability across environments. Figure 3.14 shows: (a) an example photo from our SfM model; (b) line-of-sight estimation accuracy; and (c) spatial clustering accuracy.

Spatiotemporal Clustering

Figure 3.15 presents a dynamic scenario for FOCUS, tracking four video streams across a period of minutes. Each stream is tracking one of two "athletes" (fit volunteers) on the stadium running track. The athletes ran around the track in opposite directions (clockwise and counterclockwise), crossing paths multiple times. In the figure, each stream is represented by a marker ($\triangle$, $\circ$, $+$, $*$). Each grouping along the Y-axis denotes a particular video stream. Along the X-axis, we show markers at every second of a four-minute video. A dark (black) marker denotes a true positive result: a stream, for the corresponding one-second interval, was placed in the same spatial cluster as was the true videographer intent (same as that of symbol along Y-axis). A light (red) marker denotes a false positive result: a stream, for the corresponding one-second interval, was placed in the same spatial cluster as that of symbol along the Y-axis, but contradictory to to the true videographer intent. The figure illustrates that, typically, FOCUS is able to place a pair of videos with matching content into a matching cluster. It also captures all the data required



Video	$\triangle$	$+$	$\circ$	$*$
$\triangle$	-	88	151	63
$+$		-	115	149
$\circ$			-	86
$*$				-

FIGURE 3.15: (a) Y-axis shows four videos, $(\triangle)$ and $(\circ)$ capture athlete A, $(+)$ and $(*)$ capture B. Markers show streams placed into the same cluster, by time. Dark markers are matching subject. (b) Spatiotemporal matrix M: M_{ij} denotes the number of spatial clusters where stream i and j were mutually placed.

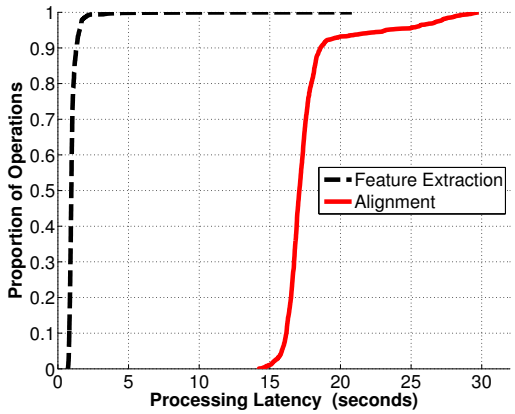


FIGURE 3.16: CDF of processing latency: video frame feature extraction, SfM frame-to-model alignment.

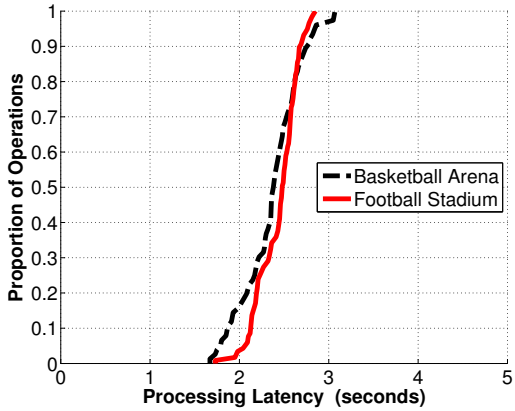


FIGURE 3.17: CDF of processing latency: frame-to-tree search.

to construct the spatiotemporal matrix of content similarity for these clips. The final, overall spatiotemporal clustering successfully outputs the correct stream-to-athlete matching ($\triangle$ and $\circ$ grouped for athlete A subject, $+$ and $*$ grouped for athlete B subject). This result was successful even though stream $\triangle$ suffered an extended 80-second period during which SfM alignment failed (due to deficiencies in our stadium model), leveraging our gyroscope-based dead-reckoning for seamless tracking.

Computer Vision Computational Latency

FOCUS' end-to-end performance is largely a function of vision processing, especially the dominating subtask: alignment of extracted features to a precomputed SfM model. Figure 3.16 plots single-threaded performance. In ongoing work, we are investigating ways to further expedite processing leveraging state-of-the-art techniques for fast alignment Zhao et al. (2005); Li et al. (2012); Crandall et al. (2011).

While our emphasis in this paper has been on accuracy and real-time operation, we explore alternative ways of video clustering without continuous alignment to 3D model, combating alignment latency to enable true real-world deployment. We use a vocabulary tree based method proposed in Nister and Stewenius (2006); Agarwal et al. (2009) to build an analytical cache of model-to-model image matches and respective camera poses (technique details in the references). Note that SfM model construction step outputs camera pose of model images. The idea is to use camera pose of best matched model image as a proxy to bypass a large portion of frame-to-model alignment latency (Figure 3.16). We achieve this because frame search in the vocabular tree is much more efficient than frame alignment in 3D model. We have used SIFT descriptors of the model images to construct the vocabualary tree. This optimization reduces median pose estimation time to X seconds from 17 seconds. The tradeoff of using this approach would be possible degradation of overall clustering accuracy, specially when the model images are not dense enough (due to erroneous camera pose approximation). Figure 3.17 shows a CDF of optimized camera pose estimation time.

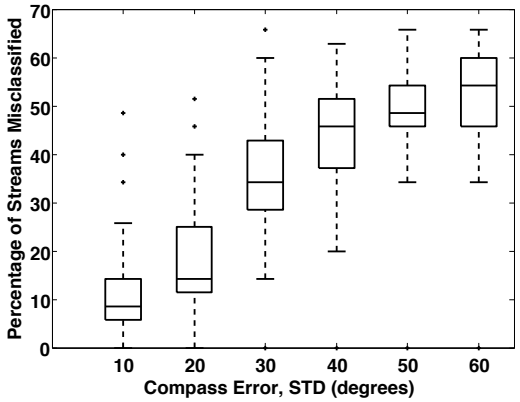


FIGURE 3.18: Box-and-whisker plots showing performance of GPS/compass-only failover by compass accuracy. X-axis shows standard deviation of introduced Gaussian compass errors (angles in degrees).

Tolerance of Sensing-only Failover to Compass Error

FOCUS' sensing-only failover (GPS/compass-based line-of-sight estimation) must be tolerant to substantial compass error. We tested failover by introducing random Gaussian errors to ground truth compass angles (GPS locations were used unmodified from experimental locations). Figure 3.18 shows diminishing clustering accuracy with extreme compass error, but solid performance with angular error having a standard deviation of 20 degrees, clustering at 85% accuracy.

User Study: Comparison to Human-created Clusters

We recruited 70 volunteers (demographics in Figure 3.19) to compare FOCUS' clusters to human groupings. Participants manually formed clusters from 10 randomly-selected videos (either from the football stadium or basketball arena datasets) by "dragging-and-dropping" them into bins. We adapt metrics from information retrieval to quantify the correctness of both human and FOCUS-created clusters: *precision*, *recall*, and *fallout*, using predefined labels of videographer intent.

For understanding, precision roughly captures how consistently a volunteer or FOCUS is able to create groupings where each member of the group is a video of

the same intended subject as all other members. Recall captures how completely a group includes all videos of the same intended subject. Fallout captures how often a video is placed in the same group as another video that does not capture the same intended subject (lower values are better). More precisely:

Let $V = \{v_1, v_2, \ldots, v_n\}$ be a set of videos under test. Let $C(v_i, v_j) = 1$ if v_i and v_j are placed in the same cluster, 0 otherwise. Let $G(v_i, v_j) = 1$ if v_i and v_j *should* be placed in the same cluster, according to ground truth, 0 otherwise.

$$\text{PRECISION} = \frac{|\{\forall v_i, v_j \in V \text{ s.t. } G(v_i, v_j) \wedge C(v_i, v_j)\}|}{|\{\forall v_i, v_j \in V \text{ s.t. } C(v_i, v_j)\}|}$$

$$\text{RECALL} = \frac{|\{\forall v_i, v_j \in V \text{ s.t. } G(v_i, v_j) \wedge C(v_i, v_j)\}|}{|\{\forall v_i, v_j \in V \text{ s.t. } G(v_i, v_j)\}|}$$

$$\text{FALLOUT} = \frac{|\{\forall v_i, v_j \in V \text{ s.t. } \neg G(v_i, v_j) \wedge C(v_i, v_j)\}|}{|\{\forall v_i, v_j \in V \text{ s.t. } \neg G(v_i, v_j)\}|}$$

As shown by Figure 3.20, FOCUS' precision, recall, and fallout percentages compare favorably to that of volunteers. Note that the performance of FOCUS occasionally (though not typically) exceeds that of our volunteers. We found that for humans, as well as for FOCUS, videographer intent is subjective and can be ambiguous to a third party. In some (randomized) cases, the volunteers were asked to cluster images which look, subjectively, quite similar but actually capture different physical locations. FOCUS was able to find small details to distinguish the locations that were not obvious to our volunteers. We observed a few trial participants and found them surprised when we debriefed them of errors in their clustering choices.

3.5 Limitations and Discussion

Difficult Scenarios for Structure from Motion

FOCUS is designed to work in locations where it is feasible to visually reconstruct a sound 3D model of the physical space. Reconstruction efficacy is subject to the properties of the feature detection algorithm, algorithms to identify edges or corners of a rigid structure. Stadiums and open areas between buildings, for example, are compliant environments as they contain large rigid structures, likely to produce many consistent keypoints across multiple images. Even when filled with spectators, the rigid structure of a stadium grandstand is still preserved (and thus are so many keypoints in the SfM model, see Figure 3.7). However, in other contexts, environmental dynamism may hide rigid structures, such as in a parade with large floats. Further, open fields, areas heavily occluded with trees, and tight indoor spaces will present a challenge to SfM, yielding poor results with FOCUS. We have not systematically explored the performance of FOCUS under heavy occlusions. In such cases, we assume SfM to fail and expect to failover to sensing-only techniques. Unsurprisingly, the efficacy of SfM is also dependent on lighting conditions. Dimly lit environments and outdoor environments at dawn or dusk yielding sun glare are especially challenging. For all such extreme cases, overall accuracy will roughly equate to sensing-only performance, as SfM alignment rarely results in silent failures.

Average Reported Age	45
% Male / % Female	79% / 21%
% own a smartphone	82%
% use smartphone for taking photos	88%
% use smartphone for taking video	60%
% share multimedia on social networks	41%

FIGURE 3.19: User study volunteer demographics.

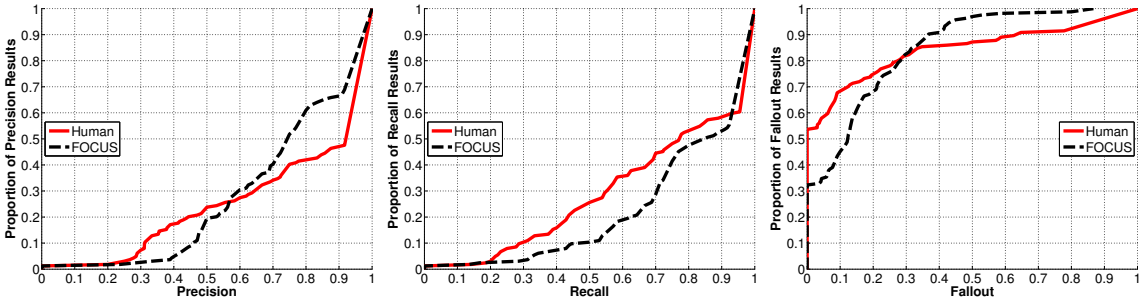


FIGURE 3.20: CDFs comparing FOCUS accuracy versus ground truth to human-generated clusters from the user study. FOCUS' clustering accuracy is comparable to that of volunteers; (a) Precision; (b) Recall; (c) Fallout.

Optimized Frame Selection for Alignment

Predictably, not all video frames are equally amenable to SfM alignment. Leveraging sensory inputs or lightweight computer vision, it is possible to identify which video frames are likely to be the "best" for further processing. For example, any of accelerometer, gyroscope, or visual inputs can be applied to identify shaken and blurred views. Similarly, simple visual metrics, such a color histogram or spatiograms Birchfield and Rangarajan (2005), would be useful in detecting a change in the presence of occlusions. Gyroscope-based dead reckoning is again useful here, making it easy to select, align, and leverage a compliant frame.

Reconstructing 3D Models "on the Fly"

In our evaluation, we have considered environments where it is feasible to precompute the SfM 3D model. In a stadium for professional or collegiate sports, for example, we imagine an employee or fan taking photographs of the arena, in advance, as input to our Hadoop-based model generation pipeline. However, with a goal to streamline practical deployment of the FOCUS system, we believe it is also possible to construct the model dynamically, using the content of the video streams themselves. Fundamentally, SfM is able to build a model by leveraging a diversity of perspective across multiple views of the same object: normally achieved by tak-

ing photos while moving through a space. A similar diversity is inherently available across the various feeds coming into system.[6] During the bootstrap period, during which sufficient images are gathered and the model is constructed, FOCUS would leverage its sensing-only failover technique, operating at a reduced accuracy until the model is complete. We tested model generation from streams as part of our indoor arena evaluation — from 60 video clips, we reconstructed an SfM model of comparable quality to that generated from still images.

Applications

We exclude a thorough treatment of how one might leverage (e.g., query) FOCUS' output video clusters. Section 3.4 evaluates in the context of sports, but many novel presentations are possible. Rather than domain-specific, our target is a generic construct to extract "app-enabling" metadata: logical pointers to videos capturing the same subject.

3.6 Related Work

Structure from Motion and Related

FOCUS applies structure from motion (SfM) to generate 3D models from multiple images, leveraging Bundler Snavely et al. (2006). Agarwal et al. (2009) uses Bundler to develop a large-scale 3D model of Rome from a corpus of photographs available on image sharing websites. Fitzgibbon et al. (1998); Zhu et al. (1999); Grzeszczuk et al. (2009) generate SfM models from continuous video sequences. Such models can be directly used by FOCUS. Zhao et al. (2005); Li et al. (2012); Crandall et al. (2011) propose various methods for fast alignment of an image into an existing SfM 3D model using motion patterns and inertial sensing. These techniques are complementary to FOCUS, improving the speed of line-of-sight es-

[6] Historical videos can also be used (e.g., from YouTube).

timation. Herranz et al. (2011) uses LED markers to estimate camera pose. Tuite et al. (2011) develops an iPhone photography game for city-scale SfM. Manweiler et al. (2012) applies SfM and sensing to estimate the location of a remote object.

Visual Similarity Analysis and Tracking

Video content matching and grouping is not unique to FOCUS, especially leveraging computer vision. Heath et al. (2010) constructs graphs to connect related imagery in a large collection. Kanade et al. (1998) considers the use of video sensors for cooperative tracking of surveilled subjects. Efros et al. (2003); Mohan (1998) find similar actions in sports games, identifying videos and the time of action. Sand and Teller (2004); Kim and Vasudev (2005); Kim et al. (2012) apply image processing such as background subtraction, compositing, and increasing dynamic range to identify similar content in video and images. Zhao and Nevatia (2004); Hampapur et al. (2005) visually track a shared subject from multiple cameras in complex situations. Shen et al. (2011) enables geo-referenced video search, similar in spirit to our GPS/Compass sensing-only failover approach.

Crowdsourcing and Localization

The power of crowdsourcing is widely confirmed. The ubiquity of mobile phones is helping to enable such systems on a large scale. Qin et al. (2011); Bao and Roy Choudhury (2010); Chon et al. (2012) propose various methods on collaborative and continuous sensing using smartphones, enabling multimedia tagging and organization by mining over contextual cues generated from inertial sensors and image processing. Reddy et al. (2007); Yan et al. (2010) use crowdsourcing for image search over large collections. Manweiler et al. (2012); Hightower and Borriello (2001); Hile et al. (2008) apply sensing to localization.

Sports and Industries

Startup Exa (2013) deploys/monitors laser cameras on commercial stadiums for tracking player movements. Schleicher et al. (2011) is a connects collocated TV viewers to enhance a social viewing experience. Chuang and Narasimhan (2010); Yin (2013) personalize sports multimedia feeds for different viewers. Str (2013b); Swi (2013); Vyc (2013); Str (2013a) enable users to create and share collaborative video experiences. Cro (2014) considers GPS and compass in image metadata to organize content from related angles.

3.7 Conclusion

The value of user-uploaded video is both immense and fragile. YouTube and other sites depend on a haphazard collection of manual tags and machine-mineable comments. Real-time content, prior to the availability of this crowdsourced context, is difficult to index. With the trends towards enhanced wireless data connectivity, improved smartphone battery life, and adoption of the cloud for low-cost, scalable computation, we envisage widespread distribution of user-uploaded real-time video streams from mobile phones. FOCUS is a system to analyze this live content, in realtime, finding groups of video streams with a shared subject. As an immediate high-value target, we have thoroughly evaluated FOCUS in a collegiate stadium environment. Once fully hardened, FOCUS can be deployed, for example, to enable a crowdsourced "instant replay," enabling the viewer to inspect multiple angles of a contentious play. More generally, we believe that FOCUS is broadly enabling for a variety of next-generation streaming multimedia applications. Nonetheless, FOCUS is a prototype, and a work-in-progress. Finally, we see further research opportunities in advancing a cloud platform for additional automated analyeses and distribution of live user video streams.

4
Practical Mobile Augmented Reality

The idea of augmented reality – the ability to look at a physical object through a camera and view annotations about the object – is certainly not new. Yet, this apparently feasible vision has not yet materialized into a precise, fast, and comprehensively usable system. This paper asks: *What does it take to enable augmented reality (AR) on smartphones today?* To build a ready-to-use mobile AR system, we adopt a top-down approach cutting across smartphone sensing, computer vision, cloud offloading, and linear optimization. Our core contribution is in a novel *location-free* geometric representation of the environment – from smartphone sensors – and using this geometry to prune down the visual search space. Metrics of success include both accuracy and latency of object identification, coupled with the ease of use and scalability in uncontrolled environments. Our converged system, *OverLay*, is currently deployed in the engineering building and open for use to regular public; ongoing work is focussed on campus-wide deployment to serve as a "historical tour guide" of UIUC. Performance results and user responses thus far have been promising, to say the least.

4.1 Introduction

The concept of mobile augmented reality is tantalizing. Researchers, designers, and the authors of science fiction have converged on the vision of using a hand-held camera as a sort of "magnifying glass" to browse the physical world. Digital annotations may seamlessly appear when the camera is pointed at an object. Looking at spaghetti in a grocery store may trigger recipes and product reviews; pointing the camera at a sculpture in a museum may pop up the artist's bio data; viewing a corridor in an airport can display restaurants located downstream in that path. Despite such exciting potential applications, a generic and usable solution remains elusive. The core challenge lies in carefully mitigating the tradeoff between accuracy and latency, i.e., the ability to precisely recognize an object and pop-up its annotation without perceivable time-lag. Given that hundreds of objects may be annotated in the same vicinity, slight inaccuracies or delays can degrade the quality of human experience. Moreover, authoring and retrieving annotations should be simple, and on the spot. If Alice is the first to annotate a painting in a museum, Bob should be able to view her annotation in the very next moment.

It is natural to wonder *why this is not a solved problem despite substantial research in augmented reality.* We began this project with the same question and realized that while several prototypes have been built in isolation – each making technical contributions in vision, sensing, and even information fusion – to the best of our knowledge, no system has made a holistic effort end to end Rolland et al. (2001); Arth and Schmalstieg (2011); Azuma et al. (2001); Wagner and Schmalstieg (2003); Girod and et al. (2011).

Pure sensor based approaches such as Wikitude Wik (2013) rely on the smartphone GPS to position the camera, the compass to infer direction, and the ac-

celerometer to identify the tilt. By computing the camera's 3D line of sight (LoS) from these data, Wikitude estimates the object that should be within the camera's viewfinder, and pops up the corresponding annotation. Unfortunately, sensory data has proven largely insufficient. Compass error measurable in tens of degrees severely derails the LoS accuracy; GPS errors of $5m$ or more exacerbates the condition. The application works when large objects are located near the camera, such as a user looking at the White House while standing at its fenced perimeter. For less favorable cases, especially when the annotation density is high, results are far from acceptable. Indoor environments lacking precise localization are of course out of scope.

In contrast to sensory approaches, computer vision enables a tradeoff of latency for better accuracy. Latency arises from matching the image in the camera's view against various images in the annotated database. When the match occurs, the results are precise – the annotation perfectly pops up on the screen atop the object. However, in uncontrolled environments, users view an object from different angles, under different lighting conditions, and from different distances – all adding to the complexity of the problem. If the database contains many images of the same object (e.g., Alice and many others have annotated the same painting), image matching exploits this diversity to gain robustness. However, it takes longer. With only one or few images in the database per object, the operation is faster, but at the cost of accuracy.

Recent years have witnessed hybrid approaches that fuse sensor data and computer vision Qin et al. (2011); Manweiler et al. (2012); Jain et al. (2013). By utilizing the location of the user and orientation of the phone, authors in Takacs and et al. (2008) prune the search space for image matching. This certainly offers improvement, however, the lack of precise location and large compass fluctuations

make them unreliable indoors. The latency is also in the order of several seconds, requiring users to take a picture and wait for the annotation. Real-time object browsing – like a magnifying lens – really warrants sub-second latency.

Based on extensive literature search, we conclude that despite meaningful technical advances, and many released apps and media articles, no single effort has fully delivered on the complete vision. There are holes in the end to end processing pipeline – achieving the strict latency and accuracy targets would warrant a holistic look at the system. The system also needs to relax any assumptions on location to be deployable universally in the near future. Finally, sheer engineering is necessary to cope with corner cases in uncontrolled environments, including hand vibrations in some users, some objects being identical, weak network connections, etc.

We aim to complete and test the vision – to deliver, from a top-down design, a viable mobile AR framework that enables a useable and rewarding experience. While many of the challenges that must be addressed are not fundamental, they are "hard" and previously undemonstrated. It needed many design iterations to get the pipeline sufficiently optimized so that a viable solution could even be within grasp. Initially, we expected our efforts to be primarily engineering. *However, we also found ripe opportunity for novel heuristics to improve accuracy, responsiveness, and ultimately, the human user experience.* Especially, we developed an optimization framework that underpins computer vision with a geometric representation of objects in the surroundings. We are able to learn that geometry from the natural human movements across space and time, bringing the system to a desirable level of robustness.

Finally, we evaluate our approach not only through micro benchmarks, but also live, through the natural usage of independent and unbiased volunteers. For the first time, we were able to observe real human behaviors when interacting with

a viable mobile AR solution.

In developing our solution, we learned the key contribution of this paper: *Mobile AR is best addressed not through sensing, computer vision, or any modality in isolation. Instead, each may be most advantageously combined within a stateful model, underpinned by the geometry and time of motion. With this spatiotemporal awareness, its possible to deliver content to the human user, at consistent precision and sub-second latency.*

For a sense of the accuracy and realtime responsiveness of the prototype, we invite the reader to watch the following video demonstration of our live system. http://synrg.csl.illinois.edu/projects/MobileAR

4.2 Measurements and Guidelines

OverLay aims to enable real time augmented reality on today's smartphone platforms. This section develops basic design guidelines through measurements and observations.

