Title

Vision-based Appliance Identification and Control with Smartphone Sensors in Commercial Buildings

Author

Chen, Kaifei

Publication Date

2018

Vision-based Appliance Identification and Control with Smartphone Sensors in Commercial Buildings

by

Kaifei Chen

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Randy Katz, Co-chair
Professor David Culler, Co-chair
Professor Edward Arens

Summer 2018

Vision-based Appliance Identification and Control with Smartphone Sensors in Commercial Buildings

Abstract

Vision-based Appliance Identification and Control with Smartphone Sensors in Commercial Buildings

by

Kaifei Chen

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Randy Katz, Co-chair

Professor David Culler, Co-chair

Appliances in commercial buildings are connected to the Internet and becoming programmatically controllable. However, as the number of smart appliances increases, identifying and controlling one instance among thousands in a building becomes challenging. Existing methods have various problems when deployed in large commercial buildings. For example, proprietary remote controllers and smartphone apps become unmanageable. Voice or gesture command assistants require users to memorize many control commands in advance. Attaching visual markers (e.g., QR codes) to appliances introduces considerable deployment overhead and cannot work at a distance.

In this dissertation, we introduce new approaches for easier appliance selection and interaction. We first study how several different indoor localization approaches can be used to generate a list of nearby appliances for users to choose from. Wi-Fi signal strength fingerprinting can provide a rough estimation with an error of 10 meters. It can also be fused with different types of information, such as acoustic background noise. However, indoor localization can only reduce the displayed appliance list and is not sufficient to provide a quick and intuitive appliance selection mechanism.

In comparison, identifying an appliance by simply pointing a smartphone camera and controlling the appliance using a graphical overlay interface is more intuitive. We introduce SnapLink, a responsive and accurate vision-based system for mobile appliance identification and interaction using image localization. Compared to the image retrieval approaches used in previous vision-based appliance control systems, SnapLink exploits 3D models to improve identification accuracy and reduce deployment overhead via quick video captures and a simplified labeling process. To evaluate SnapLink, we collected training videos from 39 rooms to represent the scale of a modern commercial building. It achieves a 94% successful appliance identification rate among 1526 test images of 179 appliances within 120 ms average server processing time. Furthermore, we show that

SnapLink is robust to viewing angle and distance differences, illumination changes, as well as daily changes in the environment.

On top of SnapLink, we build MARVEL (Mobile Augmented Reality with Viable Energy and Latency) to provide a continuous appliance identification and interaction experience. MARVEL identifies appliances with imperceptible latency (~100 ms) and low energy consumption on regular mobile devices. In contrast to conventional MAR systems, which recognize objects using image-based computations performed in the cloud, MARVEL mainly utilizes a mobile device's local inertial sensors for recognizing and tracking multiple objects, while computing local optical flow and offloading images only when necessary. We propose a novel system architecture which uses local inertial tracking, local optical flow, and visual tracking in the cloud synergistically. On top of that, we investigate how to minimize the overhead for image computation and offloading. We have implemented and deployed a holistic prototype system in a commercial building and extensively evaluate MARVEL's performance. It reveals that the efficient use of a mobile device's capabilities significantly lowers latency and energy consumption without sacrificing accuracy.

To my family, friends, colleagues, and advisors

Contents

List of Figures

List of Tables

Acknowledgments

The years I spent in Berkeley were invaluable to my life. I cannot express my gratitude enough to my advisor, Professor Randy Katz, for his advise and support to my work, as well as his enlightenment and encouragement when I struggled. Professor David Culler was a great advisor and provided guidance at many stages of my PhD. I would also like to thank Professor Avideh Zakhor and Professor Edward Arens, who were both in my qualifying exam committee. Professor Zakhor provided many insightful items of feedback on my work, especially on details of the computer vision algorithms, without which I would have spent countless nights trying different approaches. Professor Arens helped a lot on potential directions and applications where my work can be applied.

None of my work can be accomplished without the help from all my colleagues in the group. Jonathan Fürst was a good friend and close collaborator on many different projects. Many in-depth discussions between us made it possible for me to move forward with my research. Hyung-Sin Kim was a great mentor who guided me with many useful good practices on conducting research. Michael Anderson, Gabe Fierro, Jack Kolb, Sam Kumar, and Kalyanaraman Shankari all collaborated with me on several projects, from whom I've learned a lot. In addition, I've mentored several undergraduate students: Beidi Chen, Takeshi Mochida, Siyuan He, and Tong Li. They contributed much to our research by providing both good engineering efforts and inspiring ideas. They also helped me learn a lot on providing good mentorship. Colleagues who have overlap with me in my first several years, including Arka Bhattacharya, Stephen Dawson-Haggerty, Andrew Krioukov, Prashanth Mohan, Jorge Ortiz, and Jay Taneja, helped me to get familiar with the group and get started with my first research projects.

Many of my friends in Berkeley not only shared good memories in life with me, but also helped me on my research with many discussions. I want to thank Xiang Gao, Dezhi Hong, Xiaohe Hu, Xin Jin, Gautam Kumar, Zhi Liu, Aishwarya Parasuram, Qifan Pu, Dimin Xu, Neeraja Yadwadkar, Jiao Zhang, and Ben Zhang for being with me and learn with me.

Many staff members in Berkeley made a lot things possible for me to conduct research, including Kattt Atchley, Domenico Caramagno, Carlyn Chinen, Albert Goto, Jon Kuroda, and Boban Zarkovich. Albert helped me in many aspects of my work. He was always meticulous in preparing us for conferences or retreats, such as setting up demos, or printing and cutting posters late at night. He also helped collected a large volume of data in the SnapLink evaluation. Albert is also a great friend, and we've spent countless late nights chatting in the lab.

I also want to thank people who gave me opportunities to work on several research projects prior joining UC Berkeley. Professor Pei Zhang from Carnegie Mellon University invited me to visit his lab and join research projects while I was an undergraduate student. I worked with Shijia Pan, Aveek Purohit, and Zheng Sun on several very interesting projects on indoor localization and sensor networks. Dr. Chieh-Jan Mike Liang and Dr. Xiaofan Jiang accepted me as an intern in Microsoft Research Asia, where I enjoyed my life, learned and worked a lot, and made a lot of good friends. These experiences gave me a good understanding of research, as well as the determination to pursue a PhD myself.

This work is supported in part by the National Science Foundation under grant CPS-1239552 (SDB), California Energy Commission, and gifts from Qualcomm Inc. and Intel Corp.

Last but not least, I want to deliver special thanks to my loved ones. My girlfriend, Yuwen Dai, has been very supportive in all years before and during my PhD. My father, Xinde Chen, and mother, Weigui Li, provided me a good environment to grow and study along the way, and gave me freedom and courage to face many challenges. My accomplishment cannot be made without their support.

Chapter 1

Introduction

The number of programmatically controllable appliances is increasing rapidly in both commercial and residential buildings. Directly controlling or interacting with them has become challenging for building occupants, especially in commercial buildings, where appliances are dense and shared. In this chapter, we recognize the trends and the problems with appliance identification and control, and lay out the roadmap for this dissertation.

1.1 Trends in Smart Appliances

We first categorize and describe different types of smart appliances in this Section. Then we discuss the trends of the number and complexity of the appliances, as well as the complexity of the building infrastructures that integrate all the appliances.

1.1.1 Types of Smart Appliances

Many appliances in today's buildings are gradually being replaced by "smart" ones that are connected to the Internet and programmatically controllable with manufacturer-provided APIs. For example, smart thermostats learn occupant activity patterns using motion sensors and adjust temperature settings accordingly to reduce energy consumption. Smart light bulbs can be controlled using smartphone apps to change color and brightness, and be scheduled to turn on/off at a later time.

These appliances come with a on-board processor that runs a server on top of TCP/IP (e.g., a HTTP server that receives JSON objects in POST requests) and act based on received requests. Under the network layer, some have a direct Ethernet connection, and others communicate via low power protocols (e.g., Bluetooth Low Energy (BLE), ZigBee, Z-Wave) to a dedicated relay, a.k.a. a hub, that is plugged in and connected to the Internet. The manufacturers usually provide documentations of APIs for developers to integrate the appliance to third party applications.

In addition to everyday appliances that are easy and affordable to replace (e.g., printers, lights), buildings also have many internal components that are not frequently replaced, such as Variable Air Volume (VAV) boxes, air handling units (AHU), damper, water chiller and heater, and different types of sensors (e.g., air flow, CO2, temperature). Since decades ago, these components have been designed to be connected and controlled programmatically, especially in commercial buildings. For example, many of them are controlled by a Programmable Logic Controller (PLC), and communicate using BACNet. Controlling these devices requires knowledge of not only the proprietary programming languages the PLC supports, but also the physical configuration of the building, which are usually not well documented and obscure. In recent years, the development of computer networks (e.g., Ethernet, TCP/IP) and programming languages (e.g., Python) over the last decades have been applied to commercial buildings [1], allowing these conventional components to be controlled synergistically with modern smart appliances. This is important because many applications, such as personalized HVAC configuration, require direct control of these internal components in the building.

1.1.2 Increasing Number and Complexity

According to Statista [2], the number of smart home devices is predicted to increase from 370 million worldwide in 2018 to 913 million in 2025. Many useful but sophisticated building applications, which usually require a good infiltration of smart appliances in the buildings, also incentivize people to deploy more smart appliances, especially after successfully making some buildings more energy efficient, comfortable, and occupant friendly. Examples of such applications include personalized comfort control [3], anomaly detection [4] for building analytics, and demand response [5] for energy efficiency.

In addition to the increasing number of smart appliances, the complexity of each device is increasing. This is enabled by richer and more flexible APIs, as well as today's ubiquitous connected personal devices, such as smartphones. Thermostats and light bulbs are among those that have been providing sophisticated functions. Their smartphone control applications add many possibilities in how they can be controlled, such as setting up complex heating, cooling, and lighting schedules based on occupancy in different zones at different times. In recent years and the near future, we have seen and will see many more devices augmented with new functionality. For example, door locks can be activated and deactivated using a smartphone, and the ability to control can be delegated to other people for certain period of time [6]. Security cameras can recognize faces and send notifications with a scene description [7]. However, while the complexity adds more possibilities, it also imposes more requirements on the systems that manage the devices.

1.1.3 Building Infrastructures

To allow many applications to use all the smart appliances, we must integrate them and provide a uniform and safe execution environment, especially when the appliances are shared and applications are not fully trusted. In commercial buildings, building management systems (BMS) have been used to control all devices at a central computer. Balaji, et al. [8] shows that a commercial building already has thousands of sensors and control points today. With the increasing number and complexity of new devices, existing building management systems will not be able to handle the volume of data, provide security guarantee, or adapt to different device vendors. To mitigate these problems, new building management systems have been proposed, abstracting hardware and services in buildings to a new standardized high-level programming interface [1, 8]. They are to building appliances what traditional operating systems are to computer hardware, providing an application execution environment while protecting hardware from faulty applications.

1.2 Motivating Application: Direct Appliance Control

While many building applications can be automated with a modern BMS, building occupants still need to directly control appliances in many cases. For example, a user may want to connect a projector to a particular laptop she has, or change the temperature or light brightness to a personally preferred value in a room. However, it is impractical for an application to automatically and unambiguously detect the user's attention without any action from her.

Conventionally, hardware controllers (e.g., light switches, remote controllers) are used to interact with appliances. These controllers cannot scale as the density of appliances in buildings increases, or be extensible when new functions are desired. In the era of smart appliances, many manufacturers now provide a mobile application or website for direct control. However, each manufacturer has their own proprietary silo of system architecture, including the control app itself. To control a particular appliance, a building occupant has to find the correct app on her smartphone among possibly tens of others to control an appliance, which is tedious and cumbersome.

This problem has been mitigated by smart voice assistant in residential buildings, such as Amazon Echo [9] and Google Home [10]. Users can configure names and associate voice control commands with manufacturer-defined functions, such as turning on/off lights or setting temperature. However, many appliances in commercial buildings are shared and occupants cannot memorize names for thousands of appliances or their voice control commands, especially for people who are in the building for only a short period of time (e.g., in a meeting).

In commercial buildings, we can potentially build an application that allows occupants to select and control all appliances in a single application or website. This eliminates the need of having one app for each manufacturer, but is still challenging for occupants to identify the exact appliance instance among thousands effortlessly. Because the user may not know exactly where the appliance is, what its type is named in the BMS, and how to find it using the app (i.e., what to search for given the location and name).

In this dissertation, we aim to solve the aforementioned problems for our target application: direct appliance control. We aim to build a system that must meet the following requirements:

- the system must allow users to conveniently, quickly, and unambiguously identify any appliance they want to control among many in the building.
- the system must be robust under different circumstances, such as time of day, network condition, user behavior, and change of environment.
- the system must be able to scale in both identification. accuracy and latency as the number and complexity of appliances continue to increase in the future.
- the system must be able to adapt to future developments in building management systems and smart appliances.

1.3 Dissertation Roadmap

The rest of this dissertation is organized as follows. Chapter 2 introduces existing approaches for users to identify and control smart appliances in related work. We conclude that existing approaches cannot be directly used for our application in commercial buildings.

Chapter 3 describes our first attempt to use indoor localization to narrow down the displayed list of appliances to only the nearby ones. By adding algorithm-aware confidence values to our system, we can select the best localization results among two popular indoor localization algorithms under different circumstances. However, even though it can help us reduce the appliance list, we still cannot disambiguate appliances of the same type that are close to the user.

Chapter 4 starts to solve the problem using computer vision. We introduce SnapLink, a vision-based appliance identification system that allows users to start controlling an appliance by taking a picture of it using their smartphones. We compare different computer vision algorithms and decide 3D image localization is the best fit for our purpose. To improve the response time without sacrificing image localization accuracy, we add a feature sub-sampling step in the computation pipeline. We also conduct a comprehensive evaluation to show that SnapLink is scalable with our large data set. In addition, it is robust with different illumination conditions, different angles and distances of the image view, as well as everyday changes in shared environments.

In Chapter 5, we remove the requirement of image capturing in SnapLink and build MARVEL (Mobile Augmented Reality with Viable Energy and Latency). MARVEL is an augmented reality application that displays annotation (as opposed to rendering 3D objects) built on top of SnapLink, and provides real-time continuous augmented display of controllable appliances in the camera view. It allows users to explore the environment without prior knowledge of what are controllable. To make MARVEL responsive and energy efficient, we explore the trade-offs for both latency and energy consumption between local motion inertial tracking and correction, and vision-based tracking in the cloud. We show that MARVEL can achieve high annotation display accuracy with only 100ms latency, which is the limitation of the Android operating system in the smartphone we use.

Chapter 6 concludes this dissertation with several lessons learned, talks about other potential applications using our systems, and discusses several problems to solve and features to add in future work.

Chapter 2

Background

Many research efforts have been focused on providing more convenient ways for occupants to directly interact with appliances. Some display a subset of all appliances based on the estimated location of the user. This approach still cannot disambiguate appliances of the same type. Others use different sensors and actuators to assist users to select appliances, some with even feedback to users. These approaches either require significant deployment overhead, or are not suitable for commercial buildings. In this chapter, we describe these approaches and conclude that we need more investigation into some approaches and potentially a better approach.

2.1 Building Management Systems

Most appliance manufacturers have their own propriety protocols and applications to control their devices today. To build a centralized control application for all appliances, the BMS is the ideal platform to send control commands. Modern commercial buildings contains thousands of sensors and actuators distributed across the building, such as temperature sensors, heating coils, and ventilation dampers. These devices communicate with a centralized BMS over various communication protocols (e.g., BACNet/IP, 6LoWPAN) and follow different data schema as they are deployed by different vendors. In addition to these built-in devices, appliances are being integrated into the BMS as well. Applications access these building components through the BMS to provide various functions, such as demand response, HVAC scheduling, safety monitoring, and our application — human-appliance interactions.

Today's smart buildings aim to build a BMS with unified interface for these heterogeneous devices to form an easily programmable building platform. Figure 2.1 shows an example smart building architecture proposed in BOSS (Building Operating System Services) [1]. BOSS runs sMAP (Simple Measuring and Actuation Profile) [11] drivers as a Hardware Representation Layer to abstract different hardware and network protocols with *points*, which expose a read and write interface for a time series data stream. A Hard-

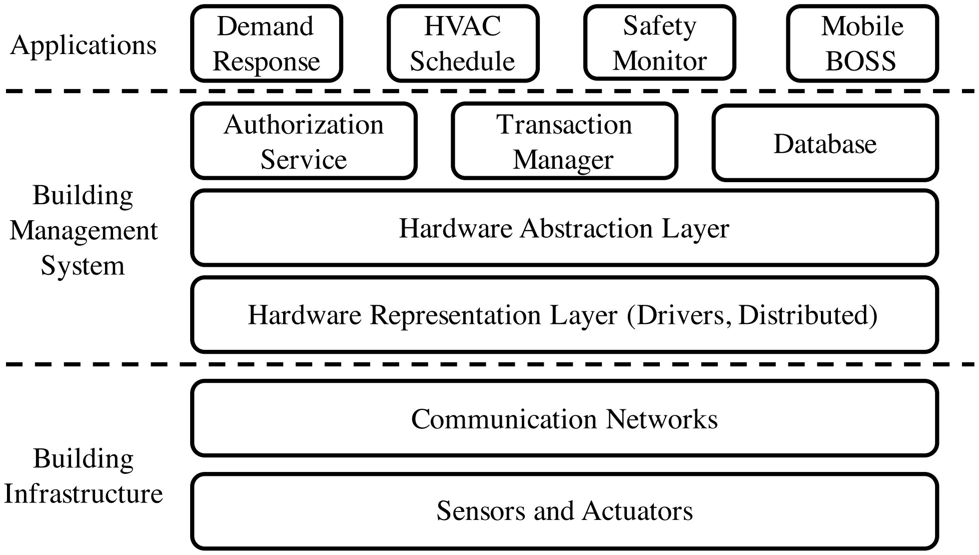


Figure 2.1: Example Smart Building Architecture. Building infrastructure includes existing and new deployment of sensors and actuators. The Building Management System abstracts them at different layers and provide necessary services for protection. Applications implement logic like energy consumption optimization and smart appliance control.

ware Abstraction Layer builds relationships (e.g., spacial, electrical) between points and exposes higher level interfaces. For example, changing temperature involves controlling multiple points (e.g., chillers, heating coil, damper) in a specific feedback loop. The Authentication Service handles access permissions from applications based on their function and developers. The Transaction manager guarantees the consistency of building states because multiple states can be modified by one operation or by multiple applications. A database stores all history time series data for applications.

2.2 Indoor Localization

Location information can be used to filter out appliances that are far from the user, assuming she is only interested in controlling those nearby. This is especially true in an open environment, such as a commercial building, where appliances are shared by many occupants. Therefore, localizing the user in the building will be extremely helpful to simplify the appliance identification process. However, GPS is not suitable for indoor scenarios because buildings usually block GPS signal [12].

2.2.1 Fingerprinting

One popular approach to address indoor localization is the use of fingerprints. Any measurable information that is dependent on location and also consistent over time with environmental changes can be used as fingerprints. Some fingerprints can be obtained from universal existing infrastructures, such as Wi-Fi received signal strength (RSS) [13] or acoustic background features [14]. Others require extra deployments, such as ultrasound beacons [15], Bluetooth beacons [16], and magnetic beacons [17].

Fingerprinting-based localization systems require a site survey to build a database that annotates ground truth locations with collected fingerprints. Most surveys today are performed manually, but others have looked at automating this process using extra localization inference [18, 19]. To perform a location estimation, an occupant needs to collect a new fingerprint and ask the system to find one or more nearest fingerprints in the database to infer her location. These approaches can have different performance depending on the fingerprints itself, the environment, and quality of the database. We investigate more about how much fingerprinting-based indoor localization can help in reducing the displayed appliance list in Section 3.

Some other work deploy beacons to broadcast zone-based fingerprints. The challenge is to allow signal reception and zone detection without ambiguity, which can be caused by interference or a weak signal. Fürst, et al. [15] deploys a ultrasonic beacon in every isolated room, using the building walls to perfectly eliminate interference. Jiang, et al. [17] creates virtual zones with sharp boundaries using a magnetic signal, which is mostly unaffected by objects in the environment, such as a human body or metal.

2.2.2 Multilateration and Multiangulation

Multilateration localizes people using measured distances from no less than three fixed stations (signal beacons or receivers). It finds the intersection point of the circles centered at each station with the distances as the radii. This approach works best in open spaces when Line-of-Sight (LoS) distance can be directly measured, which is how GPS works in outdoor environments. Time-of-Flight (ToF) of signal is one of the most intuitive ways to measure distance, but it requires accurate time synchronization between transmitters and receivers, which is challenging. Cortina [20] uses Round-trip Time-of-Flight (RToF) to estimate distance. Cricket [21] uses the Time Difference of Arrival (TDoA) of RF and ultrasonic signal sent from beacons for distance measurement. BeepBeep [22] uses two acoustic chirps sent from two commodity smartphones back to back to derive the distance without the need of accurate time synchronization.

In addition to ToF, received signal strength (RSS) can also be used to measure LoS distance using the propagation model. However, RF signal strength can be influenced by many immeasurable factors in the environment, such as multi-path effects, human body absorption, and background noise, and therefore cannot be used for accurate distance measurement [23]. Compared to RF signal, visible light has much fewer multi-path effects and partial absorption thanks to its short wavelengths. There is also less interference because LED can be modulated. Therefore, visible light can be better modeled using the Lambertian radiation pattern [24], which can be used to derive the distance based on both the transmission power and RSS. Li, et al. [25] shows that an LED can be used to measure distance with sub-meter errors in most cases.

Both ToF- and RSS-based LoS measurements require an open space, which may not exist in many indoor environments because of walls and furniture. In many cases, the signal propagates between transmitter and receiver on a Non-Line-of-Sight (NLoS) path, which can include the attenuated signal on the LoS path in some literature [26]. Using an Ultra Wideband (UWB) signal, we are usually able to filter out noises and find the smallest NLoS distance as a good approximation to the LoS distance [26].

Sometimes, time synchronization is feasible among stations, either using wired or wireless signal. In these situations, it is possible to obtain Time-Difference-of-Arrival (TDoA) between the target and every two pairs of stations, allowing us to draw several hyperbolas with the stations as the foci. The target will be the intersection point of these hyperbolas. Acoustic signals [27, 28] are usually used to obtain TDoA, because they propagates more slowly compared to an RF signal, which imposes a loose accuracy requirement for the time synchronization.

Similar to multilateration, multiangulation finds the intersection point based on the Angle-of-Arrival (AoA) of the signal between fixed beacons and the target. Different approaches can be used to detect the AoA between a transmitter and a receiver, such as a microphone array [29] or RF antenna array [30].

2.2.3 Sensor Fusion

Instead of using a single indoor localization algorithm or source of information, researchers also combine multiple sources using various statistical models. The Bayesian Filter [31] is one of the most popular models. It computes the possibility of the target at each location based on the different sensor readings (or estimated locations from different algorithms), as well as the previous location's possibility distribution. This requires us to know the relationship between the ground-truth location and sensor readings, which is not specified in a Bayesian Filter. There are different implementations to represent these relationships. One of the most popular variant is the Kalman Filter, which models the location possibility distribution as a Gaussian distribution. It assumes a linear relationship between ground-truth location and sensor readings. The Extended Kalman Filter generalizes to non-linear relationships using Taylor expansions. When the location probability distribution is not unimodal, a Particle Filter can be used to better model the location possibility distribution [18]. Instead of computing the distribution in the continuous space, a Particle Filter computes the possibilities at discrete sample locations, where locations with higher possibility in the previous distribution are more likely to be sampled.

2.3 Direct Appliance Identification

Reducing the displayed appliance list can mitigate our problem, but still suffer when there are multiple appliances of the same type are presented nearby, which is very common in commercial buildings (e.g., lights, projectors, etc.). In addition to using location information, many other work have looked at how to make direct appliance identification more convenient for users.

2.3.1 Sensor-based Approaches

Various sensors have been used to help users to identify an appliance. Traditional remote controllers transmit infrared control signal to appliances (e.g., TV, air conditioner), which have built-in infrared receivers connected to their internal control logic. To extend this to appliances without infrared receivers, some research studies install extra infrastructure on them, such as laser receivers [32], or infrared receivers [33], Bluetooth dongles [15], and NFC or RFID tags [34]. As the number of appliances increases, traditional remote controllers will simply incur significant usage overhead and cannot benefit from the flexibility and extensibility of modern BMS.

With more smart appliances connected to the BMS, researchers have built apps for users to browse appliances, either in an inventory list [11] or on a 2D/3D-graphic map of a building [35]. However, finding an appliance quickly from thousands requires a user to be aware of her current location and orientation on a large building map. As we discussed in the previous Section, indoor localization is used to mitigate this problem [15].

In addition to displaying, some works allow users to input query statements [8, 11, 36], which requires manual typing as well as adequate background knowledge of the building. Voice- and gesture-based appliance identification systems [37, 38] allow users to speak a statement or perform a gesture as a control command. They have been successfully applied in many residential buildings [9]. However, because they require users to memorize a unique voice command (or gesture) for every single appliance and control command, they cannot be deployed in a commercial building, which generally has hundreds to thousands of appliances and is shared by occupants who may not be familiar with the building.

2.3.2 Vision-based Approaches

Instead of utilizing or deploying sensors in the building, researchers have also looked at using computer vision to recognize appliance images. Users can take an image of an appliance and query for its appliance ID with the image, which is usually offloaded to a server because of high computation complexity of most computer vision algorithms. Note that we need to recognize not only the type of the appliance in the query image, but also which exact instance of appliance it is in the building because eventually we need to send control commands directly targeting one specific appliance instance.

One way to achieve this is to use image retrieval, which finds the most similar image to the query image among a set of pre-collected images. Different features extracted from an image can be used to compute similarity, such as SIFT [39] and SURF [40]. However, to use image retrieval, a database of reference images of all appliances is needed, preferably from different angles and distances, covering different parts of the background of the appliance. Collecting, labeling, and updating the reference images can involve significant manual effort.

Instead of using image retrieval, we can also localize the image in the building using the geometry relationship between image features. This requires a 3D model being built out of one short video of the appliance, which take significantly less effort to do. More details about the comparison will be discussed in Section 4.2.3.

2.4 Summary

In this chapter, we described different layers of abstraction in modern Building Management Systems (BMS) that run commercial buildings today. Instead of using proprietary APIs provided by different manufactures, our appliance control application should programmatically interact with appliances on top of the BMS.

One way to mitigate the problem of overwhelming appliances is to use indoor localization to reduce the displayed appliance list. Several ways of approaching indoor localization have been extensively studied, including fingerprinting, multilateration/multiangulation, and sensor fusion. We evaluate how much indoor localization can help our application in Chapter 3.

When there are multiple appliances of the same type presented to a user, we still need to display all of them even if her location is known. However, there is no easy and unambiguous way to differentiate between them. We discussed how sensors and computer vision can be used to directly help users to identify appliances, either using a signal transmitter/receiver or an image of the appliance. We explore more of the computer vision approach in Chapter 4, and how it can be used together with IMU-based indoor localization for a better user experience.

Chapter 3

Location-based Appliance Display

In this chapter, we discuss our first attempt to reduce the appliance list to those within vicinity of the user. This potentially reduces the size of the list from thousands to hundreds or tens, and makes browsing and finding the target appliance less tedious. We implemented two popular indoor localization systems and fused their results based on algorithm-generated confidence values. Our results show that indoor localization can greatly help to reduce candidate appliances, but cannot resolve all problems we are facing. This work was done in collaboration with Karthik Vadde at UC Berkeley [41].

3.1 Introduction

We start with the assumption that users usually control only the appliances they are utilizing, especially in a shared environment like commercial buildings. Knowing the user's location will allow us to reduce displayed appliance list, which in turn makes appliance selection easier for the user.

To achieve this location-based appliance listing, our system needs to know the location of the user (or her smartphone) inside the building. However, unlike outdoor environments, where GPS [42] can be used to calculate a rough location using triangulation from satellites, indoor environments usually have undetectable GPS signals, attenuated by building materials [12]. Fortunately, researchers have been working on indoor localization solutions for years.

As we discussed in Chapter 2, one group of the solutions are based on the idea of fingerprinting, which are essentially sensor measurements of the physical environment that are unique at different locations. Examples include WiFi RSSI from all access points in the building [13], FM radio signal features (e.g., RSSI, SNR, multi-path) [43], acoustic background spectrum [14]. Some of the solutions require extra infrastructures as beacons in the building, such as ultrasound [21, 44], infrared [45], magnetic induction [17], RFID [46], and Doppler effect on radio signals [47]. Moreover, sensors available on smartphones stimulate researchers to bring the idea of dead reckoning into trajectory estimation and localization [18, 48]. Basically, they use IMU embedded in commercial smartphones (e.g. accelerometer, gyroscope, compass, magnetometers) to estimate the velocity and direction of users, and in turn estimate the trajectory and the location given the known start point.

However, none of the state-of-the-art indoor localization techniques can be generally applied to all buildings. Different systems operate under different assumptions, which in turn would be their limitations in real-world deployments. As examples, IMU tracking requires small sensor measurement error. Fingerprints need to be both unique and consistent at a location over time. Certain specialized fingerprints can only be used in specific scenarios, such as wall color distribution [49] in a shopping mall.

To use indoor localization for our application, we need to eliminate these limitations and combine various techniques. The basic idea is that they don't always fail at the same time. Liu et al. [43] found it very likely that at least one of WiFi fingerprint and FM fingerprint are consistent at one location. In addition, we can usually infer the fidelity of estimated location. For example, abnormal movements (e.g., high speed, move through physical objects) measured from IMU can imply erroneous tracking.

We build BearLoc [50], a framework to combine different indoor localization solutions to provide a location service with high fidelity. Using BearLoc, we implemented two indoor localization algorithms, one using WiFi RSSI-based fingerprinting and the other using Acoustic Background Spectrum (ABS) fingerprinting [14]. Both of them compute the confidence of every localization estimation. We also use a linear weighted average to combine their results. Based on the fingerprint database we build for a university building, we evaluate our confidence algorithms and framework with three trajectories, along which both WiFi and ABS fingerprints are sampled. The results show that confidence is an effective way to combine results while eliminating inaccurate estimations of both WiFi and ABS.

The rest this chapter is organized as follows: in Section 3.2, we give an overview on related work in indoor localization. Section 3.3 describes our system architecture. We discuss the ideas of calculating confidences with detailed examples in WiFi and ABS in Section 3.4, and data fusion methods to combine multiple sources of estimated location in Section 3.5. The evaluations are described and analyzed in Section 3.6. We summarize this chapter in Section 3.7.

3.2 Related Work

Much work has been done to combine multiple indoor localization systems. Some of these efforts studied how different sources of localization information can be combined for a more accurate estimation. Azizyan, et al. [49] take WiFi, sound, light, and color features from smartphone sensors as signatures, and narrow down possible positions to one using one or more signatures sequentially. The accuracy improvement using this cascade approach increases as the number of available sensors increases. However, they also observe that a sound filter sometimes rules out the correct positions, which gives us more motivation to look into the confidence level before using their results. Chen, et al. [43] combined WiFi and FM signal indicators as one signature. They found the interference to WiFi and FM signals happen independently. Thus, using combined signature increases the localization accuracy from around 80% up to 98%. We believe this is also applicable to other localization techniques. Rai, et al. [18] built a fully-automated indoor localization system called Zee, which uses a particle filter by combining IMU-based dead reckoning and WiFi signature-based localization, with the knowledge of movement constraints on the map. Zee uses estimated trajectory to determine the positions on the map, and records the WiFi signatures simultaneously to build the signature database from zero. WiFi signatures in history are used to calibrate the localization methods. Pandya, et al. [51] introduced the idea of combining different wireless signals because none of them is available everywhere, and the collective coverage will provide a more continuous service.

Others have looked at how data fusion can be approached, regardless of what specific sensors and algorithms are used. Gwon, et al. [52] propose two indoor localization fusion algorithms: Selection Fusion Location Estimation (SELFLOC) and Region of Confidence (RoC). SELFLOC is a linear weighted sum of multiple indoor localization results, where the weights are determined using historical localization performance in pre-defined regions. RoC is an algorithm to determine the final location when using triangulation. However, triangulation is not generally feasible in buildings where line-of-sight distance measurements can hardly be accurate. Bayesian Filters [31], such as the Kalman Filter and the Particle Filter, are used to combine estimations from different sources. However, they rely on localization systems that have consistent performance, which may not be true in some cases.

3.3 BearLoc – An Indoor Localization Platform

To study how well indoor localization can help reduce appliance list, we built BearLoc, a platform with abstractions to develop and run indoor localization algorithms. This work was done in collaboration with Siyuan He, Beidi Chen, and John Kolb at UC Berkeley [50].

The BearLoc architecture is shown in Figure 3.1. From bottom to top, there are three layers: a topic-based pub/sub network, the BearLoc framework, and component implementations by developers. BearLoc provides wrappers for all three categories of components: sensors, algorithms, and applications. Developers build and deploy their components using relevant wrappers. All components can be distributed and run on any Internet-connected device, communicating with each other on an overlay pub/sub network. For example, a sensor driver can run on a low-power mote or a web browser, and an application can be an HVAC system or on a light actuator.

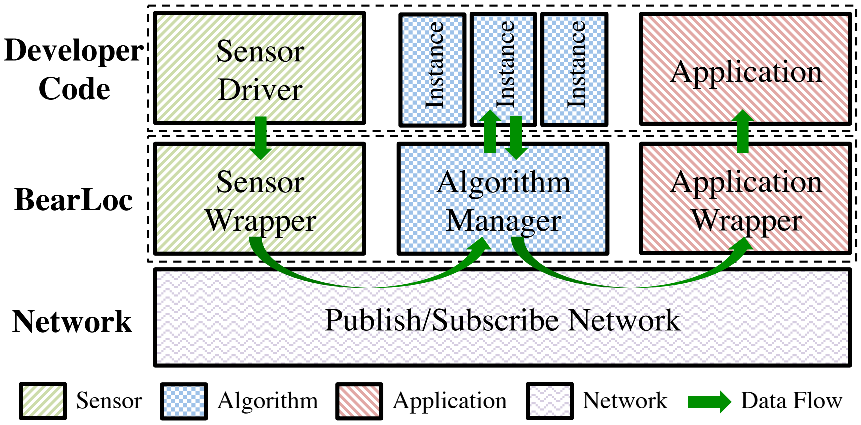


Figure 3.1: BearLoc System Architecture. BearLoc abstracts sensors, algorithms, and applications, allowing components from different developers to work together, on top of a pub/sub overlay network.

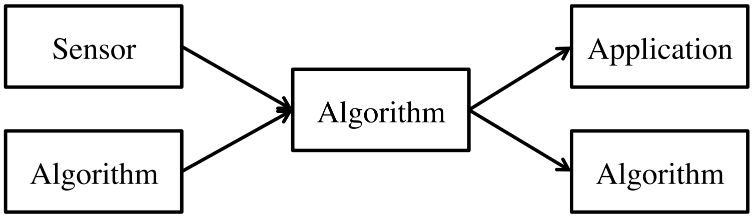


Figure 3.2: BearLoc *binding* process connects inputs and outputs to an algorithm. The inputs and outputs can be algorithms as well.

3.3.1 Data Flow: The Concept of Binding

A *binding* specifies how an algorithm is “wired up” with other components, which act as either input or output data representations. The concept of binding is illustrated in Figure 3.2 and a typical data flow between components is shown in Figure 3.1. In a binding, an algorithm takes data from either sensors or other algorithms and publishes locations to applications or other algorithms. More complex configurations, such as chaining and multiplexing, can be composed using multiple bindings. A functioning indoor localization system consists of both running components and bindings connecting them together. Since BearLoc is built on top of a topic-based pub/sub network, a binding is created by publishing to an algorithm’s control topic, along with topics of required sensors or algorithms, and a new output topic. Other applications and algorithms can get the estimated locations by subscribing to the new output topic. This procedure is defined by our Binding Control Protocol in Section 3.3.2.

Algorithm 3.1: BearLoc Binding Control Protocol

```
1 When an algorithm starts, it automatically subscribes to its control topic.
2 An application sends a Start Binding request to the control topic. The request must
    contain (1) output topics of required sensors and algorithms, (2) a new algorithm
    output topic, (3) a keep-alive topic. In addition, the request must include any (4)
    algorithm-specific configurations.
3 After receiving an request, the algorithm automatically (1) subscribes to the
    sensor/algorithm output topics and the keep-alive topic, and (2) publishes
    estimated locations to the algorithm output topic.
4 The application periodically sends keep-alive messages to the algorithms it is
    interested in, otherwise the algorithm instance is killed.
```

3.3.2 Control Flow: Binding Control Protocol

BearLoc provides a *Binding Control Protocol* between algorithms and applications for easy binding creation and maintenance. An application can start a new binding following Algorithm 3.1. In Step 2, there are four elements in a start binding request. The first two tell the algorithm where to "wire" the inputs and output respectively. The keep-alive topic is used for applications to preserve the binding and continue processing sensor data. The optional configuration data is useful for algorithms using shared sensors to filter data for specific targets. For example, a vision-based human tracking algorithm may only report locations of particular people. After a new binding is created, other applications and algorithms can subscribe to this algorithm and get location updates as well.

The first two steps require the application to know the algorithm specifications such as control topic, input sensor list, and configuration options. This should be simplified using a sensor and algorithm discovery service. We leave this to our future work.

3.3.3 Component Abstractions

BearLoc provides abstractions for three types of components: sensors, algorithms, and applications. Developers of different components can run their codes in a decoupled way. We discuss the details of each components here.

Sensor

In Figure 3.1, the sensor driver is a data generation implementation of a *Sensor Driver* interface defined in BearLoc. The interface specifies the sensor class, and lets the sensor wrapper register a callback function for data updates. When the sensor generates new data, the sensor wrapper relays it to the sensor's topic on the pub/sub network.

BearLoc provides a library of commonly used sensor classes. A sensor class specifies the data schema of a sensor, and provides data serialization and deserialization functions.

To ensure interoperability between sensors and algorithms, sensor developers must use the sensor classes supported by BearLoc.

Algorithm

An algorithm manager implements the binding control protocol and multiplexes an algorithm. For every start binding request it receives, it starts one algorithm instance, which is a process that runs its algorithm executable. The algorithm executable is implemented by algorithm developers using a BearLoc algorithm interface. The interface is an RPC server wrapper that invokes localization computation methods. The algorithm manager relays sensor data to an algorithm instance and then directs localization results back as an RPC invocation.

The algorithm manager also subscribes to the keep-alive topics for all bindings. A binding expires when no message is published to its keep-alive topic after a timeout period. Once a binding expires, the algorithm manager kills the associated algorithm instance, and unsubscribes from its topics. By managing algorithm instances, BearLoc hides the multiplexing overhead from algorithm developers.

Application

An application uses a localization service by initiating a binding and registering a callback function for location updates through the *Application Wrapper*. The application wrapper implements the binding control protocol. It also subscribes to the algorithm output topic, and has the registered callback function invoked on new estimated locations.

3.4 Localization Algorithms and Confidence Value

In this section, we introduce two indoor localization algorithms, WiFi RSSI fingerprinting and Acoustic Background Spectrum fingerprinting, and discuss how they can compute confidence values for every location estimation. Our intuition is that they don't have a consistently good (or bad) performance at different time or places. Specificailly, we focus on both the WiFi and ABS indoor localization algorithms, and extend the discussion a little in other algorithms. In our discussion, we normalize all confidence values be a real number in $[0, 1]$, where 1 means the systems are absolutely sure the estimation is accurate.

3.4.1 WiFi RSSI Fingerprinting

WiFi RSSI fingerprinting takes a set of RSSI values from all nearby WiFi APs as a signature. We collect signatures (a.k.a. fingerprints) on selected survey locations in advance and store them in a database. When an application queries for the estimated location, it looks for the closest fingerprint in the database. The distances could be any appropriate distances that can be applied to a scalar array, such as Euclidean distance or Manhattan distance.

The location of the closest fingerprints will be the estimated location. Practically, we find several closest fingerprints and use the average of their locations. This process is called KNN averaging, and is commonly used in fingerprint-based localization algorithms.

One way to compute confidence values for WiFi RSSI fingerprinting is using the distance we computed. Intuitively, smaller distance should yield higher accuracy, and therefore higher confidence. If the algorithm uses more than one fingerprints, the sparsity of their location coordinates can also be used for confidence levels. The more sparse they are, the less confidence it should be.

In addition, context information can be used to provide confidence. For example, we can determine whether the signal is consistent at one location by recording multiple signatures over time. Other indicators include the background noise or interference, the network delay, and numbers of users connected to an AP. These information are all observable with existing infrastructure, but how exactly they influence the accuracy of the estimations require more in-depth research.

Historic performance can also be used, such as the consistency of RSSI of each AP under different circumstances (e.g. time of the day, numbers of connected clients, interference, network throughput).

3.4.2 Acoustic Background Spectrum

Acoustic Background Spectrum (ABS) [14] is another fingerprint-based approach and uses sound features in the background as signatures. The idea is that each room has its unique pattern of background noise, thanks to different components (e.g., pipes, machines, HVAC) behind the wall that make sounds.

Similar to WiFi RSSI, ABS can also use signature distance and fingerprint sparsity to compute confidence. It can also use algorithm-specific context information. For example, fingerprints in large open areas should be used with less confidence than isolated rooms, because there are less acoustic changes over open space. Performance history is also a good information to factor into the confidence value computation. One point to emphasize here is that the idea for history performances in the framework and in each IPS are different. According to the evaluation results of ABS [14], it only works properly in relative quiet environments. Based on the noisiness of the record, ABS can adjust the confidence value accordingly.

3.5 Data Fusion

With multiple sources of estimated location list with confidence values, we need another algorithm to combine them for a final location estimation. Too achieve this, it needs to decide the weights of each localization algorithm. Unfortunately, the confidence provided by these algorithms can be inaccurate as well, because of the heterogeneity of the algorithms and their implementations. We propose an credit-based method, which tracks the fidelity of the confidence values of each algorithm and assign weights based the credits. To combine the results, we use both linear average of coordinates as well as basic Kalman Filter.

3.5.1 Performance Credit

To combine localization results, the fusion algorithm assigns credits to participating algorithms based on their estimated location and confidence values. Intuitively, if one algorithm always provides accurate estimations (estimations close to the final decision) with high confidences, it gains credits. Inaccurate estimations with high confidence will harm the credits. If an algorithm has low confidence on some results, it shouldn't affect its credit no matter how accurate the estimations are.

3.5.2 Weight Assignment

The weight of an algorithm represents how much the fusion algorithm uses its estimation. For simplicity, we require weights to be normalized before use. It means all weights W_i $(i = 1, 2, 3, ..., n)$ for n algorithms must sum to 1:

$$\sum_{i=0}^{n} W_i = 1, \ W_i > 0 \tag{3.1}$$

The most intuitive way to assign weight is using the product of credit and confidence value. Moreover, as the purpose of weights is to highlight the most possibly accurate result, and reduce the impact from erroneous ones, when the framework detects large sparsity among estimated locations, it may consider amplifying the differences between all weights, to potentially reduce the effects from less-trusted algorithms.

3.5.3 Location Granularity Unification

Different indoor localization algorithms yield different representations of locations. For example, ABS with a room-level database can only generate room IDs (or semantic representations). WiFi RSSI fingerprinting with surveyed locations can compute an estimated coordinate in the spatial form. To combine these results, the fusion algorithm must have a map containing both the semantic and coordinate representations. Map creation is out of the scope of this chapter. With a map, we propose two ways to unify the granularity as follows.

- **Semantic to Spatial:** To convert estimated room to a coordinate, the fusion algorithm can simply use the center point of the room. For large rooms, multiple coordinates in the room can be used to fuse with other results. These coordinates can be uniformly selected in the room at a proper granularity.

- **Spatial to Semantic:** For the other direction, we can simply convert a coordinate to room IDs they locate in. However, because of the lower granularity of semantic representations, we may end up having no overlapped rooms from different algorithms. We leave solutions to this problem to future work.

3.5.4 Linear Weighted Average

With weights and unified representations of locations, the fusion algorithm can combine them for the final estimation. In our work, we only use "Semantic to Spatial" described previously. One of the most intuitive data fusion approaches is linear weighted average. Specifically, the final estimated location l is

$$l = \frac{\sum_{i=0}^{n} (w_i \cdot l_i)}{n} \tag{3.2}$$

where vector l_i as the estimated location from the i-th algorithm, and w_i are their weights.

3.5.5 Kalman Filter

Compared to linear weighted average, the Kalman Filter considers the rationality in the trajectory. We used a simple form of the Kalman Filter to combine the data. It provides Bayesian recursive estimate of state space X_k using knowledge of previous state $[X_1, X_2, X_3, ..., X_{k-1}]$ in a linear state space system with Gaussian noise. Since the project focus was on building the framework rather than the Kalman Filter itself, we used a basic form of Kalman Filter with certain assumptions. Firstly, a person's movement in a building is a linear system where he is moving with a constant velocity. This is a simplest case of a person's movement in building. So a simple variation of Kalman filter would suffice for data fusion. Secondly, the measurements are taken from right top corner of the floor map. Position of both sensors are considered to be fixed at $(0, 0)$ position.

Each sensor module feeds the Kalman Filter with a location list and confidence value. Another important assumption is that the sensors are synchronized, meaning they provide measurement of target state at same time interval. Using the linear system equations for position estimation and the measurements from sensors Kalman Filter corrects the errors in sensor measurements. We combine the data using weighted average after the estimation. Weights are calculated based on the confidence value supplied to the framework.

3.6 Evaluation

To learn how well indoor localization practically helps to reduce appliance list, we evaluate the linear weighted average fusion algorithm combining both WIFI RSSI and ABS fingerprinting. Our experiment continuously localizes a smartphone while walking along a campus office space. We first show that the confidence values are accurate for both algorithms. With the confidence values, accuracy can be improved after combining two algorithms, either using linear weighted average or the Kalman Filter. We also explore how the database size can impact our results, which can help us reduce the survey overhead.

3.6.1 Experimental Setup

We implemented both WiFi RSSI and ABS fingerprinting, where data collection is on Android and data processing is in MATLAB. In WiFi RSSI fingerprinting, we ran a mobile application on a LG Revolution VS910 with Android 2.3.4 to sample and log WiFi RSSI values. Every sample was saved as a $(MAC\ address, RSSI)$ tuple. A WiFi signature is represented as a list of tuples that are collected about every 800 milliseconds. In the database, every fingerprint is corresponded to a 2D coordinate on the map.

For ABS fingerprinting, we used the default sound recorder in an Apple iPhone 5 with iOS 6.01, which stored the audio in .m4a files. At each survey location, we recorded at least 60 seconds for higher quality fingerprints. To make the audio file easy to analyze, each of them is converted to a .wav file in Audacity. To make it feasible to combine with WIFi RSSI fingerprinting, we bind every ABS fingerprint to a coordinate in the database. Using fine-grained should not reduce the accuracy at the semantic level, because estimated coordinates can be easily converted to an room it is in, given we have an accurate map.

In the data survey phase, we collected both WiFi RSSI and ABS signatures at 98 selected locations in a campus office space, marked as red cross markers in Figure 3.3. At every location, we record more than 60 seconds of both audio signal and WiFi RSSI values. We manually measured and input the coordinate into the database. The time spent on collecting all data in database spans one week.

Figure 3.3 also shows three paths we tested for localization. While walking along a path, we continuously collected both WiFi RSSI and ABS signatures, while using another smartphone camera to capture the ground truth location, as shown in Figure 3.4. We manually extracted the ground truth locations from the video, and used the collected signatures for localization in MATLAB. We selected an subset of these signatures for localization, shown as circles on the path in Figure 3.3. For every circle, both WiFi RSSI and ABS signatures collected at the location are used. The size of marker represents the time the user spent at that point. Both path 1 and path 2 started from the bottom part of the map, whereas path 3 started from the top part. Starting points are tags as S1, S2, and S3 in Figure 3.3 respectively. To help the discussions in Section 3.6.2 on how the environment can influence ABS fingerprinting performance, we also mark isolated rooms and open areas on the map.

With the signatures sampled along the path, WiFi RSSI and ABS fingerprinting algorithm localizes the signatures we selected, as we described before. We use Euclidean distance as signature distance. In WiFi RSSI fingerprinting, the default values of RSSI is set to $-150\ dBm$ if it is missing in either signatures compared. Instead of fixing the

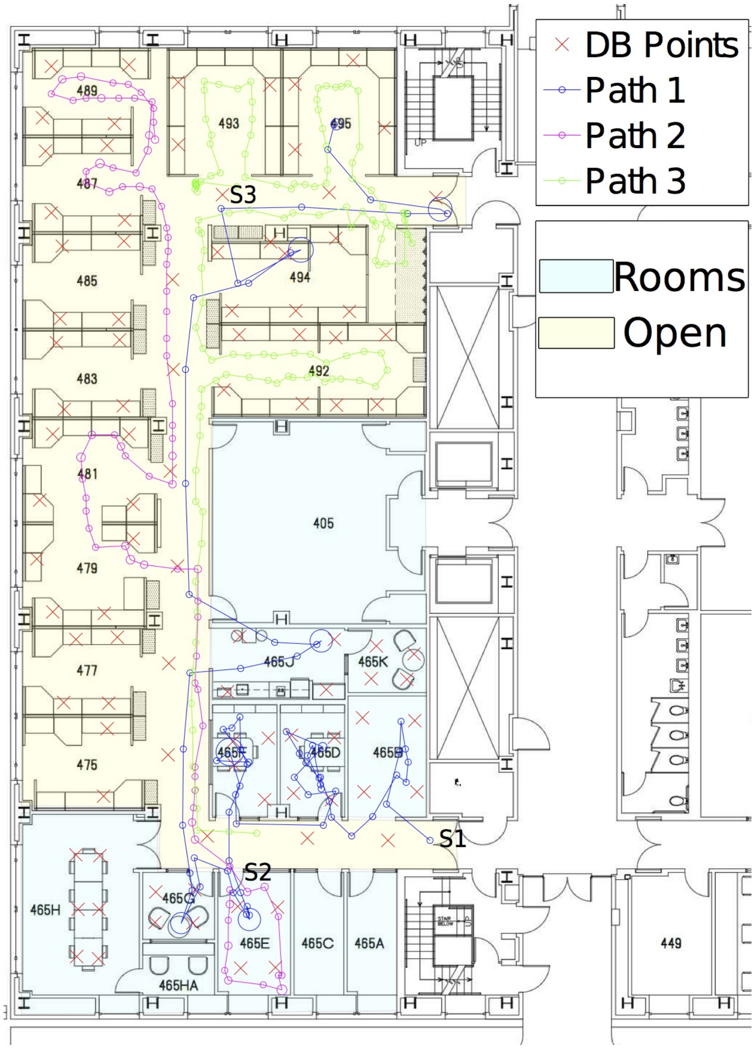


Figure 3.3: Experiment setup contains 98 survey points (red cross markers), and three localization sessions (paths). A circle on a path indicates we performs a localization computation based on the signatures captured at the location.

Figure 3.4: Experimenter holds two smartphones while walking in the office space. One smartphone is collecting both WiFi RSSI and ABS fingerprints, and the other one is capturing a video for manual groud truth inference.

K in KNN, we average among all fingerprints within a distance threshold $T = \alpha \cdot D_{min}$, where D_{min} is the smallest signature distance, and $\alpha = 1.3$ is a empirally decided constant. Currently, we only use the sparsity of this location list to determine the confidence value for both WiFi RSSI and ABS fingerprinting. In particular, the confidence C_i is defined as

$$C_i = \frac{1}{2 \cdot e^{\frac{std_x}{b}}} + \frac{1}{2 \cdot e^{\frac{std_y}{b}}} \tag{3.3}$$

where Std_x and Std_y denote the standard deviations on each dimension of coordinates, and $b = 100\ (inch)$ is an empirical ratio.

After getting the estimated locations and confidence values, the fusion algorithm obtains assigns normalized weighted W_k based on C_i using

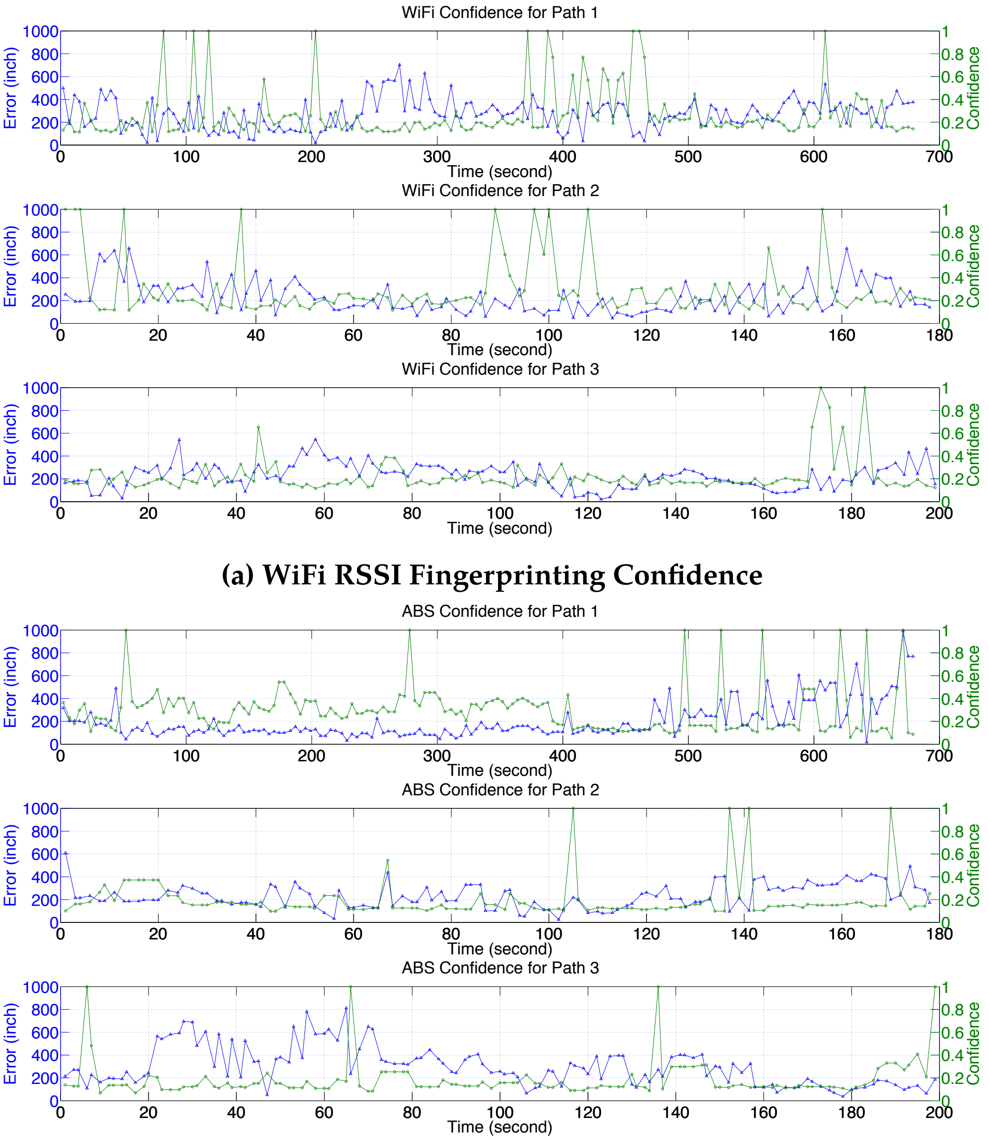
$$W_k = \frac{(C_k)^3}{\sum_{i=0}^{n} (C_i)^3} \tag{3.4}$$

The purpose of the power computation is to amplify the differences between the two algorithms. With weights normalized, the fusion algorithm applies linear weighted average and Kalman Filter to the estimated coordinates from both input estimations.