Location and Orientation Sensors Inadequate

We believe that sensor-only approaches are inadequate to realize OverLay in any generalized settings. Even in outdoor settings, where GPS locations are precise to around $5m$, sensor based approaches fall short. Figure 4.1 shows measurements performed with Wikitude Wik (2013), a popular app on the app store that uses the phone's location, compass, and accelerometer to display annotations. The graph plots Wikitude's error in line of sights (LoS) measured by computing the perpendicular distance from the true object to the LoS. The median separation is around $12m$, implying that an object has to be at least $12m$ wide for the annotation to still be correct. Clearly, this offers limited applicability.

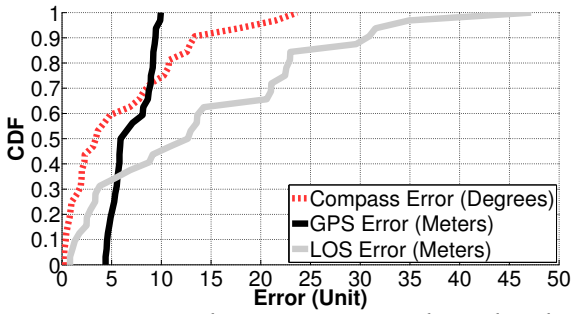


FIGURE 4.1: Difference in estimated LoS vs. ground truth when measured through the Wikitude app.

Indoor environments are far worse. Figure 4.2 measures compass errors due to ferromagnetic material in the ambience – as a user walks through a straight corridor, compass angle deviations are measured against ground truth. The median deviation is more than 15° with a heavy tail of up to 180°. Finally, indoor location is still not universally available, and some that are starting to roll out in a few places are limited to $5m$ accuracies, at best. Such accuracies easily derail a augmented reality approach. Thus, to make OverLay robust and immediately deployable in all environments, we must desensitize the solution to localization and orientation estimates.

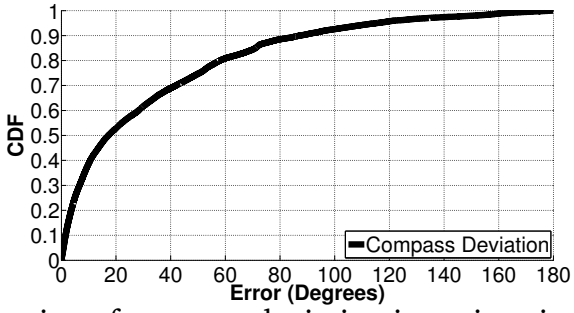


FIGURE 4.2: Distribution of compass deviation in various indoor environments.

Computer Vision and Cloud Offload Essential

The reality we intend to (digitally) augment is visual. In light of this, it seems most intuitive to take advantage of the camera, a sensor that "sees" the world similar to

humans[1]. Of course, computer vision algorithms (needed to "perceive" objects and display annotations) warrant non-trivial computational support. The way in which this computation is to be delivered presents a fundamental design decision.

Google's Project Tango pro (2014) convincingly presents the premise that tomorrow's hardware might have computational and (therefore) visual sensory powers far beyond anything on the marketplace today: a laptop-grade GPU able to perform heavyweight computer vision on-device, such as simultaneous localization and modeling (SLAM) through bundle adjustment. While the concept is fascinating, today's reality is quite different, and we target a nearer-term solution. It is also worth noting upfront that as we are without access to this hardware, we will not be able to compare our techniques against it.

Excluding such forward-looking hardware prototypes, we establish our second design guideline. Mobile devices do not have sufficient computational capability, even with embedded GPUs, to perform suitable computer vision for immersive (and thus compelling) Mobile AR. Contrastingly, today's cloud environments afford on-demand provisioning of vast computational resources, to include dedicated GPUs (especially, Amazon EC2 and IBM SoftLayer). As evidence of the performance contrast, Figure 4.3 shows CDFs of computational latency in extracting local image features from a 1080p video frame (using the state-of-the-art SURF heuristic) on a desktop CPU, GPU, and a current-generation mobile CPU. Mobile CPU performance is a factor of 1000 – *three orders of magnitude* – slower than desktop GPU. Any hope to attain real time AR today will probably be infeasible without cloud offloading.

[1] We assume current generation cameras in smartphones. While 3D cameras would open further opportunities, we wish to ensure the widest generalizability across devices today.

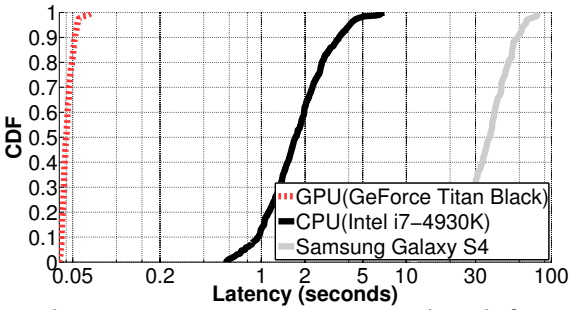


FIGURE 4.3: Compute latency: extracting SURF local features on GPU vs CPU (single thread) vs mobile CPU (single thread) for 1 HD-quality video frame.

Network Latency Dictates Lower Bound

The latency gain from cloud offloading is obviously offset by the network latency in moving images to the cloud. Figure 4.4 shows a CDF of latency for moving a single HD video frame from a mobile device, over Wi-Fi, to a local server for processing. Even with a low-latency connection, limited throughput makes realtime operation untenable. While reduced image resolution will reduce the data burden (and hasten transmission times), a commensurate reduction in computer vision efficacy is an undesirable penalty. In response, OverLay must be highly selective in *which* imagery it uploads. Selected portions of selected frames will need to be transmitted to reduce data transfer by orders of magnitude.

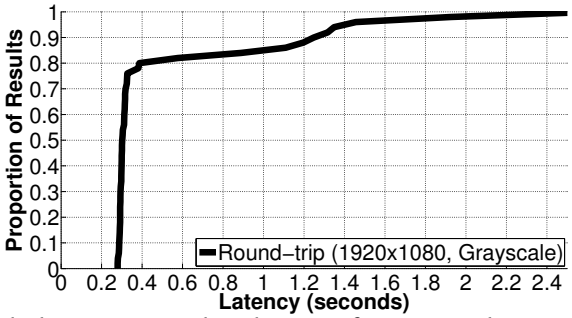


FIGURE 4.4: Network latency: upload time for a single HD-quality video frame (1920 x 1080).

Computer Vision: Still Too Slow

Cloud offloading only alleviates the resource crunch. Even when running on the cloud, the accurate computer vision techniques are still heavyweight (hence slow and perhaps unscalable to many users), while the lighter and quicker ones are less accurate. The choice of the suitable technique is a matter of engineering. Figures 4.5(a) and (b) plot image matching accuracy and computational latency of a range of well-established feature extraction and matching techniques. Unsurprisingly, there is a tradeoff. Our intuition is that accuracy must not be overly sacrificed – false positive and negative pop-ups will simply "break" the user's sense of immersion. The latency penalty has to be mitigated some other way, suggesting that computer vision is necessary but insufficient for delivering the end to end experience.

As a side note, its possible to bring down the latencies by parallelizing the image matching computation on many CPU/GPUs in the cloud. However, that would not scale to real-world many-user deployments. With an eye towards scalability, we perform our experiments on a single consumer-grade desktop GPU (resembling a cloudlet Satyanarayanan et al. (2009)), and support tens of users. The same techniques should hold with real-world user densities on a real-world cloud.

Opportunities for Geometric Optimization

The computational latency in Figure 4.5(b) is dominated by image matching, which is in turn a function of the number of candidates in the image database. Pruning the candidate set can aid in bringing down the latency to sub-second. To this end, it may be useful to develop a spatial understanding of the objects in the physical surrounding. If objects A, B, C, and D are known to be in spatial proximity, it may be possible to "prefetch" objects B, C, and D when the user is currently viewing

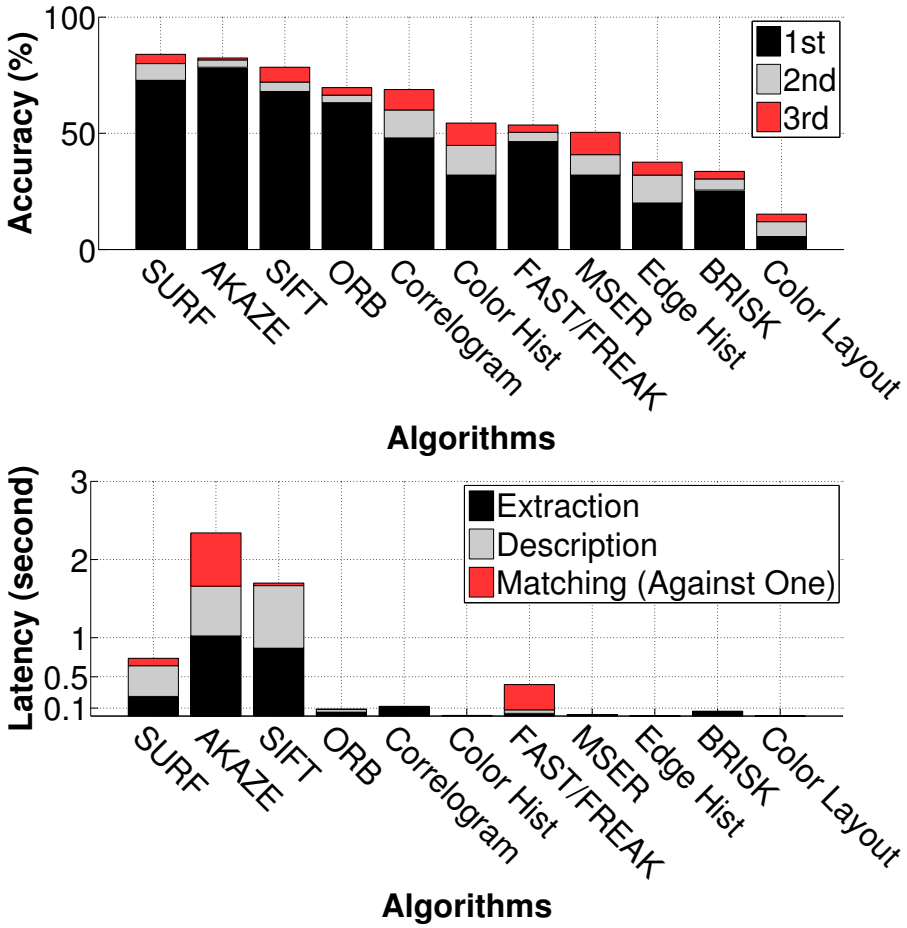


FIGURE 4.5: (a) Accuracy/latency of image matching based on local or global features. Accuracy for 1st, 2nd, or 3rd-best match plotted from an 100-image database. (b) Latency bars reflect stages of the matching process (all numbers for server CPU)

.

object A. If among these four, A and B are known to be in angular proximity, only B can be prefetched. In the absence of location information, spatial proximity may be statistically inferred from the temporal separation observed between various pairs of objects. Angular proximity can be deduced from gyroscope rotation as users scan across objects.

By synthesizing sensor data through a geometric optimization framework, it may be possible to infer a spatio-angular representation of objects in a non-absolute coordinate system. In other words, anchoring any given object at the origin of such

a coordinate system, it may be feasible to understand how other objects are relatively located. This allows for prediction and prefetching, offering opportunities to attain our real-time goals. Humans use such multi-sensor anchoring to reason about their movement through environments; OverLay makes an attempt to mimic some of these abilities.

4.3 System Overview

Desired User Experience

Our ideal end goal is as follows. As a user Alice points her mobile camera at an object in the physical world, an appropriate annotation pops up immediately atop the corresponding object. Alice moves her hand to browse other objects, and tags keep popping up with non-perceivable lag. If Alice wishes to annotate a particular object, she simply brings the object in the center of her viewfinder and enters the annotation (which could be text, multimedia, or even software code). Bob passing by that object soon after can look at the same object and see Alice's annotation pop-up on his screen. When multiple annotated objects are in the viewfinder, all the annotations are overlaid atop the respective object.

We have not been able to achieve this "seamless" goal – the current system requires a median of $480ms$ (includes $302ms$ network latency) to pop-up annotations, along with more than 95% image matching accuracy. Thus, when using OverLay, Alice must pause her camera at an object of interest, and the annotation pops up within a second (except in rare occasions). Figure 4.6 presents a screenshot of our prototype running on an Android smartphone. Even this relaxed goal was non-trivial.

FIGURE 4.6: Live object retrieval using OverLay.

Main Technical Modules

OverLay's end to end design firmed up after multiple iterations – in retrospect, we see three logical phases in the development process. (1) As an initial target, we focused on laying the groundwork with computer vision techniques, characterizing the best possible tradeoff between accuracy and latency. This phase included building the complete processing pipeline, evaluating a wide range of vision techniques, understanding their assumptions and possible modifications for our application, and finally reducing latency to the extent possible, through GPU parallelization, code optimizations, and sheer engineering. We do not make novel contributions here, nonetheless, the exercise was "hard" and time consuming to attain a desired degree of robustness and predictability. (2) With this framework in place, we heavily leveraged sensing (especially gyroscope) to reduce the amount of imagery that would be uploaded from device to server. This alleviated load on the GPU cutting the response time in half. (3) Finally, using sensor data from users, we developed an optimization framework to create the geometric representation of annotated

objects. This pruned down the search space, translating to $4x$ load reduction in image matching and ultimately translating to sub-second response latency.

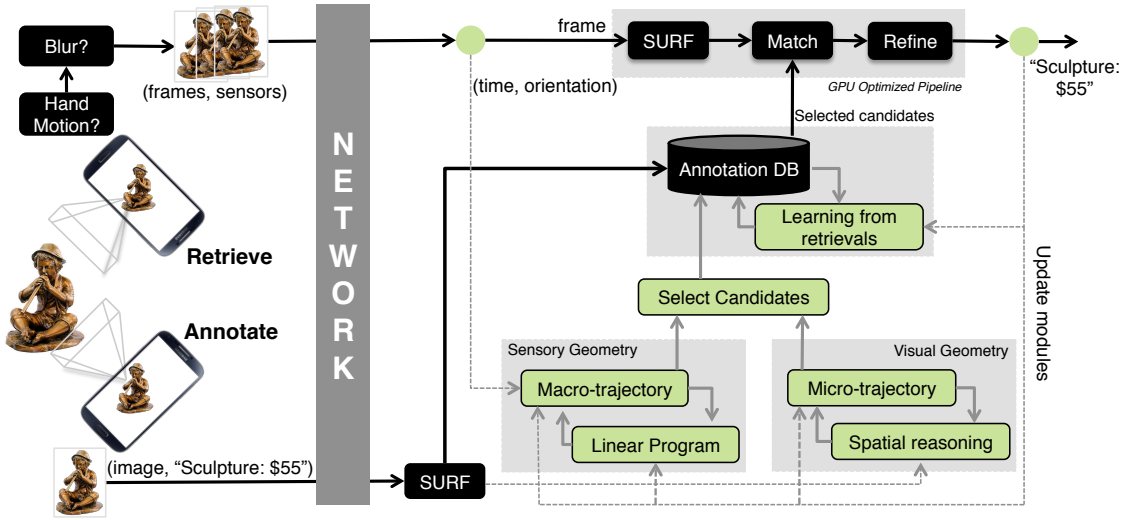


FIGURE 4.7: System Overview. Left: client-side. Right: offloaded compute, database on server (cloud).

Terminology

Tag or Annotation: Used interchangeably, they refer to content associated to physical objects. The user may tag or annotate an object, and these tags/annotations pop up when a OverLay–enabled camera is pointed at it. Tags and annotations can be text, multimedia, or a trigger for more actions (e.g., purchase of an item).

Constant Scan: The ability to continuously browse the physical environment with the camera, even when the user is moving. Tags expected to pop up when camera pauses on an object.

Search Space: Set of candidate objects against which a given object is being matched. Our search space is a spatial/angular region surrounding the most recently matched objects (better explained later).

Micro/Macro Trajectory: Observed or predicted human path through physical

space, exploited to refine the search space. A macro trajectory is substantial movement with translational and rotational components, such as walking and taking turns. A micro trajectory is minor hand movement, likely while standing in a fixed position – common when entering a room and rotating the camera to scan objects on the wall.

4.4 System Design

Figure 5.7 illustrates our simplified system architecture. In the basic execution flow, an annotated object (bottom left) is uploaded to the OverLay server, where SURF features are extracted and systematically indexed in an annotated (image) database. When objects are retrieved later (top left), the phone uses vision and sensing data to filter out unusable frames caused by hand vibration and blur. The uploaded frames are processed through a GPU optimized pipeline – the operation includes SURF feature extraction, matching with selected candidate sets, and refining the match. If a confident match is found, the server returns the corresponding "annotation" to the smartphone for on-screen display.

OverLay underpins this basic execution pipeline with optimizations that prune the search space during retrieval. To this end, 3 modules are invoked at different places in the overall system. (1) Sensor data received during the retrieval process (i.e., time and gyroscope orientation) is fed into a "Sensory Geometry" module responsible for inferring the the spatiotemporal relationships between objects. Two objects retrieved in a corridor can be "connected" in terms of their relative time and angular separation. As new retrievals arrive, and as images get correctly matched (top right), all these information are fed back into a *linear program* to ultimately converge on a geometric representation of annotated objects in the environment. (2) Such inter-object relationships are also extracted in the visual domain – if object

A and B are viewed in the same image, OverLay infers relationships such as "A is on the left of B". The "Visual Geometry" module uses data from annotated images to develop this understanding. (3) Finally, correctly matched images are fed back into the annotated database to improve/complete the visual models of an object – this allows for accurate recognition even when users are viewing objects from different locations, angles, lighting conditions. The details of these modules are described next.

4.4.1 Object Geometry (Sensory and Visual)

When Alice's camera points at an object, the video frames are uploaded to the OverLay server, which matches the frame against annotated images in its database. Matching against all database objects will be prohibitively slow. For maximized performance, both in terms of accuracy and speed, it is important to prune the matching search space to only the likely candidate objects. GPS location, erroneous to $30m$ or more in indoor environments, prunes the candidate set to the order of a building or a wing in a shopping mall. A real-world deployment could easily present more than 100 annotations in such areas – far more pruning is necessary to attain our accuracy and latency targets.

Towards this goal, we prune across *spatial* and *temporal* dimensions by learning a spatiotemporal relationship among annotated objects. At a high level, this results in a graph of objects where the shape of the graph essentially reflects how humans observe the (angular and temporal) separation between objects, as they walk through them. As a simplistic example, imagine we have three tagged objects A, B, and C in sequence on along a hallway (Figure 4.8). Unsurprisingly, if we observe A then B, it is quite likely we will soon observe C. Further imagine a fourth object, D, located on the left in a perpendicular hallway as shown in Figure 4.8. To make a left turn, we would expect a rotation $\approx 90^{\circ}$ counterclockwise. If instead, a

clockwise rotation is observed, the user has likely turned away, and the possibility of observing D is dramatically reduced. As a result, once OverLay recognizes an object X, it is able to infer the user's location in the object graph. Only objects near X are now candidates for the next recognition task, resulting in substantial pruning. Once the next object Y is recognized, OverLay knows that the user is now close to Y and selects objects near Y as the new candidate set. This operation carries on and the user's motion path is tracked through this spatiotemporal object graph. Of course, the first object recognition must rely on computer vision alone and crude GPS location.

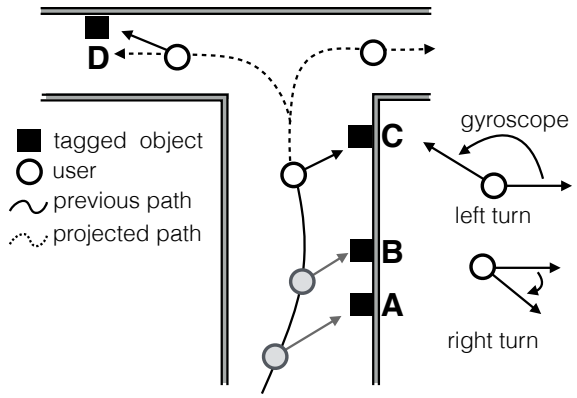


FIGURE 4.8: Example macro trajectory. Counterclockwise rotation observed after C is predictive of D.

We limit our understanding of this natural human trajectory to time and rotation only (note that no form of localization is necessary). The gyroscope is accurate enough to capture a high-fidelity awareness of user rotation. Further, we learn and leverage these trajectories at both *macro* (long-lived with substantial motion) and *micro* scales (small movements of the device in hand). The details follow.

Learning Macro Trajectories

As the server matches images to its object database, it tracks which objects the user has observed in the recent past (across minutes of walking motion). When

observing object i followed by object j, the server records an event k represented by a net rotation R_k (normalized between $0 \leqslant R < 2\pi$) and by measured time M_k (in arbitrary units). Of course, not all users will view i and j from the same location, and hence R_k will vary. Let E_k^R denote some unknown "error" by which R_k will deviate from the mean rotation from i to j (computed across all users). Similarly, let E_k^M denote some unknown "error" by which M_k will deviate from the mean time taken between the observations of i and j.

With K observation pairs, we may construct a pair of optimizations for N tags, independently solving for rotation and time. In addition to the known values R_k, M_k and unknown values E_k^R, E_k^M, we further introduce unknowns for each annotation i, P_i and $\forall j > i | T_{ij}$. Values of P_i may be understood as the "rotation" of annotation i relative to an arbitrary frame of reference (consistent across all annotations). Thus, $|P_i - P_j|$ may be understood as the the mean observed rotation between annotations i and j. T_{ij} may be understood as the mean time to observe annotation j after observing i.

Values of P_i and T_{ij} do not represent absolute properties of the annotated object. Instead, they reflect *where the object is typically observed*, relative to others – how much rotation is typical from annotation A to B to C, and how much time typically elapses in between – illustrated in Figure 4.9.

Tables 4.1 and 4.2 provide optimizations for rotation and time. To ensure computational tractability in dense areas with many annotations, it was important to formulate each as a linear program (LP). An earlier attempt using a mixed integer linear program (MILP) was unsolvable in days of compute time – these solve in seconds using CPLEX. Given low latency, the server re-solves both LPs for each new annotation, immediately as it is added to the database.

These optimizations explicitly track error attributed to each observation

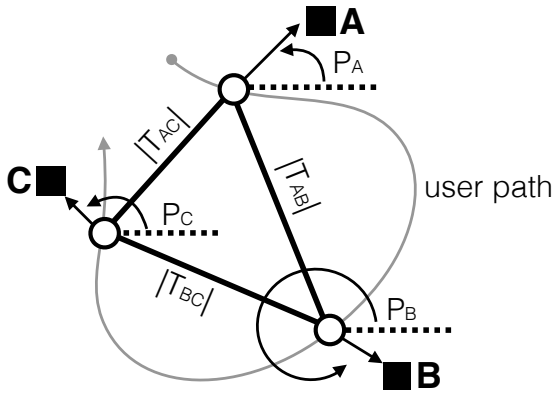


FIGURE 4.9: User macro trajectory as it relates to rotational and temporal optimizations. P values reflect rotation of the user as she views multiple tagged objects. T values reflect time elapsed walking.

recorded in the annotation database. The solutions to these error terms yield "confidence" estimates for the solved rotation and time estimates, per annotation. For annotation i, median error per observation is computed as:

$$E_*^R(i) := \forall j,k : \text{median}\{E_k^R | \exists R_k^{i\to j} \vee \exists R_k^{j\to i}\}$$
$$E_*^M(i) := \forall j,k : \text{median}\{E_k^M | \exists M_k^{i\to j} \vee \exists M_k^{j\to i}\}$$

OverLay combines rotational (P) and temporal (T) values with these error terms (E_*^R, E_*^M) to score which annotations should be prioritized as the best *candidate set* for a user.