In the remaining parts of this section, we examine the confidence value accuracy, data fusion performance, and how the database size and density can impact the localization results.

3.6.2 Confidence Value Accuracy

Figure 3.5 shows the confidence values and localization errors (distance from estimated location to the ground truth location) over time for both algorithms in all 3 paths. Every



(a) WiFi RSSI Fingerprinting Confidence

(b) ABS Fingerprinting Confidence

Figure 3.5: Confidence Accuracy of WiFiLoc and ABSLoc. Localization error is generally low when the confidence value is high.

marker corresponds to a marker in Figure 3.1. For path 1, it samples a WiFi and ABS signature nearly every 4 seconds, and path 2 and 3 sample every 1 or 2 seconds. We eliminate the samples between adjacent markers along the curve to reduce the manual labor on round truth extraction.

As we can see, in most cases when the confidence is high, the errors are low, with only one exception in ABS fingerprinting at the end of path 1. High confidences do not happen frequently, but the confidence between two algorithms are not highly correlated. This fits with the intuition that WiFi signal should have nothing to do with acoustic background noise. Experiments can be conducted to empirically future verify this, such as [43].

In path 1 of Figure 3.5b, it is also very clear that ABS fingerprinting works better in isolated rooms, which starts near the beginning and ends after 470 seconds. In comparison, performance in open space shows a larger variation. At the same time, the confidence values of ABS fingerprinting are also higher in isolated rooms than in the open area. This can not be observed in path 2 and 3, because they are almost all in the open area. It suggests that we can also use this information to infer confidence in the future.

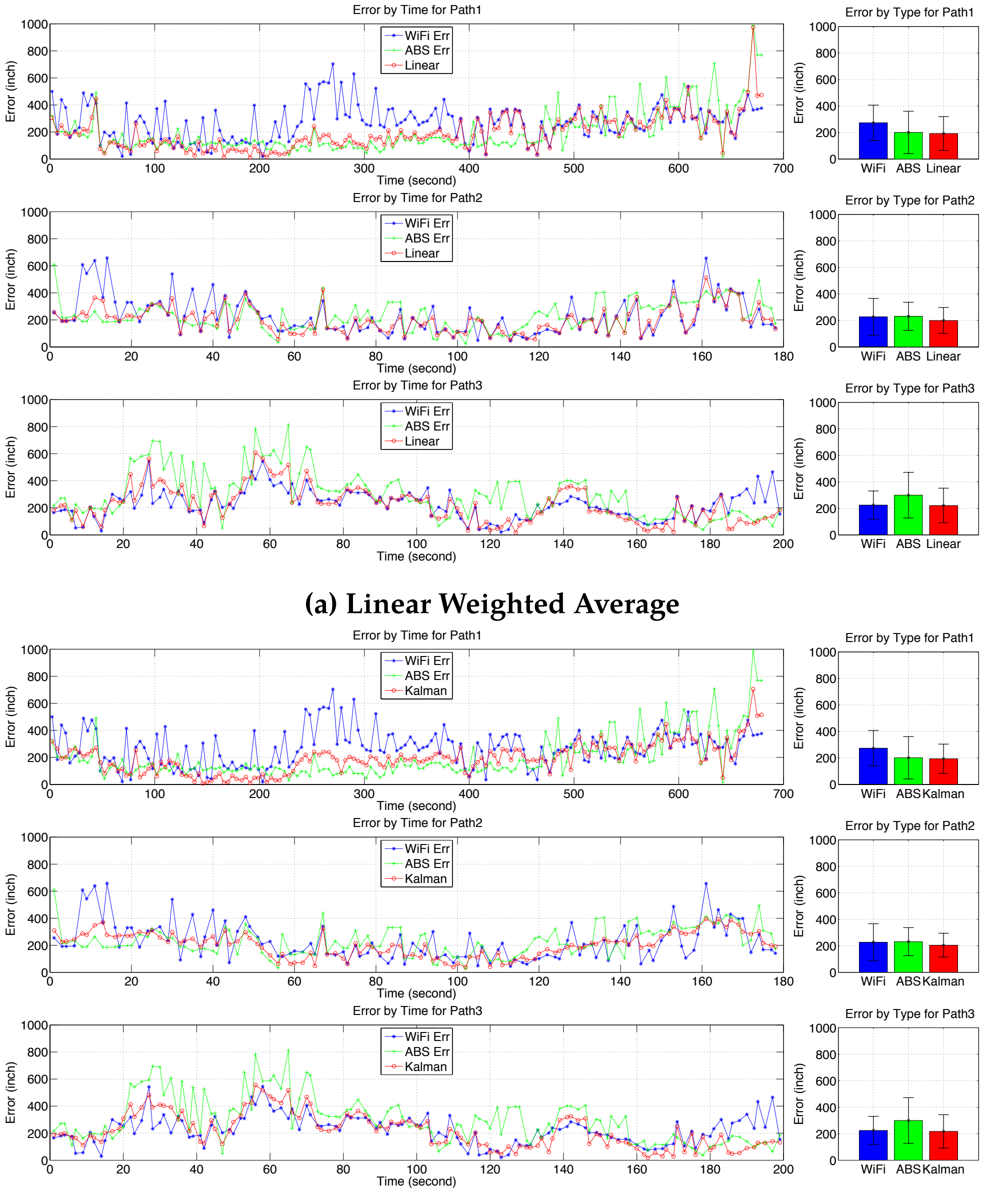
3.6.3 Data Fusion Algorithm

Figure 3.6 shows the errors over time for WiFi RSSI and ABS fingerprinting, as well as the linear weighted average data fusion algorithm in all 3 paths. It also shows the mean and standard deviation of the errors in a bar plot. In the barplot, we can see that ABS fingerprinting is more accurate than WiFi RSSI fingerprinting in path 1, but less accurate in path 2 and 3. This is because path 1 is mostly in isolated rooms, whereas path 2 and 3 are mostly in the open area.

More importantly, no matter which algorithm performs better, both linear weighted average and Kalman Filter can almost give a final result that has a equal or smaller error. However, at the end of path 1, because ABS fingerprinting gives a result with both high error and high confidence, the linear weighted average method generates a high error too. Instead, Kalman Filter takes the human movement limitation into consideration, it manages to reduce the spike by nearly 50%. Moreover, Kalman Filter eliminates most other high errors where linear weighted average cannot. In the following experiment, we choose to use Kalman Filter for better performance.

3.6.4 Fingerprint Number Per Point

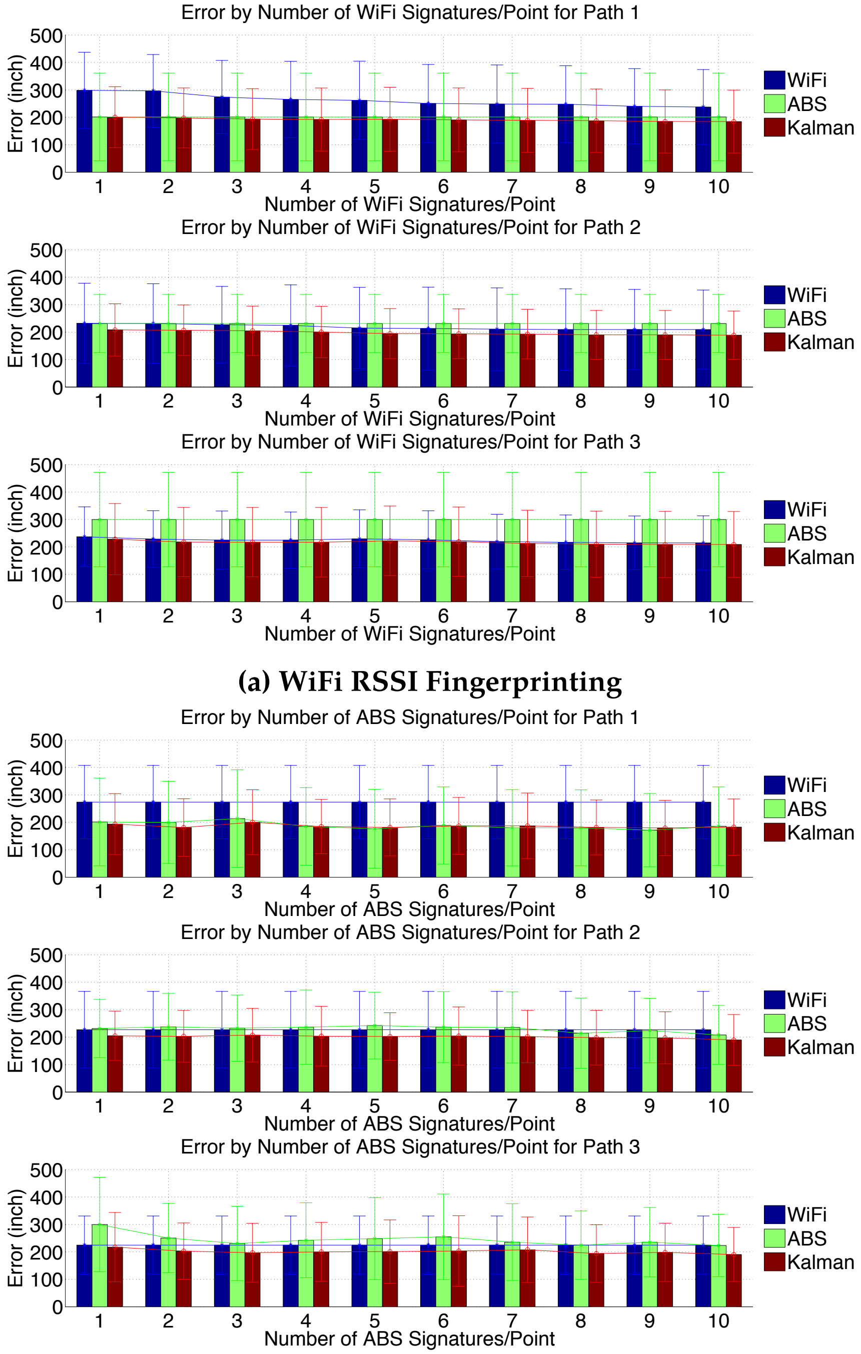
At every location in the 98 survey points in the database, we collected multiple fingerprints. In WiFi RSSI fingerprinting, a 60-second log can contain more than 70 signatures. A 60-second audio record can also be divided to several small audio sections. Because we collect the ABS fingerprints when it was quiet, the signature can even be extracted from a 5-second section. However, the time spent on KNN search increases linearly with the number of fingerprints in the database (unless you perform approximate KNN search, such as using a KD-Tree). Using more than enough fingerprints can only increase the



(a) Linear Weighted Average

(b) Kalman Filter

Figure 3.6: Performance improvements by combing WiFi RSSI and ABS fingerprinting using Linear Weighted Average and Kalman Filter



(a) WiFi RSSI Fingerprinting

(b) ABS Fingerprinting

Figure 3.7: Localization errors when using different number of fingerprints at every survey location in the database. It shows we usually do not need more than three fingerprints at each location before the accuracy saturates.

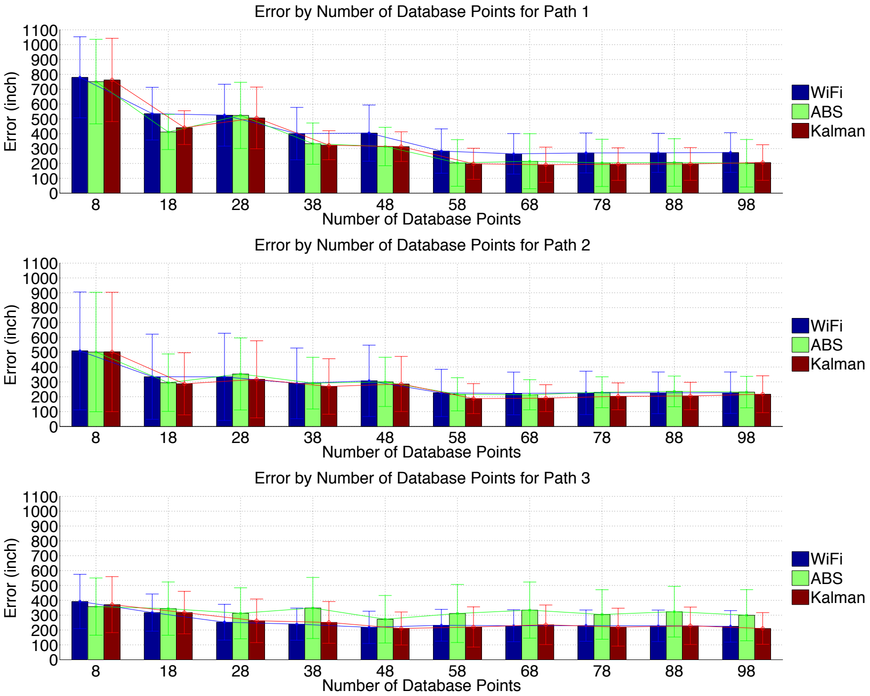


Figure 3.8: Error Trends by the Number of Database Points. It shows we only need certain number of survey points in each building.

system response time unnecessarily. In our experiment, when we use 10 WiFi fingerprints at each survey location, it can take up to 10 seconds to localize a signature.

Figure 3.7 shows indoor localization errors with different number of fingerprints we use at every location. We can see the errors do not reduce much beyond three fingerprints at each location.

3.6.5 Number of Survey Points

Ideally, the more survey points we have, they smaller the localization errors should be, and the more manual work it is to build the database. To find how many survey points we need to build a sufficient database for an typical office space, we conduct our experiment with different number of survey points randomly removed from our database. Based on the results in Figure 3.7, we use 3 fingerprints in for WiFi RSSI and 1 fingerprint for ABS in this experiment. Figure 3.8 shows the errors when different number of survey points used. When removing points from database, we manually tried to remove points uniformly from on the map. As expected, more survey points help decreasing the localization errors, but it saturates at around 58 points for this particular lab space. Ideally, we should optimize the number of survey points and their locations before building a database. We leave this to future work.

3.7 Summary

In this chapter, we studied how indoor localization and data fusion algorithms can be used to narrow down displayed appliance list by showing only those near a user. To conduct the study, we design and implemented an indoor localization development platform BearLoc. It abstracts sensors, algorithms, and applications, and connects them using an overlap pub/sub network. In addition to conventional indoor localization systems that only output estimated locations, we propose to have each algorithm developer provide a confidence value along with each estimation. Confidence value computation requires a well understanding of the mechanism of the algorithm, which is usually occlusive to the algorithm consumers. We also implemented an data fusion algorithm that uses linear weighted average to combine results from multiple other indoor localization algorithms.

To study how accurate indoor localization systems can be, we implemented two popular indoor localization algorithms, WiFi RSSI fingerprinting and ABS fingerprint. We also implemented confidence value computation based on the variance of every KNN search result. Both algorithms are deployed in an office space. The experiment results show that both algorithms can product accurate confidence values. Taking advantage of the confidence values, the data fusion algorithm can almost always find the better estimation among the two algorithms. Compared to the simple linear weighted average, Kalman Filter generates a more smooth trajectory because it considers limitations of human movements.

In conclusion, indoor localization algorithms can have an average of 200 inches (or 5 meters) in estimation errors. This constitutes a very good mechanism to narrow down appliance list to a room. However, many rooms in commercial buildings contain multiple instances of the same type, such as ceiling lights, projectors, and power outlets. A reduced appliance list still have to display all these appliances, which is hard to disambiguate just by name for smart building occupants.

Chapter 4

SnapLink: Using Single Image Localization

Since appliances of the same type can hardly be disambiguated with indoor location technologies, we try to look at the problem from the perspective of direct control. As discussed in Chapter 2, sensor-based approaches generally require installation of sensors and even modifications to existing appliances, which is not ideal or salable in commercial buildings that is already occupied everyday. In this chapter, we look into what are the best solution for vision-based approach, and demonstrate this idea with a system we build, dubbed SnapLink. We demonstrate the SnapLink is robust to different image capture angle and distance, and works with different illumination conditions. We also show the natural everyday environment changes will not affect our system performance over a long period of time, which allows us to update the database infrequently. This work was done in collaboration with Jonathan Fürst at IT University of Copenhagen, Xin Jin at Johns Hopkins University, as well as John Kolb and Hyung-Sin Kim at UC Berkeley [53].

4.1 Introduction

We start with the intuition that it is natural for humans to use their visual senses when interacting with appliances: *"What you see is what you control."* This motivates us to design a *vision*-based appliance identification and control system that allows users to interact with any appliance they can see. Such a system needs to include a simple and intuitive user interface that converts human vision (an appliance image) to a control interface. We use *smartphones*, one of the most ubiquitous devices in modern life, as the vision sensor (i.e., capture an image of an appliance within view) and appliance controller (i.e., control the appliance in the captured image). Specifically, we aim to enable users to control an appliance through a smartphone application by simply pointing at it, as depicted in Figure 4.1. Users of this application should not be concerned with taking pictures from a specific angle or distance from the appliance. Whatever image is taken, users

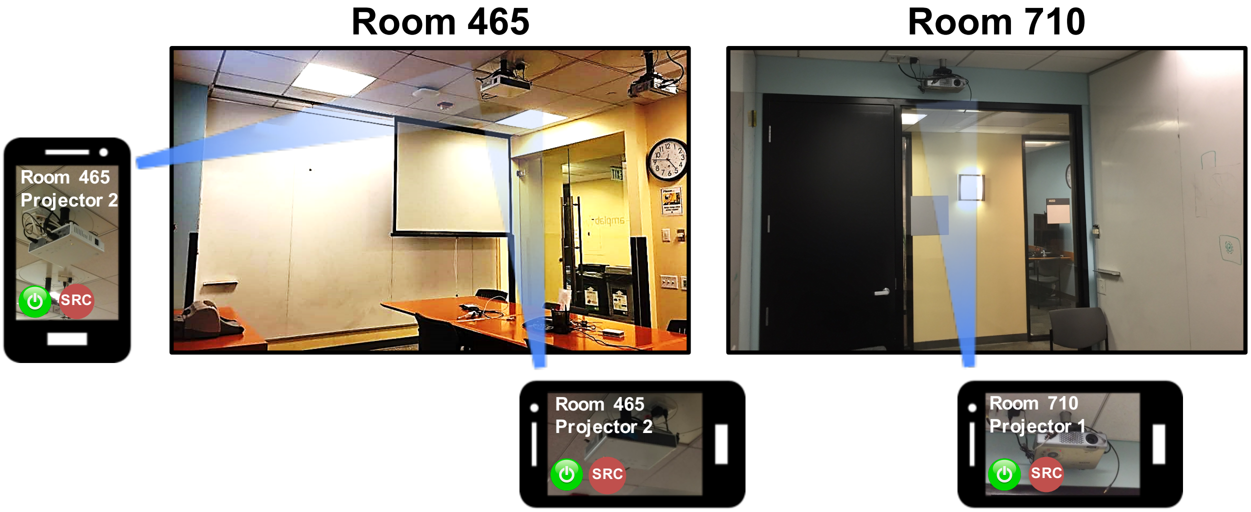


Figure 4.1: Application scenario for our vision-based appliance identification and control system. Users can get a control interface of an appliance by capturing an image from an arbitrary angle and distance, even when other similar appliances exist in the same or different rooms.

should get a proper control interface of what they are pointing at (e.g., on/off button for projector 2 in room 465) on the smartphone screen. Furthermore, no matter how many similar appliances exist (e.g., the same projector in various meeting rooms), users should still enjoy convenient appliance control by receiving an accurate and fast identification response when they point their smartphone at an appliance.

Even though a number of studies have investigated the problem [33, 37, 38], their proposals cannot scale to commercial buildings containing hundreds to thousands of appliances. Some approaches require additional infrastructure for each appliance, such as laser or infrared signal receivers [32, 33], introducing a large deployment overhead. Some are not applicable when controlling appliances from an arbitrarily large distance and angle, such as QR codes [54, 55]. Finally, some approaches are inconvenient for users, especially in a large building, such as typing textual queries [11], memorizing vocal or gesture commands [37, 38], or manually browsing and searching on a 2D/3D map [35].

In this chapter, we design and implement **SnapLink**, a prototype system that can quickly and accurately identify an appliance among hundreds to thousands of (possibly similar) appliances based on a query image from an arbitrary location and orientation. SnapLink uses *image localization* [56] to identify an appliance by finding where a query image is located in a pre-constructed *3D space* in the database. This design choice comes from two motivations:

1. From the user perspective, image localization provides high identification accuracy with arbitrary query images.

2. From a deployment perspective, 3D-model construction (the database for image localization) requires only capturing a video with a commercial off-the-shelf RGB-Depth camera while walking through a building.

With a 3D model, appliance labels only need to be applied once per appliance, whereas previous vision-based systems [55, 57, 58] require many images for every appliance and every image to be labeled. In addition, we propose two ways to further improve the accuracy and latency of image localization-based appliance identification in commercial buildings:

1. a *Feature Sub-sampling* mechanism that eliminates redundant features from query images.
2. a *Geo-partitioned* 3D-model construction that does not build a monolithic 3D-model but a group of small 3D-models (of each room) to represent the entire building.

We have built a SnapLink prototype system including an Android smartphone application and integrated it with BOSS [1], a representative modern BMS. To evaluate it, we construct a database with 3D models by collecting 67 minutes of video footage from 39 different rooms from 5 different buildings, which is equal to the size of a small commercial building [59]. We also label an arbitrarily selected subset containing 179 appliances in these rooms. To our knowledge, this dataset size has one order of magnitude more modeling images and covers a much larger space than those featured in prior vision-based object identification work [55, 57, 58], which allows us to optimize and verify the scalability of our approach. For 1526 test query images of these appliances from various angles and distances, our SnapLink system achieves **94%** identification accuracy and **120 ms** server processing time per query image, whereas our image retrieval-based baseline system takes 177 ms to achieve only 66% accuracy.

Our contributions are as follows:

- We introduce a novel and useful application, *arbitrary image*-based appliance identification and control with ubiquitous *smartphones*. We describe several technical requirements in designing a system to support this application.
- With the application requirements and technical challenges as the basis, we present a set of key approaches in order to maintain low latency and deployment overhead in large buildings: *image localization* in *3D models*, *Feature Sub-sampling*, and *Geo-partitioning*.
- We build an end-to-end system that integrates SnapLink with a modern BMS (i.e., BOSS [1]) and provides an Android smartphone application as a user interface.[1]

[1] https://github.com/SoftwareDefinedBuildings/SnapLink

- We validate the performance of SnapLink at the scale of a commercial building that has 39 rooms and 179 appliances, which is an order of magnitude larger than in prior work [55, 57, 58]. In this large real-world scenario, SnapLink achieves 94% identification accuracy and 120 ms server processing time, which are both sensitive to the database size.

While this chapter focuses on appliance identification and control, we believe that it is only one example of a new generation of smart building applications enabled by SnapLink, including location-based authorization, augmented information display, indoor navigation, and building diagnosis and re-commissioning. The rest of this chapter is organized as follows: In Section 4.2 we introduce our application scenario and its requirements. We also discuss candidate vision-based identification methods for our application. Next, we introduce details on SnapLink's design in Section 4.3 and its implementation in Section 4.4. We present deployment results of SnapLink in Section 4.5. We position SnapLink among prior work in Section 4.6, and conclude with a summary in Section 4.7.

4.2 Target Application: Vision-based Appliance Identification and Control

Recent work on smart buildings has investigated how to unify various vertically integrated, isolated BMSs and Internet of Things (IoT) devices into a common set of abstractions. For example, BOSS (Building Operating System Services) develops a building operating system on which various applications can run to improve building performance and occupant comfort [1]. Applications identify the underlying heterogeneous appliances by standardized metadata and unique identifiers [8]. Such identification models work well for centralized building applications, but fail when occupants want to directly interact with appliances. Our hypothesis is that most humans want to control appliances within their sight. Thus, user-centric, cyber-physical applications should leverage visual identification.

4.2.1 Application Scenario and System Architecture

We aim to design a vision-based appliance identification and control system that provides an intuitive mobile user interface and can be applied in an environment as complex as a commercial building. Consider the following application scenario. While walking through a building, Alice encounters a networked printer in a conference room and wishes to use it to print out a document from her smartphone. She turns on the SnapLink app and takes an *arbitrary* picture of the printer (as in Figure 4.2). The app then processes the picture and quickly recognizes it as the specific printer in the conference room. The app overlays the printer's control interface on the smartphone screen so she can interact with the printer. Alice clicks on the "Upload and Print" button to upload the document, which is then printed out by the printer.