Learning Micro Trajectories

Macro trajectories reflect the general, typical behavior of users as they move from annotation to annotation. This is quite useful for users who also move in this typical fashion. For others, the value degrades. For example, if most users walk down the center of a hallway, the angles at which various annotations are observed will reflect observation from this typical trajectory. If a user walks atypically against the wall, the angles she observes will be shifted, and the spatial optimization will

Table 4.1: LP, relative angular (rotational) position.

$$
\begin{aligned}
\textbf{Minimize} \quad & \sum_{\forall k \in 1 \ldots K} E_k^R \\
\textbf{Subject to} & \\
\forall R_k^{i \to j} | i < j \quad : \quad & P_i - P_j \leqslant R_k^{i \to j} + E_k^R \\
& P_i - P_j > R_k^{i \to j} - E_k^R \\
\forall R_k^{i \to j} | j < i \quad : \quad & P_i - P_j \leqslant -R_k^{j \to i} + E_k^R \\
& P_i - P_j > -R_k^{j \to i} - E_k^R \\
\forall i \in 1 \ldots N \quad : \quad & 0 \leqslant P_i < 2\pi(N-1) \\
\forall k \in 1 \ldots K \quad : \quad & 0 \leqslant E_k^R < 2\pi \\
\textbf{Solving for} & \\
\forall i \in 1 \ldots N \quad : \quad & P_i \\
\forall k \in 1 \ldots K \quad : \quad & E_k^R \\
\textbf{With parameters} & \\
\forall k \in 1 \ldots K | \exists i, j \quad : \quad & 0 \leqslant R_k^{i \to j} < 2\pi
\end{aligned}
$$

Name	Parameter / Variable Interpretation
$R_k^{i \to j}$	Observation k; rotation from anno. i to j
P_i	Rotational position of a anno. i
E_k^R	Rotational error for observation k

become less correct *for her*. Equivalently, a user walking exceptionally quickly will incur faster timings than the typical walking pace. To accommodate these atypical motion patterns, OverLay builds a simple model of *invariant spatial relationships* that hold true even for the atypical user. Simply, these relationships characterize if an annotation A is observed, some annotation B may be known to appear to the *left*, *right*, *above*, or *below* A. Especially as the user makes *micro* (rotational) motions (e.g., scanning around a room looking for annotations), these pairwise spatial relationships enable OverLay to shortlist those annotations the user is most likely to encounter next.

Smartphone cameras have a substantial field of view at their widest setting (zoom is not used in our prototype). In environments with dense annotations,

Table 4.2: LP, relative temporal spacing.

$$
\begin{aligned}
&\textbf{Minimize} && \sum_{\forall k \in 1 \ldots K} E_k^M \\
&\textbf{Subject to} && \\
&\forall M_k^{i \to j} | i \leqslant j && : \quad T_{ij} \leqslant M_k^{i \to j} + E_k^M \\
& && \quad\; T_{ij} > M_k^{i \to j} - E_k^M \\
&\forall M_k^{i \to j} | i > j && : \quad T_{ji} \leqslant -M_k^{j \to i} + E_k^M \\
& && \quad\; T_{ji} > -M_k^{j \to i} - E_k^M \\
&\forall i < j \in 1 \ldots N && : \quad 0 \leqslant T_{ij} \\
&\forall k \in 1 \ldots K && : \quad 0 \leqslant E_k < \max\{\forall k | M_k\} \\
&\textbf{Solving for} && \\
&\forall i < j \in 1 \ldots N && : \quad T_{ij} \\
&\forall k \in 1 \ldots K && : \quad E_k^M \\
&\textbf{With parameters} && \\
&\forall k \in 1 \ldots K | \exists i, j && : \quad 0 \leqslant M_k^{i \to j}
\end{aligned}
$$

Name	Parameter / Variable Interpretation
$M_k^{i \to j}$	Observation k; time from anno. i to j
T_{ij}	Time separation between anno. i and j
E_k^M	Temporal error for observation k

often multiple annotations will be in view simultaneously – detected when a camera frame matches to two or more annotations concurrently. Figure 4.11 provides pseudocode for inferring *micro trajectories* when two annotations appear simultaneously in view. When a visual match is made we can compute the 2D centroid of the match in the image. Centroids are compared to infer the general direction from one to the other (e.g., right). Figure 4.10(a) shows the centroid for B appearing right of A (the circle in the figure denoting the camera).

Under certain conditions, *these micro trajectory relationships will hold invariant, regardless of where in the room A and B are observed.* $A \leftrightarrow B$ is *spatially invariant* when A and B are coplanar with a room's wall (e.g., mounted). OverLay assumes $A \leftrightarrow B$ *could* be spatially invariant, if they have been found simultane-

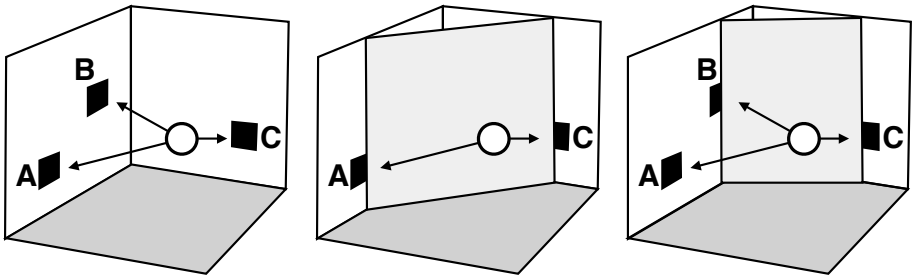


FIGURE 4.10: Micro trajectories for annotations A, B, C on walls of a room. (a) $A \leftrightarrow B$ invariant. (b) $A \leftrightarrow C$ *conditionally invariant* – effectively invariant, so long as the observer remains outside the shaded boundary. (c) $B \leftrightarrow C$ also conditionally invariant.

ously in view by one or more observers. Assume w.l.o.g. A is positioned left of B. At the time of retrieval, once A is observed, we may immediately shortlist B as a potential candidate. If the user rotates right (tracked by gyroscope), we increase our confidence: B should shortly appear in view.

Shortlisting is most productive when A, B are truly spatially invariant, the condition does not have to hold universally to be useful. Often, a *conditional invariance* occurs. That is, a region in which the observer may view A, B such that the condition holds. In Figure 4.10, $A \leftrightarrow B$ reflects true spatial invariance. $A \leftrightarrow C$ are invariant under the condition that the observer remains outside the illustrated shaded "barrier." Similarly, $B \leftrightarrow C$ are also conditionally invariant.

Prioritizing Candidate Tags

At the *macro* and *micro* scales, OverLay develops an understanding of which annotation a user is most likely to encounter next, given a current observation. OverLay prioritizes candidate annotations according to this understanding before matching against the user's live camera imagery. The prioritization is applied as follows.

Consider Figure 4.12. OverLay considers time then rotation. First, we construct a spatiotemporal radius M_U around the user, reflecting the set of annotations

Input : t_i = Descriptors of image i
t_j = Descriptors of image j
Output: Spatial relationships $\in$ {A, B, L, R, NONE}
$match_{ij} = descriptorMatch(t_i, t_j)$
if $|match_{ij}| \geqslant threshold$ **then**
$centroid_i = computeCentroid(t_i \in match_{ij})$
$centroid_j = computeCentroid(t_j \in match_{ij})$
$W_i = Width_i; H_i = Height_i;$
`/** inferRelation example **/`
if $centroid_i \in [(W_i/3, 2W_i/3), (0, H_i/3)]$ and $centroid_j \in [(W_j/3, 2W_j/3), (2H_j/3, H_j)]$ **then**
`relationship is above-below`
end if
`/** inferRelation example end **/`
return $inferRelation(centroid_i, centroid_j)$
end if
return $none$

FIGURE 4.11: Pseudocode: micro trajectory inference.

it would have been *possible* for the user to visit since the last match (illustrated as A) in M_U time. For each annotation i, we expand the radius to $M_U + E^M_*(i)$, to account for the computed temporal uncertainty. Annotation i is accepted to step two only if $T_{Ai} < M_U + E^M_*(i)$, otherwise it is immediately rejected.

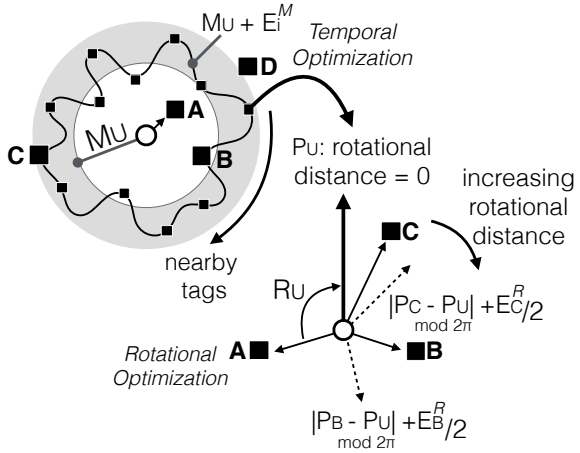


FIGURE 4.12: Prioritizing the annotation candidate set. We compare the user's last matched tag A with all other tags. Elapsed time T_U defines a radius (containing B), expanded by error term $E^M_*(i)$ for each annotation (containing C, D excluded.) B, C, and others in this expanded radius proceed to step 2 (rest rejected). They are prioritized by rotational deviance from the user, including error terms $E^R_*(i)$.

Next, we are able to impose an ordering based on rotation. Given a user's rotation R_U since tag A, we find orientation $P_U := (P_A + R_U) \bmod 2\pi$, the user's current orientation. We consider the rotational distance from any tag i as $|P_i - P_U| \bmod 2\pi + E^R_*(i)/2$. We include only half of the error term to reflect the 50-50 probability that the error should work in favor or against accepting tag i.

Since micro trajectory is rotation, we may factor it into the (macro) rotational distance scores. If a micro trajectory relationship $\{\text{ABOVE}, \text{BELOW}, \text{LEFT}, \text{RIGHT}\}$ is inferred with annotation i and the user's gyroscope reflects motion in the corresponding direction (i.e., clockwise for right, counterclockwise for left, up for above, and down for below) from i, we subtract a value $w\pi$ from the rotational score. w is a weight which lets us (de)emphasize micro trajectory context.

Regionalized Analysis

When the OverLay application is first launched, the user has no history of recent matches – OverLay has no notion yet of the user's macro/micro trajectory. To limit the search space, OverLay leverages the user's rough GPS location. The user's location and the location of every annotation in the database is discretized using the "Geohash" algorithm. OverLay attempts matches only against annotations in the user's square Geohash "cell" or in any of the eight neighboring cells.

4.4.2 Learning from Retrieval

Initially, an annotated object is only represented by a single image in the database. Intuitively, the crowdsourced effect of many users viewing the same annotated object (over time) might be useful: more views → more visual context → a refined database → higher precision for later users. OverLay exploits this potential for *learning from retrieval*.

Learning from retrieval is a periodic four step process, run as a background

server daemon. (1) All past retrievals and the original annotation images are used to reconstruct a 3D model of the object and surroundings (a process known as *structure from motion*, or specifically, *bundle adjustment*). The 3D model provides a geometric representation of both the object and each *camera pose*, the 3D position and orientation of the user during retrieval. (2) Due to false positive matches ($\approx$ 5% in our tests), 3D reconstruction Agarwal et al. (2011) can yield erroneous camera poses. Fortunately, these retrievals are outliers in the resultant 3D *point cloud*. We remove these false camera poses by performing outlier removal based on the point cloud centroid and variance of the model points. (3) All remaining retrievals contain the same object, but often contain redundant visual information. Including all retrieval imagery in the annotation database would create bloat and increase matching latency. Instead, OverLay identifies the most diverse retrievals (i.e., angles from which the object appears sufficiently different). For this, we use KMEANS clustering on 3D camera pose angles to cluster similar retrieval poses together. (4) We select one representative view from each of the retrieval clusters to construct a secondary matching database for each annotation, to be invoked only when primary matching with the original fails.

Since learning from retrieval creates a secondary database, it imposes zero latency for image frames which match the primary database. When the primary database is insufficient, the secondary database provides valuable matches. The overall experience is thus improved as users who would otherwise see no annotation content are presented with results from the secondary database, albeit at a slight delay.

4.4.3 Real-time Cloud Computer Vision

A primary requirement for Mobile AR is that annotations must quickly appear as the user scans around a room. This section describes necessary engineering refine-

ments to ensure that OverLay can surpass a tight usability bound.

Sub-selecting Frame Regions from Gyroscope

Often the user will hold the device reasonably steady while reading on-screen annotations. Frames generated during this time are near duplicate of the previous frames and need not be uploaded. We leverage accelerometer and gyroscope to cheaply identify such frames. No further processing steps are invoked; simply the previously-identified annotation remains on screen.

Even when the user makes nontrivial movements, there is still often partial overlap between the prior and current frames. This overlap can be inferred by (1) monitoring accelerometer and gyroscope to deduce that a user's movements are primarily rotational (and not from walking); and (2) inferring the angular difference from gyroscope to see which portions of the view are new. The gyroscope measures the relative three-axis angular movement of device from a previous position (a point in time), and infers the percentage of new content that should have appeared in the field of view. Specifically, for a lens projecting a rectilinear image, angle of view may be computed in radians as $\alpha = 2\arctan(d/2f)$ where d is the sensor size and f is the focal length. Assume w.l.o.g. that the gyroscope records a clockwise rotation $0 < \beta < \alpha$ radians. Only $100\beta/\alpha$% of the screen contains new content, and only that portion should be shipped to the server for analysis.

Managing Frame Motion Blur

User hand movements from "scanning" the physical environment result in blurred camera frames (motion blur and active adjustment from the autofocus system), causing computer vision underperformance. By applying a Canny edge detector for blur detection, we score frames to select only those most likely to contain usable imagery. As Canny is relatively robust to image resolution, we can apply it at low

computational latency to a down-sampled preview image – the first step of the algorithm is to apply Gaussian blur to remove image speckle, roughly equivalent to resolution downsampling. By throwing away useless frames without further processing, we achieve a higher effective frame rate of useful image data, both improving accuracy and latency.

Extracting, Matching, and Refining Image Features

Once a crisp frame (or frame region) has been uploaded to the server and a prioritized candidate set of annotated images has been identified, we must then match the frame imagery to these annotations. Each annotation is represented in the database as a collection of local image features, computed using the SURF Bay et al. (2006) feature extractor and descriptor. Each feature is represented as a keypoint, an x, y position in the image at which the feature has been detected and a descriptor vector of 64 floating point values. Two features are considered similar as an inverse function of the Euclidean distance (l^2-norm) between their respective descriptor vectors.

We find M_{AB}, the minimum distance bipartite matching between feature vectors for a pair of images I_A and I_B. Each descriptor value d_i^A of I_A may be compared with each descriptor value d_j^B of I_B. Let m_i^{AB} be the feature descriptor of I_B which has the lowest Euclidean distance from feature i in I_A. Formally, $m_i^{AB} := \forall d_j^B,\ \text{argmin}\ l^2(d_i^A, d_j^B)$. Let $M_{AB} := \{\forall i, m_i^{AB}\}$, the set of all such best matches from image I_A into image I_B.

Although optimal, M_{AB} likely has many false positive, poor image descriptor matches. We may now refine to $M_{AB}^* \subseteq M_{AB}$, the subset of high quality matches from I_A into image I_B. From common practice, we apply the following tests to construct M_{AB}^* from M_{AB}:

Distance Ratio Ratio of the Euclidean distance of the best match value (m_i^{AB}) to the distance of the second best match value. Typically, threshold $\geqslant 0.8$.

Cross Check Ensures the inverse match $I_B \rightarrow I_A$ would result in the same pairs; $m_i^{AB} = j \Leftrightarrow m_j^{BA} = i$. Thus, $M_{AB}^* = M_{BA}^*$.

Homography Runs RANSAC (random sample consensus) to find an approximate projective transformation from I_A to I_B. A match m_i^{AB} is rejected if it is an outlier to this transformation.

Early testing revealed an excess number of false positive or negative matches, depending on the thresholds given to homography. So, we add a final binary test for the entire image.

Slope Variance Imagine I_A and I_B composited into a single image, I_A on left and I_B on right. For each match $m_i^{AB} = j$, we draw a line from I_A to I_B between the corresponding feature keypoints. We compute the slope for each line. If the variance of slope is low, all remaining values m_i^{AB} are accepted. Otherwise we reject all matches and set $M_{AB}^* := \varnothing$.

OverLay performs all computer vision (SURF feature extraction, feature matching, and feature filtering) on GPU.

Asynchronous GPU Pipeline

To exact the value of mobile-to-cloud computer vision computational offloading, we must leverage the extreme parallelism available on a modern GPU. The cost of that GPU is only acceptable if we can amortize across multiple concurrent users. However, we found that available implementations of feature extraction and matching assume a single thread of execution (i.e., a single CPU core controlling the entire GPU for each synchronous operation).

When multiple CPU threads attempt concurrent access, conflicts arise. While it is possible to use a CPU semaphore to isolate access to the GPU, this approach

leads to intolerably poor GPU utilization. Instead, we heavily modify the CUDA (NVIDIA architecture GPU programming language) implementations of SURF, feature matching, and feature filtering to utilize CUDA Streams. CUDA Streams provides an asynchronous GPU processing pipeline based on a series of micro operations (e.g., copying data into or out of the GPU memory, memory allocation, and kernel execution).

4.5 Evaluation

From the onset, we were keen on a real-time evaluation of OverLay with a live fully-functional prototype in the hands of unbiased volunteers. However, we did not want to burden these volunteers with tasks of identifying correct versus incorrect annotations, or other feedback on ground-truth. This could slow them down or distract them from the seamless (environment scanning) experience that OverLay should offer. Thus, for purposes of evaluation, we added instrumentation to the server side of our system. All uploaded imagery and sensory data were recorded to disk. Later we were able to replay the image and sensing data, and exactly observe the ground truth – what the users saw in each video frame, their camera pose, their camera motion, the time between viewing, etc. We labeled $\approx 10K$ video frames based on whether the annotations displayed on screen were correct or not. Note that in some frames, one or many annotations should appear (if multiple objects are present). In others, "NO ANNOTATION" is expected. Importantly, this form of "offline" control allowed us to improve the system even after the live user studies, and test the results through accurate data playback and emulation.

4.5.1 Experiment Methodology

We annotated ≈ 100 objects on the 2^{nd} floor of the coordinated science lab (CSL) – the floor is around $50m$ long and $10m$ wide, as shown in Figure 4.13. Objects

are annotated just once, and selected arbitrarily – they include printers, exit signs, posters, research prototypes, water fountains, objects on display, etc. Now, with a completely functional prototype, we invited volunteers[2] to use it live. With limited guidance, volunteers were free to explore the building, scan whatever interested them, and they would see the matching annotations live on their phone screen – all without our interference. In this way, their walking and scanning with the device reflects their natural behavior. If they found the experience unenjoyable, they were free to give up at any point. Thankfully, our volunteers were universally curious and excited by the experience, each spending far longer exploring the space than we asked or anticipated (table below)

Number of tagged objects	100
Volunteers, Men / Woman / Total	6 / 3 / 9
Images captured, Min / Mean / Max	700 / 1188 / 1800
Volunteer time, Min / Mean / Max	12 / 18 / 27 min

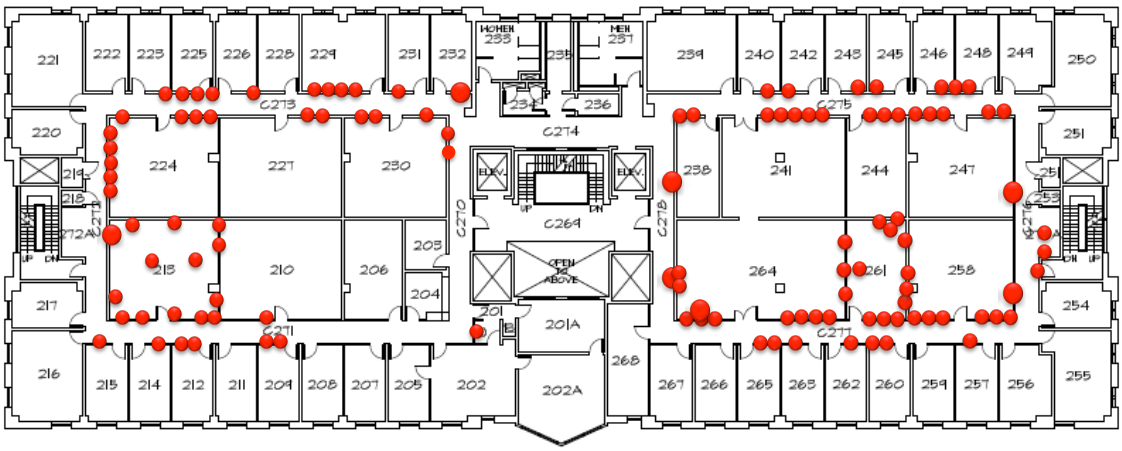


FIGURE 4.13: 2^{nd} floor CSL: dots denote annotations.

Our volunteers were only asked to use a single version of our system, CONSERVATIVE. The CONSERVATIVE version includes our complete solution except for optimizations to prune the search space of candidate annotations. From the CON-

[2] In compliance with our institutions' policies for IRB.

SERVATIVE test data, we emulate (offline) results for our ROTATION and TIME macro trajectory optimizations as well as with micro Trajectory identification. In all subsequent versions, the processed client imagery and sensory data are identical (e.g., gyroscope-based frame sub-selection is used in all). As CONSERVATIVE performs worse in terms of latency, the end user experience can be assumed to be the lower bound of what OverLay can achieve.

4.5.2 Metrics

Across all volunteers, we evaluate accuracy and latency *per frame* and *per annotation*. Each frame uploaded from the device app to the server is considered in isolation. It may be blurred or crisp; it may capture an annotated object or not; it may be captured intentionally (while the user is actively looking at an object) or incidentally (as the user moves about the space). Similarly, each annotation has unique performance characteristics – it may be easy or hard to identify; the search space optimizations may characterize its location well or poorly. Therefore, each graph presents a CDF (empirical distribution) of accuracy or latency with 100 points, one for each annotated object, generated from $\approx 10K$ samples, one for each frame.

Let Y be the set of all processed frames. Let $V \subset Y$ be the set of frames containing annotated object k. Let $P \subset Y$ be the set of frames identified by our system as having k. Then, $V \backslash P$ denotes our system's false negative predictions, $P \backslash V$ denotes false positive predictions, and $Y \backslash V$ denotes the set of frames which do not contain object k. *We evaluate OverLay's prediction efficacy by the following standard metrics of information retrieval:*

$\mathsf{Precision}_k = |V \cap P|/|P|$

i.e., among all frames that OverLay believes has object k, what fraction truly has k.

Recall$_k$ $|V \cap P|/|V|$

i.e., among all frames that truly have object k, what fraction was identified by OverLay.

Fallout$_k$ $|(Y \backslash V) \cap P|/|Y \backslash V|$

i.e., among all frames that truly do not have object k, what fraction was believed to have k.

4.5.3 Comparison with Approximate Matching

Our image-to-database matching process employs brute force (on GPU) to find the optimal match. As an alternative, a number of heuristics exist for approximate matching Muja and Lowe (2009). By design, these approximate schemes are computationally inexpensive (i.e., runs faster), at the cost of greater error. To understand this tradeoff, we compare all proposed schemes against APPROXIMATE. Briefly, APPROXIMATE matching computes a set of FLANN indices and loads them into CPU memory – a one-time operation. Later, each incoming image descriptors are matched in real time against these indices (using schemes like KDTREE, KMEANS). The best matched descriptor is up-voted, ultimately resulting in a best match image. To be favorable to APPROXIMATE, we pick the $top - 5$ images, and perform a brute-force search on it – as long as the correct candidate is in this set of 5, brute force should output the correct match. This reflects an optimistic view of APPROXIMATE, and we will plot its results alongside ours.

4.5.4 Overall Results: Accuracy and Latency

Figure 4.14 presents CDFs of overall accuracy as (a) precision, (b) recall, and (c) fallout. Graphs compare the accuracy of our CONSERVATIVE scheme against optimizations ROTATION, TIME, and FULL, as well as APPROXIMATE. Figure 4.15 presents a CDF of end-to-end latency (system responsiveness). Each curve reflects

100% of volunteer data, every frame captured by the system. None of the volunteers had any prior exposure to the system nor were given any special knowledge of its technical approach.

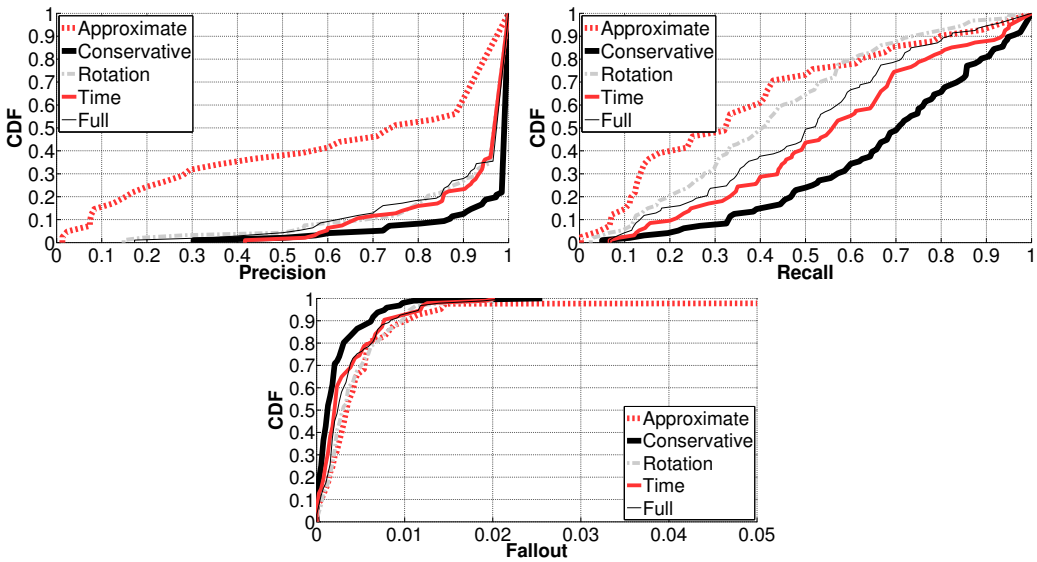


FIGURE 4.14: Main accuracy results: (a) Precision; (b) Recall; and (c) Fallout. Graphs compare accuracy of our CONSERVATIVE system against enhancements through optimizations, ROTATION, TIME, and FULL.

We would expect, and confirm here, that the strongest accuracy performance is achieved by the CONSERVATIVE version of our system. Precision is consistently high indicating that when an annotation is shown, it is almost always accurate. ROTATION, TIME, and FULL reflect optimizations aimed to improve system responsiveness (not accuracy). ROTATION is found to be comparatively unreliable in isolation. TIME is highly accurate, and to our surprise, often outperforms even the FULL version of the system (which jointly optimizes across time and rotation, along with micro trajectory context). Upon deeper inspection of the data (the $\approx 10K$ camera frames recording by our volunteers), we find our volunteers were more consistent in time taken to move between objects than they were for angular separation. All these schemes consistently outperform APPROXIMATE, implying how latency reduc-

tion indeed incurs a strong penalty in vision-only schemes.

Of course, recall is not impressive, implying that some annotations are not captured/displayed in time. We partly expected this since – if a user's hand shakes, if lighting is poor, if the person views the object from a corner, they will all affect recall. This clearly motivates the need to learn from new retrievals made by users over time – Figure 4.17 will later demonstrate the efficacy of retrieval.

Latency results also align well with intuition (Figure 4.15). The sensor optimizations (ROTATION, TIME, and FULL) heavily outperform CONSERVATIVE. APPROXIMATE performs slightly worse than sensor optimizations. Since GPU implementation of APPROXIMATE is unavailable, we estimate latency numbers using 3-dimension implementation of KDTREE and scale accordingly for 64-dimension SURF descriptors. For each of the optimization-based approaches (ROTATION, TIME, and FULL), our budgeted candidate set size is 10 annotations, one tenth of the total 100 annotated objects. As shown in Figure 4.16, matching is the primary latency component in the CONSERVATIVE system version, reduced here by 90%. Overall speedup from the added optimizations is around $4x$.

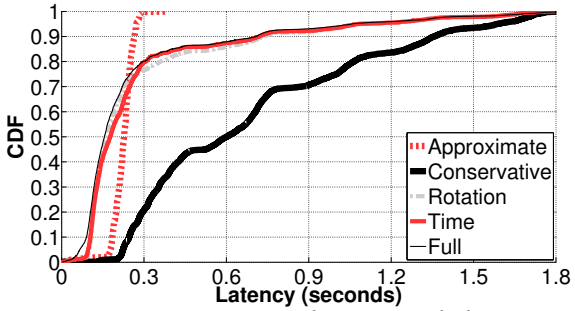


FIGURE 4.15: Latency: Optimizations reduce candidate set from 100 to 10, decreasing median end-to-end latency by more than a factor of 4, to 180ms.

4.5.5 Learning from Retrievals

OverLay only requires users to annotate objects once, with a single image. While

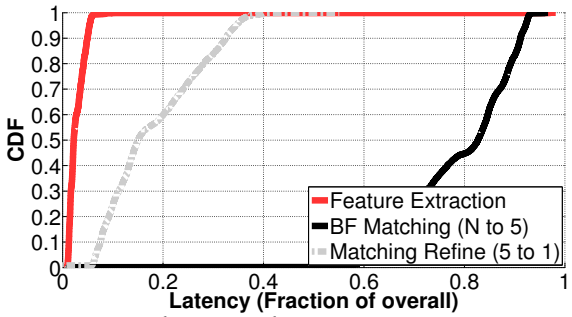


FIGURE 4.16: Computer vision latency by component, CONSERVATIVE system version. Optimized versions eliminate 90% of the primary component, matching.

this places minimal burden on an end user, diverse perspectives of the object cannot be captured. Thus, when another user retrieves from a sufficiently different location/angle/lighting condition, the object is sometimes not recognized, diminishing recall (as noted earlier). The issues can get pronounced over time, with minor changes in object's appearance and background. Robust multi-view feature matching is still an unsolved problem in computer vision Schaffalitzky and Zisserman (2002), so its mandatory that features of a stored object are updated periodically. Figure 4.17 shows the efficacy of such learning from retrievals. Consistent improvements are evident in both precision and recall. Some objects were still unrecognized – delving into the data, we recognized cases of poor network connection (even transient disconnection), heavily delaying the frames from reaching the server. Such cases are entirely out of our control for OverLay.

4.5.6 Micro Benchmarks

We evaluate some details of OverLay's performance through micro-benchmarks. Figure 4.18 shows decrease in responsiveness with multiple simultaneous clients on the same server, under (a) the CONSERVATIVE (most GPU intensive) version of the system and (b) the optimized FULL version. The CONSERVATIVE version cannot support more than three concurrent clients; FULL supports six at equal user latency.

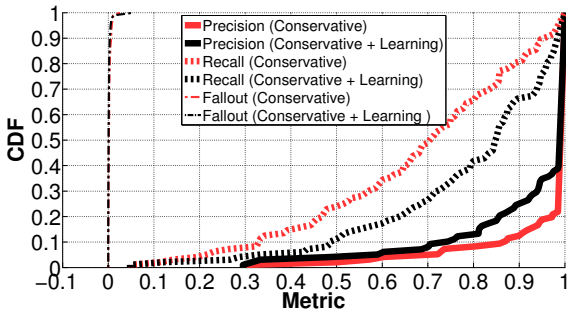


FIGURE 4.17: Enhanced accuracy results based on learning from past three retrievals chosen based on diversity in the camera pose in the 3D reconstruction when compared against CONSERVATIVE system.

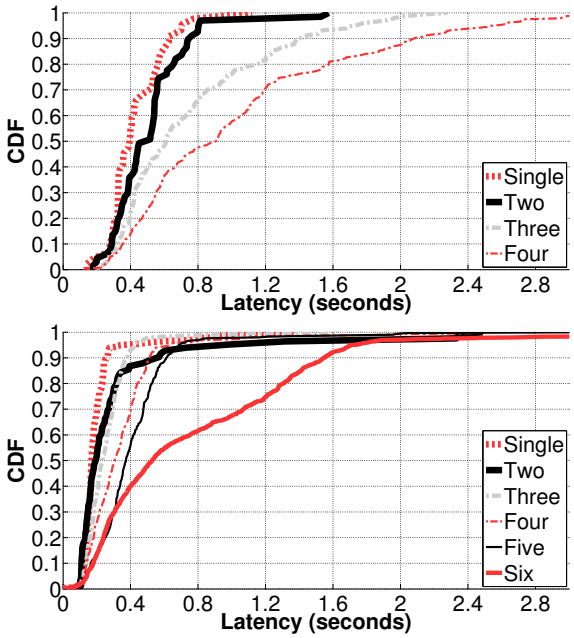


FIGURE 4.18: Responsiveness, simultaneous clients, (a) CONSERVATIVE and (b) FULL optimized version.

Figure 4.19 shows energy consumed (Samsung Galaxy S4 Android). System power measured using Monsoon Solutions hardware monitor through the battery contacts. Figure 4.20 shows app UI performance in display frames per second (FPS). Running the complete system, the app maintains 8 FPS (median) on a Samsung Galaxy S4.

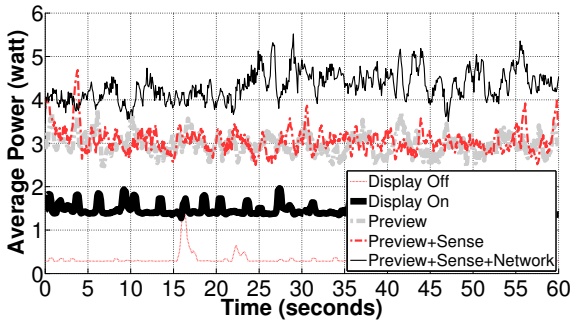


FIGURE 4.19: Energy consumption. DISPLAY + PREVIEW (camera enabled) + SENSE (GPS, gyroscope, accelerometer, and compass) + NETWORK TRANSFER reflects the complete system (average ≈ 4.5 watts).

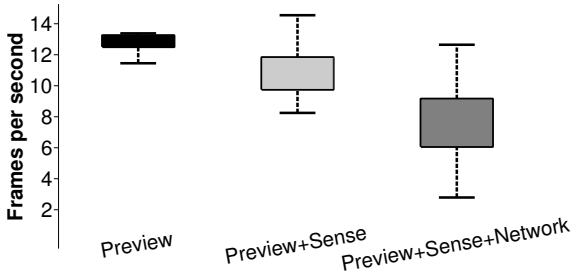


FIGURE 4.20: App UI responsiveness, frames/second.

Figure 4.21 shows the percentage of blurred frames rejected without upload to the server, per volunteer. Overall, 46% of frames are rejected without upload, saving substantial upload bandwidth and server processing.

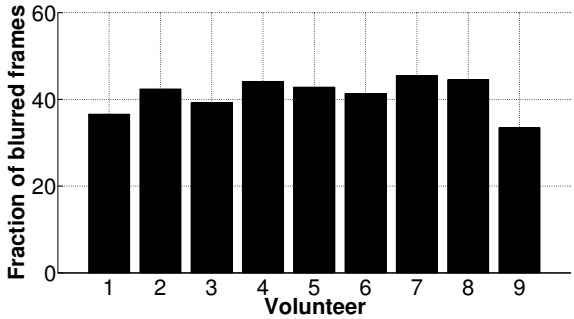


FIGURE 4.21: Percentage of frames rejected due to blur.

4.6 Limitations and Discussion

We consider several limitations of the OverLay prototype as implemented today, and avenues for future enhancement.

Exact placement of the annotations on screen

OverLay displays object annotations at a fixed screen location. An enhanced user interface might display the annotation directly atop or adjacent to the corresponding object. Our rationale for this this simplification is twofold: (1) when the annotation is authored, the user does not explicitly mark which part of the image corresponds to the object of interest – *multiple objects* might be in view, and (2) during retrieval, the annotation would need to remain aligned to the object, even as the user makes fine hand movements.

The "multiple objects" issue may be addressed in varied ways: (A) requiring the user to draw a rectangle or denote the intended object; (B) marking *candidate objects* on screen and allowing the user to make a simple multiple-choice selection; or (C) assuming the center of the user's screen most likely corresponds to the intended object, and tolerating errors when this assumption does not hold. The "annotation alignment" issue can be roughly accommodated by tracking fine hand movements and shifting the annotations on screen in the opposite direction. Optical flow and gyroscope based techniques are both possible for such motion compensation; recent work has even combined the two for enhanced precision and speed Hwangbo et al. (2009).

Searching for annotations

OverLay is most compelling when annotations are dense – when many objects around are annotated. Value decreases with sparsity: the user must "hunt" more

between annotated objects. One possibility is to use rough location estimates passively to alert the user when annotations are nearby, for example, through a signature vibration. Once the user opens the app, on-screen arrows might indicate which direction the user should rotate until a nearby annotation appears in view. OverLay is being readied for the *Illinois Distributed Museum* project on the UIUC campus dis (2014), and such "on-screen arrows" will be added in the next release.

Handling appearance changes

OverLay records image data to identify the physical location of an annotation. However, OverLay must cope with environmental dynamism, including changes to the object's appearance. One of our users annotated his office desk, and its appearance changed every day. While we cannot expect to address extreme changes, micro-movement of objects or minor appearance changes could be accommodated through continuous database enhancement, discarding older visual data to be replaced with that from later retrievals. OverLay does not apply to objects such as digital displays — whose appearance changes continuously. Similarly, OverLay does not immediately apply if a retrieval is made from a significantly different angle to that of annotation. However, some of these limitations are addressable using techniques discussed in 4.5.5.

Distinguishing objects in close proximity

OverLay cannot distinguish visually-similar objects in close proximity. This might be problematic in environments with repetitive design features, such as different name tags on adjacent doors in a corridor. False positives will undoubtably result, although our geometric linear optimizations will partly help (Section 4.4.1).

4.7 Related work

Mobile augmented reality has been studied for more than a decade. Pence (2010); Henrysson and Ollila (2004) outline a broad vision and discuss various possible applications. Wagner and Schmalstieg (2009) considers the challenges, strategies, and limitations one needs to overcome in building mobile AR systems. LiKamWa et al. (2013) and Cuervo et al. (2010) address two important aspects of Mobile AR – low-power continuous vision and code offloading to cloud. Several works establish sensing-based primitives for Mobile AR: GPS-compass triangulation Feiner et al. (1997); Piekarski and Thomas (2002); Gammeter et al. (2010); camera-pose estimation using Kinect depth sensors Izadi et al. (2011); and gyroscope based camera-pose tracking You et al. (1999). Some have leveraged physical markers deployed in the environment such as: QR codes Rekimoto and Ayatsuka (2000), color markers Mistry and Maes (2009), and RFID tags Tenmoku et al. (2003). Others have applied 3D camera pose estimation, finding a correspondence between 2D image features and a 3D world coordinate system Wagner et al. (2008); Klein and Murray (2009); Zhou et al. (2008). Takacs and et al. (2008) is the closest research prototype to our work, using SURF feature extraction and location based pruning to enable mobile AR in outdoor environments. Our primary advantage beyond this effort is in applicability of our search space optimizations to indoor environments, *with zero reliance on location information*.

While prior art has often considered computer vision and mobile sensing in isolation, there has been some recent success in hybridization. Smartphone inertial sensors can be used to imitate or enhance the computation of various vision algorithms (e.g., bundle adjustment, optical flow) Jain et al. (2013); Manweiler et al. (2012). Kopf et al. (2014); ins (2014) fuse vision and sensing to create hyperlapse video summaries (a form of offline augmented reality). Kopf et al. (2014) takes

a vision-oriented approach, strongly relaxing any latency requirement. ins (2014) priorities lightweight computation, emphasizing gyroscope. Yan et al. (2008) proposes energy efficient design of a distributed image search engine. Most of these research are bottom-up and do not deliver an end-to-end application – the challenges of very little training, physical indoor space, and human authoring of content, combined with opportunities of geometric optimizations, makes this paper's design constraints unique.

Object recognition (OR) for mobile devices is a overlapping research effort to Mobile AR, such as Amazon's Fire phone or Google Goggles Chen et al. (2009); goo (2014); Amazon (2014). Visual MIMO Ashok et al. (2010), communication between LED screens and a camera, can be used for Mobile AR but the approach is not generalizable. In contrast to mobile AR, OR systems: (1) typically operate on a trained model of multiple images of the same object Girod and et al. (2011); Föckler et al. (2005); Amazon (2014); goo (2014); (2) are typically invariant to the user's context; and (3) typically do not allow dynamic insertions to the content database as retraining costs are often high. Qualcomm Vuforia vuf (2014) is a commercial Mobile AR SDK for object recognition and 3D object tracking. Videoguide vid (2014) is a Vuforia app used to animate architecture work in Barcelona museum. Contrary to Vuforia which requires deployment in advance, OverLay is an anywhere, anytime system for everyone.

4.8 Conclusion

Mobile AR has been an exciting, yet unrealized, vision. This paper attempts to complete the vision within the constraints of today's smartphones. We combine cloud-offloaded computer vision with an optimization framework on space and time – capturing typical human behavior and pruning the computer vision search

space. In conjunction with a suite of engineered systems optimizations, these techniques enable a practical system for mobile AR. We demonstrate a response time well within tolerable bounds yet while preserving strong accuracy. Our approach provides a ready-to-use platform for enabling a broad spectrum of compelling mobile AR applications, and is currently deployed in our building and being readied for a campus wide roll-out.

5

Concise Hi-fidelity Environmental Fingerprinting

Environmental fingerprinting has been proposed as a key enabler to immersive, highly contextualized mobile computing applications, especially augmented reality. While fingerprints can be constructed in many domains (e.g., wireless RF, magnetic field, and human motion patterns), visual fingerprinting is especially appealing due to the inherent heterogeneity in many indoor spaces and high fidelity of modern smartphone cameras. This fidelity, however, is also its Achilles' heel – matching a unique visual signature against a database of millions requires either impractical computation for a mobile device, or to upload large quantities of visual data for cloud offload. Further, most visual "features" tend to be low entropy – e.g., homogeneous repetitions of floor and ceiling tiles. Our system, *Concise High-fidelity Environmental Fingerprinting* (CHEF), proposes a means to offload only the *most distinctive* visual data, that is, only those visual signatures which stand a good chance to yield a unique match. CHEF enables cloud-offloaded visual fingerprinting with efficacy comparable to using whole images, but with an order of magnitude reduction in network transfer.

5.1 Introduction

Indoor localization has remained something of a grand challenge for mobile computing. While many schemes have been proposed, substantial drawbacks remain: in deployment practicality, in efficacy, or in robustness. Without hardware deployments of RF "beacons" LaMarca et al. (2005), specialized ligtbhulbs byt (2015), or advanced Wi-Fi access points Sen et al. (2013), instantaneous or continuous indoor localization remains challenging. Moreover, with sustained excitement for indoor augmented reality applications Jain et al. (2015a), we need mechanisms to track where a user holds their phone to virtually engage with the nearby physical world.

Indoor spaces naturally lend themselves to high visual diversity. Patterns of color in paintings and photographs on walls, stylized light fixtures, corners on furniture, and even minuscule imperfections in drywall and baseboards collectively create a unique visual "signature" or "fingerprint" of a particular space. Due to the combinatorics, it is highly unlikely that any two places would have a very similar pattern of visual features. Our goal is to create a lightweight mechanism to enable "visual fingerprinting" at scale. As shown in Figure 5.1, we want to build a lookup service by which it is possible to snap a photo and instantly map it to a point in space, a unique point, by capturing only the essence of the scene – the visual fingerprint. Crucially, the mechanism must be low latency – quickly able to identify a smartphone user's location (and more specifically, the user's rough *camera pose* while the data is still relevant. Further, it must not be overly burdensome in computation, bandwidth, energy, or storage.

Visual fingerprinting "at scale" implies huge volumes of image-derived data – more than could be stored, indexed, or searched, on an iPhone or Android. Especially, high dimensional search is computationally challenging. Some amount

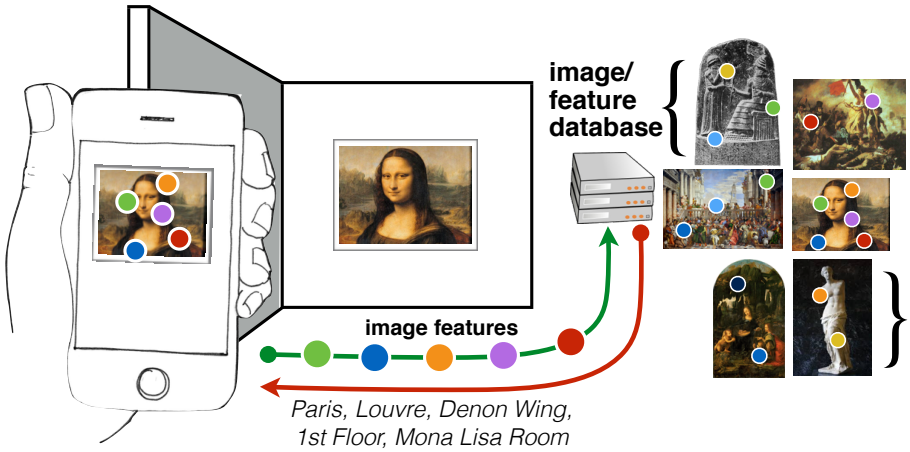


FIGURE 5.1: User views scene of interest on smartphone display, image or image features shipped to cloud processing server, cloud returns with precise localization result.

of server or cloud offloading appears necessary. Typically, one would ship either whole images or image-derived *keypoints* to the cloud for further processing against a massive visual database – the well-established problem of *image-based content retrieval*.