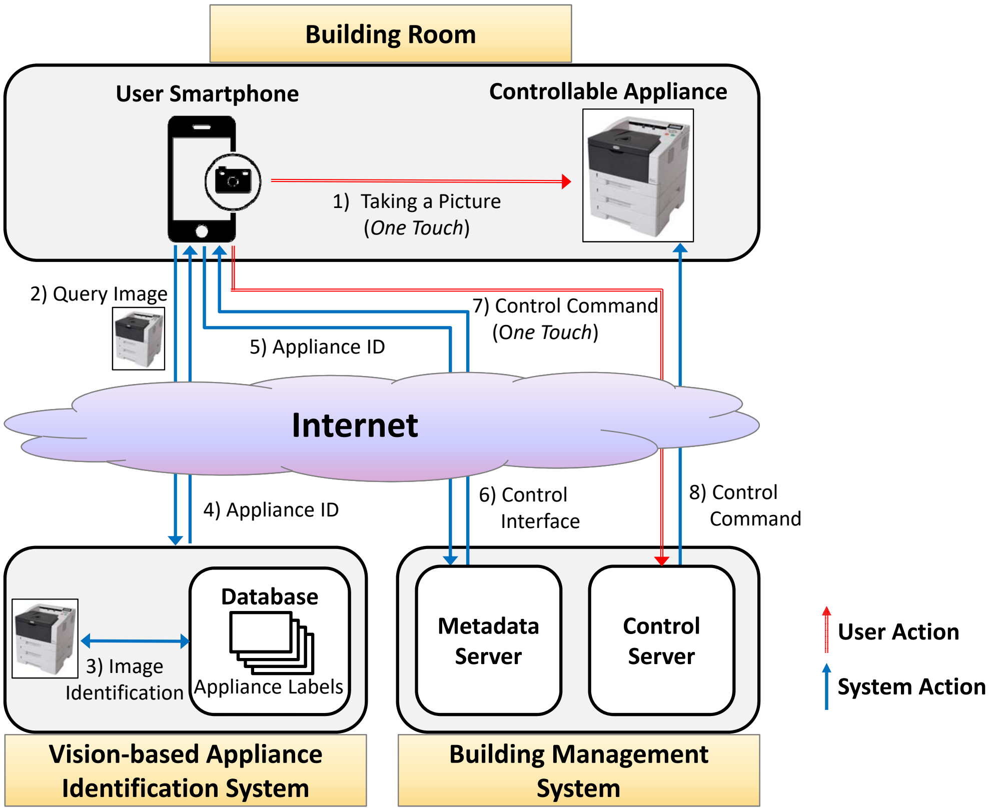


Figure 4.2: Application scenario for our vision-based appliance identification and control system, consisting of controllable appliance, user smartphone, vision-based appliance identification system, and BMS.

At a higher level, the entire workflow of this application and the system architecture are depicted in Figure 4.2, which includes both user actions and the underlying system's behavior. It consists of a controllable appliance, the user's smartphone, a vision-based appliance identification system, and the BMS. After Alice takes a picture of the target appliance, her smartphone sends the image to a vision-based appliance identification system. Using the query image, the identification system selects the most likely appliance from those in its database and sends the appliance ID to Alice's smartphone. The smartphone sends the appliance ID to the metadata server of the BMS and receives the control interface for the target appliance.[2] Finally, Alice can now see the control interface on her smartphone screen and control it as needed. When she touches a command button, the smartphone generates and delivers the control command to the BMS's control server,

[2] Alternatively, the same server could identify the appliance, use its ID to retrieve the proper handler from the BMS, and send a control interface back to Alice's phone. However, this is an implementation detail that does not affect the core functionality of the system, and the round trip time between the phone and a local BMS server is likely to be small.

which finally controls the target appliance by sending it the proper actuation command. Note that, during the whole appliance identification process, all Alice needs to do is touch her smartphone screen once to take a picture before she can interact with the appliance.

4.2.2 Technical Requirements

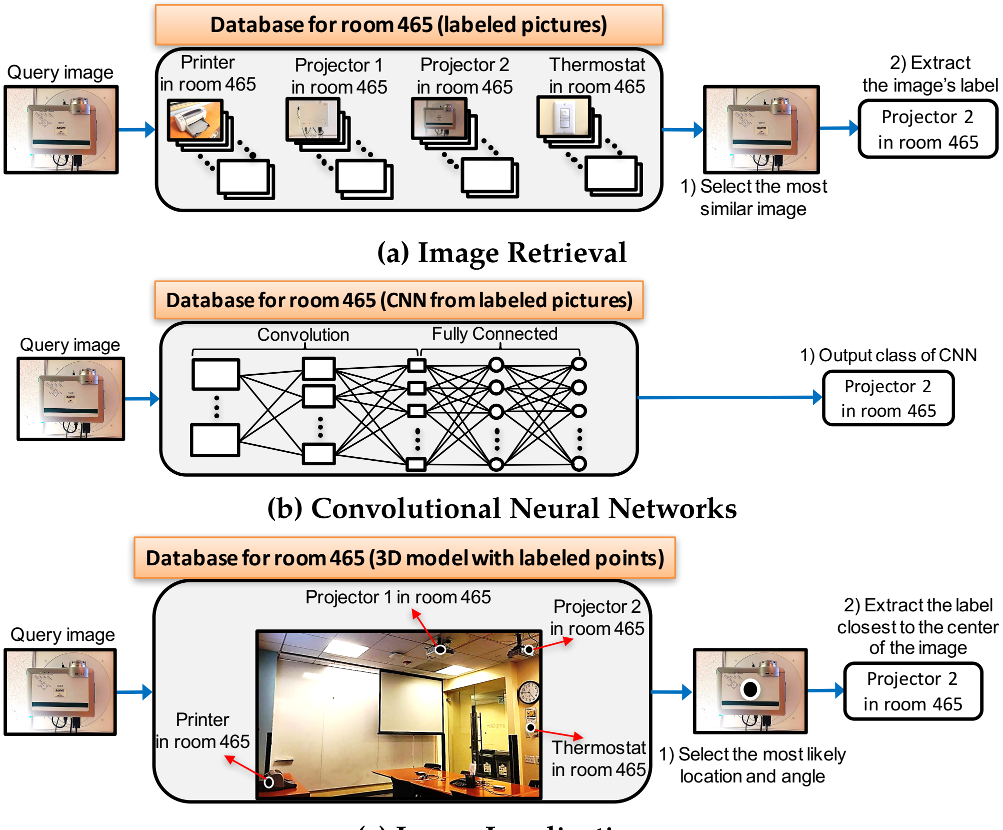
Our system needs to meet the following requirements to enable the target application, which motivate our subsequent design of SnapLink detailed in Section 3.

Identification Accuracy. High appliance identification accuracy is our first goal. Our system needs to accurately identify a target appliance from an arbitrary image and distinguish it from other similar appliances. However, even a single incorrect result will be frustrating. Therefore, while achieving high identification accuracy, our system should also allow manual appliance selection as a fail-over. Identification accuracy should be high under changing environments. We assume most appliances are not moved frequently (e.g., printers, projectors, lights), but the environment they sit in can always have daily occupant activities and changes in lighting conditions.

Low Latency. As an interactive system, our system needs to be responsive so that users do not perceive a considerable delay between an action and its response. We aim to achieve no larger than 100 ms latency, which is the time requirement to create the impression of an instantaneous reaction [60]. However, since the most important performance metric is identification accuracy, minimizing latency should not sacrifice accuracy.

Scalability. Given that our system targets large commercial buildings with thousands of appliances, the database of our appliance identification system should hold a large number of appliance labels. Since finding the most relevant appliance among a large number of candidates creates a computational burden, our system should provide scalability in terms of database size to achieve both high accuracy and low latency when applied to large commercial buildings.

Low Deployment Overhead. To enable the application, system managers/staff need to construct a database with the information on all controllable appliances in a large commercial building. This involves nontrivial deployment overhead. Our system should be user-friendly but also deployment/management-friendly. To this end, we need to carefully consider what method to use for vision-based identification since it significantly impacts deployment overhead (e.g., image retrieval requires taking a large number of pictures in the deployment phase).



(a) Image Retrieval

(b) Convolutional Neural Networks

(c) Image Localization

Figure 4.3: Comparison between three design options for vision-based instance identification: image retrieval, Convolutional Neural Networks, and image localization.

4.2.3 Design Options for Vision-based Appliance Identification

We now discuss how to enable vision-based appliance identification, the most important function of our system, and present our core design choices for SnapLink. A modern commercial building has hundreds to thousands of appliances, but most fall into a limited set of categories (e.g., thermostats, projectors, lights). This means that many appliance instances fall into the same category. To enable users to control a single target appliance, we need to recognize an appliance *instance* (i.e., a specific physical object) rather than its *category*.

There are several representative ways to perform instance recognition: *image retrieval*, *Convolutional Neural Networks (CNNs)*, and *image localization*, as depicted in Figure 4.3. An image retrieval-based instance identification system (Figure 4.3a) needs to construct a database of labeled appliance images. When receiving a query image as an input, it searches for the most similar labeled image among the database based on the number of common salient visual features (e.g., SURF (Speeded Up Robust Features) [40]). To our knowledge, image retrieval is used by all previous vision-based instance identification systems for appliance control, such as [55, 57, 58, 61]. However, it has several drawbacks that preclude its deployment at the scale of a large commercial building. First, existing techniques require every database image to contain only one appliance and each image must be individually labeled. Second, recognition succeeds when the query image is taken from a similar angle and distance as the corresponding labeled image in the database. In a commercial building with hundreds to thousands of appliances, this results in significant deployment overhead for building managers/staff[3] to take pictures of all appliances from various angles and distances and to label each of them (i.e., number of labels = appliances × angles × distances). The accuracy and deployment overhead would not be an issue when it comes to a well-constructed huge database, which already has an extremely large number of labeled images [62]. Unfortunately, this is not the case for our target application, which requires constructing a *local* database for each commercial building.

Convolutional Neural Networks (CNNs), which have been extensively used for category recognition [63, 64], have recently been applied to instance recognition as well (Figure 4.3b). A system trains labeled pictures into a CNN model in advance with the instance labels as classes. A query image is classified by the model and the output class is simply the recognized instance. However, as summarized in [65], CNNs only outperform image retrieval on instance recognition with scenery [66] and landmarks [62, 67] datasets, where training and testing images have a similar data distribution (e.g., number of images of an instance) and little occlusion of the instances is present. In building environments, the usage of appliances is unknown in advance, and occlusions are very common (especially when the viewing angle is large, such as in Figure 4.24b), therefore image retrieval is preferable to a CNN-based approach for our purposes. Moreover, an image classification process can take seconds to finish on a modern CPU [65], making CNNs prohibitively expensive without a GPU for our application, which requires low latency.

An image localization-based instance identification system (Figure 4.3c) involves constructing a database in the form of a 3D model of a building and one labeled point for each appliance in the 3D space [56]. To infer which appliances are visible within a query image, it localizes the query image in the 3D-space database using the most likely location and angle, captures the image of that specific location/angle (i.e., a small fraction) from the 3D space, and finds the appliance label point closest to the center of the captured image. Several aspects make image localization a better fit for our target application compared to image retrieval. First, the system deployment process is extremely simple because a 3D model that covers various angles can be built by capturing a video with a modern modeling tool, such as Project Tango [68], while walking through a building. Labeling on a 3D model is also simple because every appliance only needs to be labeled once as a single point in the 3D space, irrespective of the number of images containing that appliance in the database. Second, because a 3D model incorporates all image features into

[3] Given that each label of the vision-based identification system must be translated to its appliance ID in the BMS (metadata server), the labels must be added by professional building managers/staff who have knowledge of the BMS instead of crowdsourced from building occupants.

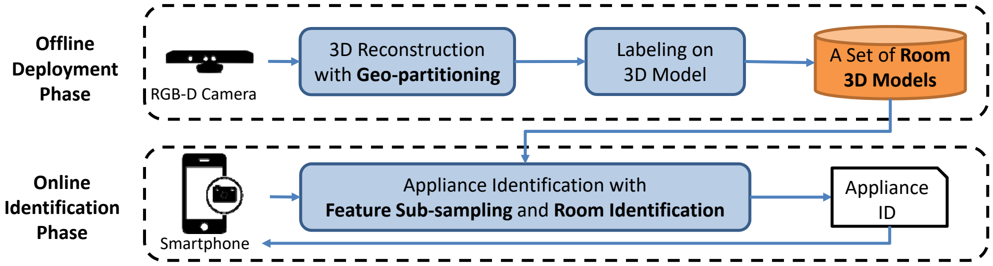


Figure 4.4: SnapLink overview. The offline deployment phase builds a 3D point cloud with a set of partitioned 3D models and labeled appliances. The online identification phase recognizes an appliance instance through an image localization process including feature sub-sampling and room identification based on the set of 3D room models.

a 3D space, it can represent all angles and distances, resulting in a higher identification accuracy with an arbitrary query image compared to image retrieval. As an example, in a room that contains 100 appliances, each of which needs to be captured from 10 different viewing locations, image retrieval would require 1,000 reference images, whereas image localization only requires a video and 100 labeled points. Thus, our system uses image localization for vision-based appliance instance identification.

However, a naive approach to image localization is not suitable for our target application due to a number of problems. Constructing a properly functioning 3D model for a large commercial building involves many unsolved problems, such as failure to identify a previously visited place (loop closure detection [69]) caused by cumulative errors while constructing a 3D model. Furthermore, image localization imposes a significant computational burden, resulting in unsatisfactory latency according to our system requirements. To apply image localization to our target application, these issues need to be addressed, which is the focus of our SnapLink design in Section 4.3.

4.3 SnapLink System Design

In this section we introduce the details of the SnapLink design, which addresses three issues of image localization to support our target application: (1) how to construct a properly working 3D model in a large building (i.e., *geo-partitioning*), (2) how to identify appliances through image localization with a given 3D model, and (3) how to minimize the computation time without losing accuracy (i.e., *feature sub-sampling*). We first give an overview and then introduce the details of each design element.

4.3.1 SnapLink Overview

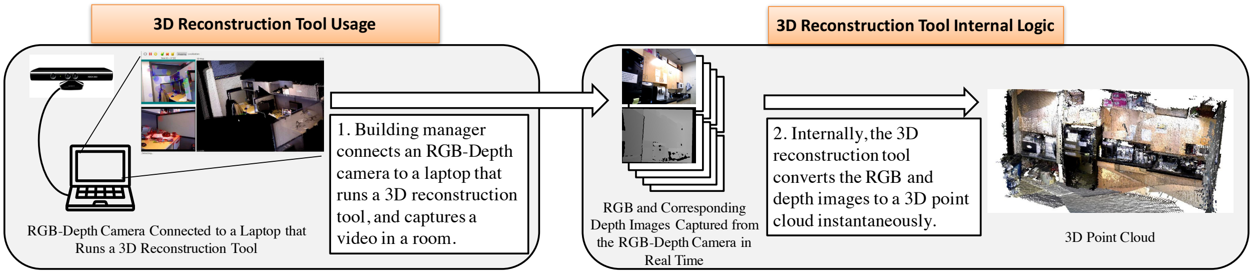
Figure 4.4 shows an overview of SnapLink, which comprises the *offline deployment phase* and the *online appliance identification phase*. In the deployment phase, a user or a building manager needs to capture videos inside the building using an RGB-Depth (RGB-D) camera. In doing this, SnapLink uses *geo-partitioning* to enable easy and scalable 3D model construction without cumulative errors [69]; it uses a set of room-based 3D models, not a single building 3D model, to represent the whole building. Each video for a room is fed into a standard 3D reconstruction tool, which generates a 3D point cloud for the room. Then, the manager labels each appliance point within each room's 3D model and stores these labeled 3D models in the database. We also build our own labeling tool to simplify the labeling process.

In the appliance identification phase, SnapLink localizes a query image in a set of 3D models in the database to infer the labels that are visible in the image. On top of the standard image localization process, SnapLink adds two pre-processing stages before image localization: *feature sub-sampling* and *room identification*. The feature sub-sampling stage *randomly* selects features in the query image to reduce computation time while still providing adequate information for correct localization. The room identification stage is necessary to operate with the database partitioned by room.

4.3.2 Building a 3D Model Database with Geo-partitioning

In the offline deployment phase, we build a 3D model of the building and label appliances for the runtime appliance identification phase. To model a building, we collect a separate video for each room. This partitions our database into a collection of 3D room models, which is critical for a successful building model for several reasons. First, it simplifies the database construction process. We need to capture only a short video to model a room, which takes about 100 seconds on average in our case. When a room's environment is changed (e.g., furniture is rearranged or the room is remodeled), a building manager is required to recapture a video only for that room, not the whole building. Our 64-day evaluation described in Section 4.5.9 demonstrates that a room model is robust to everyday changes (e.g., content on whiteboard, items on table) as long as the appliance itself is still present. Second, geo-partitioning reduces cumulative errors in the 3D reconstruction process because it is applied for each room (small scale). Note that 3D reconstruction, which converts a video captured by an RGB-Depth camera to a 3D point cloud, is a necessary procedure for building an image localization database. Also note that it causes cumulative errors at a large scale due to drifting and loop closure detection failures [69]. Third, since geo-partitioning constructs the whole building dataset as a group of room datasets, the whole dataset can easily be hierarchically partitioned (e.g., to floor datasets). This enables the system to use the closest local server (e.g., a floor-specific server) for a partitioned dataset, instead of a single server for the whole dataset, which reduces network latency in a large scale building. The closest local server can be identified by using localization methods, such as WiFi fingerprints or GPS.

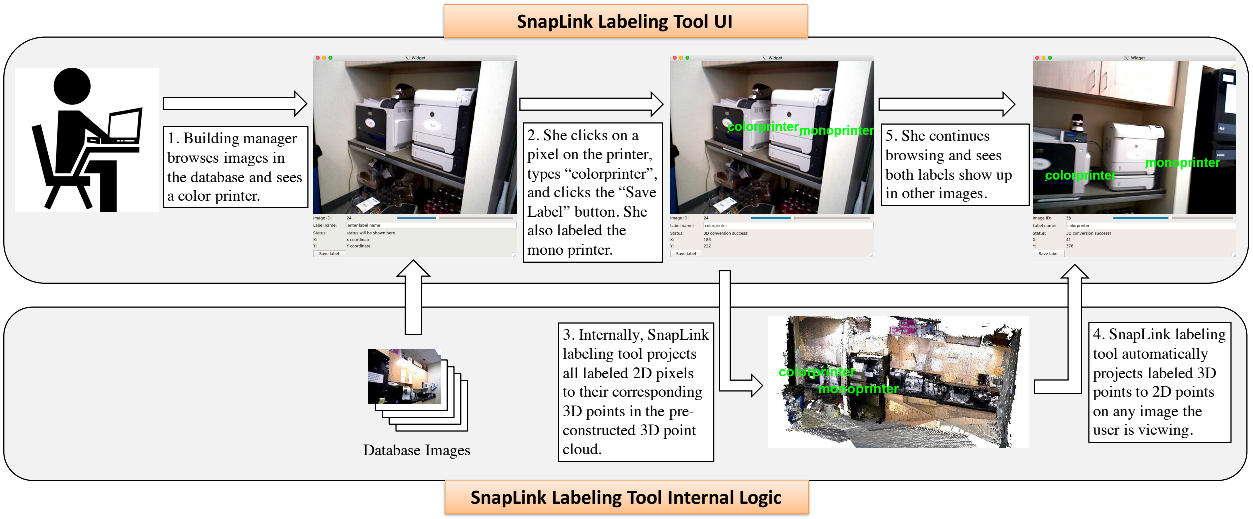
Figure 4.5 shows how a building manager can construct a 3D room model. She simply needs to connect a commercial off-the-shelf RGB-Depth camera to a laptop, run a 3D reconstruction program (e.g., Project Tango [68] or RTABMap [70]) on the laptop, and


3D Reconstruction Tool Usage
RGB-Depth Camera Connected to a Laptop that Runs a 3D Reconstruction Tool
3D Reconstruction Tool Internal Logic
RGB and Corresponding Depth Images Captured from the RGB-Depth Camera in Real Time
3D Point Cloud

1. Building manager connects an RGB-Depth camera to a laptop that runs a 3D reconstruction tool, and captures a video in a room.

2. Internally, the 3D reconstruction tool converts the RGB and depth images to a 3D point cloud instantaneously.

Figure 4.5: An example of SnapLink's 3D modeling. Using an off-the-shelf RGB-Depth camera, a building manager captures a video of a room, which is automatically converted to a room 3D model by a 3D reconstruction program, such as RTABMap [70].


SnapLink Labeling Tool UI
colorprinter
monoprinter
Database Images
SnapLink Labeling Tool Internal Logic

1. Building manager browses images in the database and sees a color printer.

2. She clicks on a pixel on the printer, types "colorprinter", and clicks the "Save Label" button. She also labeled the mono printer.

3. Internally, SnapLink labeling tool projects all labeled 2D pixels to their corresponding 3D points in the pre-constructed 3D point cloud.

4. SnapLink labeling tool automatically projects labeled 3D points to 2D points on any image the user is viewing.

5. She continues browsing and sees both labels show up in other images.

Figure 4.6: An example of SnapLink's labeling process. A building manager can label an appliance by clicking on a pixel and typing a label while browsing captured images. Multiple appliances can be labeled in a single image. All labels are projected to the 3D space and therefore show up in all images containing it.

point the camera at appliances while walking through the room. With the captured room images, the 3D reconstruction program automatically computes the relative 3D transformations between similar image pairs by using the three-point-algorithm inside a RANSAC loop [71], projects the images onto a 3D space, and creates a list of 3D points, resulting in a 3D point cloud of the room.

Once a 3D point cloud is constructed for a room, a building manager needs to label each appliance in the room only one time on a 3D point in the 3D point cloud. To simplify this process, we built a labeling tool as depicted in Figure 4.6. A building manager browses room images captured by the camera. When she finds an appliance in an image, she clicks on any point on the appliance, types its label, and clicks on the "Save Label" button. Since

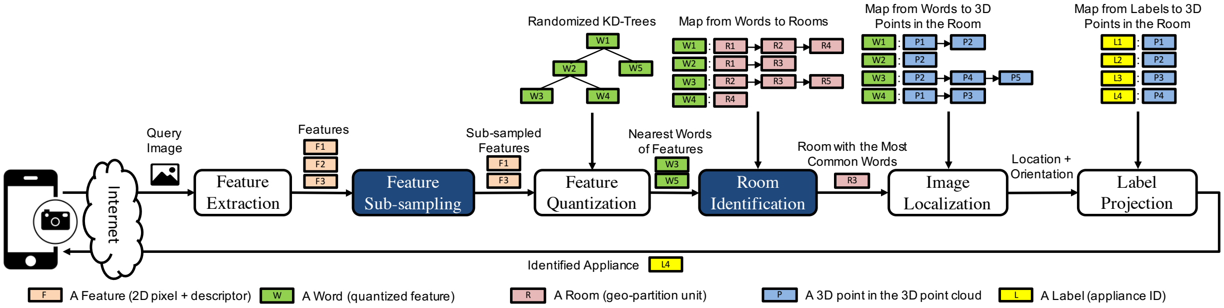


Figure 4.7: SnapLink appliance identification pipeline. The extracted query image features are subsampled and quantized into words using KD-trees. Subsequently, room, image location and orientation are identified. Finally an appliance is identified by projecting labeled 3D points onto the query image.

our labeling tool can match each 2D point in a captured image to a 3D point in the 3D point cloud, it automatically labels the appliance on the corresponding 3D point. While she browses further, the tool displays the appliance label if the browsed image includes the labeled point, which prevents her from relabeling the same appliance. Furthermore, she is allowed to label multiple appliances in a single image (e.g., "colorprinter" and "monoprinter" in Figure 4.6), which is not the case for image retrieval because each image is expected to contain only a single appliance instance. Undoubtedly, there are many possible ways to simplify the labeling process, such as using voice recognition to label appliances in a smartphone camera view.

4.3.3 Identifying Appliances in Partitioned 3D Models

Figure 4.7 depicts the appliance identification pipeline of SnapLink. As in the standard image localization approach, SnapLink first extracts salient features from the query images. We use SURF for its speed and accuracy [40, 57]. However, unlike previous work that uses all features, we only use randomly sub-sampled features to achieve lower latency without sacrificing accuracy, which we discuss in Section 4.3.4. We then quantize the sub-sampled features to their closest cluster among many clusters of features, whose centers are dubbed *words* [72]. These clusters are generated from all database features in advance using the distance ratio test [40]. We use randomized KD-Trees of these words to find the nearest cluster of a feature, because it outperforms other approaches in both speed and accuracy on visual feature descriptors [73]. The feature quantization greatly reduces the computation in later steps. We use an empirically optimal distance ratio threshold 0.7 and use the default of 4 randomized KD-Trees, as implemented in FLANN [73].

As we discussed in Section 4.3.2, to overcome problems in large scale 3D reconstructions, our database consists of 3D models of separate rooms. Consequently, we need to identify the room before we can localize the query image. We define the similarity between the query image q and room r as

$$s_{q,r} = \frac{\left|W_q \cap W_r\right|}{|W_r|} \tag{4.1}$$

where W_q is the set of words in image q, and W_r is the set of words in room r. To quickly compute $\left|W_q \cap W_r\right|$, we tally all words for room r contained in query image q. Then we compute $s_{q,r}$ for every room and find the one with the highest similarity. Note that this is done efficiently in one traversal of all words in image q, maintaining a running total for each room r and selecting the room with the highest total once all words have been processed. This approach is also described in [74].

With a room r identified, we can localize the query image q within it. We solve this as a standard Perspective-N-Point (PnP) [75] problem with a list of 2D points in image q and their corresponding 3D points in room r. To get the list, for every 2D SURF point p in image q, we find its closest 3D point whose SURF descriptor is in the same cluster of p. Specifically, we compute

$$\underset{i, w_p = w_i^r}{\operatorname{argmin}} \|\mathbf{f}_p - \mathbf{f}_i^r\| \tag{4.2}$$

where w_p and $\mathbf{f}_p$ are the word and SURF descriptor of p, and w_i^r and $\mathbf{f}_i^r$ are the word and SURF descriptor of the i^{th} 3D point in r. In addition, we perform two optimizations when obtaining the list, inspired by [56]. First, we start our search with 2D points whose SURF descriptors are in smaller clusters to reduce noise. Second, we limit the size of the list to reduce redundant information. We use the empirically optimal value of 100, as in [56], for our list size limit.

After we know the location and orientation of the query image, we can compute where every appliance appears in the image by projecting their labeled 3D points onto the query image plane. Our server returns the appliance ID whose label is closest to the center of the query image. As described in Section 4.2, the mobile client can then query for a control interface from the BMS server using the identified appliance ID to enable user interactions with that appliance.

To examine the placement of computation for our identification pipeline, we conducted two similar experiments as in [57]. The first experiment measures the computation time of SURF [40] using OpenCV [76] on a server GPU, server CPU, and smartphone CPU. The second experiment measures the transmission time of an image from a smartphone to four servers at different locations: on campus, San Jose, Virginia, and Ireland. Figure 4.8 and Figure 4.9 show the respective results. As we can see, it takes 70 ms on average to transmit an image and compute SURF on a server GPU, but more than 1000 ms to compute SURF on the phone. Figure 4.9 also substantiates the necessity of geographically partitioning the database in order to run computation nearby and reduce end-to-end latency. In SnapLink, we therefore offload all appliance identification computation to a server that is physically close.

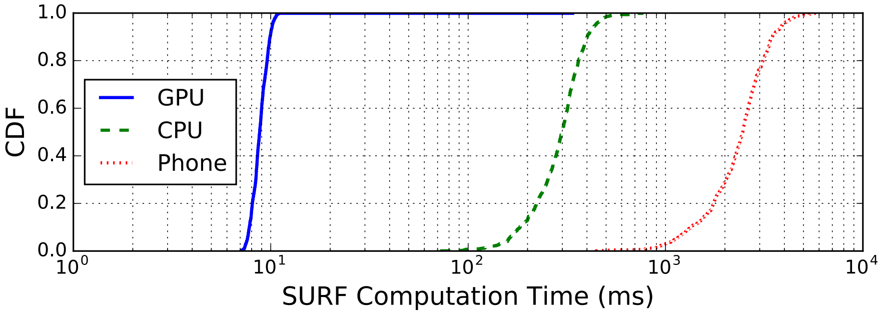


Figure 4.8: CDF of SURF computation time of a 640 × 480 image on a server GPU (NVIDIA GeForce GTX 970), a server CPU (Intel i7-4790), and a smartphone CPU (Qualcomm Snapdragon 400).