Practical Challenges of Cloud Offload

While existing approaches for image-based content retrieval might be sufficient for identifying a user's location from a photograph, they would quickly become impractical for continuous, real-time use. A stream of images or video implies a high sustained bandwidth, especially if we take full advantage of modern high-resolution smartphone cameras. As shown in Figure 5.2, at these resolutions, even 10 frames per second (FPS) requires 2 Mbps using state-of-the-art H264 compression. Worse, such levels of compression result in an almost unusable reduction in the quantity of extractable keypoints (Figure 5.3). Thus, we would really need lossless compressed frames, such as PNG – at much higher bitrates. On a cellular data connection, even LTE, such a stream is at worst infeasible, or at best, wasteful of the user's data quota and battery life, not to mention finite cellular bandwidth.

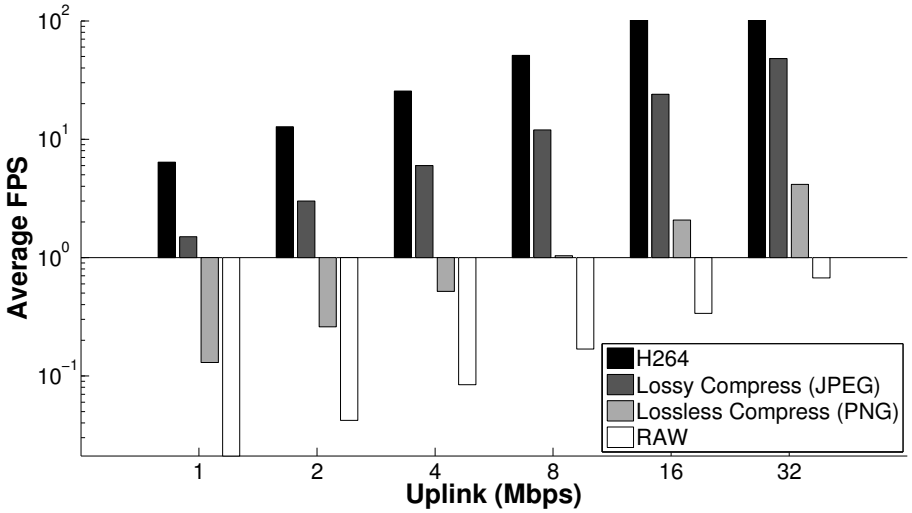


FIGURE 5.2: Uplink bandwidth versus sustainable frames per second (FPS), by encoding. Note: log-log scale plot.

Sending keypoints instead is also not practical: extracted keypoints typically require *at least* as much space as the image itself. Even after heavy GZIP compression, keypoints require comparable space for most images, and five times more uncompressed (Figure 5.4). We envision an alternative. What if, instead of shipping the entire image data (or all image keypoints), we ship only those parts which

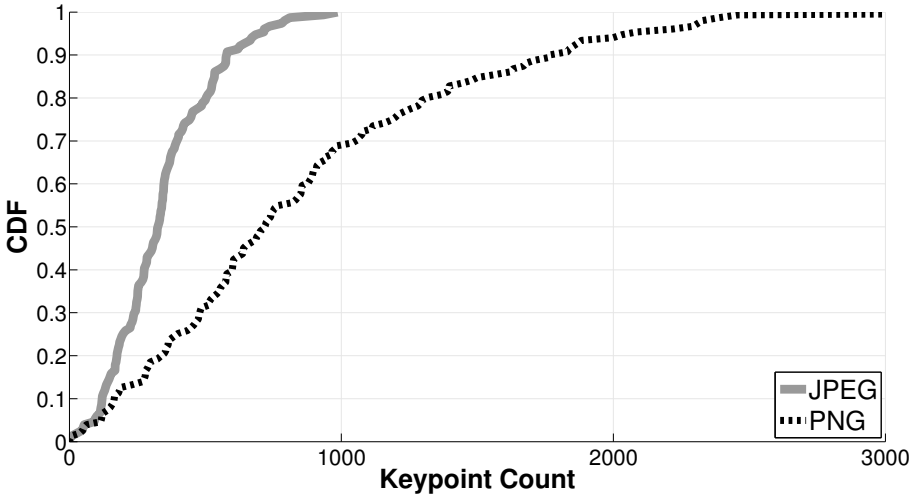


FIGURE 5.3: CDF of number of SIFT keypoints, PNG versus JPEG compression (same compression ratio as Figure 5.2). Under compression, SIFT feature extraction efficacy drops substantially. Peak performance is achieved using lossless compression, such as PNG images.

define a unique fingerprint?

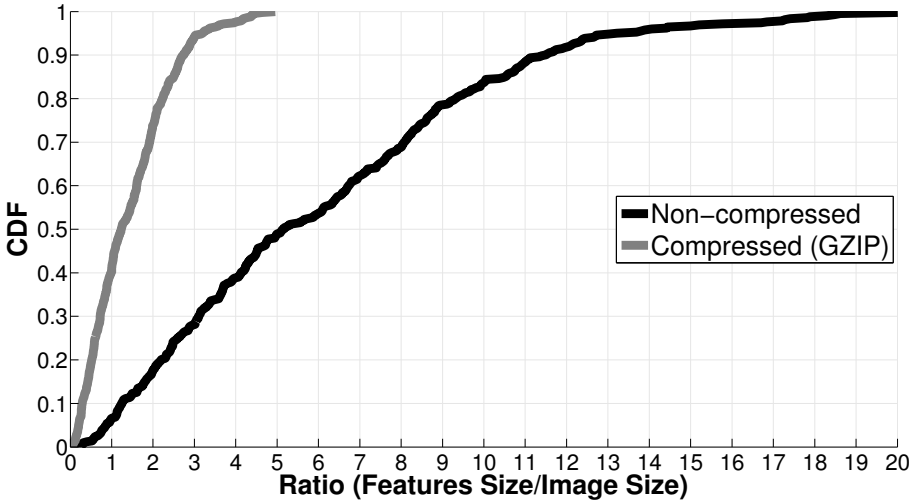


FIGURE 5.4: CDF of SIFT feature size (in bytes) ratio to image size. Even after heavy compression, features require comparable space to the original image.

CHEF: A Uniqueness "Oracle"

Imagine waking up in a maze of blank, white walls. It is hard to find your way through, given that each area looks the same as the one preceding it. Now imagine you have a marker. By drawing a unique symbol on each wall – equivalent to a few bits of information – you could quickly create a signature to identify where you are at any time. This paper is about finding those few bits of information that naturally occur in real-life environments.

To construct a visual fingerprint, not all parts of an image are equally useful. To reduce cloud-offload upload bandwidth, we need to distill to only those high-entropy bits. This is analogous to delta (difference) encoding for video compression: all data is thrown away, except the information gain. The challenge here is that the *entropy of any pixel cannot be deduced from strictly local information*. For example, imagine an art gallery. The one-of-a-kind paintings are likely to contain good candidate, highly-unique keypoints. The corners of a checkerboard floor or the regular

pattern of ceiling tiles are less so. Many keypoints could be quickly discarded – for example, if two very similar keypoints appear in the same image. However, some might be unique in a particular image, yet common across several. For example, a door knob or light switch might be unique in a room, but repeated in every room of a building. To recognize this global repetition, we need something of a uniqueness "oracle."

We want to create a visual lookup mechanism whereby a smartphone app can capture a photograph or stream of video and quickly identify a handful of image keypoints that are highly unique, to define a short fingerprint description. Instead of 2,000 keypoints from a single photograph (not an unusual result from a high-resolution photo using SIFT), we wish to capture the same essence in only 200 – an order of magnitude fewer. The app can then upload this short description (approximately 30KB) of the scene to a cloud service. On that service, we can apply existing techniques of image-based content retrieval (like Google Reverse Image Search or TinEye.com). Instead of similar images, the service replies with metadata, an estimate of location and camera pose.

We call our approach *CHEF* for *C*oncise *H*igh-fidelity *E*nvironmental *F*ingerprinting. CHEF uses only local information (on the smartphone) to leverage a global awareness of which visual data is worthwhile (curated on the server and downloaded to the client in advance). Since this table can be aggressively probabilistic – false positives create a minimal performance penalty – the representation can be extremely compact. Our design ensures that lookups are cheap – constant time per image keypoint – so we can filter all captured image data to only the most essential (the "compact" fingerprint).

In Figure 5.5, we present our intuition for the opportunity. Most of the 128 dimensions of a SIFT keypoint feature descriptor contain minimal information to differentiate it from all others. Generally, only a few dimensions are required to

isolate a descriptor from its closest neighbor. Locality-sensitive hashing (LSH) is a natural choice to re-project a minority of valuable dimensions into a more manageable low-dimensional space – enabling efficient high-dimensional nearest neighbor search.

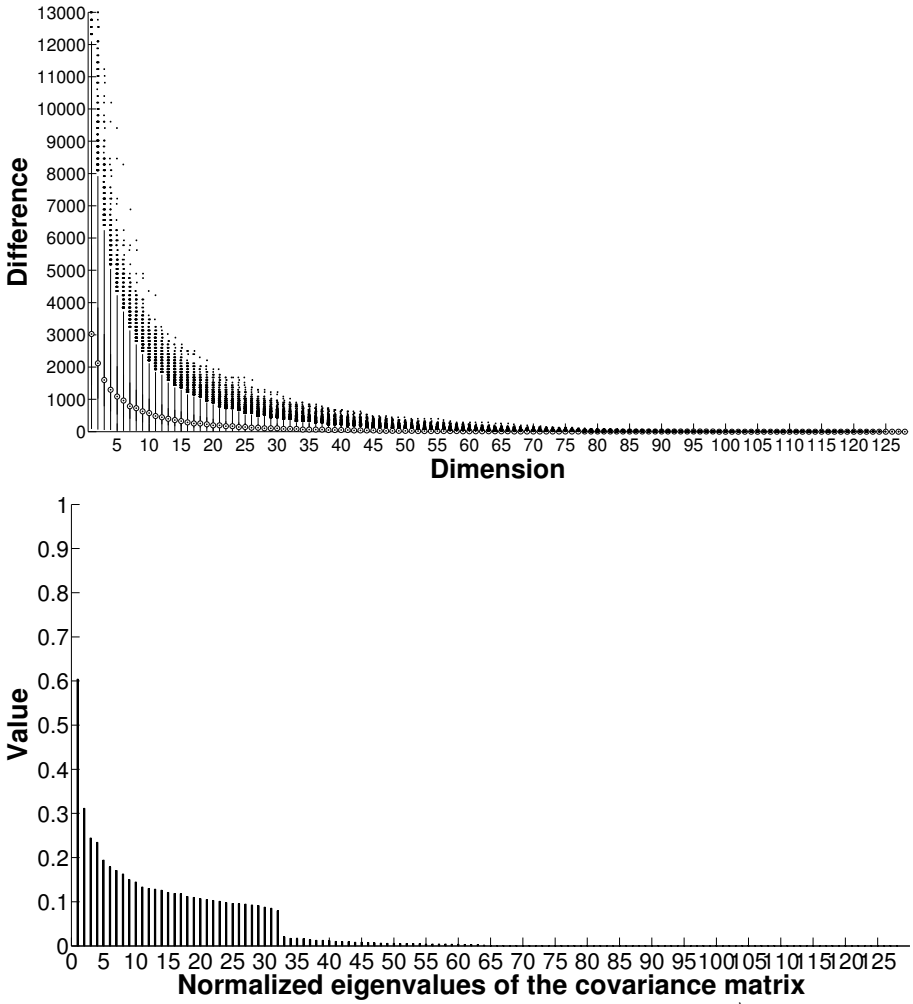


FIGURE 5.5: (a) For 500 images, each feature descriptor $\vec{A}$ matched to nearest neighbor in the database $\vec{B}$. Boxplots show `sort_reversed`$[(\vec{A} - \vec{B})^2]$. Few dimensions provide most of the Euclidean distance between $\vec{A}$ and $\vec{B}$. (b) Principal component analysis (PCA) confirms this intuition, as only a few PCA dimensions (far less than 128) are enough to account for the majority of covariance.

With feature descriptors distilled using LSH, our key technique applies counting Bloom filters to build small, probabilistic lookup tables. These tables (10s of

MBs) are downloaded to a user's device in advance to summarize vast quantities of visual data (1000s of MBs). An image keypoint can be quickly tested against these tables to quantify its uniqueness – how often it appears in other images, globally. Quickly, thousands of keypoints can be sorted according to these uniqueness estimates. The several (200) most unique are then chosen for matching on a cloud service – others immediately discarded.

Beyond the core intuition, our end-to-end implemented CHEF prototype consists of three major system components: (1) a wardriving app based on Google Project Tango pro (2014) using Simultaneous Localization and Mapping (SLAM) techniques to quickly construct a 3D database of visual features, applicable to indoor environments; (2) a cloud storage and compute service to curate this database, by identifying the most unique visual features, and respond to fingerprint queries with a location estimate; and (3) a smartphone app to capture photographs and extract visual features, subselect features by estimated uniqueness to a concise "fingerprint," and perform location lookups on the cloud service.

CHEF makes the following contributions:

1. Unique application of 3D SLAM to construct a location database suitable for lookup on 2D cameras.
2. Novel visual-uniqueness lookup tables, constructed as a fusion of LSH indexing and Bloom filters.
3. Client-side data filtering from global uniqueness: offloading upload reduced an order of magnitude.
4. Instantaneous, 3D visual indoor localization with median error of 2.5m from ground truth.

5.2 Related Work

CHEF builds upon a large body of prior art, especially in the areas of image-based content retrieval, cloud offload support for mobile systems, visual approaches for localization and mapping, and multi-sensory indoor localization. Additionally, the CHEF approach can enhance several emerging mobile applications beyond localization.

Image-based Retrieval: Image-based content retrieval has been extensively studied, including consideration for resource constraints applicable to mobile devices. Zhang et al. (1995) proposes a method to match two uncalibrated stereo images robustly. Lowe (1999); Bay et al. (2008) identify robust features in an image to achieve high accuracy. Rublee et al. (2011); Alcantarilla et al. (2012) are binary variants of image features which can be used to perform faster image matching based on Hamming distances. While better feature design improves matching latency between a pair of images – this is insufficient to scale to a database of a large size. Approximate matching schemes such as Silpa-Anan and Hartley (2008); Datar et al. (2004) address this issue to some extent, but memory and computation overheads of indexing hinder their applicability to resource-constrained smartphones. Similarly, running matching algorithms on GPUs Gong and Yang (2005) is feasible on a server but not readily on smartphones. Kirsch and Mitzenmacher (2006); Hua et al. (2012) design locality-sensitive Bloom-filters to address memory limitations of approximate high-dimensional matching. We extend these approaches for CHEF, but in a novel application to identify unique image features.

Cloud Offloading for Mobile Applications: Computation offloading to a remote server has been used to overcome computational, memory, and energy limitations of a smartphone Cuervo et al. (2010); Kemp et al. (2012). However, real-time offloaded image processing remains challenging for several reasons: upload

throughput between the phone and the cloud is often limited (especially over cellular, even LTE uplinks); wireless network latencies between the phone and cloud are unpredictable and can not guarantee a consistent user experience Lee et al. (2010); and high power consumption in continuous uploading defeats a primary incentive for offloading Jain et al. (2015a); Kumar and Lu (2010). Hybrid architectures leveraging the phone to do some local computation to (e.g., to pick frames to be uploaded on the cloud) partially address this issue Hu et al. (2015). Satyanarayanan et al. (2009) proposes an alternate architecture of bringing computation power in the proximity of mobile device.

Visual Simultaneous Localization and Mapping (VSLAM): VSLAM refers to problem of simultaneously constructing/updating a map while tracking the location of a device, using a camera and other onboard sensors Karlsson et al. (2005). The Project Tango hardware, used in CHEF, is essentially VSLAM put into practice. VSLAM heavily relies on statistical techniques such as Kalman or particles filter to smooth and correct dead reckoning errors of noisy sensor data. Bundle adjustment Triggs et al. (2000) is used to generate a 3D model of the scene, while simultaneously estimating and correcting optical parameters of the camera. Later, image registration Lucas et al. (1981) and inertial tracking is used to find an instantaneous localization of the device Sattler et al. (2011). VSLAM is implicitly applied in CHEF, through our reliance on Google Tango during the initial 3D wardriving phase.

Indoor Localization in Other Domains: WiFi and inertial sensing have been applied for indoor localization Bahl and Padmanabhan (2000); Wang et al. (2012). Our approach to visual fingerprinting is complementary to, and can be further enhanced by, these works: location context can be used to reduce image search on the server. However, this context is not ubiquitously available. Thus, we do not assume such a location service, to the benefit of generalizability.

Applications: Crowdsourced video is rising in importance, as evident in the popu-

larity of Youtube, Periscope per (2015), and Meerkat mee (2015). Several applications built atop demand real-time understanding of location context in the video streams dex (2015), as can be provided by CHEF. Image-derived location context can also immediately benefit applications for augmented reality ocu (2015); hol (2015), mobile object recognition Amazon (2014), movable-camera surveillance nes (2015), life-logging Hodges et al. (2006), sports video analytics Jain et al. (2013), mood-sensing LiKamWa et al. (2011), and reaction sensing Bao et al. (2013). We believe that CHEF can be productively applied to each of these domains.

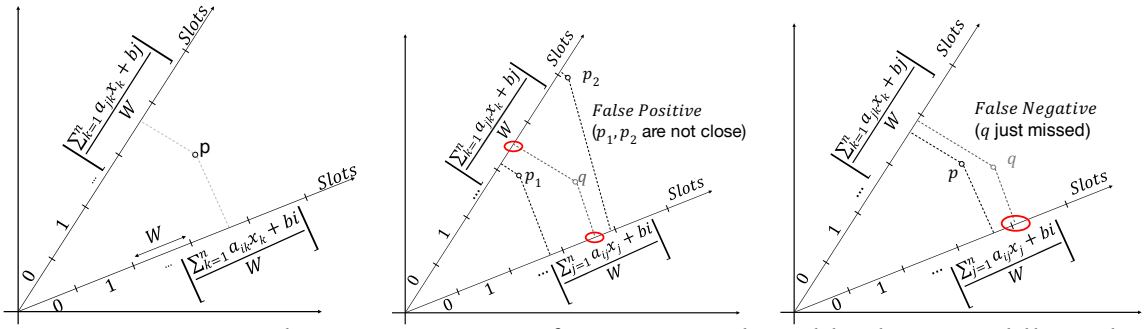


FIGURE 5.6: LSH Scheme Overview. Left: Projection based hashing. Middle: False Positives. Right: False Negatives.

5.3 System Design

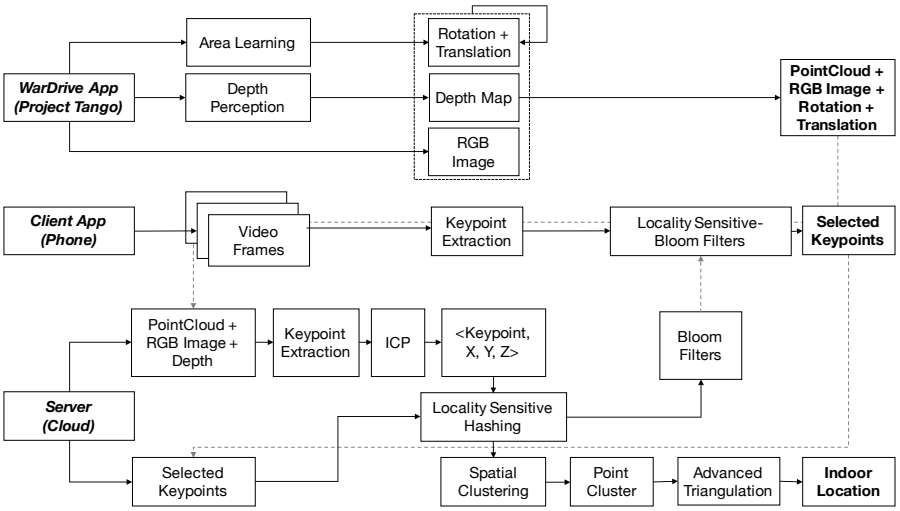


FIGURE 5.7: System overview. Top row: wardriving-app based on Google Tango. Top right: output of wardriving app consists of 2D image keypoints with corresponding 3D metadata. Middle row: phone (client) app captures video frames, extracts keypoints, identifies a subset of highly-unique keypoints, and queries to server. Bottom row: server both processes wardriven image 2D/3D data and also response to client queries. Bottom right: end of query-response pathway, final result is a 3D location inferred from a client query (2D, comprised of several highly-unique keypoints).

Figure 5.7 illustrates CHEF's simplified system architecture, consisting of three major components: (1) a client app for commodity off-the-shelf Android smartphones, providing visual fingerprinting and localization; (2) a cloud server module, providing visual fingerprint pre-processing and fingerprint-to-location lookup services; and (3) a wardriving app for Google Tango app, to populate the server with visual fingerprint data, each time a new building is added to the database.

The smartphone client "app" is really a proof of concept. It exists only to demonstrate and test the CHEF client library, which could be easily incorporated into any Android app requiring fingerprinting and/or localization services. Or, it might be plausibly incorporated as a cross-app system-level service – enabling camera-based localization applicable indoors when GPS is untenable.

When the library loads the first time, it contacts the CHEF cloud service to download up-to-date feature-uniqueness tables (to be refreshed periodically). These are structured as a set of counting Bloom filters, indexed using locality-sensitive hashing. We describe this next.

Uniqueness "Oracle" Construction

Primer – Bloom Filters: A Bloom filter is a probabilistic data structure, designed for efficient (constant time and memory) set membership queries. That is, one can look for the presence of a particular (exact) element: range or nearest-neighbor queries are not supported. There is some uncertainty to a lookup, hence considered probabilistic. The filter can determine that an element is either *definitely not* in the set or *may be* in the set. Bloom filters are typically implemented using a fixed-size bit vector. To insert, an element is hashed multiple times using a cryptographic hash (typically, a hash is selected for execution speed over cryptographic guarantees, such as Murmur-3). The hash output is taken modulo the size of the bit vector as an index – setting these positions to true. To test set membership, the query element is simply hashed again and the corresponding bits are checked. If all the hashed bits set to true, the element is probabilistically assumed to be present. If a single hashed bit is set to false, the element is definitely not present.

A *counting* Bloom filter is a variant that maintains counters (rather than binary bit flips), enabling accumulation for multiple occurrences of an element. CHEF leverages counting Bloom filter with a low saturation point (beyond which additional insertions of the same value have no effect). Once saturated for a particular feature descriptor, we can be confident that the corresponding keypoint is fairly common, not useful to our purpose.

Primer – Locality Sensitive Hashing (LSH): LSH techniques widely apply to efficient searching in a high dimensional dataset. The intuition is to reduce dimensions

of the data in such a way that similar items map to similar or nearby buckets with high-probability. Assuming the number of buckets is much smaller compared to the original data, an efficient nearest-neighbor lookup can be performed. For CHEF, we wish to enable an efficient lookup mechanism of image descriptors. The closeness of two descriptors is measured by the Euclidean distance between them. Due to this requirement, from a variety of LSH variants, E^2LSH Andoni and Indyk (2004) is most natural for CHEF.

E^2LSH belongs to a family of random projection-based LSH schemes. Specifically, a point to be indexed is projected randomly on several hyperplanes. The expectation is that for two nearby points, most projections will also yield nearby scalar values for each. Therefore, once discretized, most nearby points will be indexed to the same LSH "bucket." What distinguishes E^2LSH from related schemes is in how it selects coefficients to each of the random projection hyperplanes. Critically, it has been shown that when one constructs hyperplanes with coefficients drawn as random samples of a p-stable distribution, the projection values preserve the L_p norm. Thus, under a 2-stable distribution, the L_2 norm – the metric of Euclidean distances – is preserved. The Gaussian distribution is 2-stable, and thus an appropriate choice. Hence, E^2LSH (under Gaussian coefficient selection) supports nearest neighbor lookup by Euclidean distance, appropriate for SIFT keypoints.

Locality-Sensitive Bloom Filters

CHEF blends the advantages of LSH and Bloom filters. LSH enables imprecise queries based on Euclidean distances; Bloom filters provide a compact representation. Figure 5.8 illustrates our locality-sensitive Bloom filter construction. From top to bottom, we begin with a single keypoint descriptor. A SIFT descriptor exists in 128-dimensional space (each dimension being a one-byte integer value). The first step is to take M random projections of the 128-dimension descriptor. Each

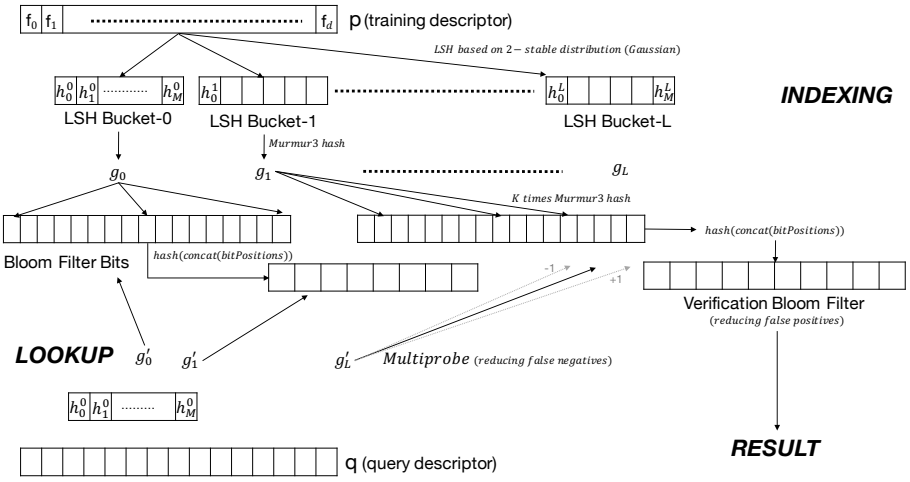


FIGURE 5.8: Locality-sensitive Bloom filter "oracle" construction. From top (indexing): image keypoint descriptor passes first through a locality-sensitive, then secondly, a cryptographic hash, yielding several bit indices into counting Bloom filters (middle). From bottom left (lookup): keypoint descriptor checked against Bloom filters. "Multiprobe" protects against off-by-one false negatives. Non-counting verification Bloom filter (far right) protects against false positives.

singular projection yields a single scalar value, with M projections yielding a M-dimensional vector. Each projection is quantized into a discrete space based on width parameter W. We construct L such M-dimensional vectors to create L LSH "buckets." Each of the $M \times L$ randomly-chosen projections is held constant for the life of the data structure. For each of the L buckets, we apply one of L cryptographic hash functions g_i from the same family (Murmur-3) on the M-dimensional vector. The hash output provides K indices into a counting Bloom filter (K for each of the L LSH buckets). For each index, we increase the corresponding count by one. Each bit index counter is represented in 10 bits, for a count saturation (maximum count that can be represented) of 1024. Beyond 1024, we treat a keypoint as not unique enough for consideration. We empirically optimize the various parameters to LSH and Bloom filters from a generic image keypoint dataset: $L = 10$, $M = 7$, $W = 500$, and $K = 8$.

Challenge, False Positives: Three sources of false positives exist in our scheme: (1)

Bloom filters, (2) LSH, and (3) the interplay of Bloom filters and LSH. The probability of Bloom filter false positive is tunable – trading off memory consumption. We tune Bloom filters to support up to 2.5M unique feature vectors with less than 1% false positives. LSH false positives are mitigated through empirical tuning of L, M, and W. After tuning, we observe few false positives from LSH alone. However, the interplay of LSH and Bloom is more problematic. The primary cause is due to the quantization of features (as tuned using LSH parameter W). Quantization must be coarse enough to capture similarity of two nearby feature descriptors. However, when W is selected to be more coarse, there are fewer unique "bins." Thus, the Bloom filter is used less uniformly – bits are only flipped to represent this smaller number of bins. After many insertions, "hotspots" in the Bloom filter may result. We address these in two ways. First, we choose a relatively higher saturation point for counting Bloom filters: we use 10 bits per Bloom filter index, for saturation at 1024, when fewer would otherwise be required. Secondly, we leverage an established technique from prior art – a *verification* Bloom filter. For each insertion to a primary Bloom filter, a second insertion is performed into the verification filter. However, instead of hashing the original data, we hash the bit positions of the insertions to the primary filter. During the query, a positive result is returned if and only if a positive match is found in both the primary and verification Bloom filters. *Challenge, False Negatives:* False negatives may occur in LSH, due to the quantization boundaries. Irrespective of the choice of quantization parameter W, it is possible that two nearby feature descriptors would be mapped to different quantization buckets (if the quantization boundary happens to fall in between). Fortunately, the error can be at most a single quantization bucket. Borrowing a technique from Lv et al. (2007), we perform "multi-probe" checks into adjacent quantization buckets to reduce these cases. During retrieval, we consider off-by-one partial matches in the Bloom filter – those with $K - 1$ of K bits matching. This requires a total of

$2K + 1$ Bloom filter checks – easily accommodated, as each Bloom filter test is extremely fast. If such a partial match is found, we declare this as a *potential* false negative case – accepting it a "pass" of the primary Bloom filter. The verification Bloom filter provides the final check. Given that multi-probes increase the probability of false positives, the verification Bloom filter becomes all the more crucial to our scheme.

Keypoint-to-3D Position Wardriving

The core data structure of the CHEF cloud service is a large-scale lookup table – mapping image features to their corresponding 3D spatial positions. Before CHEF can be used to *retrieve* visual fingerprints in an indoor space, the environment must be wardriven – populating this table. We build a visual-wardriving app for Google Tango pro (2014) hardware. While the user walks around a building, the app captures photographs (from a traditional RGB sensor), a depth map (from an integrated IR depth sensor), and the location of the user (relative to the start position, tracked through visual-SLAM).

During the wardriving phase, the CHEF app for Tango is used to collect a series of snapshots with corresponding metadata: the six degree of freedom (6-DoF) pose of the device, an RGB image (from which keypoints will be extracted), and a lower resolution depth map of the corresponding view (from an embedded IR-based depth sensor). The 6-DoF pose is relative to the initial position (where the app was initialized). It includes three dimensions of translation in (x, y, z) and three dimensions of device rotation/orientation (yaw, pitch, roll). To wardrive a venue, a user needs to walk throughout the indoor space to be captured. This process is manual (the user must physically walk), but could be automated using area-covering robots like Roomba. During collection, the Tango app pushes images, depth context, and positioning metadata to the CHEF cloud service. As described

in the next subsection, the CHEF cloud service pre-processes this data to establish mappings between unique image keypoints and their corresponding 3D locations. The number of such mappings in a building can be exceedingly large – hundreds of thousands to millions.

The Tango prototype does not immediately provide all of the required data for CHEF. Practical limitations, due to Tango's unpolished beta status, complicate our implementation. Principally, when leveraging Tango visual-SLAM and depth mapping functionality, all visual sensing hardware is locked, and cannot be otherwise accessed by third party Tango (a form of Android) apps. We leverage area learning and depth perception to achieve an understanding of where a depth map exists in relation to others in the same indoor space. Unfortunately, Tango blocks access to the RGB camera when using these services tan (2015). So, it is less straightforward to map a 3D depth estimate, projected into a shared indoor space via area learning, to a portion of an RGB image frame. Note that much of Tango exists in closed source, so we were not able to circumvent the hardware protections. While it is possible to grab the RGB screen buffer, this is too low resolution for our needs. Instead, we used a low-tech workaround: we taped a second Android device to the back of our Tango and created a simple network API to synchronously grab image frames in concert with Tango-recorded depth maps (see Figure 5.9). Of course, limited RGB access is not fundamental, and we expect this shortcoming to be addressed in the future. Figure 5.10 shows how we merge Tango-acquired depth data into RGB.

Challenge, Positioning Error and Uniqueness: Unfortunately, but unsurprisingly, there are inaccuracies in Tango's visual-SLAM positioning. As the user walks around, location estimates naturally reflect some amount of drift from true positions. While small amounts of positioning error might not cause undue harm to many potential Tango applications, for CHEF, this can be problematic.

FIGURE 5.9: Tango connected with phone using double-backed tape and foam board.

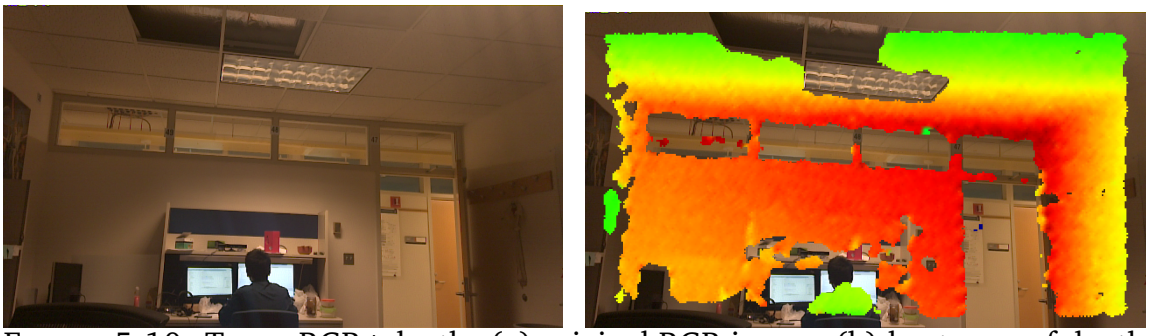


FIGURE 5.10: Tango RGB+depth: (a) original RGB image; (b) heat map of depth from observer, red is farther away.

If two photographs appear to be of different objects (due to erroneous location estimates), truly-unique keypoints may appear to be repeated – a false negative for our uniqueness tracking. We resolve this issue by post-processing Tango's depth map output – a 3D *point cloud*. We apply iterative closest point (ICP) heuristics to merge Tango 3D depth maps (from separate snapshots) into a single coherent point cloud for the entire indoor space. Only from this converged, comprehensive depth map we can be sure that two keypoints reflect truly independent locations. This post processing also has a secondary advantage of reducing some error in our location estimates.

Cloud Processing and Localization Service

CHEF cloud service accepts keypoint-to-3D position mappings from the Tango wardriving app. As they are received, the service updates two data structures: (1) the keypoint-to-3D position lookup tables and (2) LSH-indexed Boom filters for keypoint uniqueness.

The lookup table is really a large-scale image-based content retrieval table, as one might apply for reverse image search. For our prototype, we apply standard LSH. Rather than a reference to an image or other content, the LSH table stores the 3D position of the keypoint. The service accepts queries to this table in the form of keypoint-plus-2D coordinate pairs (the x, y coordinate of the keypoint in the camera frame). The service performs an LSH lookup on each keypoint to retrieve its 3D position. It then applies a localization step, described next, and returns the estimated client location and pose.

The second data structure, the LSH-indexed Bloom filter, is the same exact data structure downloaded by CHEF clients as the uniqueness oracle. As new keypoint-to-3D mapping arrive at the CHEF cloud service, we simply perform an LSH lookup followed by insertion to the primary and verification Bloom filters. With this simple design, new keypoint-to-location mappings can be incorporated continuously, in constant time and memory.

Keypoint-to-3D Client Localization

For each client localization query, the cloud service retrieves the 3D position associated with each image keypoint. This is a straightforward query to a standard LSH-indexed lookup table. However, the retrieved 3D positions are only the first step towards localizing the client user. When combined with the 2D data of how the 3D position is perceived by the client device, it is possible to infer the clients 6-

DoF camera pose, including translation (x, y, z) and orientation (yaw, pitch, roll). In the final effect, we achieve a similar positioning fidelity as Google Tango, but *with only a standard, 2D, RGB camera*. Unlike several approaches in prior art Wang et al. (2012); Sen et al. (2013); Li et al. (2014), CHEF achieves instantaneous localization – continuous tracking not required.

From a single image I, a client queries CHEF with a set of keypoints K. From the LSH-indexed lookup table, CHEF finds $|K| \cdot n$ nearest-neighbor matches (since there is uncertainty in the match), and accordingly $|K| \cdot n$ 3D points. On these points, CHEF applies spatial clustering to filter down to only those 3D points in the largest cluster P, discarding others. $K' \subseteq K$ keypoints remain, for $|P|$ 3D points, where $|K'| \leqslant |P| \leqslant |K'| \cdot n$. For each 3D point $p \in P$, there is a single corresponding keypoint $k^p \in K'$. We also know that k has a corresponding 2D pixel coordinate (x_p, y_p) in the original client query image I.

Figure 5.11 illustrates the spatial relationship between a CHEF client camera capturing image I and the known 3D points corresponding to keypoints found in I. Assume the client's camera exists in 3D space at point A. B is at the center-right (along the edge) and C is at the center of the image. Therefore, $\angle CAB$ is half of the horizontal field-of-view of the camera. The diagram shows how one can calculate angles from A, w.r.t to the center C. Let (p_x, p_y) be the pixel coordinates of a keypoint P and γ_x be the angle $\angle CAP$ (P's projection on the x-axis). Using known 2D pixel coordinates $B = (width, height/2)$ and $C = (width/2, height/2)$, and the camera's field of view (FoV_h, FoV_v), we may estimate γ_x and γ_y.

By the same logic, for a pair of keypoints P_0 and P_1, we may compute x-axis angles $\gamma_x(P_0)$, $\gamma_x(P_1)$, and y-axis angles $\gamma_y(P_0)$, $\gamma_y(P_1)$. Now, the x-axis angle between P_0 and P_1 is $\gamma_x(P_0) + \gamma_x(P_1)$, if P_0 and P_1 fall on opposite sides of C, or $|\gamma_x(P_0) - \gamma_x(P_1)|$ if the are on the same side. The same method applies to find the y-axis angle.

From several such point pairs (in the same image), CHEF estimates the client's location in 3D space. CHEF models this geometry as a nonlinear optimization. Figure 5.12 presents the objective function, constraints, variables, and parameters of the optimization.

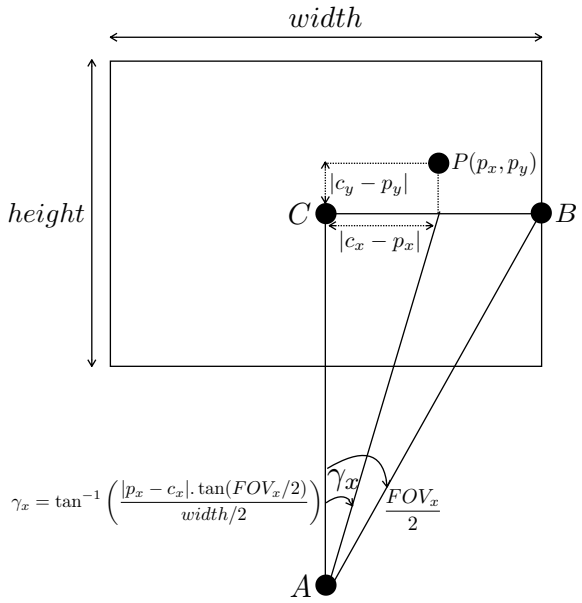


FIGURE 5.11: Geometry of angular separation from a keypoint P and the center of the screen C, respective to the 3D location of the client's camera at A. The angle $\angle CAP$ (called γ_x) is of interest – the angle from center to a projection of P along the x-axis. All expressions treat the distance of line segments $\overline{AC}$, $\overline{AP}$, and $\overline{AB}$ as unknown.

Client Android App

The client app serves as a proof of concept for our client library, intended to bring visual fingerprint matching to any application. The app demonstrates CHEF's capabilities for realtime localization, based on visual features.

When launched for the first time, the app retrieves the current state of the uniqueness LSH Bloom filter ("oracle") from the cloud server. In our testing, this download is approximately 10MB, but may be more or less, varying with the number of unique keypoints on the server. Although the Bloom filters are of fixed size,

$$
\begin{aligned}
\textbf{Minimize} \quad & \sum_{\forall i<j \in 1 \ldots K} E_{ij}^x + E_{ij}^y \\
\textbf{Subject to} & \\
\gamma_{ij}^x &= \gamma(p_i^x, C_x, F_h, W) - \gamma(p_j^x, C_x, F_h, W) \\
\gamma_{ij}^y &= \gamma(p_i^y, C_y, F_v, H) - \gamma(p_j^y, C_y, F_v, H) \\
\cos(\gamma_{ij}^x + E_{ij}^x) &= \frac{d(x,z,x_i,z_i) + d(x,z,x_j,z_j) - d(x_i,z_i,x_j,z_j)}{2.\sqrt{d(x,z,x_i,z_i)}\sqrt{d(x,z,x_j,z_j)}} \\
\cos(\gamma_{ij}^y + E_{ij}^y) &= \frac{d(y,z,y_i,z_i) + d(y,z,y_j,z_j) - d(y_i,z_i,y_j,z_j)}{2.\sqrt{d(y,z,y_i,z_i)}\sqrt{d((y,z,y_j,z_j)}} \\
\gamma(p,C,F,S) &= \tan^{-1}\left(\frac{|p-C|.\tan(F/2)}{S/2}\right) \\
d(x_1,y_1,x_2,y_2) &= (x_1-x_2)^2 + (y_1-y_2)^2 \\
\textbf{Solving for} & \\
\forall i<j \in 1 \ldots K &: x, y, z, E_{ij}^x, E_{ij}^y \\
\textbf{With parameters} & \\
\forall i<j \in 1 \ldots K &: p_i^x, p_i^y, p_j^x, p_j^y, x_i, y_i, z_i, x_j, y_j, z_j
\end{aligned}
$$

Left Side	Interpretation
i, j	Keypoint indices, $i < j$.
γ_{ij}^x	Angle from keypoint i to j, projected on the x-axis.
$\cos(\gamma_{ij}^x + E_{ij}^x)$	Geometric constraint derived from law of cosines.
$\gamma(p, C, F, S)$	Angle from client at A to keypoint, as a function of pixel location, image center, field of view, and side length (width or height).
$d(x_1, y_1, x_2, y_2)$	Square of Euclidean distance $(x_1, y_1) \leftrightarrow (x_2, y_2)$.
(x, y, z)	Optimized 3D coordinate of client (position A).
$E_{ij}^x + E_{ij}^y$	Minimized angular error between keypoints i, j.
(x_i, y_i, z_i)	Known 3D position of i.
(p_i^x, p_i^y)	Viewed 2D position of i.

FIGURE 5.12: Nonlinear optimization to estimate client camera position A at (x, y, z).

we compress them with GZIP for efficient retrieval, and compressibility reduces as the Bloom filter becomes more saturated. The app periodically refreshes its copy of the Bloom filter to stay current with the server. We could reduce data transfer by sending only a compressed bitmask representing the diff between versions (not yet implemented).

Once ready, the app activates the smartphone camera. The app continuously extracts video frames. It performs a quick check on each frame to detect blur (often due to quick motion), discarding such frames. It also rejects frames when processing falls behind the realtime stream. That is, the app only processes extremely recent frames. For each frame passing all checks, the app extracts SIFT keypoints – up to several thousand per frame, depending on image content. The app then queries the uniqueness LSH Bloom filter for each keypoint. Recall that the primary Bloom filter is of the counting variety. Thus, uniqueness counts (up to the saturation point of 1024) yield a partial ordering, ranking keypoints from highly unique to common. The app then upload several of the most unique keypoint descriptors (experiments with 200 and 500 are presented in the next section).

Upon receiving these unique keypoint descriptors, the cloud service performs the keypoint-to-3D client localization, returning the 3D camera pose estimate to the client. The app then displays the estimate on screen.

5.4 Evaluation

Our evaluation considers these research questions:

1. *How does CHEF's accuracy compare with other image/keypoint matching schemes?* In particular, we compare CHEF to a BruteForce Euclidean distance match (implemented on GPU), a traditional LSH implementation (as would be typical of a large-scale reverse image search), and Random where we uniformly subsample image keypoints.
2. *Are client-side overheads acceptable?* Does CHEF reduce data upload compared to conventional cloud offload? Memory? Computation? Energy?
3. *How effective is CHEF's localization performance?*

Matching Accuracy

We photographed 100 non-overlapping scenes across the three floors of our research facility, the Coordinated Science Lab (CSL). Each CSL floor is 50m $\times$ 10m in dimension. We captured each scene with a single image. We also capture 400 additional distractor images. These images consists of ceiling, floor, name-plates, furniture, etc. in the building. Since these images naturally contains repeated patterns, they are likely to have repeated visual features as well. For each of the 500 images, we use OpenCV's default SIFT implementation to extract visual features. Across these 500 images, the database contains a total of $\approx$2.5M feature descriptors (128 bytes each).

The query database consists of five additional photographs of each scene in the database. We systematically captured these five photograph from substantially different angles. As SIFT matching efficacy degrades substantially with angular separation, these diverse angles are intended to challenge all matching schemes.

The average number of features per query image was 3,500. From these key-

points, we evaluate CHEF across five different system regimes. Random picks 500 random keypoints from the query image and uploads them to the server for matching. Random can be understood as lower-bound on CHEF's performance (one with no intelligence in feature subselection). For CHEF-200 and CHEF-500, our complete system, respectively, selects the 200 and 500 most unique keypoints. LSH applies the reference E^2LSH locality-sensitive hashing implementation for nearest-neighbor search. BruteForce finds the "optimal" nearest neighbor match. Note that unlike Random and CHEF-200/500, LSH and BruteForce use all image keypoints. Note that while it may be tempting to consider BruteForce an upper bound on accuracy, it can be mislead by homogeneous keypoints across images.

Metrics: Let Y be the set of all processed query frames. Let $V \subset Y$ be the set of frames containing scene k. Let $P \subset Y$ be the set of frames identified by our system as capturing k. Then, $V \backslash P$ denotes our system's false negative predictions, $P \backslash V$ denotes false positive predictions, and $Y \backslash V$ denotes the set of frames which do not contain scene k. *We evaluate CHEF's prediction efficacy by the following standard metrics of information retrieval:*

Precision$_k$ $= |V \cap P|/|P|$

i.e., among all frames that CHEF identifies as capturing scene k, what fraction truly captures k.

Recall$_k$ $|V \cap P|/|V|$

i.e., among all frames that truly have capture scene k, what fraction was identified by CHEF.