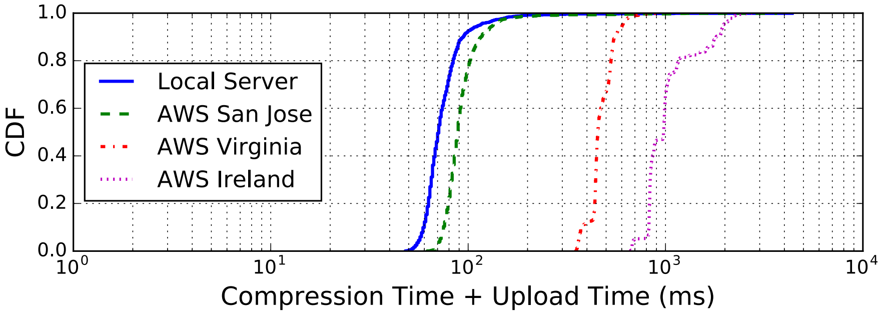


Figure 4.9: CDF of JPEG compression time + upload time of a 640×480 image to different servers: on campus, in San Jose (43 miles away), in Virginia (2350 miles away), and in Ireland (5050 miles away).

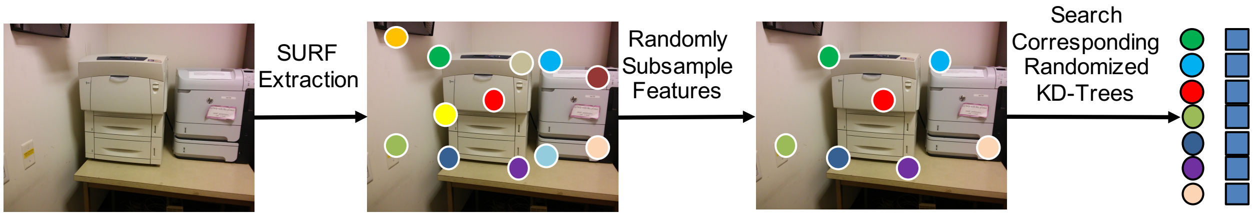

SURF Extraction
Randomly Subsample Features
Search Corresponding Randomized KD-Trees

Figure 4.10: Feature Subsampling

4.3.4 Feature Sub-sampling

Since appliance identification is an interactive process, we aim to reduce the end-to-end latency to an imperceptible range. Even though it is not practical to achieve our 100 ms response time goal [77] while network transmission takes 80 ms on average (as shown in Figure 4.9), we try to minimize server computation without sacrificing accuracy.

Table 4.1 shows the time complexity of every step in the standard image localization process. Feature extraction complexity depends on the content of the query image, but can

Stage	Mean Time Complexity	Note
Feature Extraction	Depends on image	Can run on GPU.
Feature Quantization	$O(F \log W)$	F: № of features, W: № of words.
Room Identification	$O(F + R)$	F: № of features, R: № of rooms.
Image Localization	$O(P)$	P: № of 2D-3D point pairs.
Label Projection	$O(L)$	L: № of labels.

Table 4.1: Time complexity of every stage in the standard image localization pipeline. Note that F, R, P, and L are usually several orders of magnitude smaller than W.

usually finish in 10 ms when parallelized on a GPU, as shown in Figure 4.8. The average KD-Tree search time in the feature quantization stage is $O(\log F)$, where F is the number of features. However, because of the recursive nature of KD-Tree search, it cannot benefit from GPU parallelization, especially when data dimensionality is high [78], such as SURF descriptors. Since feature quantization reduces image features to a much smaller set of words, it makes room identification, image localization, and label projection consume little time. Therefore, feature quantization is the bottleneck of the pipeline.

KD-Tree search time increases linearly with the number of features. Therefore we try to use a subset of the image features. A 640 × 480 image can generate hundreds to thousands of features, but not all of them are useful. Conceptually, the features that are both unique and robust are the most useful because they help match 2D points to 3D points with less ambiguity. Jain, et al. [79] use a counting bloom filter to get a uniqueness score for every word in the query image. However, this approach only works after features are quantized to words.

Instead of trying to find unique and robust features, we propose a simple random feature sub-sampling scheme. We argue that enough randomly sub-sampled features can provide the requisite information for image localization, while keeping the KD-Tree search time low. Furthermore, by introducing a tunable number of sub-sampled features, the system can adjust the trade-off between accuracy and latency based on the average KD-Tree search time, which depends on the number of words in the database. Similarly, we can use this mechanism to adjust end-to-end latency based on current network transmission time. Our evaluation in Section 4.5.4 shows the effectiveness of this approach.

4.4 System Implementation

In this section, we describe our implementation of the labeling tool, appliance identification runtime, and Android mobile client.

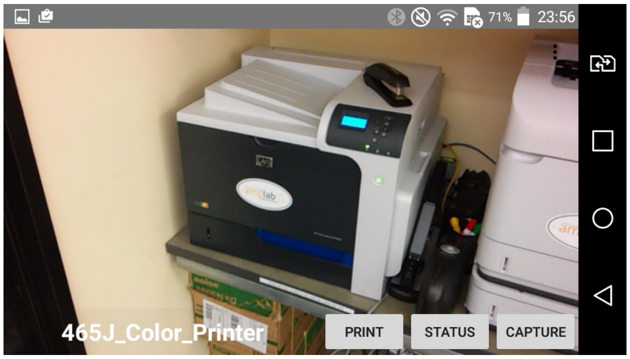


Figure 4.11: SnapLink android application. The capture button sends a new image to a server. When an appliance is identified, it displays its name and control widgets retrieved from a BMS server.

4.4.1 SnapLink Server

For the deployment phase, we choose to use RTABMap because it is open source and supports various types of hardware (e.g., stereo camera, RGB-Depth camera). We also implemented a SnapLink labeling tool in 400+ lines of C++ code, using Qt as the GUI library. It currently reads RTABMap [70] databases and presents images to users for labeling. Labels are saved to the original database file. In the future, we plan to add adapters to other popular 3D reconstruction tools, such as Google Tango [68] and Bundler [80].

We implemented our SnapLink runtime in 3200+ lines of C++ code. It receives queries in a RESTful interface using HTTP. We use the Qt event system to construct the image localization pipeline. In the pipeline, we use OpenCV [76] to extract SURF, solve perspective in point (PnP) problems, and project points between 2D and 3D. We use the randomized KD-Tree implementation in FLANN [73] and transformation utilities from PCL [81].

4.4.2 Android Mobile Application

We developed an Android client using 1700+ lines of Java code. Figure 4.11 shows an example screenshot. It has a full screen see-through camera view with control widgets overlaid on top. A capture button is at the bottom right. The bottom left shows the currently identified appliance. Appliance control buttons are shown at the bottom center.

To minimize the identification latency, we disabled auto focus, auto exposure, and auto white balance, which can take up to 1 second to complete if enabled. However, if a user clicks on the capture button while moving the smartphone, it will create a blurry image that deteriorates salient feature extraction performance [39], and the user can choose to capture another still image if she sees an identification failure. Several approaches have been proposed to detect blurry images, such as using a Canny edge detector [57] or gyroscope readings [82]. We are currently implementing blurry image detection in the app, which will either ask the user to capture another image or automatically find a clear image in the image view stream. After the client sends a clear image and receives the identified appliance ID from a SnapLink server, it retrieves its control interface and metadata from BOSS [1] and updates the UI accordingly. In addition, it uses the secure publish/subscribe overlay network [83] designed for BOSS to send control commands.

4.5 Evaluation

In this section, we first compare the accuracy and latency of image retrieval, image localization, and image localization with feature subsampling. Then we examine the scalability of SnapLink when database size increases as well as its robustness for identifying appliance instances in the same category, viewing angles, distances, illumination conditions, and changes in the environment. We also conduct a micro-benchmark and a field study to show how QR codes and SnapLink can compensate each other. Finally, we examine failure cases as well as the energy consumption of our Android mobile client.

4.5.1 Experimental Setup

We use an RGB-Depth camera (an ASUS Xtion PRO LIVE in some rooms, and a Kinect in others) to capture videos and use RTABMap [70] to create 3D models. Our dataset contains videos of 39 rooms across 5 buildings in two university campuses, and 4 of the rooms are captured by a first-time user. The rooms include conference rooms, office cubicles, lounges, kitchens, hallways, etc. We estimate the average room size to be 150 sq. ft., meaning our deployment covers about $5,850$ sq. ft. of total space, which is about the size of a typical small office building [59]. To our knowledge, our dataset covers a much larger space than those in previous vision-based object identification systems [55, 57, 58]. Our videos are sampled at 1 frame per second, and our entire dataset contains 4034 images, which means we can capture a room in 100 seconds and an entire small office building in about 67 minutes.

After recording a video, we use Android phones (an LG G2 Mini in some rooms, and a Nexus 5x in others) to capture 480×640 (or 640×480) test pictures of appliances from different angles and distances. The appliances are arbitrarily selected from all appliances and objects in the rooms, including both controllable objects (e.g., lights) and non-controllable objects (e.g., bookshelves) in our evaluation. Note that if we want to include more appliances from the captured rooms later, there is no need to go back and capture extra data (as required in image retrieval), since they are already included in our 3D models. In total, we added 263 labels for 179 appliances (some large appliances are labeled on multiple points, but will work with only one) using our labeling tool, and collected 1526 images for testing.

Our setup includes a SnapLink server running in a docker container on an Ubuntu machine, which has an Intel Xeon E5-2670 CPU and 256 GB of memory. We run a Python script as a client in the same docker container to send test images to the server over HTTP.

To compare SnapLink with other vision-based systems ([55, 57, 58]), we also implemented an image retrieval based system, which simply finds the image with the most salient features in common with the query image. Specifically, we find the image p such that

$$p = \underset{i}{\operatorname{argmax}} \left| Words_i \cap Words_q \right| \tag{4.3}$$

where $Words_i$ and $Words_q$ are feature words of image i and the query image, respectively. To achieve efficient search, we build a word-image map, then vote and tally all database images while iterating through all words in the query image as described in [74]. It uses the same database we collected, but with every image labeled by the appliance closest to the image center.

4.5.2 Metrics

Our evaluation focuses on two figures of merit: accuracy and latency. Since users capture a query image of an appliance to initialize interactions, we argue that our system performance is more important for test images containing an appliance, as opposed to those containing no appliance (e.g., a blurry image captured by accident, or a white wall). Therefore, we define accuracy as *recall*:

$$\text{Accuracy} = \frac{\text{Number of Correctly Identified Images}}{\text{Number of Images Containing an Appliance}} \tag{4.4}$$

When a test image contains multiple appliances, we regard the appliance closest to the center of the image as the ground truth, which is also how the SnapLink server identifies the target appliance when multiple appliance labels are visible in the query image (Section 4.3.3).

Given that we have already showed the average local network latency to upload an 640 image is 70ms in Figure 4.9, we only focus on server side latency in our evaluation.

4.5.3 Accuracy and Latency

We first look at the accuracy and latency of image retrieval, image localization, and image localization with feature subsampling, by running and testing these three configurations on all our data. We choose to subsample 300 features, which is empirically optimal as shown in Section 4.5.4. Figure 4.12 shows the average accuracy as well as the average server processing times with standard deviations for each technique. Image retrieval only yields 66% accuracy with 177 ms average computation time, whereas image localization yields 94% accuracy with 198 ms average computation time. After applying our feature subsampling mechanism, SnapLink maintains 94% accuracy while reducing the computation time by 39% to 120 ms with a smaller standard deviation. This time reduction significantly helps to push the end-to-end latency down to an imperceptible value.

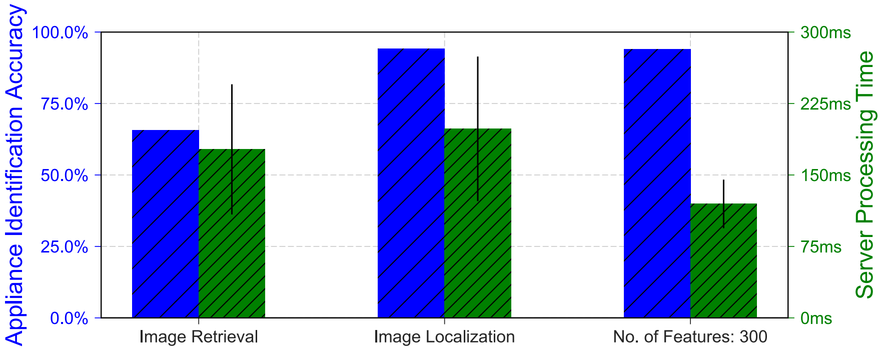


Figure 4.12: Accuracy and identification time for image retrieval, image localization, and image localization with feature subsampling.

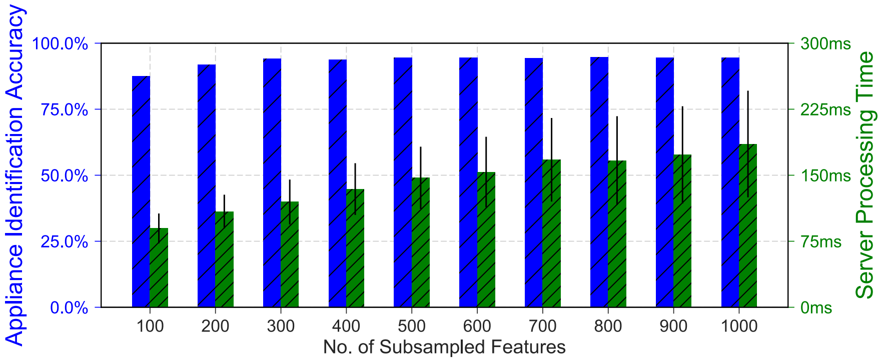


Figure 4.13: Accuracy and identification time for different numbers of subsampled features. Accuracy does not further improve when the number of features exceeds 300.

4.5.4 Feature Subsampling

To further study how feature subsampling influences the accuracy and latency, we run image localization using our entire database while varying the number of subsampled features per image. Figure 4.13 shows that the accuracy saturates at 94% with 300 features subsampled for the kind of appliances we need to recognize in building environments, and adding more features only increases the server computation time. This shows that we do not need all features from the query image. Therefore, we choose to subsample 300 features in our system, as well as in the following evaluations.

4.5.5 Scalability

One of our design goals is to make SnapLink scale to the size of a commercial building. To further understand how feature subsampling reduces our server processing time, we run image localization with and without feature subsampling on different numbers of rooms and test with images from the respectively used rooms. Figure 4.14 shows a breakdown

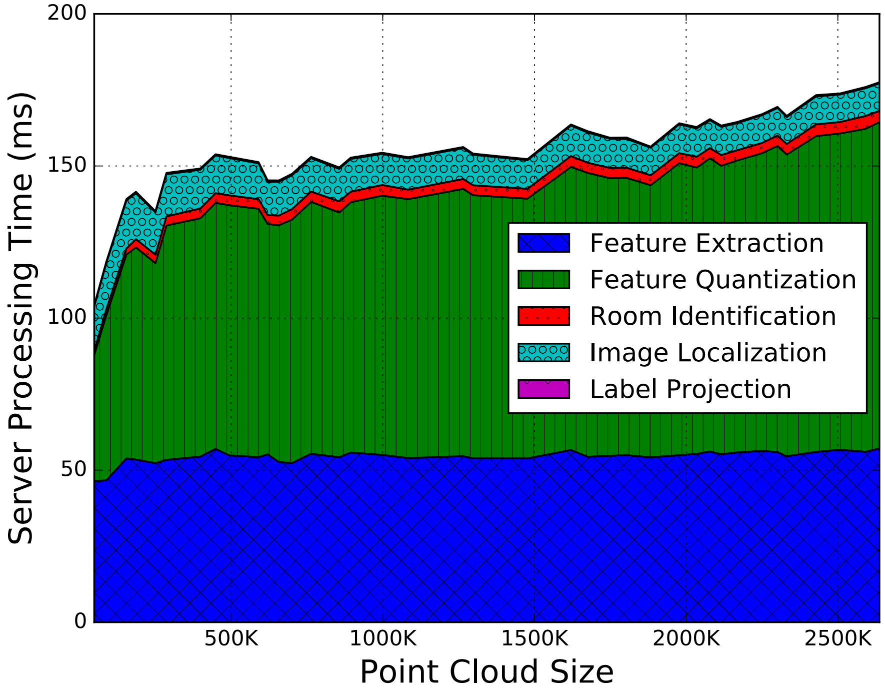


Figure 4.14: Time breakdown vs. the size of point cloud using image localization without subsampling

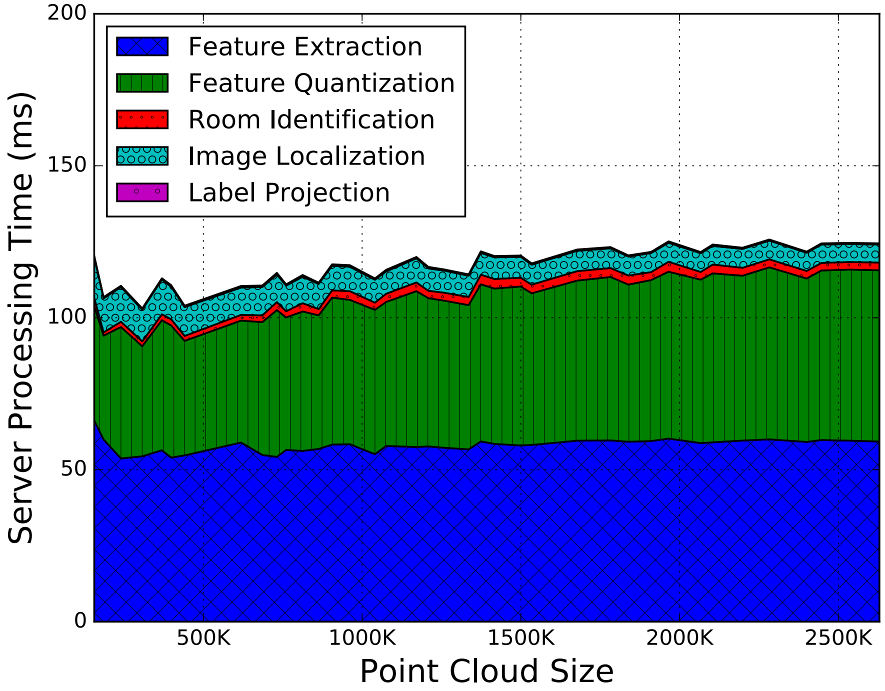


Figure 4.15: Time breakdown vs. the size of point cloud using image localization with 300 features subsampled

of the server processing time when performing image localization without subsampling for different sizes of 3D point clouds. The point clouds are created by selecting different combinations of rooms from the overall database. SURF takes about 50 ms, but can easily be reduced to less than 10 ms by using a GPU. Because we perform feature quantization, room identification and image localization both take a small amount of time. Label projection also takes a negligible amount time because of the small number of labels. However, because of the nature of KD-Trees, the feature quantization time increases logarithmically with the number of words. This generally increases as the size of the point cloud increases. This results in noticeable latency in the appliance identification process, which gets worse when deployed with more data in a larger building.

Figure 4.15 shows that feature subsampling effectively reduces the feature quantization time when the data size increases. It provides a tunable parameter to choose between accuracy and latency. When the data size gets larger, we can reduce the number of subsampled features to further reduce the latency, while sacrificing a small amount of accuracy. In future work, we plan to explore different subsampling strategies to select "better" features (rather than random ones) before the quantization.

4.5.6 Instance Recognition In Categories

Since a building contains many instances of appliances from the same category, it is important for SnapLink to distinguish instances among the same category with image localization. To study how SnapLink performs, we select the six most common categories among all 179 labeled appliances and summarize their identification accuracy among all instances in the category in Table 4.2. SnapLink achieves more than 95% accuracy for four out of the six categories and 89.2% for projector. However, it can only get 77.1% accuracy when identifying lights. We investigated this in our dataset, and found that many lights were turned on when we built the 3D model, which caused the image to underexpose, such as in Figure 4.16. This likely degrades the feature extraction performance of the image. In addition, many lights are located on the ceiling, where few distinct visual features can be found. In future work, we plan to add continuous trajectory estimation between query images using smartphone motion sensors to mitigate this problem.

4.5.7 Angle and Distance

Since users can control appliances from arbitrary locations, SnapLink should be robust to different viewing angles and distances. To evaluate this, we select two sufficiently large lounges (i.e., Lounge 1 and Lounge 2, both in the 39 rooms in our dataset) and create a 3D model of each. Each lounge has three appliances labeled, one of which is our target appliance in this evaluation. We capture test images (50 per location in Lounge 2, and 10 in Lounge 2) of the target appliance from 5 different angles with 30° intervals, each with different distances with 50 cm intervals (Because Lounge 1 is not spacious enough, we cannot take images from more than 200 cm except when the viewing angle is 0°). All test

Appliance Category	№ Instances	Accuracy in Category
Printer	25	99.1%
Monitor	15	98.3%
Projector	12	89.2%
Light	12	77.1%
Wall Clock	10	98.7%
Microphone	9	95.8%

Table 4.2: Summary of instance identification accuracy among the six most common categories. SnapLink achieves high accuracy among each category except lights, which suffer from underexposure and a lack of visual features present on ceilings.

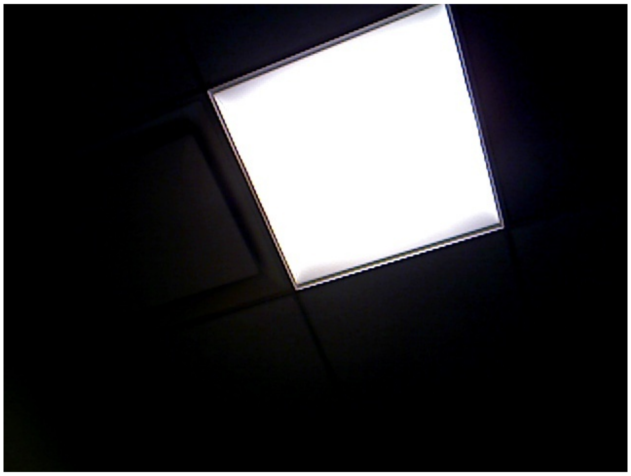
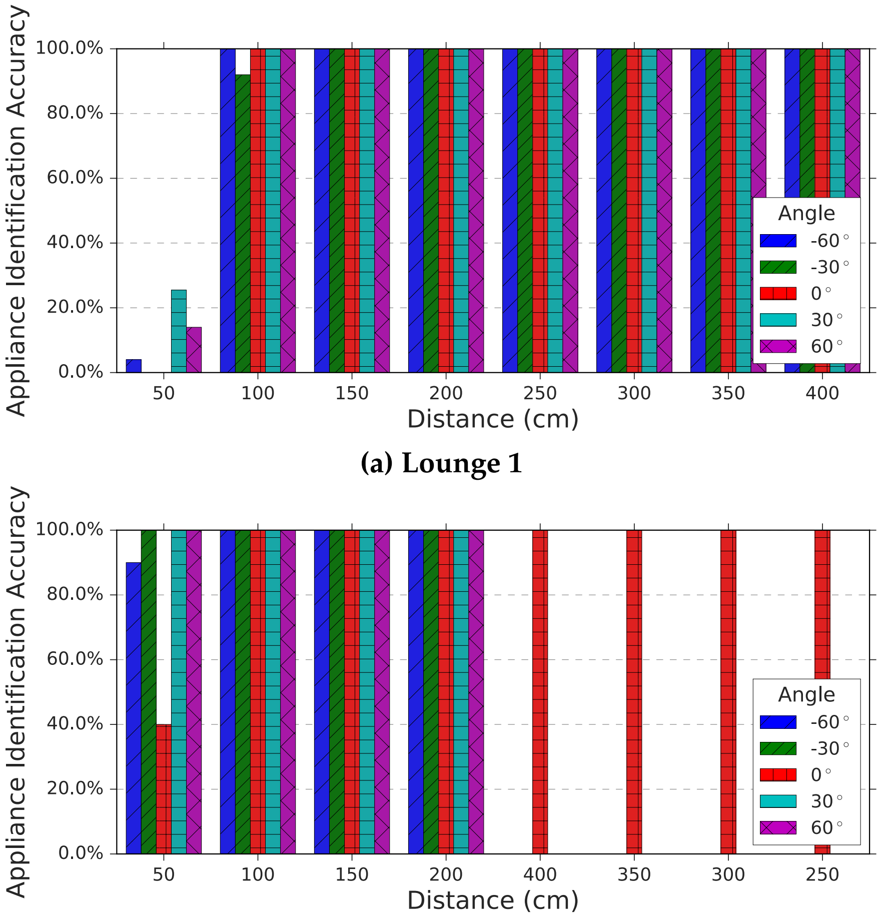


Figure 4.16: Example underexposed image caused by a light, which can degrade feature extraction.

images are sent to a SnapLink server to be identified among the three appliances in the lounge along with all appliances in the other 38 rooms in our dataset. All tests are done with 300 features subsampled.

Figure 4.17 shows the identification accuracy at all locations. SnapLink only experiences significant failures when the viewing distance is small (i.e., ≤ 50 cm). The main reason is that when the query is taken from a close distance, not enough features may be captured, especially those providing distinctive contextual information. We argue that this will not happen often in practice, because most times users try to control appliances outside their arm range, such as a ceiling light or a projector. Another option to solve this problem is the attachment of a visual marker (e.g., QR code) to appliances that are usually within an occupant's arm range (see section 4.5.10).



(a) Lounge 1

(b) Lounge 2

Figure 4.17: Identification accuracy at different angles and distances with Image Localization and 300 Features Subsampled. Query images close to the appliance tend to fail because they contain less visual context. Note there is no data from oblique angles when distances > 200 cm in Lounge 2 due to limited physical space.

Testing 3D Model	Lights On	Lights Off
Lights On + 39 Rooms	90.4%	93.7%
Lights Off + 39 Rooms	91.9%	100%

Table 4.3: Identification accuracy when images are tested against the 3D model and the 39 rooms in our dataset. It shows SnapLink is robust to illumination changes.

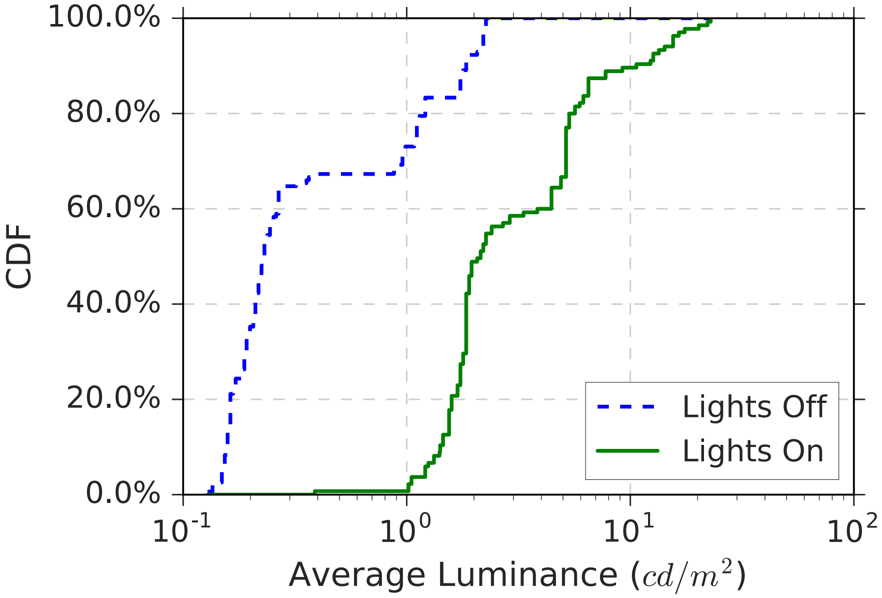


Figure 4.18: CDF of average scene luminance (cd/m^2) ($C = 1$) when lights are on and off.

4.5.8 Illumination Conditions

The illumination conditions in a building can be affected by various factors, such as lights turning on and off and window shades being opened and closed. SnapLink needs to be robust to common illumination changes when deployed. We conduct an experiment in an office room with 5 appliances (2 monitors, 2 speakers, and 1 fan) under two different illumination conditions: all lights on (bright) and all lights off (dark), while all window shades are put down. Under each illumination condition, we capture a video to construct a 3D model, and capture at least 25 test images of each appliance from different viewing angles and distances.

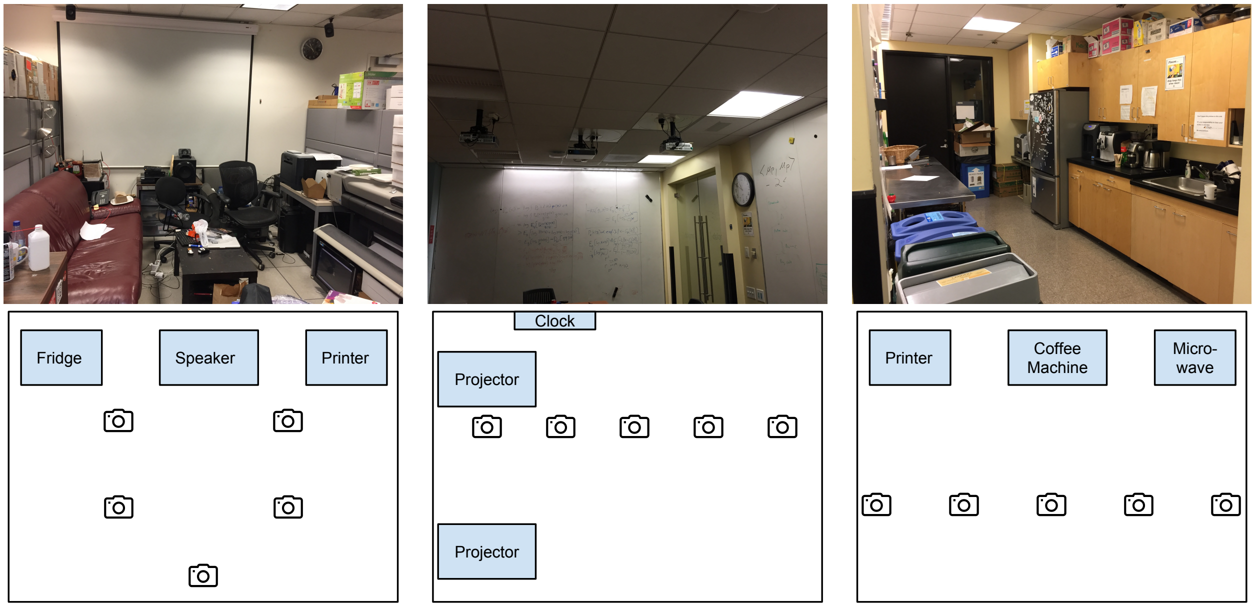
To quantify the illumination condition when lights are on and off, we compute the average scene luminance (in cd/m^2) from the EXIF data of the test images using the following equation (described in [84]):

$$L = C\frac{k^2}{S \cdot t} \tag{4.5}$$

where L is the average scene luminance (in cd/m^2), k denotes the f-number, S denotes the ISO speed setting value, t denotes the exposure time (in seconds), and C is the hardware-dependent reflected-light meter calibration constant. k, S, and t can be found in the EXIF data, and we arbitrarily set C to be 1. Since we are only comparing relative luminances captured using the same camera, the value of C can be an arbitrary constant. Figure 4.18 shows the CDF of the average luminance (when $C = 1$) with a log-scale x axis. We can see that the scene is about 10 times brighter when lights are on than off.

Room	Appliances
Lounge	1 Fridge, 1 Speaker, and 1 Printer
Conference Room	2 Projectors, 1 Clock
Kitchen	1 Microwave, 1 Coffee Machine, and 1 Printer

Table 4.4: Summary of appliances in the changing environment experiment.



(a) Lounge **(b) Conference Room** **(c) Kitchen**

Figure 4.19: Changing environment experiment setup. Each room contains 3 appliances and is shared by many people in a commercial building, where changes to the environment happen every day. We capture 20 test images of each appliance at each capture location every day (57,600 images in total).

Table 4.3 summarizes the identification accuracy when test images are tested against a 3D model and the 39 rooms. Test images captured under different illumination conditions do not yield worse identification accuracy than those captured under the same illumination condition. This is because of the robust nature of salient features [40] as well as the auto exposure adjustment in modern smartphone cameras. Note that dark test images have higher identification accuracy than the bright test images in the bright 3D model, which is because the two test image sets are taken from different arbitrary angles and distances.

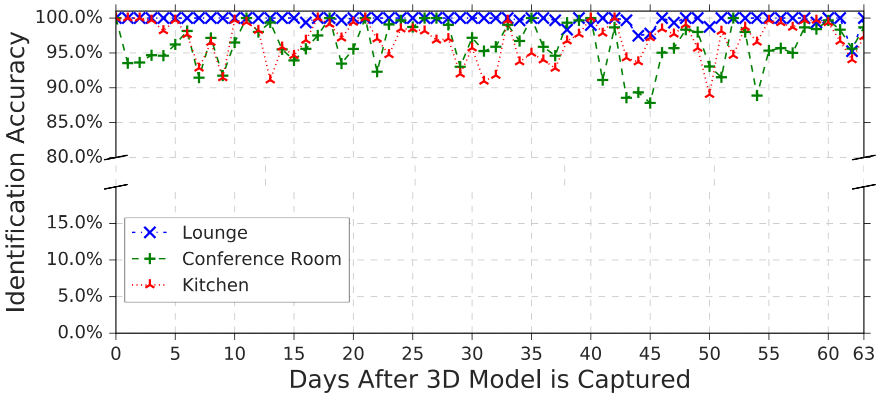


Figure 4.20: Identification accuracy over time as the environment changes. The accuracy fluctuates caused by everyday changes in the environment and occasional blurry test images. We see no clear trend of degrading accuracy, meaning the database can be used for a longer period of time.

4.5.9 Changes in the Environment

To show how SnapLink is robust to everyday changes after a 3D model is captured, we construct 3D models of 3 rooms (which are among the 39 rooms in our dataset) containing 9 appliances, and evaluate SnapLink every day thereafter for 64 consecutive days. Table 4.4 summarizes the 3 rooms and 9 appliances, and Figure 4.19 shows pictures of the 3 rooms, which are all shared by many occupants in a commercial building and naturally change all the time without our intervention. As examples, actively changed items include: objects on the coffee table and chairs in the lounge, content on the whiteboard and projector screen in the conference room, and items on the table and countertop in the kitchen. To ensure images captured every day have the same distribution in terms of viewing angles and distances, we capture 20 test images of every appliance from five locations in each room, with minor viewing point movements in the arm's range between images. The lower part of Figure 4.19 shows test image capture locations.[4] In total, this experiment generates 57600 test images (20 images ×5 locations ×3 appliances ×3 rooms ×64 days). To reduce the authors' bias, we also asked a volunteer to collect about half of all the test images, captured on dates when they were available. While identifying appliances in the test images, out search space also include all appliances from the remaining 36 rooms.

Figure 4.20 shows the identification accuracy for each of the three rooms as an increasing number of days have elapsed since the initial 3D models were captured. SnapLink's accuracy is mostly above 90% during the entire experiment, but varies day by day due to environmental changes and varying quality in the captured test images. For example, the dip in accuracy for the conference room from day 43 to day 45 is caused by blurry test images. Furthermore, our 64-day experiment shows no clear trend of degrading accuracy,

[4] The room dimension ratios are not to exact scale.

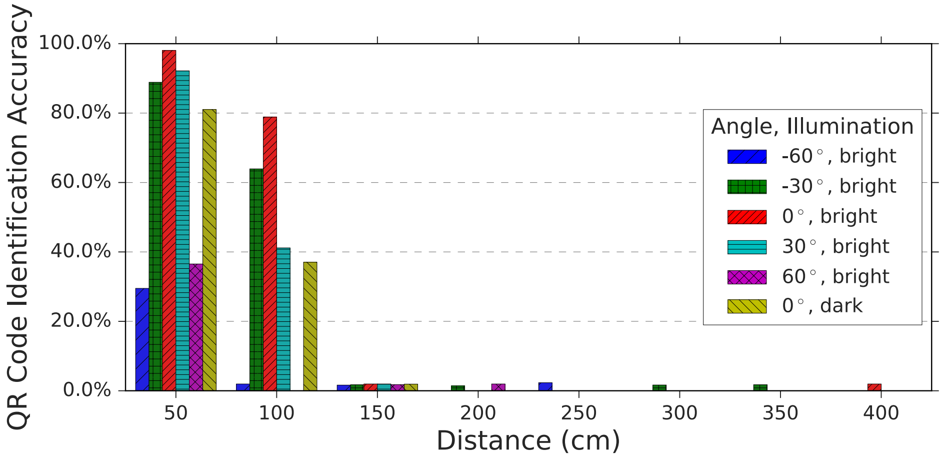


Figure 4.21: QR code recognition rate at different viewing distances and angles, as well as under different illumination conditions.

which suggests that SnapLink can retain high accuracy throughout an even longer period. This robustness comes from two factors: (1) the RANSAC algorithm that we use while solving the PnP problem, and (2) the environments did not change dramatically, which we argue is generally true for most commercial buildings.

4.5.10 Comparison with QR Codes

QR Codes are two-dimensional machine-readable barcodes that are quick to decode with error correction. They form the basis of a very popular technique for object identification. We use a micro-benchmark to show the advantages and disadvantages of QR codes for our application. We also show how QR codes can help improve SnapLink performance with a field study involving 7 appliances.

Micro-benchmark

In the micro-benchmark experiment, a QR code encoding an 11-byte string with error correction level “L” [85] is printed at size 7.5 cm × 7.5 cm and attached to a wall in a well-lit conference room. From different viewing distances and angles, we capture 50 images of the QR code at each location using an LG G2 Mini Android phone. Since QR codes can be quickly decoded on phones with no need for offloading, the images are captured at resolution 2448 × 3264, which is much higher than the image resolution (i.e., 640 × 480) used in SnapLink. We also did the same experiment from a viewing angle of 0° in a darker environment, where with all lights are turned off while there is still ambient light from the windows and adjacent rooms. We use ZBar[5], a popular open source barcode reader, to decode QR codes in the images on a Linux machine. Since QR codes yield few false

[5] http://zbar.sourceforge.net/

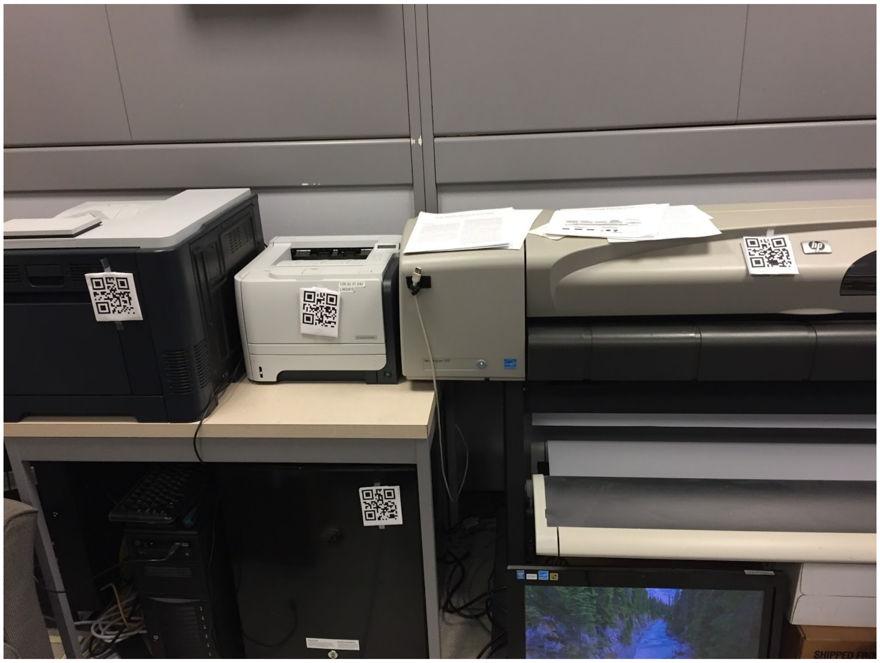


Figure 4.22: 4 of the 7 appliances in the field study setup, with 7.5 cm × 7.5 cm QR codes attached.

positives with Reed-Solomon error correction [85], we consider a QR code to be correctly decoded if the encoded string is among the set of decoded content.

Figure 4.21 shows the QR code recognition accuracy at different locations and under different illumination conditions. It shows that accuracy drops sharply when the viewing distance is larger than 150 cm or when the viewing angle is larger than 60°. A darker environment also diminishes the recognition accuracy. As we show in Section 4.5.7, SnapLink performs poorly when viewing distance is low with insufficient salient features. Hence, QR codes perfectly compensate for this drawback of SnapLink.

Field Study

To show how QR codes can be integrated into SnapLink and compensate for its drawbacks, we conduct a field study in a room (which is among the 39 rooms in our dataset) with 7 appliances (3 fridges, 3 printers, and 1 speaker). Each appliance is attached with a 7.5 cm × 7.5cm QR code encoding its unique ID. We capture more than 50 test images of each appliance from different angles and distances, including from low distances (e.g., < 30 cm) where SnapLink performs poorly. All images are tested using both ZBar [86] and Snaplink, as well as a fusing system, which first uses ZBar and subsequently uses SnapLink if no QR Code is decoded. Because QR code identification features a low rate of false positives, we consider the identification successful as long as the ground truth appliance ID is among all decoded QR codes from ZBar. When using Snaplink, we distinguish among the 7

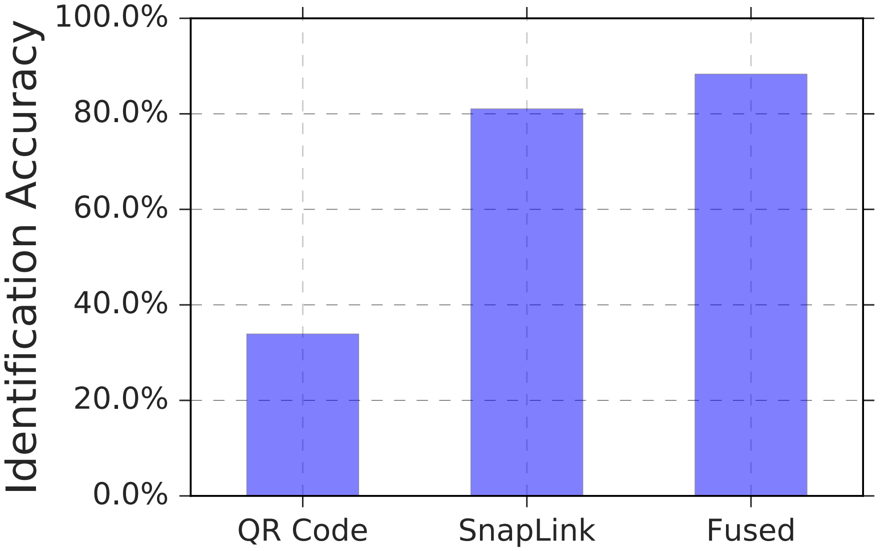


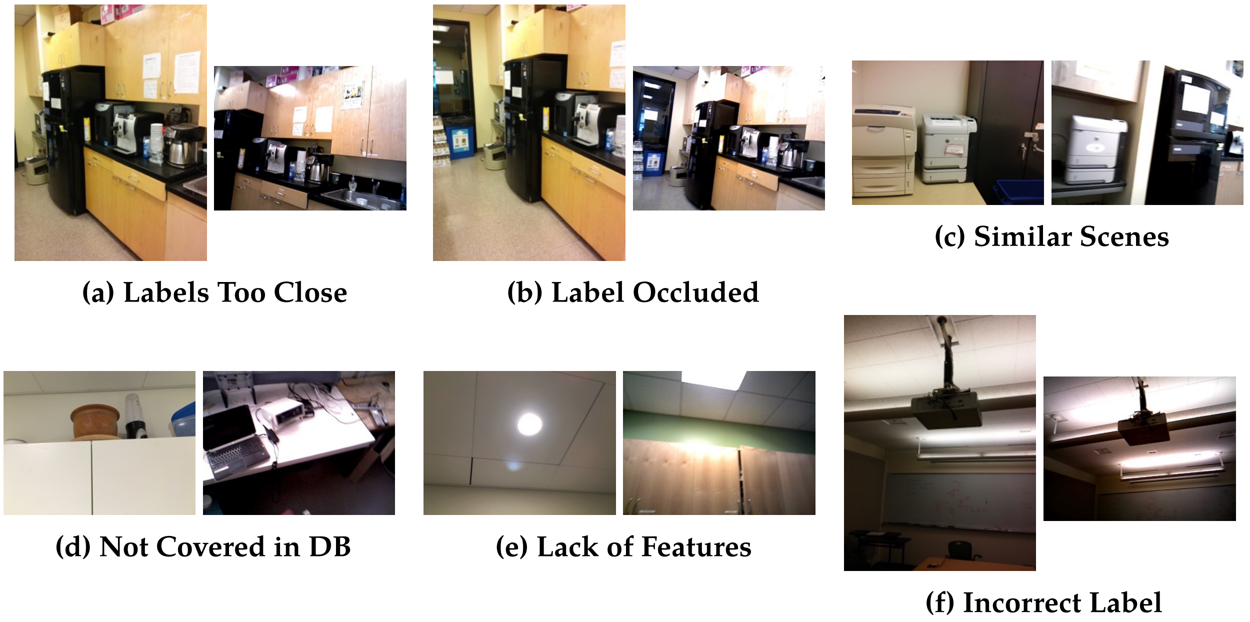
Figure 4.23: Identification accuracy of QR codes, SnapLink, and a fusing system that combines both.

appliances as well as all appliances in the other 38 rooms in our dataset.

Figure 4.23 shows the identification accuracy achieved by each of the three approaches. QR codes and SnapLink yield 34% and 81% accuracy respectively, and the fused result improves the accuracy to 88%. This shows that QR codes and SnapLink can complement each other very well for the purpose of appliance identification from an arbitrary viewing distance and angle.

4.5.11 Failure Analytics

To examine the causes of failures in our system, we summarize six categories of common failures that we have found over our deployment period. Figure 4.24 shows these failure cases with examples. In each example, the lefthand image is the query image, and the righthand image is its closest image using image retrieval, which helps demonstrate the problem. In the results, (a) contains a water machine and a coffee machine that are close to each other, and the same is true of their respective labels. This situation is sensitive to image localization accuracy. (b) shows that the user intends to identify the fridge, but a printer is occluded by the fridge and has a label that is closer to the image center. (c) shows two scenes that look similar, and our system cannot distinguish between them using their SURFs. In (d), the flowerpot is the target appliance, but it is barely captured by any image in the training video. (e) shows that the ceiling light does not have a sufficiently unique visual context, so our system mistakenly localizes it as another ceiling light. (f) shows the query image was identified as the light, but labeled incorrectly as the projector. These failures cases can be useful to guide our future work on improving the success rate



(a) Labels Too Close

(b) Label Occluded

(c) Similar Scenes

(d) Not Covered in DB

(e) Lack of Features

(f) Incorrect Label

Figure 4.24: Common failure cases (left is query image, right is the image in database found by image retrieval)

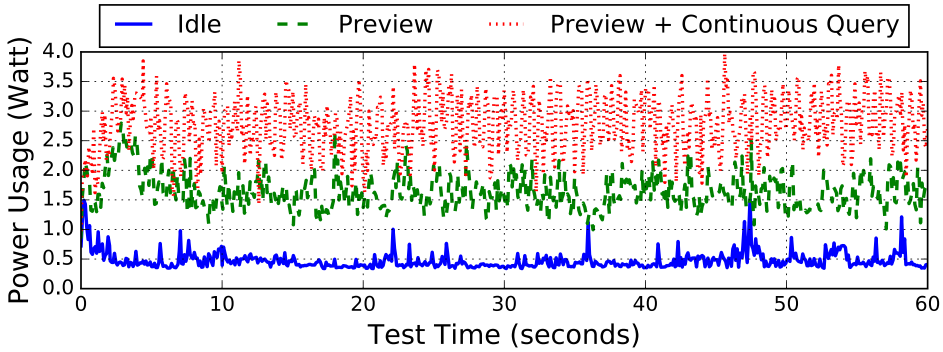


Figure 4.25: Energy Usage over Time

and demonstrate that most failures are due to external factors rather than artifacts of the system itself.

4.5.12 App Energy Usage

Our design goal is to make the SnapLink mobile client power efficient. Because the LG G2 Mini uses the Qualcomm MSM8226 Snapdragon 400 SoC, we use the Qualcomm Trepn Power Profiler [87] to measure accurate per-app power usage by leveraging specific Snapdragon features. We adjust the screen brightness to 100% and measure three different power consumption traces for 60 seconds: overall power usage while showing the home screen (idle), SnapLink power usage while streaming a camera view to screen (preview), and SnapLink power usage while sequentially sending query images as fast as possible using WiFi. Figure 4.25 shows the power usage trends, where idling consumes on average 0.48 Watts, preview consumes on average 1.64 Watts, and preview with continuous queries consumes on average 2.78 Watts. With the 3.8V 2440 mAh battery in the test phone, the expected battery life when using SnapLink to preview and continuously identify appliances is 2.84 hours ($\frac{(2.44 \text{ Amp hour})*(3.8 \text{ Volt})}{0.48+2.78 \text{ Watt}}$).

4.6 Related Work

In this section, we discuss existing appliance identification approaches and compare them with SnapLink.

With many recent improvements in computer vision (e.g., deep learning [63, 64] augmented reality [57]), vision-based human object interactions have been well studied. While general augmented reality systems focus on understanding a scene and how to naturally overlay virtual objects, we focus on quick and accurate image localization based appliance instance identification in a commercial building. Mayer, et al. [61] built a mobile interface to display information and to control appliances, which internally uses image retrieval to find the appliance in the current view. Overlay [57] is an object tagging application that uses image retrieval to find the object in the current camera view. It relies on an always-on camera to provide a continuous video stream to narrow down the search space, whereas our application requires the user to be able to take her phone out to start controlling appliances at an arbitrary time. Kong, et al. [58] used convolutional neural networks to recognize the category of appliances in the current camera view and identify the instances using unsupervised activity recognition and WiFi-based indoor localization systems. However, activity recognition and indoor localization cannot differentiate appliances of the same type in the same room, such as ceiling lights and printers in a copy room. Snap-To-It [55] allows users to take a picture to identify an appliance using image retrieval, using smartphone sensors for location validation. All of these works use image retrieval for instance recognition that requires many images to be captured for every appliance and each of the images to be labeled, which is a prohibitive effort for large scale deployment in a commercial building. Instead, SnapLink overcomes this problem by constructing a 3D model and allowing people to label 3D points with a labeling tool.

Fiducial markers (e.g., QR codes [54], ARToolKit [88], AprilTags [89]) are also used for vision-based object interactions. They encode appliance IDs and are attached to appliances for visual identification. However, as we discussed in Section 4.5.10, markers of practical sizes are hardly recognizable from a room size distance (e.g., 3 meters) or a large angle. Nevertheless, fiducial markers complement SnapLink very well when query images are taken from a small distance.

4.7 Summary

In this chapter, we presented SnapLink, an accurate and responsive vision-based appliance identification system that enables users to control building appliances in their sight by using ubiquitous smartphones. This work started with an intuition: "What we see is what we control." SnapLink leverages image localization for better appliance identification accuracy and much lower deployment overhead in commercial buildings. On top of the standard image localization process, we introduce geo-partitioning to enable easy and accurate 3D model construction, and provide a labeling tool to simplify appliance labeling process. In addition, SnapLink uses a feature sub-sampling mechanism which reduces computation time and scales well when database size increases, without losing identification accuracy.

We built an end-to-end system, including SnapLink and a smartphone application, and test it at a building scale using 1526 test images among 39 rooms captured by 4034 images. Our results show that SnapLink achieves 94% identification accuracy with 120 ms of server processing time and is robust to different viewing distances and angles, as well as changes in the environment (e.g., illumination, daily occupant activities).

We believe that SnapLink forms a significant step in enabling users to interact with their environments using computer vision technologies. However, it still requires user to take a picture of the target appliance to initiate an identification process. It is inconvenient if the user wants to explore what appliances are actually controllable, especially if many appliances are still not "smart" in the building, which is quite common today. Augmented Reality can solve this by continuously performing image localization and displaying identified appliance on the smartphone screen. We discuss the challenges and how we pursue in this direction in Chapter 5.

Chapter 5

MARVEL: Continuous Tracking

In this chapter, we explore the idea of continuous 6DOF (Degrees of Freedom) localization and appliance identification, which is essentially one form of Mobile Augmented Reality (MAR) that only shows annotations. We identify the challenges of image offloading latency and propose to use local motion sensor 6DOF tracking to minimize the need of offloading. Our system shows we can provide continuous appliance annotation display with very low latency (< 100ms), which is basically limited by smartphone hardware and the operating system. This work was done in collaboration with Tong Li and Hyung-Sin Kim at UC Berkeley [90].

5.1 Introduction

Augmented Reality (AR) has seen wide adoption with recent advances in computer vision and robotics. The basic concept of AR is to show a user additional information overlaid on the original view when she sees the real world through the screen. A user expects to see the augmented information both accurately and in real time, while moving her device. To this end, an AR device must perform intense computations on a large amount of (visual) data with *imperceptible latency*, from capturing an image to extracting the corresponding information (e.g., image localization and surface detection) and overlaying it on the proper location of the screen.

While this challenge has been addressed by using powerful and expensive AR-oriented hardware platforms, such as Microsoft HoloLens [91] and Google Tango [68], another approach has been also investigated: AR with *regular mobile devices*, such as a smartphone, called Mobile AR (MAR). MAR is attractive in that it does not require users to purchase and carry additional devices [92]. However, MAR has its own challenges since mobile devices have significantly less *storage* and *computation speed* than AR-oriented devices. In addition, given that mobile platforms are not only for MAR but also for many other daily tasks (e.g., social network services and web search), MAR is expected to avoid excessive *battery* consumption. There are relatively lightweight MAR software tools, such as Google

ARCore [93] and Apple ARKit [94], but they operate only on the latest generation of smartphones and consume considerable energy.