Figure 5.13 presents CDF of (a) precision and (b) recall results. Unsurprisingly, Random performs poorly in both precision and recall. The results for BruteForce are more nuanced. BruteForce precision is poor, likely due to confusion among homogeneous keypoints. BruteForce recall is most favorable, likely some valuable keypoints are missed by CHEF, or it is able to differentiate even among homoge-

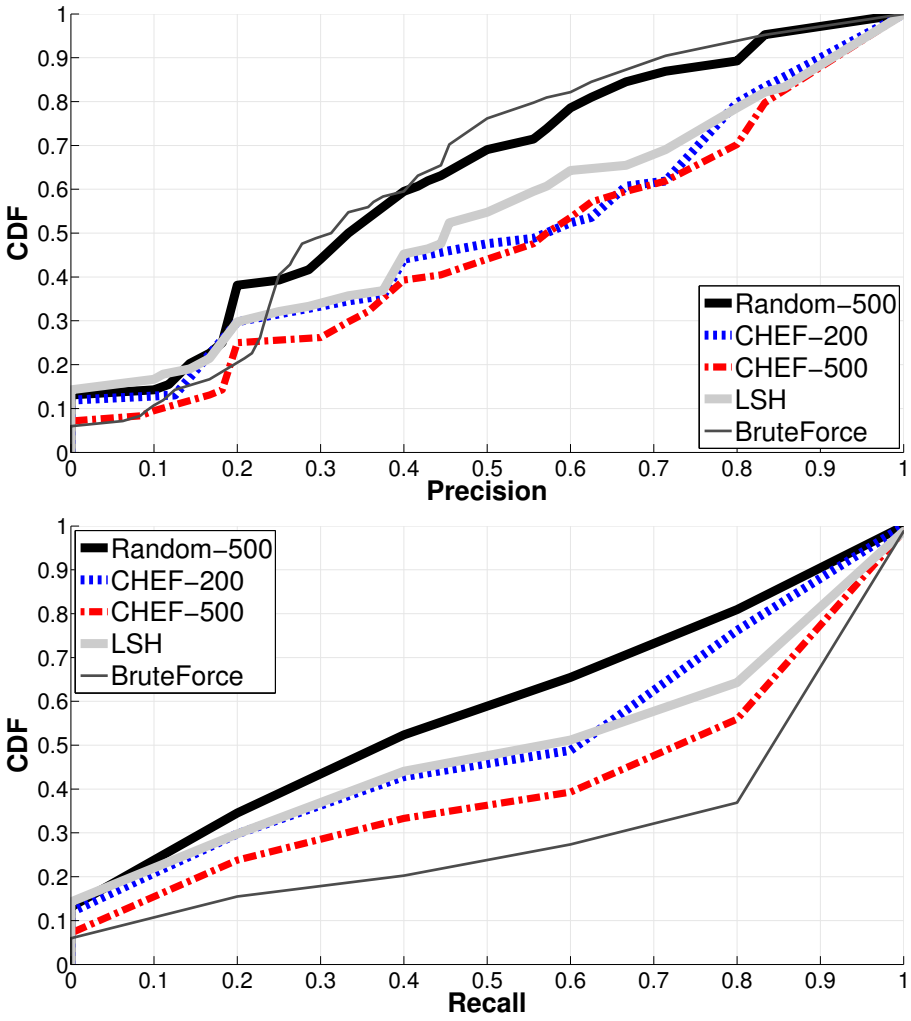


FIGURE 5.13: CHEF's precision bests even LSH and BruteForce as CHEF automatically eliminates useless, distracting non-unique keypoints (homogeneous repetitions can lead to false positive matches). Recall suffers slightly against BruteForce (some additional false negatives), but still bests LSH and random keypoint subselection.

neous keypoints based on their relative quantities (the entire distribution is scored in the BruteForce heuristic). LSH performs comparably to CHEF-200, but is inferior to CHEF-500 – likely due to the same confusion-under-homogeneity afflicting BruteForce.

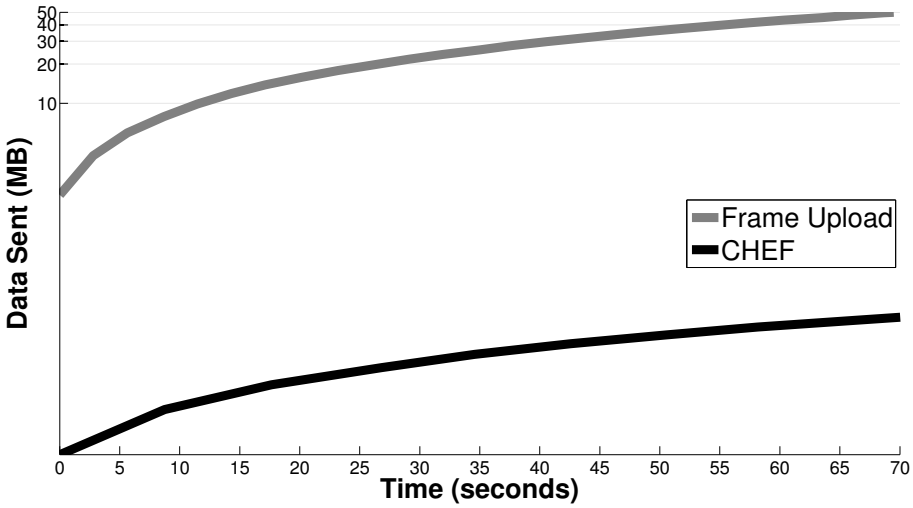


FIGURE 5.14: Cumulative data upload by execution time. CHEF reduces data consumption by at least one order of magnitude compared to raw frames (sending all raw keypoints would be more than frames, as shown in Figure 5.4).

Client Overheads

Data Upload Savings: Figure 5.14 compares upload data for CHEF versus whole frame upload. Note that, as shown in Figure 5.4, uploading keypoints (even compressed) saves no upload data versus the original frame.

Client Storage and Memory Overhead: Figure 5.15 compares the client-side storage/memory footprints of Random, CHEF, LSH, and BruteForce. Random requires effectively no memory (there is no index). CHEF has a substantial memory footprint, due to the Bloom filters. LSH has a extremely large memory footprint, much larger than the input data, due to multiple replications supporting multiple projections. BruteForce has a memory footprint comparable to the database size – here this is implemented on GPU as a SIMD matching (CPU matching would be nonsensical due to extreme latency). It would be possible to reduce BruteForce memory footprint by loading the database in parts, but with a trade-off of excessive loading latency, rather than a one-time cost.

Client Computational Latency: Figure 5.16 presents a CDF of CHEF computational

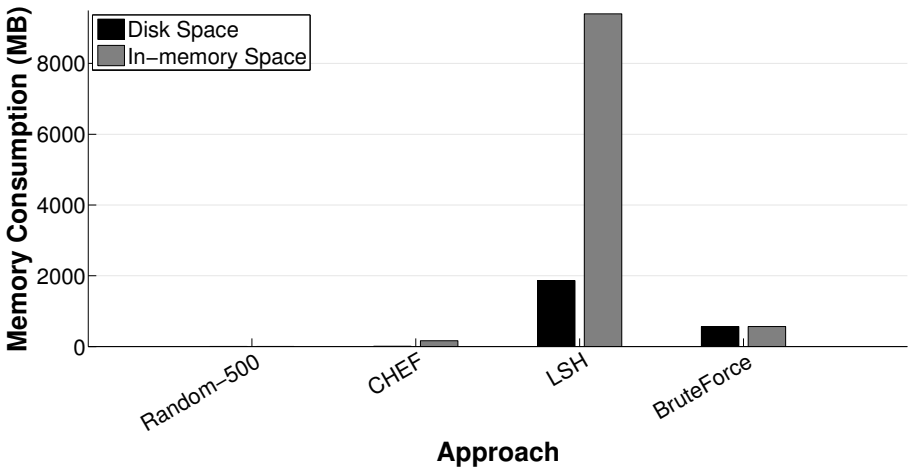


FIGURE 5.15: Client disk/memory consumption. CHEF uses substantially less memory than conventional LSH, which caches the entire image database. CHEF requires somewhat more memory than random keypoint subselection to maintain Bloom filters in memory. BruteForce memory consumption reflects loading all database keypoints into memory for a GPU SIMD parallel execution.

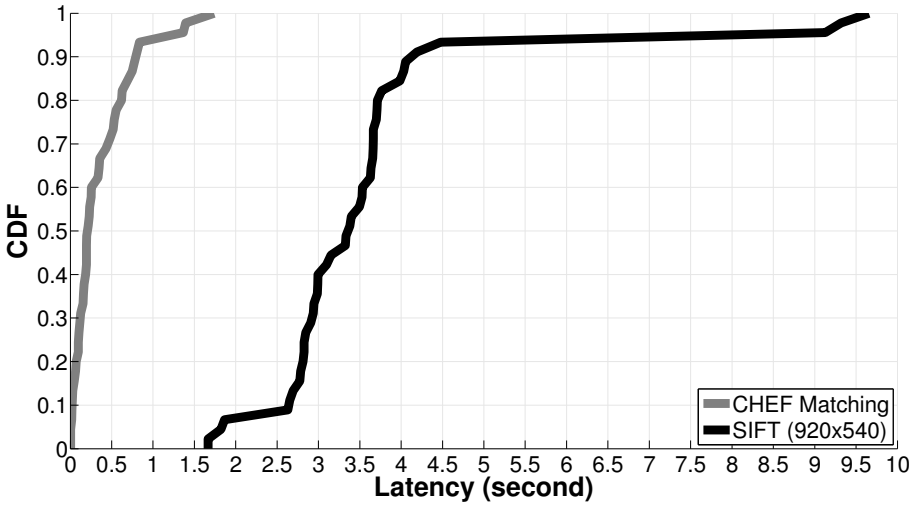


FIGURE 5.16: Computational overheads on Galaxy S6. CHEF requires an order of magnitude less computation than SIFT feature extraction. Ample room for end-to-end improvement by running SIFT on GPU or a coprocessor.

latency, as compared to SIFT feature extraction. Principally, CHEF latency is due to the LSH Bloom filter lookups for each keypoint descriptor, plus sorting. SIFT delay dominates CHEF.

Energy: All non-energy experiments were performed on the latest-generation Sam-

sung Galaxy S6 phone. However, as the battery compartment of the S6 is not removable, we revert to the prior generation S5 phone for energy measurements (only). The measurement setup is shown in Figure 5.17. We use a Monsoon power monitor to measure average power consumption of the phone. No additional applications apart from CHEF and required background services were running during the experiment. The phone was put on airplane mode to avoid cellular communications and WiFi was turned on for CHEF tests (only). As energy consumption is impacted by the number of extracted keypoints, lighting conditions and scene were kept same across all schemes. See Figure 5.18.

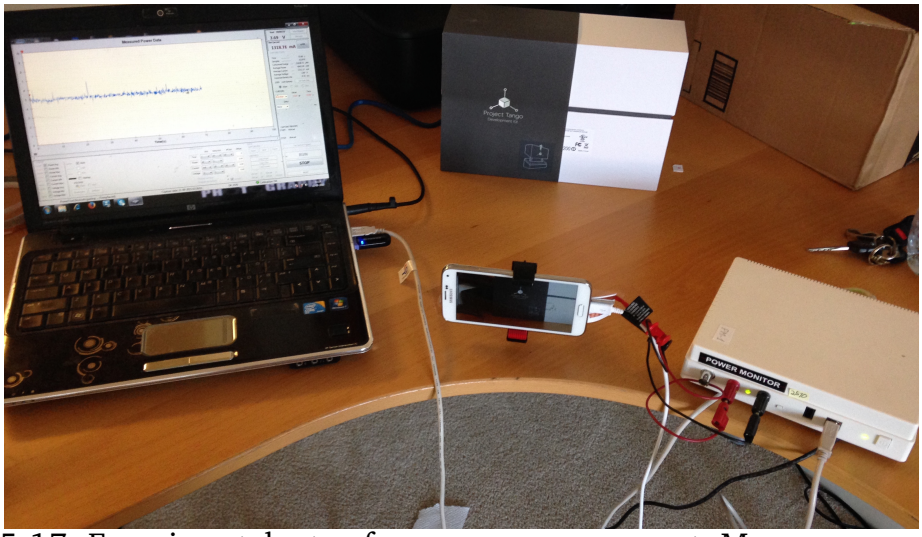


FIGURE 5.17: Experimental setup for energy measurement. Monsoon power meter provides current to Galaxy S5 phone in place of battery. Measurement at 5,000 Hz.

Localization

We test CHEF's localization performance in three indoor environments: an office space, employee cafeteria, and grocery store. The office environment consisted of similar looking cubicles, a kitchen area, and lounge. In the office, we wardrive a rectangular space of approximate dimensions 50m $\times$ 20m. The cafeteria consists of many identical chairs and tables, food menu boards, foodservice and checkout

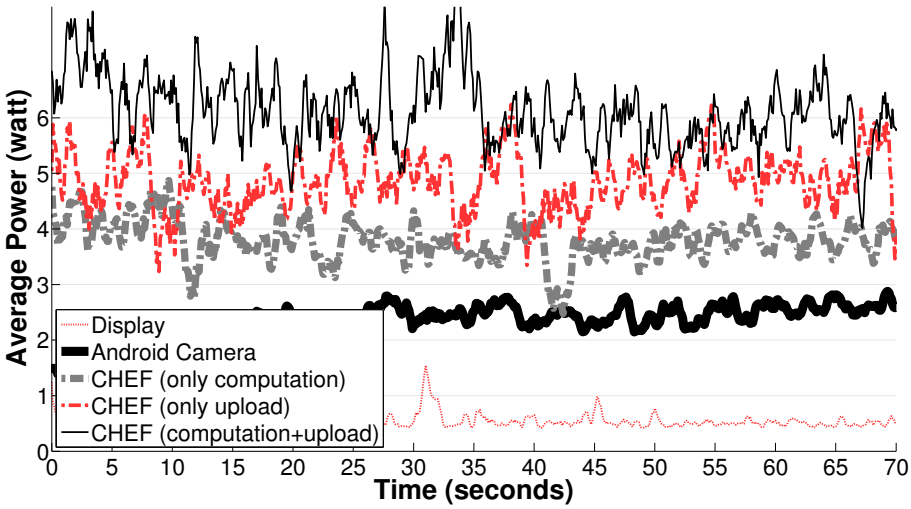


FIGURE 5.18: Energy consumption. Including display and camera, complete CHEF averages ≈ 6.5 W. Not shown: whole-frame cloud offload (no local compute) ≈ 4.9 W.

counters, and registers in a 50m $\times$ 15m room. The grocery store (80m $\times$ 50m) had standard aisle based layout, filled with household items.

During wardriving, the Tango was mounted with a phone (Galaxy Note 3) as shown in Figure 5.9. During query phase, a different phone (Galaxy S6) was used to take photographs, in arbitrary orientations. We use Tango to collect a "ground-truth" reference to compare query results. Wardriving and query data were collected along similar routes.

Figure 5.19 shows cumulative localization errors. Figure 5.20 shows isolates those errors to X, Y, and, Z directions. While localization is generally quite accurate, some failure cases do occur. We believe that most cases of substandard performance may be attributed to suboptimality at local minima (we solve the localization optimization using a time-bounded differential evolution).

Evaluation Takeaways

The key findings of our evaluation are as follows:

1. CHEF precision and recall are roughly comparable to LSH, the parent scheme

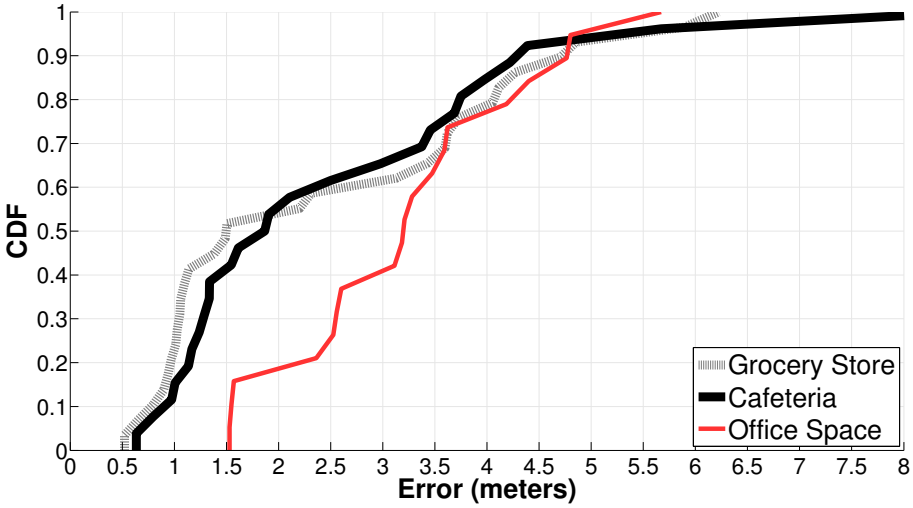


FIGURE 5.19: CDF of indoor localization error. Graph shows 3D distance from "ground truth" (as estimated from Google Tango), to CHEF's estimate on Galaxy S6.

on which CHEF is based.

2. CHEF requires $1/10^{th}$ bandwidth of whole frame uploads, 51.2 KB versus 523 KB (Figure 5.14).
3. Based on our 2.5M descriptor database, the CHEF client uses 10.5 MB disk space for Bloom filters, stored compressed – $1/124^{th}$ of the server side LSH indices compressed (1.3 GB).
4. Based on our 2.5M descriptor database, the CHEF client requires 162 MB in RAM uncompressed – $1/58^{th}$ of the LSH indices uncompressed (9.4 GB), cached in RAM by the server.
5. Client compute latency consists of SIFT extraction (3300 ms at median) and Bloom filter lookups (217 ms at median) for a total median compute latency of 3547 ms, dominated by SIFT (Figure 5.16). Substantial latency reduction is possible by running SIFT extraction on a mobile GPU or a coprocessor.
6. Client energy consumption is substantial (6.5 W), principally due to camera and local computation, especially SIFT feature extraction (Figure 5.18).
7. Results show CHEF has a median 3D localization error of 2.5 m (Figure 5.19).

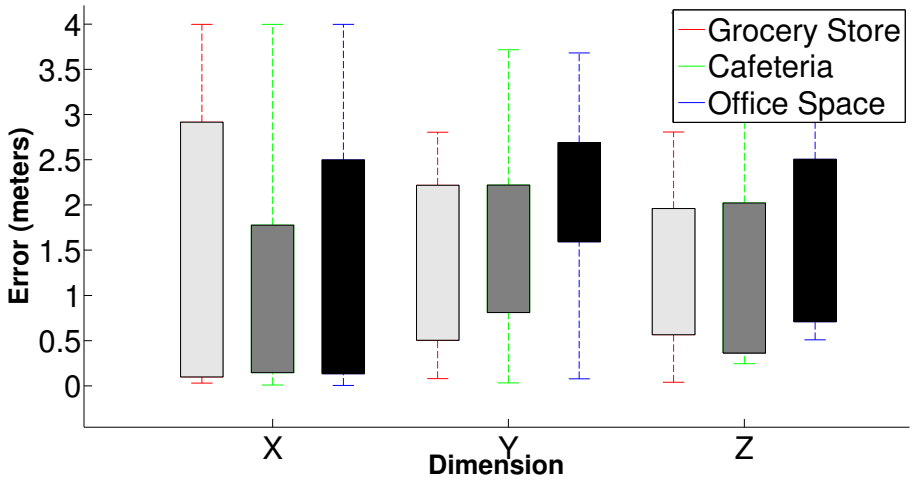


FIGURE 5.20: CHEF localization accuracy varies slightly by dimension. Localization on the horizontal X/Y plane (parallel to floor and ceiling) is somewhat more accurate than that of vertical motion (changes in elevation).

5.5 Limitations

Several practical limitations are apparent from our evaluation of CHEF. Principally, SIFT keypoint extraction incurs substantial compute overhead, yielding a large energy impact. Secondly, any visual fingerprinting scheme will incur a large energy cost, simply from turning on the camera. Fundamentally, CHEF could never be energy-competitive with lighter-weight sensory approaches.

CHEF should also not be taken as a replacement for traditional localization schemes. Rather it is an attempt to leverage visual data when it is anyways available (i.e., the camera is already turned on). Due to its visual nature, CHEF cannot always return precise user location. It can fail due to: (1) a lack of ample features in the query image, such as hallway with white walls; (2) insufficient wardriving – the environment at a location may not be well fingerprinted, causing image matching to fail; (3) false positives in keypoint matching – some environmental repetition might not be captured during wardriving; and (4) dead reckoning errors during wardriving – Tango uses IR depth sensing and IMU to track user location,

which are prone to errors (especially, in environments with natural sunlight or electromagnetic machinery).

5.6 Conclusion

Environmental fingerprint is a powerful primitive for mobile vision and augmented reality applications. However, the identification of visual fingerprints is challenging from a perspective of data storage, indexing, and retrieval. Cloud offload presents an obvious-if-challenging solution. CHEF enables selective cloud offload of only those visual fingerprints that carry the greatest global entropy – those that are highly unique, particular to the user's true location. Beyond visual fingerprints, we believe that the CHEF approach can be productively reapplied in other high-dimensional sensory domains, such as wireless RF, auditory, and hyperspectral signatures. We consider these in our ongoing research.

6

Conclusion

Real-time crowdsourced videos, emerging from handheld cameras are rapidly gaining popularity. However, the commercial value of them is yet to be explored. From our initial exploration, we realized that application building in this space is non-trivial due to many standing challenges. This thesis takes a top-down approach to address some of them. We propose several novel applications to highlight important design challenges. Clearly, the application space is rich and exhaustive and addressing all challenges is not possible. Therefore, we focus on problems which can leverage in-device sensing capabilities to reduce the burden on image processing. We demonstrate how powerful the combination of mobile sensing and computer vision is for a holistic system design. Our sensor-fusion primitives and real-time deployments show the feasibility of distant object localization, real-time video analytics, real-time tagging and retrieval for mobile AR, and visual fingerprinting of an environment at scale. Our design principles cut across multiple disciplines of computer science. This thesis leaves many challenges as future research. However, in its contributions of defining, designing, implementing, and evaluating a set of primitives, it advances the state-of-the-art in mobile computing.

Bibliography

(2013), “Exa-Tech,” http://www.exa-tech.com/.

(2013a), “Streamweaver,” http://streamweaver.com/.

(2013b), “Stringwire,” http://www.stringwire.com/.

(2013), “Switchcam,” http://www.switchcam.com.

(2013), “Vyclone,” http://vyclone.com/.

(2013), “Wikitude,” http://www.wikitude.com/.

(2013), “YinzCam,” http://www.yinzcam.com/.

(2014), “CrowdOptic,” http://www.crowdoptic.com/.

(2014), “Google Goggles,” https://play.google.com/store/apps/details?id=com.google.android.apps.unveil&hl=en.

(2014), “Illinois Distributed Museum,” http://distributedmuseum.blogspot.com/.

(2014), “Instagram Hyperlapse,” http://hyperlapse.instagram.com/.

(2014), “Project Tango,” https://www.google.com/atap/projecttango/.

(2014), “Qualcomm Vuforia,” https://www.qualcomm.com/products/vuforia.