This work investigates how to enable real-time MAR on ordinary mobile devices. Specifically, we aim to provide an *annotation-based MAR service* which understands objects that the device points at and overlays the corresponding annotations on the relevant location of the screen in real time. This application is viable even with the restricted capabilities of mobile devices, in contrast to more demanding AR that naturally embeds 3D objects into a 3D model using a game engine [91]. More importantly, it has a variety of practical usage scenarios, such as museums, buildings, and airports [53, 57]. Through the annotation service, a user can obtain information about an object by pointing at it. Furthermore, she is able to interact with (or control) the object by touching annotations on the screen.

A number of MAR studies have attempted to implement these applications. They offload computation and storage to the *cloud* to overcome the restrictions of mobile devices. However, the work in [53, 57] is excessively dependent on the cloud, using a mobile device simply to send large volumes of data to the cloud, resulting in high latency and energy consumption. Despite some effort to reduce offloading latency in [79, 95], the lowest latency is still 250 ms [96] to the best of our knowledge, which is 2.5x higher than the requirement for a smooth AR user interface (i.e., 100 ms [60])). Some work enables fast tracking of an *identified* object on the screen by using local optical flow, but the identification procedure when new objects appear still relies on the cloud [95, 97]. In addition, optical flow only works with clear images that have sufficient overlap, forcing a user to move her device slowly [97]. Furthermore, energy consumption has been significantly overlooked in this regime.

To overcome these challenges, we design MARVEL (MAR with Viable Energy and Latency) which fully utilizes the potential of both the cloud and the mobile device. To identify and track objects on the screen, MARVEL uses visual and inertial localization, rather than image retrieval and convolutional neural network (CNN) in previous MAR work, which facilitates cooperation between the cloud and the mobile device. MARVEL mainly relies on (light/fast) *local inertial data processing* while using (heavy/slow) local optical flow and cloud offloading *selectively*; it locally computes optical flow only when the device's position changes significantly and offloads images only when local results need to be calibrated. This significantly reduces latency, resulting in <100 ms for both object recognition and tracking. Inertial data is more robust to device movement than optical flow and its processing is light enough to localize *multiple objects* simultaneously.

With this low-latency system architecture, we aim to minimize energy consumption on the mobile device (both computation and offloading overhead) without sacrificing accuracy. To this end, we explore several essential issues: (1) when to offload, (2) what to offload, and (3) what to do locally while not offloading. MARVEL performs image offloading-based calibration only when it detects an inconsistency between inertial and visual data. For image offloading, the mobile device selects only a few recently captured images with two criteria: sharpness and the number of features (more features means the image contains more information). We obtain these two metrics not by heavy image computation but indirectly from simple gyroscope readings and edge detection [82]. When not offloading, the mobile device uses inertial data for an AR service and corrects its errors by processing visual data selectively. In addition, it continuously monitors inconsistency by comparing its inertial and visual information.

MARVEL's performance is evaluated in a holistic system, which implements a real-time appliance annotation service in a commercial building. The results verify that the above design choices enable MARVEL to achieve high accuracy, low latency, and low energy consumption together.

The contributions of this work are fourfold:

- We realize an annotation-based AR service on ordinary mobile devices with *imperceptible latency (<100 ms)*, which is the first practical *real-time* MAR system to the best of our knowledge.

- We propose *a novel MAR system architecture* which synergistically utilizes processing capability and storage of the cloud and a mobile device while using both visual and inertial information.

- We investigate how to effectively use visual and inertial data for identifying and tracking multiple objects with low latency, low energy, and high accuracy. This involves the following questions: (1) when to offload, (2) what to offload, and (3) what to do while not offloading.

- We implement and deploy *a holistic system*, where MARVEL's performance is evaluated and compared with other techniques, providing a better understanding of MAR system operation.

5.2 Annotation-based MAR Systems

This section describes the motivation of MARVEL. We first describe the application scenario and technical requirements of annotation-based MAR systems. Then we introduce possible design options and show our preliminary study which motivates MARVEL design choices.

5.2.1 Desired User Experience

We aim to provide an *annotation-based MAR service*, where a regular mobile device understands objects that it points at and overlays their annotations on the relevant location of the screen in real time. This application is viable with the restricted capability of mobile devices, in contrast to more demanding approaches to AR that naturally embed 3D objects into a 3D model using a game engine [91]. Furthermore, it has a variety of practical usage scenarios [53, 57]. For example, when going on a field trip to a museum, a user may see

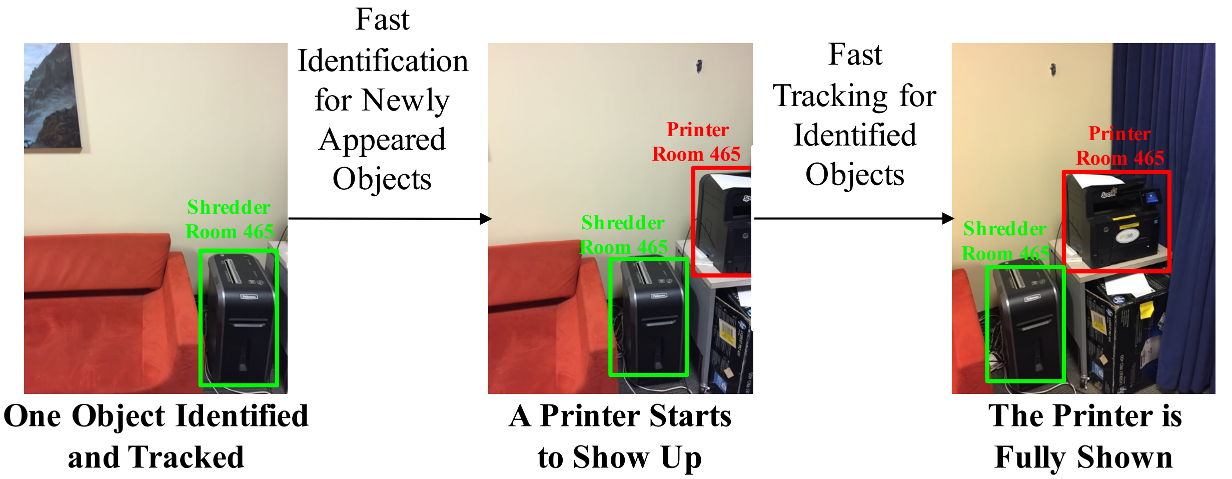


Figure 5.1: Target application: Real-time, multi-object annotation service on a regular mobile device

the description of each exhibit simply by pointing the mobile device at it. In a building, she may get a control interface for an appliance, such as a projector on the ceiling or a printer, by pointing at it.

Figure 5.1 depicts the desired user experience in this application scenario. A user expects the five following benefits, which give technical challenges for a MAR system design.

- **Fast identification:** When pointing at a new object, she expects its annotation to *immediately* pop up on the relevant location of the screen. She wants to see the annotation even before the new object is fully captured on the screen (i.e., the second case of Figure 5.1). To this end, an object recognition process should be finished within 100 ms [60].

- **Fast tracking:** After the annotation pops up, she expects it to *smoothly* move *along with* the corresponding object while moving her device. To this end, whenever the image on the screen changes, the annotation view should be updated within 100 ms [60].

- **Multi-object support:** When multiple objects are captured in the screen, she expects fast identification and tracking for each object simultaneously (i.e., the second and the third cases of Figure 5.1). To this end, the latency of annotation placement process should be decoupled from the number of objects.

- **Robustness to arbitrary movement:** The user does not have to intentionally move her device slowly to use this service. To this end, the accuracy of annotation placement should be consistent regardless of device movement.

- **Low energy consumption:** Her device is a regular smartphone, used not only for this application but also for other various purposes. She does not want to recharge the device's battery more frequently while using this application.

5.2.2 Design Options

To fulfill the above requirements, system design choices need to be made carefully. It is important to decide how to recognize and track objects and where to compute core algorithms, which impact accuracy, latency, and energy consumption.

How to Recognize and Track Objects?

In our application scenario, we are not only recognizing the categories of objects, but also *instances*. For example, two printers of the same model in two rooms need to be differentiated based on the background information. There are three representative ways to recognize instances and track objects: image retrieval [57], Convolutional Neural Networks (CNN) [65], and localization [53]. With a query image on the screen, image retrieval selects the most similar labeled image in the database in terms of the number of common salient features (e.g., SURF [40]), while CNN extracts features from the query image and use them to classify the image on a pre-trained neural network. These require heavy image computation and a large database [53]. Even with recent work on running image retrieval or CNN on smartphones [98, 99], they require a on-board GPU and still are not energy efficient.

On the other hand, localization recognizes and tracks an object by its location information. Various localization techniques have been developed for decades, which use different types of information (e.g., image, light, inertial data, and RF signal) and have different accuracy, latency, and storage. For example, image localization [53] and visual simultaneous localization and mapping (SLAM) [70] are accurate but extremely heavy, while inertial localization is lightweight but more error-prone [100]. *We choose localization-based object recognition and tracking.* Our intuition is that a synergistic combination of heterogeneous localization techniques has the potential to achieve the above five requirements by overcoming the weakness of each technique.

Where to Perform Computation?

Another question is how to efficiently use the capabilities of a cloud server and a mobile device [101]. Offloading heavy computation to the cloud reduces computation overhead on a mobile device. However, energy consumption and latency costs now come from communication rather than computation. Therefore it is important to decide what algorithm to compute locally or on the cloud, what information to offload, and when to offload, as these factors significantly impact performance.

Operation	Power (Watt)
Preview only	1.96 ± 0.64
Inertial localization	1.94 ± 0.74
Image offloading	2.24 ± 0.68
Optical flow	2.54 ± 1.06
RTABMap [70]	5.74 ± 2.44

Table 5.1: Power usage of different applications. Image offloading and optical flow consume more power than inertial localization. RTABMap (local image localization) consumes $2.5\times$ **power than image offloading.**

Operation	Time (ms)
Rotation	0.01 ± 0.03
Translation	0.03 ± 0.08
Optical flow	30.68 ± 5.16
Image offloading	≥ 250 [96]

Table 5.2: Processing time of different operations. Optical flow and offloading take significantly more time than inertial localization.

When making these design choices, a localization-based system provides more options with diverse techniques than an image-based system. However, each localization technique's characteristics need to be carefully analyzed in order for the proper design decisions.

5.2.3 Preliminary Study

To design a localization-based MAR system satisfying the five requirements in Section 5.2.1, we evaluate some candidates: (1) inertial localization, (2) image localization, (3) optical flow, and (4) visual SLAM. Image localization is offloaded on the cloud as in [53] and other algorithms are computed locally.

We run five applications on a Lenovo Phab 2 Pro Android smartphone: (1) a preview app (see-through camera), (2) a preview app that performs 6 degree-of-freedom (DOF) inertial localization in the background (translation and rotation), (3) a preview app that offloads a 640×360 image to the cloud for 6DOF image localization sequentially (i.e., it only offloads the next image when the current image result comes back), (4) a preview app that performs optical flow tracking on 640×360 images sequentially, (5) RTABMap [70], a purely local visual SLAM application. We measure the system power usage using the Trepn Profiler [87], which works with the Qualcomm Snapdragon CPUs present in smartphones, for accurate battery usage.

Tables 5.1 and 5.2 show the power consumption and processing time of each operation respectively. Since network performance varies from place to place, we use the data summarized in [96] for offloading latency. First, visual SLAM consumes much more energy than the other operations. Along with significant local storage overhead, it only runs on the latest hardware, which confirms that using the cloud is necessary for MAR. Second, both image offloading and optical flow consume significant energy, which reveals that both of them need to be triggered minimally, if at all. Between the two, offloading an image not only consumes more energy but also takes longer than computing one optical flow. Lastly, 6DOF inertial localization does not add noticeable energy overhead compared to the basic preview app. This is consistent to the prior work [102], which shows that inertial measurement unit (IMU) consumes at most ≈30 mW. Processing inertial data (both rotation and translation) is orders of magnitude faster than optical flow and image offloading.

These results confirm that local computation of 6DOF inertial localization is the fastest and the most energy-efficient localization method for MAR. Given that inertial localization is error-prone [100], the question is how to compensate for its errors while maintaining energy efficiency. We use both image localization on the cloud and local optical flow to improve localization accuracy. At the same time, we try to use these more expensive methods *only when necessary* to minimize energy consumption on a mobile device.

5.3 MARVEL Overview

This section gives an overview of MARVEL, a new *localization*-based MAR system for real-time annotation services which fulfills the five application requirements: (1) fast identification, (2) fast tracking, (3) multi-object support, (4) robustness to arbitrary movement, and (5) low energy consumption. To provide annotation services with localization, MARVEL should detect what objects are captured by the mobile device's screen (what annotations to display) and where they are located on the screen (where to display the annotations). To this end, the following two types of location information need to be obtained with low energy consumption and low latency:

- *6DOF location of the mobile device* in the 3D space (i.e., 3D location and 3D orientation of the screen).
- *2D locations of objects* on the device screen surface.

As a *holistic* MAR system, the primary contribution of MARVEL is its comprehensive system architecture. While it uses existing algorithms, such as 6DOF inertial localization with Zero Velocity Update (ZUPT) [100], 6DOF image localization [56], and optical flow [103], MARVEL's design focuses on how to combine these algorithms as well as when and where to run each algorithm considering both cloud offloading and local computing. Each design choice has a significant impact on MAR performance and a synergistic system design is the key factor to achieve both low energy and low latency.

To obtain the mobile device's 6DOF location, MARVEL uses 6DOF inertial localization and 6DOF image localization together. While inertial localization is lightweight but accurate only for a short time [48], image localization is accurate but computationally heavy and involves a large volume of data. MARVEL achieves the sweet spot of this trade-off; *it mainly performs inertial localization on the mobile device while triggering image localization on the cloud only when IMU-based results need calibration.*

Next, MARVEL uses the mobile device's 6DOF location to obtain 2D locations of objects on the screen. The mobile device projects 3D locations of nearby objects (given as the local database) onto the 2D screen surface. In addition, MARVEL uses optical flow to improve accuracy. Optical flow provides its own 2D location information for objects, which is used to correct IMU-based results and to detect inconsistency (triggering calibration on the cloud). *Optical flow is performed only when necessary, minimizing local image computation while achieving high accuracy.*

Overall, the main idea of MARVEL is to primarily use local computations involving IMU data while triggering heavy image computation (optical flow) and offloading (image localization on the cloud) only when necessary. This design choice aims to achieve low latency and low energy consumption without sacrificing accuracy.

5.4 System Design

Figure 5.2 illustrates MARVEL's architecture and operational flow, which involves a mobile client (carried by a user) and a cloud server. The client has a local database and performs AR locally most of the time with 6DOF inertial localization and optical flow. The client communicates with the cloud server for calibrating its inertial localization results, only when necessary. For calibration, the cloud server has a pre-built 3D point cloud database and provides 6DOF image localization when receiving a query image from the client.

5.4.1 Initialization: Database Construction

Initially, the point cloud of the complete service area (e.g., a building), along with all annotated 3D points, is created and stored in the cloud server as the *MARVEL database*, which has its own frame $\mathcal{M}$, as shown in Figure 5.3. It is known that a database for 6DOF image localization is easier to deploy than that for image retrieval or CNN [53].

Specifically, when a client offloads a query image, which is captured at time t_0 and has frame $\mathcal{I}$[1], 6DOF image localization on the cloud results in $\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_0)$. Note that $\mathbf{P}_{\mathcal{B}}^{\mathcal{A}}(t)$ is the 4×4 homogeneous transformation from frame $\mathcal{A}$ to frame $\mathcal{B}$ at time t, represented as

$$\mathbf{P}_{\mathcal{B}}^{\mathcal{A}}(t) = \left(\begin{array}{ccc|c} \multicolumn{3}{c|}{\mathbf{R}_{\mathcal{B}}^{\mathcal{A}}(t)} & \mathbf{T}_{\mathcal{B}}^{\mathcal{A}}(t) \\ 0 & 0 & 0 & 1 \end{array} \right) \tag{5.1}$$

[1] Note that the image frame $\mathcal{I}$ is the same as the frame of device screen and it changes over time as the device moves.

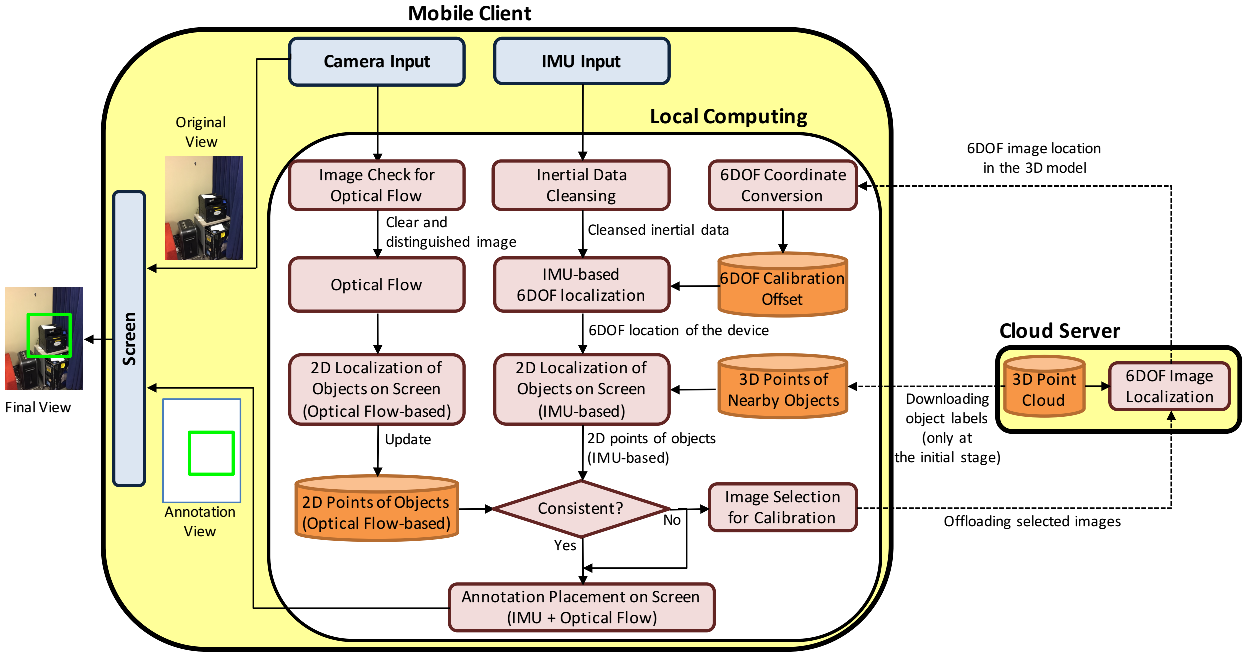

Mobile Client
Camera Input
IMU Input
Local Computing
Original View
Final View
Screen
Annotation View
Image Check for Optical Flow
Clear and distinguished image
Optical Flow
2D Localization of Objects on Screen (Optical Flow-based)
Update
2D Points of Objects (Optical Flow-based)
Inertial Data Cleansing
Cleansed inertial data
IMU-based 6DOF localization
6DOF location of the device
2D Localization of Objects on Screen (IMU-based)
2D points of objects (IMU-based)
Consistent?
Yes
No
Annotation Placement on Screen (IMU + Optical Flow)
6DOF Coordinate Conversion
6DOF Calibration Offset
3D Points of Nearby Objects
Image Selection for Calibration
6DOF image location in the 3D model
Cloud Server
3D Point Cloud
6DOF Image Localization
Downloading object labels (only at the initial stage)
Offloading selected images

Figure 5.2: MARVEL system operation overview. There are a mobile client and a cloud server. The mobile client's local computing module takes the main role for generating annotation view by using inertial localization and optical flow. The cloud does image localization for calibrating local tracking errors, which is selectively triggered.

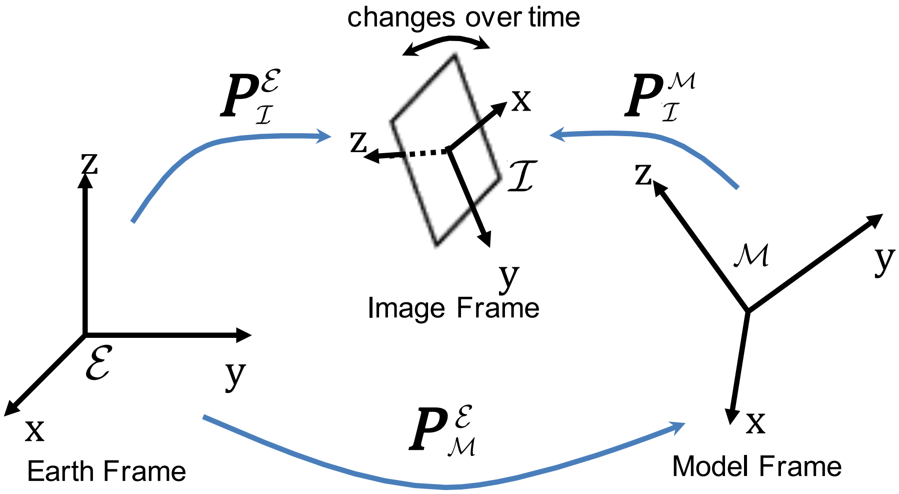


Figure 5.3: Three frames for 6DOF localization with inertial and visual Data: earth frame $\mathcal{E}$, model frame $\mathcal{M}$, and image frame $\mathcal{I}$. While $\mathcal{E}$ and $\mathcal{M}$ are static, $\mathcal{I}$ changes as the device screen moves.

where $\mathbf{R}_{\mathcal{B}}^{\mathcal{A}}(t)$ and $\mathbf{T}_{\mathcal{B}}^{\mathcal{A}}(t)$ are 3×3 rotation matrix and 3×1 translation vector from frame $\mathcal{A}$ to $\mathcal{B}$ at time t, respectively.

Unlike previous MAR systems that maintain all of their data in the cloud, MARVEL constructs a local database in the mobile client, enabling it to compute its 6DOF inertial location *completely locally*. When a user enters the service area, the mobile client constructs its *local database* by downloading *the 3D-location labels and annotations of 'nearby' objects* from the cloud. Each object's label L has its annotation A^L and its coordinate in the model frame $\mathcal{M}$, $\mathbf{p}_{\mathcal{M}}^{L} = (x_{\mathcal{M}}^{L}, y_{\mathcal{M}}^{L}, z_{\mathcal{M}}^{L})$. The mobile client converts the model frame to the earth frame $\mathcal{E}$ (from $\mathbf{p}_{\mathcal{M}}^{L}$ to $\mathbf{p}_{\mathcal{E}}^{L}$) and stores $\mathbf{p}_{\mathcal{E}}^{L}$ and A^L in the local database.

While the 3D point cloud on the server involves a large amount of data, this local database is *lightweight*. Assuming that the average annotation length is 100 bytes, an entire building that contains 10,000 labels will incur only 1.12 MB ($= 10,000 \times (3 * 4Bytes + 100Bytes)$) of memory overhead. Furthermore, this label downloading happens *very rarely* (e.g., when entering into another building), incurring negligible communication overhead.

While MARVEL is running, the mobile client's local computing module continuously receives inertial data (acceleration and rotation) (in the earth frame $\mathcal{E}$) and camera frames, and computes the two types of data in parallel as below.

5.4.2 Local Inertial Tracking

When receiving new, noisy accelerometer data from the sensor, the client first cleans this data by setting all readings below a threshold (0.2 m/s^2 in any dimension in our system) to be 0. Then the client computes its 6DOF location by using the cleaned accelerometer data, latest rotation sensor data, and the stored 6DOF calibration offset (provided by 6DOF image localization on the cloud). The client can update the calibration offset by triggering a calibration on the cloud, receiving the 6DOF image location on the model frame $\mathcal{M}$ from the cloud, and converting it to the 6DOF device location on the earth frame $\mathcal{E}$ (coordinate conversion).

Specifically, if the mobile client triggers a calibration and offloads a query image (of frame $\mathcal{I}$) at time t_0, it receives $\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_0)$ from the cloud at time $t_1(> t_0)$, due to offloading latency. Then the client converts $\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_0)$ to the calibration offset for inertial localization, $\mathbf{P}_{\mathcal{I}}^{\mathcal{E}}(t_0)$, as

$$\mathbf{P}_{\mathcal{I}}^{\mathcal{E}}(t_0) = \mathbf{P}_{\mathcal{M}}^{\mathcal{E}} \cdot \mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_0). \tag{5.2}$$

Since the mobile client obtains the calibration offset $\mathbf{P}_{\mathcal{I}}^{\mathcal{E}}(t_0)$ at time t_1, it is already outdated by as much as the offloading latency $(t_1 - t_0)$. However, this is fresh enough to accurately calibrate inertial localization. Until receiving another calibration offset, the client continues to update $\mathbf{P}_{\mathcal{I}}^{\mathcal{E}}(t)$ locally by using $\mathbf{P}_{\mathcal{I}}^{\mathcal{E}}(t_0)$ as the offset and ZUPT [100] as the inertial localization algorithm. The calibration is triggered *only when necessary* to minimize offloading overhead.

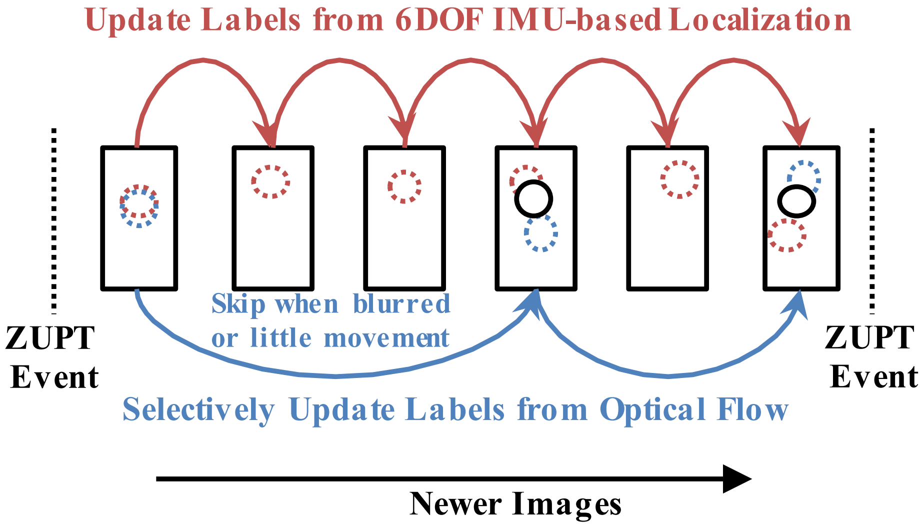

Update Labels from 6DOF IMU-based Localization
Skip when blurred or little movement
ZUPT Event
ZUPT Event
Selectively Update Labels from Optical Flow
Newer Images

Figure 5.4: Correcting the results of local inertial tracking with those of selective local visual tracking (optical flow).