(2014), “Videoguide, Antoni Gaudi Modernist Museum in Barcelona,” http://www.casabatllo.es/en/visit/videoguide.

(2015), “ByteLight: Scalable Indoor Location,” http://www.bytelight.com/.

(2015), “Dextro Stream,” https://www.dextro.co/.

(2015), “Meerkat: Live Stream Video.” http://meerkatapp.co/.

(2015), “Microsoft Hololens,” https://www.microsoft.com/microsoft-hololens/en-us.

(2015), "Nest Cam," https://nest.com/camera/meet-nest-cam/.

(2015), "OCULUS VR," https://www.oculus.com/en-us/.

(2015), "Periscope: Explore the world through someone else's eyes." https://www.periscope.tv/.

(2015), "Tango taking picture while sensing depth," http://stackoverflow.com/questions/29978991/how-to-take-high-res-picture-while-sensing-depth-using-project-tango.

Agarwal, S., Snavely, N., Simon, I., Seitz, S., and Szeliski, R. (2009), "Building rome in a day," in *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 72–79, IEEE.

Agarwal, S., Furukawa, Y., Snavely, N., Simon, I., Curless, B., Seitz, S. M., and Szeliski, R. (2011), "Building rome in a day," *Communications of the ACM*, 54, 105–112.

Alcantarilla, P. F., Bartoli, A., and Davison, A. J. (2012), "KAZE features," in *Computer Vision–ECCV 2012*, pp. 214–227, Springer.

Amazon (2014), "Amazon Fire Phone," `https://developer.amazon.com/public/solutions/devices/fire-phone`.

Andoni, A. and Indyk, P. (2004), "E2lsh: Exact euclidean locality-sensitive hashing," *Implementation available at*.

Arth, C. and Schmalstieg, D. (2011), "Challenges of Large-Scale Augmented Reality on Smartphones," *Graz University of Technology, Graz*, pp. 1–4.

Ashok, A., Gruteser, M., Mandayam, N., Silva, J., Varga, M., and Dana, K. (2010), "Challenge: mobile optical networks through visual mimo," in *MobiCom*, pp. 105–112, ACM.

Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S., and MacIntyre, B. (2001), "Recent advances in augmented reality," *Computer Graphics and Applications, IEEE*, 21, 34–47.

Bahl, P. and Padmanabhan, V. N. (2000), "RADAR: An in-building RF-based user location and tracking system," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, pp. 775–784, Ieee.

Bao, X. and Roy Choudhury, R. (2010), "MoVi: mobile phone based video highlights via collaborative sensing," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pp. 357–370, New York, NY, USA, ACM.

Bao, X., Fan, S., Varshavsky, A., Li, K., and Roy Choudhury, R. (2013), "Your reactions suggest you liked the movie: Automatic content rating via reaction sensing," in *UbiComp*, pp. 197–206, ACM.

Bay, H., Tuytelaars, T., and Van Gool, L. (2006), "Surf: Speeded up robust features," in *ECCV*, pp. 404–417, Springer.

Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. (2008), "Speeded-up robust features (SURF)," *Computer vision and image understanding*, 110, 346–359.

Birchfield, S. and Rangarajan, S. (2005), "Spatiograms versus histograms for region-based tracking," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, pp. 1158–1163, IEEE.

Chen, D. M., Tsai, S. S., Vedantham, R., Grzeszczuk, R., and Girod, B. (2009), "Streaming mobile augmented reality on mobile phones," in *ISMAR*, pp. 181–182, IEEE.

Chen, D. M., Tsai, S. S., Girod, B., Hsu, C.-H., Kim, K.-H., and Singh, J. P. (2010), "Building book inventories using smartphones," in *Proceedings of the international conference on Multimedia*, MM '10, pp. 651–654, New York, NY, USA, ACM.

Chon, Y., Lane, N., Li, F., Cha, H., and Zhao, F. (2012), "Automatically Characterizing Places with Opportunistic CrowdSensing using Smartphones," in *Proc. 14th Int. Conf. Ubiquitous Computing (UbiComp'12). ACM*.

Chuang, M. and Narasimhan, P. (2010), "Automated viewer-centric personalized sports broadcast," *Procedia Engineering*, 2, 3397–3403.

Clauset, A., Newman, M., and Moore, C. (2004), "Finding community structure in very large networks," *Physical review E*, 70, 066111.

Crandall, D., Owens, A., Snavely, N., and Huttenlocher, D. (2011), "Discrete-continuous optimization for large-scale structure from motion," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 3001–3008, IEEE.

Cuellar, G., Eckles, D., and Spasojevic, M. (2008), "Photos for information: a field study of cameraphone computer vision interactions in tourism," in *CHI '08 extended abstracts on Human factors in computing systems*, CHI EA '08, pp. 3243–3248, New York, NY, USA, ACM.

Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P. (2010), "MAUI: making smartphones last longer with code offload," in *MobiSys*, pp. 49–62, ACM.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. (2004), "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the twentieth annual symposium on Computational geometry*, pp. 253–262, ACM.

Debnath, H. and Borcea, C. (2013), "TagPix: Automatic Real-time Landscape Photo Tagging For Smartphones," .

Duell, J., Hargrove, P., and Roman, E. (2005), *The design and implementation of Berkeley Lab's linuxcheckpoint/restart*, Lawrence Berkeley National Laboratory.

Efros, A., Berg, A., Mori, G., and Malik, J. (2003), "Recognizing action at a distance," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pp. 726–733, IEEE.

Feiner, S., MacIntyre, B., Höllerer, T., and Webster, A. (1997), "A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment," *Personal Technologies*, 1, 208–217.

Fischler, M. and Bolles, R. (1981), "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, 24, 381–395.

Fitzgibbon, A., Zisserman, A., et al. (1998), "Automatic 3D model acquisition and generation of new images from video sequences," in *Proceedings of European signal processing conference*, pp. 1261–1269.

Föckler, P., Zeidler, T., Brombach, B., Bruns, E., and Bimber, O. (2005), "PhoneGuide: museum guidance supported by on-device object recognition on mobile phones," in *Proceedings of the 4th international conference on Mobile and ubiquitous multimedia*, pp. 3–10, ACM.

Gammeter, S., Gassmann, A., Bossard, L., Quack, T., and Van Gool, L. (2010), "Server-side object recognition and client-side object tracking for mobile augmented reality," in *CVPR Workshops*, pp. 1–8, IEEE.

Girod, B. and et al., C. (2011), "Mobile visual search," *Signal Processing Magazine, IEEE*, 28, 61–76.

Gong, M. and Yang, Y.-H. (2005), "Near real-time reliable stereo matching using programmable graphics hardware," in *CVPR*, vol. 1, pp. 924–931, IEEE.

Grzeszczuk, R., Kosecka, J., Vedantham, R., and Hile, H. (2009), "Creating compact architectural models by geo-registering image collections," in *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pp. 1718–1725, IEEE.

Hampapur, A., Brown, L., Connell, J., Ekin, A., Haas, N., Lu, M., Merkl, H., and Pankanti, S. (2005), "Smart video surveillance: exploring the concept of multiscale spatiotemporal tracking," *Signal Processing Magazine, IEEE*, 22, 38–51.

Hays, J. and Efros, A. (2008), "IM2GPS: estimating geographic information from a single image," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8.

Heath, K., Gelfand, N., Ovsjanikov, M., Aanjaneya, M., and Guibas, L. J. (2010), "Image webs: Computing and exploiting connectivity in image collections," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 3432–3439, IEEE.

Henrysson, A. and Ollila, M. (2004), "UMAR: Ubiquitous mobile augmented reality," in *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, pp. 41–45, ACM.

Herranz, F., Muthukrishnan, K., and Langendoen, K. (2011), "Camera Pose Estimation using Particle Filters," in *Int. Conf. on Indoor Positioning and Indoor Navigation (IPIN)*, pp. 1–8.

Hightower, J. and Borriello, G. (2001), "Location systems for ubiquitous computing," *Computer*, 34, 57–66.

Hile, H., Vedantham, R., Cuellar, G., Liu, A., Gelfand, N., Grzeszczuk, R., and Borriello, G. (2008), "Landmark-based pedestrian navigation from collections of geotagged photos," in *Proceedings of the 7th International Conference on Mobile and Ubiquitous Multimedia*, pp. 145–152, ACM.

Hile, H., Liu, A., Borriello, G., Grzeszczuk, R., Vedantham, R., and Kosecka, J. (2010), "Visual Navigation for Mobile Devices," *IEEE MultiMedia*, 17, 16–25.

Hodges, S., Williams, L., Berry, E., Izadi, S., Srinivasan, J., Butler, A., Smyth, G., Kapur, N., and Wood, K. (2006), "SenseCam: A retrospective memory aid," in *UbiComp 2006: Ubiquitous Computing*, pp. 177–193, Springer.

Hu, W., Amos, B., Chen, Z., Ha, K., Richter, W., Pillai, P., Gilbert, B., Harkes, J., and Satyanarayanan, M. (2015), "The case for offload shaping," in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pp. 51–56, ACM.

Hua, Y., Xiao, B., Veeravalli, B., and Feng, D. (2012), "Locality-sensitive bloom filter for approximate membership query," *Computers, IEEE Transactions on*, 61, 817–830.

Hwangbo, M., Kim, J.-S., and Kanade, T. (2009), "Inertial-aided KLT feature tracking for a moving camera," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 1909–1916, IEEE.

Izadi, S. et al. (2011), "KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera," in *UIST*, ACM.

Jain, P. and Roy Choudhury, R. (2014), "Demo: real-time object tagging and retrieval," in *MobiSys*, pp. 346–346, ACM.

Jain, P., Manweiler, J., Acharya, A., and Beaty, K. (2013), "Focus: clustering crowdsourced videos by line-of-sight," in *SenSys*, p. 8, ACM.

Jain, P., Manweiler, J., Acharya, A., and Choudhury, R. R. (2014), "Scalable Social Analytics for Live Viral Event Prediction," in *Eighth International AAAI Conference on Weblogs and Social Media*.

Jain, P., Manweiler, J., and Roy Choudhury, R. (2015a), "OverLay: Practical Mobile Augmented Reality," in *MobiSys*, pp. 331–344, ACM.

Jain, P., Manweiler, J., and Roy Choudhury, R. (2015b), "Poster: User Location Fingerprinting at Scale," in *Proceedings of the 21st annual international conference on Mobile computing and networking*, ACM.

Kaminsky, R., Snavely, N., Seitz, S., and Szeliski, R. (2009), "Alignment of 3D point clouds to overhead images," in *Computer Vision and Pattern Recognition Workshops, 2009. CVPR Workshops 2009. IEEE Computer Society Conference on*, pp. 63–70.

Kanade, T., Collins, R., Lipton, A., Burt, P., and Wixson, L. (1998), "Advances in cooperative multi-sensor video surveillance," in *Proceedings of DARPA Image Understanding Workshop*, vol. 1, p. 2, Citeseer.

Karlsson, N., Di Bernardo, E., Ostrowski, J., Goncalves, L., Pirjanian, P., and Munich, M. E. (2005), "The vSLAM algorithm for robust localization and mapping," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 24–29, IEEE.

Kemp, R., Palmer, N., Kielmann, T., and Bal, H. (2012), "Cuckoo: a computation offloading framework for smartphones," in *Mobile Computing, Applications, and Services*, pp. 59–79, Springer.

Kim, C. and Vasudev, B. (2005), "Spatiotemporal sequence matching for efficient video copy detection," *Circuits and Systems for Video Technology, IEEE Transactions on*, 15, 127–132.

Kim, K., Yoon, S., and Cho, H. (2012), "A faster color-based clustering method for summarizing photos in smartphone," in *Computer Science and Automation Engineering (CSAE), 2012 IEEE International Conference on*, vol. 2, pp. 561–565, IEEE.

Kirsch, A. and Mitzenmacher, M. (2006), "Distance-Sensitive Bloom Filters." in *ALENEX*, vol. 6, pp. 41–50, SIAM.

Klein, G. and Murray, D. (2009), "Parallel tracking and mapping on a camera phone," in *ISMAR*, pp. 83–86, IEEE.

Koenderink, J., Van Doorn, A., et al. (1991), "Affine structure from motion," *JOSA A*, 8, 377–385.

Kopf, J., Cohen, M. F., and Szeliski, R. (2014), "First-person hyper-lapse videos," *TOG*, 33, 78.

Kumar, K. and Lu, Y.-H. (2010), "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, pp. 51–56.

LaMarca, A., Chawathe, Y., Consolvo, S., Hightower, J., Smith, I., Scott, J., Sohn, T., Howard, J., Hughes, J., Potter, F., et al. (2005), "Place lab: Device positioning using radio beacons in the wild," in *Pervasive computing*, pp. 116–133, Springer.

Lee, K., Lee, J., Yi, Y., Rhee, I., and Chong, S. (2010), "Mobile data offloading: how much can WiFi deliver?" in *Proceedings of the 6th International Conference*, p. 26, ACM.

Li, L., Hu, P., Peng, C., Shen, G., and Zhao, F. (2014), "Epsilon: A visible light based positioning system," in *NSDI*, pp. 331–343, USENIX Association.

Li, Y., Crandall, D., and Huttenlocher, D. (2009), "Landmark classification in large-scale image collections," in *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 1957 –1964.

Li, Y., Snavely, N., Huttenlocher, D., and Fua, P. (2012), "Worldwide pose estimation using 3d point clouds," in *Computer Vision–ECCV 2012*, pp. 15–29, Springer.

LiKamWa, R., Liu, Y., Lane, N. D., and Zhong, L. (2011), "Can your smartphone infer your mood," in *PhoneSense workshop*, pp. 1–5.

LiKamWa, R., Priyantha, B., Philipose, M., Zhong, L., and Bahl, P. (2013), "Energy characterization and optimization of image sensing toward continuous mobile vision," in *MobiSys*, pp. 69–82, ACM.

Lowe, D. G. (1999), "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2, pp. 1150–1157, Ieee.

Lucas, B. D., Kanade, T., et al. (1981), "An iterative image registration technique with an application to stereo vision." in *IJCAI*, vol. 81, pp. 674–679.

Lv, Q., Josephson, W., Wang, Z., Charikar, M., and Li, K. (2007), "Multi-probe LSH: efficient indexing for high-dimensional similarity search," in *VLDB*, pp. 950–961, VLDB Endowment.

Magnor, M. (2002), "Geometry-based automatic object localization and 3-d pose detection," in *Image Analysis and Interpretation, 2002. Proceedings. Fifth IEEE Southwest Symposium on*, pp. 144–147, IEEE.

Manweiler, J. G., Jain, P., and Roy Choudhury, R. (2012), "Satellites in our pockets: an object positioning system using smartphones," in *MobiSys*, pp. 211–224, ACM.

Mistry, P. and Maes, P. (2009), "SixthSense: a wearable gestural interface," in *ACM SIGGRAPH ASIA 2009 Sketches*, p. 11, ACM.

Mohan, R. (1998), "Video sequence matching," in *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, vol. 6, pp. 3697–3700, IEEE.

Moon, S., Moon, I., and Yi, K. (2009), "Design, tuning, and evaluation of a full-range adaptive cruise control system with collision avoidance," *Control Engineering Practice*, 17, 442–455.

Muja, M. and Lowe, D. G. (2009), "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration." in *VISAPP (1)*, pp. 331–340.

Nister, D. and Stewenius, H. (2006), "Scalable recognition with a vocabulary tree," in *CVPR*, vol. 2, pp. 2161–2168, IEEE.

Pence, H. E. (2010), "Smartphones, smart objects, and augmented reality," *The Reference Librarian*, 52, 136–145.

Piekarski, W. and Thomas, B. (2002), "ARQuake: the outdoor augmented reality gaming system," *Communications of the ACM*, 45, 36–38.

Qin, C., Bao, X., Roy Choudhury, R., and Nelakuditi, S. (2011), "TagSense: a smartphone-based approach to automatic image tagging," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, MobiSys '11, pp. 1–14, New York, NY, USA, ACM.

Reddy, S., Parker, A., Hyman, J., Burke, J., Estrin, D., and Hansen, M. (2007), "Image browsing, processing, and clustering for participatory sensing: lessons from a DietSense prototype," in *Proceedings of the 4th workshop on Embedded networked sensors*, pp. 13–17, ACM.

Rekimoto, J. and Ayatsuka, Y. (2000), "CyberCode: designing augmented reality environments with visual tags," in *Proceedings of DARE 2000 on Designing augmented reality environments*, pp. 1–10, ACM.

Rolland, J. P., Davis, L., and Baillot, Y. (2001), "A survey of tracking technology for virtual environments," *Fundamentals of wearable computers and augmented reality*, 1, 67–112.

Rosten, E. and Drummond, T. (2006), "Machine learning for high-speed corner detection," in *European Conference on Computer Vision (ECCV)*, pp. 430–443.

Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011), "ORB: an efficient alternative to SIFT or SURF," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2564–2571, IEEE.

Sand, P. and Teller, S. (2004), "Video matching," in *ACM Transactions on Graphics (TOG)*, vol. 23, pp. 592–599, ACM.

Sattler, T., Leibe, B., and Kobbelt, L. (2011), "Fast image-based localization using direct 2D-to-3D matching," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 667–674, IEEE.

Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. (2009), "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, 8, 14–23.

Schaffalitzky, F. and Zisserman, A. (2002), "Multi-view matching for unordered image sets, or "How do I organize my holiday snaps?"," in *Computer Vision—ECCV 2002*, pp. 414–431, Springer.

Schall, G., Schöning, J., Paelke, V., and Gartner, G. (2011), "A survey on augmented maps and environments: approaches, interactions and applications," *Advances in Web-based GIS, Mapping Services and Applications*, 9, 207.

Schleicher, R., Shirazi, A., Rohs, M., Kratz, S., and Schmidt, A. (2011), "WorldCupinion Experiences with an Android App for Real-Time Opinion Sharing During Soccer World Cup Games," *International Journal of Mobile Human Computer Interaction (IJMHCI)*, 3, 18–35.

Sen, S., Lee, J., Kim, K.-H., and Congdon, P. (2013), "Avoiding multipath to revive inbuilding wifi localization," in *MobiSys*, pp. 249–262, ACM.

Shen, Z., Arslan Ay, S., Kim, S. H., and Zimmermann, R. (2011), "Automatic tag generation and ranking for sensor-rich outdoor videos," in *MM*, pp. 93–102, ACM.

Silpa-Anan, C. and Hartley, R. (2008), "Optimised KD-trees for fast image descriptor matching," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8, IEEE.

Snavely, N., Seitz, S. M., and Szeliski, R. (2006), "Photo tourism: exploring photo collections in 3D," in *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pp. 835–846, New York, NY, USA, ACM.

Storn, R. and Price, K. (1997), "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, 11, 341–359.

Takacs, G. and et al., C. (2008), "Outdoors augmented reality on mobile phone using loxel-based visual feature organization," in *Proceedings of the 1st ACM international conference on Multimedia information retrieval*, pp. 427–434, ACM.

Tenmoku, R., Kanbara, M., and Yokoya, N. (2003), "A wearable augmented reality system using positioning infrastructures and a pedometer," in *ISWC*, pp. 110–110, IEEE Computer Society.

Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W. (2000), "Bundle adjustment—a modern synthesis," in *Vision algorithms: theory and practice*, pp. 298–372, Springer.

Tuite, K., Snavely, N., Hsiao, D., Tabing, N., and Popovic, Z. (2011), "PhotoCity: training experts at large-scale image acquisition through a competitive game," in *Proceedings of the 2011 annual conference on Human factors in computing systems*, pp. 1383–1392, ACM.

Wagner, D. and Schmalstieg, D. (2003), "First steps towards handheld augmented reality," in *ISWC*, pp. 127–127, IEEE Computer Society.

Wagner, D. and Schmalstieg, D. (2009), "Making augmented reality practical on mobile phones, part 1," *Computer Graphics and Applications, IEEE*, 29, 12–15.

Wagner, D., Reitmayr, G., Mulloni, A., Drummond, T., and Schmalstieg, D. (2008), "Pose tracking from natural features on mobile phones," in *ISMAR*, pp. 125–134, IEEE Computer Society.

Wang, H., Sen, S., Elgohary, A., Farid, M., Youssef, M., and Choudhury, R. R. (2012), "No need to war-drive: unsupervised indoor localization," in *MobiSys*, pp. 197–210, ACM.

Yan, T., Ganesan, D., and Manmatha, R. (2008), "Distributed image search in camera sensor networks," in *SenSys*, pp. 155–168, ACM.

Yan, T., Kumar, V., and Ganesan, D. (2010), "CrowdSearch: exploiting crowds for accurate real-time image search on mobile phones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pp. 77–90, New York, NY, USA, ACM.

You, S., Neumann, U., and Azuma, R. (1999), "Hybrid inertial and vision tracking for augmented reality registration," in *Virtual Reality, 1999. Proceedings., IEEE*, pp. 260–267, IEEE.

Zhang, Z., Deriche, R., Faugeras, O., and Luong, Q.-T. (1995), "A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry," *Artificial intelligence*, 78, 87–119.

Zhao, T. and Nevatia, R. (2004), "Tracking multiple humans in complex situations," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26, 1208–1221.

Zhao, W., Nister, D., and Hsu, S. (2005), "Alignment of continuous video onto 3D point clouds," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27, 1305–1318.

Zheng, Y.-T., Zhao, M., Song, Y., Adam, H., Buddemeier, U., Bissacco, A., Brucher, F., Chua, T.-S., and Neven, H. (2009), "Tour the world: Building a web-scale landmark recognition engine," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 1085–1092.

Zhou, F., Duh, H. B.-L., and Billinghurst, M. (2008), "Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR," in *ISMAR*, pp. 193–202, IEEE Computer Society.

Zhu, Z., Xu, G., Riseman, E., and Hanson, A. (1999), "Fast generation of dynamic and multi-resolution 360 panorama from video sequences," in *Multimedia Computing and Systems, 1999. IEEE International Conference on*, vol. 1, pp. 400–406, IEEE.

Biography

Puneet Jain was born on December 17, 1986 in Bandakpur, Madhya Pradesh, India. In May 2009, he earned the M.Tech and B.Tech degrees in Computer Science from Indian Institute of Technology, located in Kharagpur, West Bengal, India. He completed his Masters of Science degree in Computer Science in May 2013 from Duke University. He also earned his Doctor of Philosophy degree in Computer Science from Duke University in August 2015 with thesis entitled, "Practical Architectures for Fused Visual and Inertial Mobile Sensing", under the supervision of Romit Roy Choudhury. Puneet's dissertation has resulted in several conference and workshop publications Manweiler et al. (2012); Jain et al. (2013); Jain and Roy Choudhury (2014); Jain et al. (2014, 2015a,b). Prior to starting his Ph.D., Puneet was a Senior Member of Technical Staff at Oracle Corporation and Adobe Systems, Bangalore, India. In 2012 and 2013, Puneet interned at IBM T.J. Watson Research, Yorktown Heights. His work titled "Practical Mobile Augmented Reality" won best demo award at HotMobile 2015 conference. In August 2015, Puneet will be joining Hewlett Packard Labs in Palo Alto, California as a Researcher.