After getting its 6DOF location information $\mathbf{P}_{\mathcal{I}}^{\mathcal{E}}(t)$, the mobile client projects 3D-points of nearby objects, $\mathbf{p}_{\mathcal{E}}^{L}$ in the *local database*, onto the screen. To this end, the client extracts $\mathbf{R}_{\mathcal{I}}^{\mathcal{E}}(t)$ and $\mathbf{T}_{\mathcal{I}}^{\mathcal{E}}(t)$ from $\mathbf{P}_{\mathcal{I}}^{\mathcal{E}}(t)$ and computes Eq. (3) [104].

$$\begin{bmatrix} \mathbf{s}_{imu}^{L}(t) \\ 1 \end{bmatrix} = \mathbf{K}_{3\times 3}[\mathbf{R}_{\mathcal{I}}^{\mathcal{E}}(t)|\mathbf{T}_{\mathcal{I}}^{\mathcal{E}}(t)] \begin{bmatrix} \mathbf{p}_{\mathcal{E}}^{L} \\ 1 \end{bmatrix} \tag{5.3}$$

Here $\mathbf{K}_{3\times 3}$ is a hardware-dependent intrinsic matrix obtained out-of-band, which converts 2D location on the screen to the 2D pixel location on the screen. The result is $\mathbf{s}_{imu}^{L}(t)$, a 2×1 vector that represents IMU-based estimation of 2D pixel location on the screen, for label L at time t. Note that $\mathbf{s}_{imu}^{L}(t)$ is generated only with local computation.

5.4.3 Selective Local Visual Tracking

After receiving a new image from the camera, the client first checks if it is sharp enough for accurate optical flow using Sobel edge detection [105], which only takes around 4 ms to compute. If the sum of detected edge intensities is above a threshold ($500,000$ for a 640×360 image in our settings), it does not consider the image for optical flow. If the new image is sharp enough, the client computes optical flow only when the new image is significantly different from the previous image, as depicted in Figure 5.4. Our idea is that optical flow between two very similar images consumes nontrivial energy (as shown in Section 5.2.3) without contributing to accuracy improvement, which should be avoided.

How can we measure difference between two images? the pixel-wise image comparison adopted by previous work on car-mounted cameras [97] does not work well for smartphones, because inconsistent hand movements can cause different levels of blurriness (even among the sharp images). Instead, we indirectly use the location difference provided by the IMU (both rotation and translation). We compare the two IMU data points, each of which is sampled when each of the two images is taken. If the rotation or translation accumulated since the previous optical flow computation is sufficiently large (more than 5° or 1 cm in our settings), the client triggers a new optical flow computation. This results in $\mathbf{s}^L_{of}(t)$, which is optical flow-based estimation of 2D pixel location on the screen, for label L at time t.

5.4.4 Smoothing Annotation Movement

Now the mobile client has two sets of 2D points (pixel location) for each object on the screen, $\mathbf{s}^L_{imu}(t)$ from inertial localization and $\mathbf{s}^L_{of}(t')$ from optical flow. t is the current time and t' $(< t)$ is the last time when optical flow was computed. Again, optical flow computation is not always triggered for energy savings and takes longer than inertial data computation. $\mathbf{s}^L_{of}(t')$ is good enough to be used at time t when the client does not move significantly between time t' and t.

To compensate for the cumulative errors of $\mathbf{s}^L_{imu}(t)$ caused by translation and to achieve smooth movement of annotations on the screen, MARVEL combines $\mathbf{s}^L_{imu}(t)$ and $\mathbf{s}^L_{of}(t')$ to compute $\mathbf{s}^L_{final}(t)$, the final 2D pixel location of label L at time t which determines where to place annotation A_L on the screen. $\mathbf{s}^L_{final}(t)$ is given by

$$\mathbf{s}^L_{final}(t) = \frac{c_{imu}\mathbf{s}^L_{imu}(t) + c_{of}\mathbf{s}^L_{of}(t')}{c_{imu} + c_{of}} \tag{5.4}$$

where c_{imu}, and c_{of} are the confidence level for $\mathbf{s}^L_{imu}(t)$ and $\mathbf{s}^L_{of}(t')$ respectively.

Given that inertial localization's error mainly comes from translation (double integration of linear acceleration) [100], we decrease c_{imu} as translation between time t_0 (when the last calibration was triggered) and t increases. On the other hand, we define c_{of} as the quality of the last optical flow performed. We measure the quality as Glimpse [97] does: using standard deviation of feature differences around all labels between the two images. Lastly, the client decides where to put each annotation A_L using Equation 5.4 and overlays the annotation view on the original view.

The whole process until generating the annotation view is performed *completely locally*. The mobile client does not wait for any response from the cloud since it has all the necessary information for processing (i.e., the local database and the 6DOF calibration offset), avoiding >250 ms of latency. The latency of MARVEL comes from local inertial data processing and selective optical flow computation.

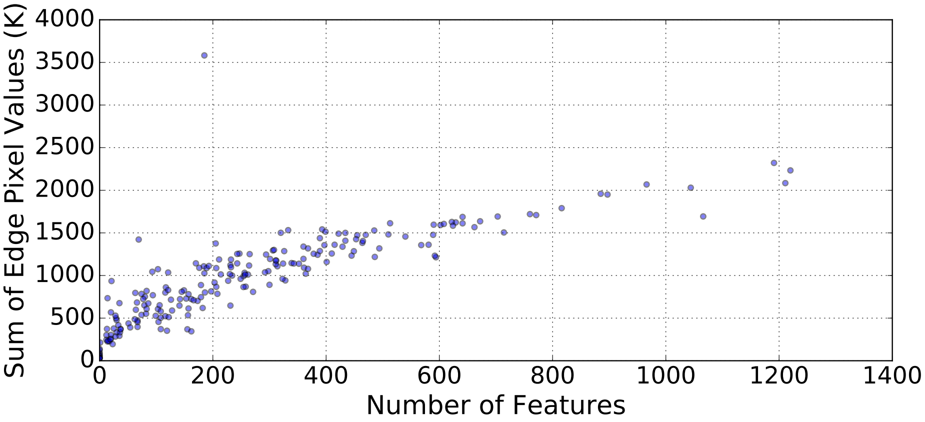


Figure 5.5: Correlation between feature number and sum of edge intensity. We use SURF as features and Sobel edge detection, both using the default OpenCV parameters. The experiment is conducted with 207 640×360 images.

5.4.5 Selective Image Offloading

When the mobile client detects an inconsistency while performing the above local computation, it triggers calibration on the cloud server. The mobile client detects inconsistency in two ways: (1) when both of two confidence values, c_{imu} and c_{of}, are zero, or (2) when $dist(\mathbf{s}^L_{imu}(t), \mathbf{s}^L_{of}(t')) > \phi$ (i.e., 2D points generated by inertial localization and optical flow are significantly different), where ϕ is called calibration threshold. As ϕ decreases, accuracy increases but offloading overhead also increases. This trade-off will be investigated in Section 5.6.2.

If the client decides to trigger a calibration, it offloads images to the cloud. In doing so, it offloads *only several of the recently taken images* to reduce communication overhead. The impact of the number of offloading images N will be investigated in Section 5.6.3. To avoid losing image localization accuracy while offloading only a few images, the client needs to extract each image's features and measure the feature uniqueness [79]. However, this process requires very heavy image computation on the client, making computation overhead exceed even the offloading overhead [57], especially when we want to use a robust feature, such as SURF [40].

To avoid this, in MARVEL, the client does not consider feature uniqueness but image sharpness and the number of features when selecting the best query images. Offloading a clear image can help the cloud to extract meaningful features from it. Another intuition is that an image with more features is likely to have more unique features at the same time (which is not always true but good enough as a rule of thumb). The client infers image sharpness (or blurriness) from both *gyroscope readings* and *edge detection*, because blurry images mostly come from rotations [82], and sharp images tend to show more edges. In addition, images with more edges should have more features, because features are essentially descriptors of high-contract areas, most of which are on edges. To verify this, we test 207 images captured in a campus building. Figure 5.5 shows that sum of edge pixels (given by edge detection) and the number of features of the 207 images are highly correlated.

To select the best images for calibration, the mobile client ranks the images in its cache based on the sum of edge pixels and the gyroscope reading at the time of the image capture, resulting r_i^{edge} and r_i^{gyro} for image i, respectively (smaller rank is better). Then it ranks the images by $(r_i^{edge} + r_i^{gyro})$ and offloads the top N images to the cloud. Overall, this image selection process is much faster and more energy-efficient than computing feature uniqueness directly.

5.4.6 Calibration

Upon receiving the N images, the server performs image localization for all of them, and returns N results to the client: 6DOF location $\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_i)$ for $i \in \{1, 2, \ldots, N\}$, where t_i is the time when image i was captured. Then, the client tries to select the most accurate $\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_i)$ among the N results and uses it for the calibration offset. Although $\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_1)$ (the result from the best ranked offloading image) mostly gives the best calibration offset, other $N-1$ results still need to be considered since sometimes $\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_1)$ can be an outlier[2].

To compare the N results together, the client shifts time t_i of each $\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_i)$ to time t_1. We define $\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_1|t_i)$ to be the estimation of $\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_1)$ derived from $\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_i)$, which can be obtained by using inertial localization results:

$$\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_1|t_i) = \mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_i)\mathbf{P}_{\mathcal{I}}^{\mathcal{E}}(t_i)^{-1}\mathbf{P}_{\mathcal{I}}^{\mathcal{E}}(t_1) \tag{5.5}$$

If both the image localization and inertial localization are ideal, $\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_1|t_i)$ is the same for all i in $\in \{1, 2, \ldots, N\}$. But in practice, both localization methods have errors, which causes difference among them.

When selecting the best result, our intuition is that if $\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_1|t_i)$ is similar to the $N-1$ others, $\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_i)$ may be secure to use for the calibration offset (i.e., it has little chance to be an outlier). To measure the similarity, the client gets $\mathbf{R}_{\mathcal{I}}^{\mathcal{M}}(t_1|t_i)$ and $\mathbf{T}_{\mathcal{I}}^{\mathcal{M}}(t_1|t_i)$ from $\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_1|t_i)$

[2] Note that we do not directly consider feature uniqueness when ranking offloading images.

and obtains the translation difference $D(i)$ and the rotation difference $A(i)$, respectively as

$$D(i) = \sum_{\substack{j=1 \\ j \neq i}}^{N} dist(\mathbf{T}_{\mathcal{I}}^{\mathcal{M}}(t_1|t_i), \mathbf{T}_{\mathcal{I}}^{\mathcal{M}}(t_1|t_j)) \tag{5.6}$$

$$A(i) = \sum_{\substack{j=1 \\ j \neq i}}^{N} angle(\mathbf{R}_{\mathcal{I}}^{\mathcal{M}}(t_1|t_i), \mathbf{R}_{\mathcal{I}}^{\mathcal{M}}(t_1|t_j)). \tag{5.7}$$

The client ranks $\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_i)$ with $D(i)$ and $A(i)$, resulting in r_i^{trans} and r_i^{rot}, respectively. Finally, it selects the best result $\mathbf{P}_{\mathcal{I}}^{\mathcal{M}}(t_c)$ where $c = \arg\min_i(r_i^{trans} + r_i^{rot})$, and converts it to the calibration offset $\mathbf{P}_{\mathcal{I}}^{\mathcal{E}}(t_c)$.

5.5 System Implementation

Most smartphone operating systems today (e.g., Android) already perform data fusion and provide virtual sensors. We implemented MARVEL in Android, and use the provided rotation vector sensor for absolute rotation in the earth frame and the linear acceleration sensor for acceleration without gravity. Our app reads them as fast as possible, at a rate of 200 samples per second, on a Lenovo Phab 2 Pro. We use OpenCV manager (with OpenCV 3.2.0), an Android application that provides system-wide service to perform OpenCV computation for all applications. The optical flow computations use the Lucas-Kanade algorithm [106] provided by OpenCV. With the Lenovo Phab 2 Pro, all images we process are at resolution 640 × 360. To ensure fastest image processing, we get raw pixels from the camera hardware (e.g., without JPEG compression), and compress only the images we decide to offload using OpenCV. Even though the Lenovo Phab 2 pro offers a depth camera, we do not use it in MARVEL.

On the server side, we build on top of SnapLink [53], an open source image localization system designed for appliance control. It provides a GRPC API, and our Android client uses GRPC 1.3.0 to communicate with it. The server is implemented in C++ and uses OpenCV 3.3.0-rc and PCL 1.8.0.

5.6 Evaluation

In this section, we conduct several micro-benchmarks to demonstrate the effectiveness of optimizations proposed earlier, as well as the end-to-end performance of MARVEL.

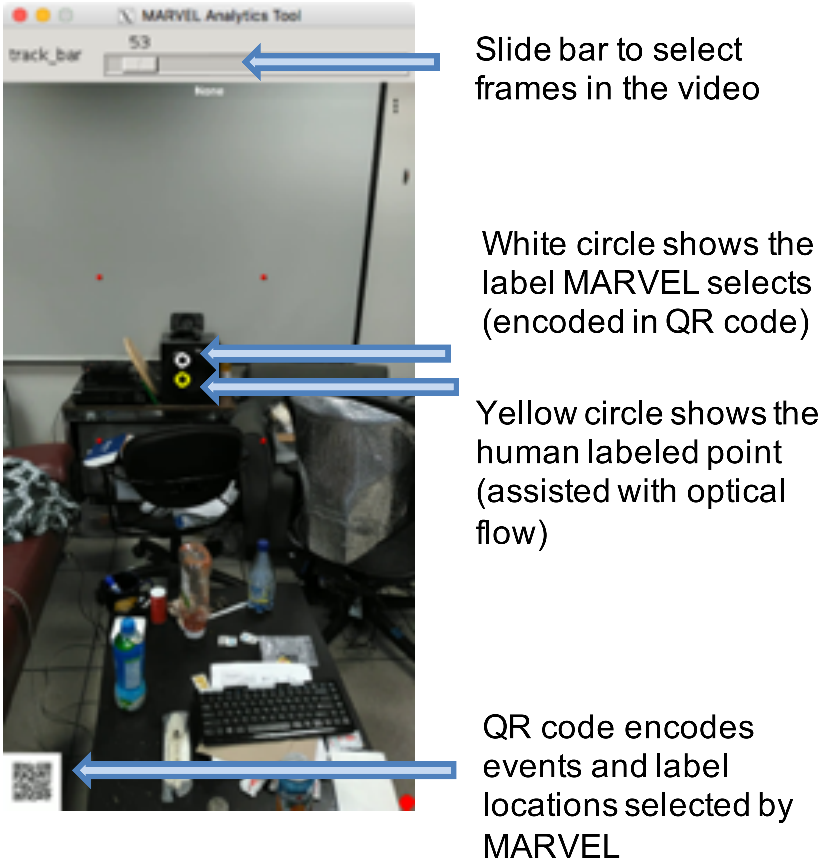

MARVEL Analytics Tool
track_bar
53
None
Slide bar to select frames in the video
White circle shows the label MARVEL selects (encoded in QR code)
Yellow circle shows the human labeled point (assisted with optical flow)
QR code encodes events and label locations selected by MARVEL

Figure 5.6: Evaluation tool to browse images, display identified labels, and manually label ground truth locations, assisted by optical flow.

5.6.1 Experimental Setup and Metrics

To ensure a accurate label placement at any point of time, our system must achieve both high accuracy and low latency. Instead of evaluating accuracy and latency separately, we examine the end-to-end label placement error from the client side. To do that, we video capture the phone screen while operating MARVEL and analyze the video using an analytic tool we built, as shown in Figure 5.6. We make MARVEL encode relevant information (e.g., label location, offloading event, and ZUPT event) into a QR code that is displayed on the left bottom corner. The tool reads all frames from the video, and shows a slide bar to allow a user to browse any image. For every image, it decodes the QR code to get the MARVEL recognized label locations and displays them on the screen, such as the white circle in Figure 5.6. To get the ground truth, a user must manually click on the correct label location on an image, which will trigger an optical flow tracking for all images afterwards. Since optical flow can fail with blurred images and large movements, the user must browse all images and manually click all incorrectly tracked ground truth

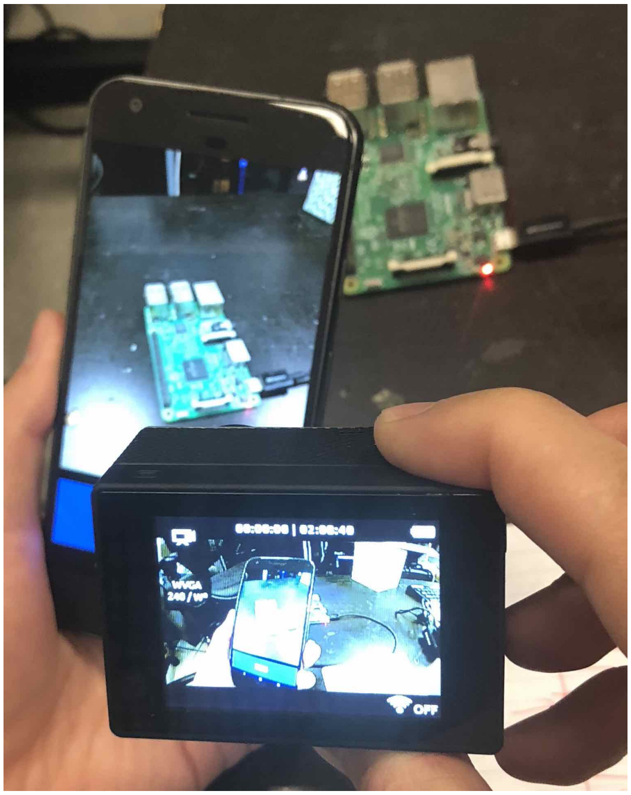
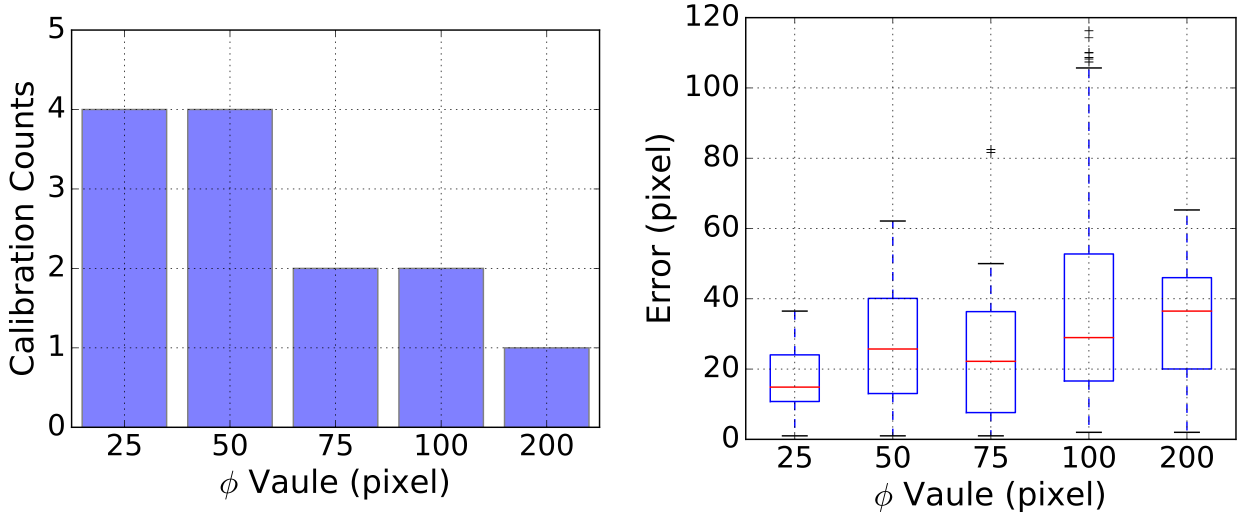


Figure 5.7: We record a 240 FPS video to measure the time for a basic camera app to reflect an LED change.

location, which will trigger another optical flow tracking starting from that frame. The user performs this labeling procedure until the ground truth locations in all frames are labeled correctly.

We deploy the server in the same local network on a Ubuntu machine with an Intel Xeon E5-2670 CPU and 256 GB of memory. It runs inside a Docker container (Docker version 17.09.0-ce). The server is lightly loaded most of the time. The smartphone communicates with the server using GRPC. When deploying the server, we collected a 3D model in a lounge in a campus building. Without loss of generality, we label one object (a speaker) in the lounge for our micro-benchmarks.

All images we use in this evaluation have the resolution 360 × 640. Our goal on accuracy is to minimize the distance (in pixel) of displayed label center and manually labelled ground truth. Note that low-resolution images are sufficient for accurate image localization [53], and MARVEL still displays identified labels on high-resolution camera feed (e.g., 1080 × 1920 on Lenovo Phab 2 Pro).



(a) Number of calibrations with different calibration threshold ϕ.

(b) Label placement errors with different calibration threshold ϕ.

Figure 5.8: Number of calibrations happened during a 20-second linear movement, and label placement errors with different calibration threshold ϕ. We select $\phi = 25$ for few calibrations and low errors.

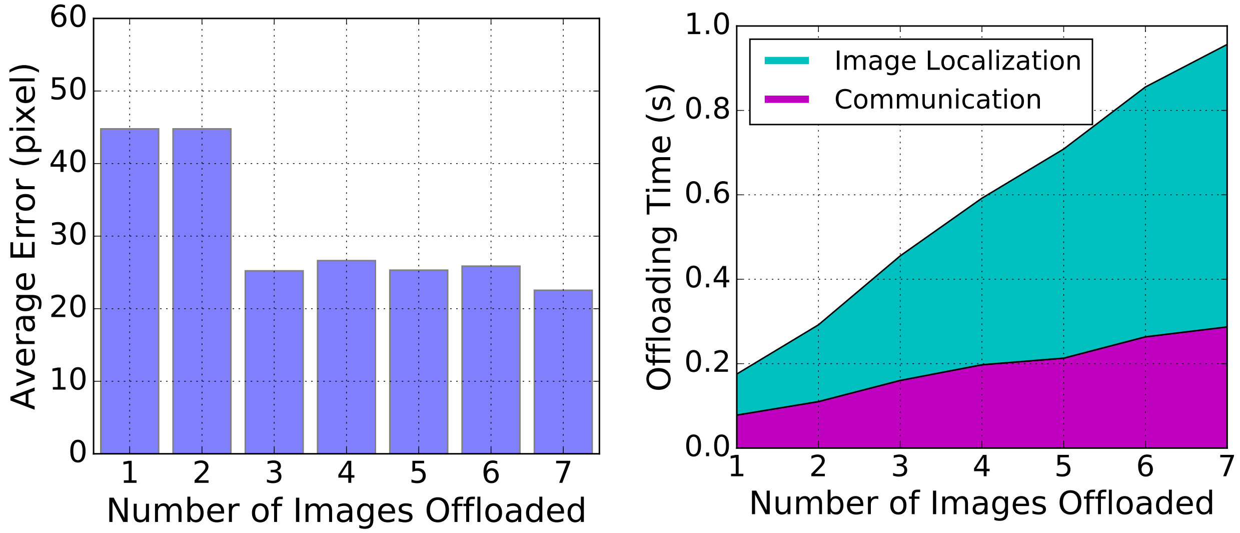
5.6.2 Calibration Threshold ϕ

Since offloading adds the most power overhead to the system, we study how the calibration threshold ϕ (in pixel) in Section 5.4.5 impacts the calibration performance. We perform the same linear movement with 5 different ϕ values during 5 runs, each takes about 20 seconds. Since image localization may return results with different accuracy between different runs, we use an AprilTag [89] in this experiment as the target object to minimize image localization errors.

Figure 5.8 shows the number of calibration happened and the errors in pixel during each run with different ϕ value. As expected, with a lower ϕ value, MARVEL client becomes more sensitive to the difference between optical flow and inertial tracking results. Therefore, as the ϕ value increases, less calibrations are triggered, and larger errors are observed. Based on our experiment, we choose 25 pixels to be the default calibration threshold in MARVEL. Note that users can easily change this value based on personal preferences on latency and accuracy.

5.6.3 Number of Images for Localization

As discussed in Section 5.4.5, we offload N selected images for calibration to ensure that the image localization is accurate by cross-validating localization results. There is a trade-off between accuracy and latency when deciding N. As N increases, accuracy



(a) Average error in pixel when offloading top N images.

(b) Offloading time breakdown when offloading top N images.

Figure 5.9: Error and offloading time according to the number of offloading images, N. Transmitting 3 images yields good accuracy as well as reasonable offloading time.

may increase but uploading time (communication overhead) and cloud processing time may also increase. To find the optimal number of images to offload, we conduct a microbenchmark by uploading the top 7 images among the recent 20 images, and obtain all the returned 7 image locations. To get the performance of offloading N ($N \leq 7$) images, we use the returned locations of only the top N images to compute the label location and encode it into the QR code. So the QR code contains 7 label locations at every moment.

Figure 5.9a shows the bar plot of the average errors of 109 samples with different N. As expected, transmitting 1 or 2 images yields larger errors than offloading more than 3 images. This happens because the 1 or 2 images can contain insufficient unique features, and more images give higher possibility of an easy-to-localize image. Note that when selecting offloading images, MARVEL does not directly extract feature uniqueness but uses indirect methods (i.e., gyroscope and edge detection) to avoid heavy image computation. In our experiment, offloading 3 images is good enough for accurate calibration even with these indirect methods.

Figure 5.9b shows the offloading time according to N. It shows that transmitting more images incurs higher offloading time, which can be problematic not only because it causes more energy consumption for communication but also because the returned calibration result can be too stale with accumulated IMU errors. Empirically, we observe acceptable IMU errors within 500 ms. Given that offloading time is just below 500 ms when transmitting 3 images, we set $N = 3$ as an optimal value in our environments. This verifies why MAR's latency should be decoupled from offloading latency. Note that this

		Pixel (ms)	Phab 2 Pro (ms)	
	Operation	Wi-Fi	Wi-Fi	Cellular
Baseline	Camera → App	67.3	71.0	
	App → Cloud → App	177.7	287.8	4299.5
	App → Screen	36.9	70.7	
	Total	**281.9**	**429.5**	**4441.2**
MARVEL	IMU → App	8.7	12.6	
	IMU Computation	0.3	0.4	
	App → Screen	36.9	70.7	
	Total	**45.9**	**83.7**	

Table 5.3: Average latency in millisecond at different steps in the baseline system and MARVEL. MARVEL has lower latency because it performs identification using only local information, including the calibration offset $\mathbf{P}_{\mathcal{I}}^{\mathcal{E}}(t)$.

500 ms latency does not affect our system latency.

5.6.4 End-to-End Latency

One of our primary design goal is to minimize the end-to-end localization latency, measured from the time of the sensor sampling (i.e., IMU or camera) to the time when identified labels are displayed on the smartphone screen. However, previous work [57, 97] overlooked the time spent on moving data between hardware buffer and user space memory, which can be categorized into two parts: (1) moving sampled data (e.g., IMU data or image) from sensor buffer to user space memory, and (2) moving images from user space memory to screen. To measure the time of (1) (denoted as $t_{\text{IMU}\rightarrow\text{App}}$ and $t_{\text{Camera}\rightarrow\text{App}}$), we simply compute

$$t_{\text{sensor}\rightarrow\text{App}} = t_{\text{app\_callback}} - t_{\text{sensor\_event}} \tag{5.8}$$

where 'sensor' can be either 'IMU' or 'Camera'. $t_{\text{sensor\_event}}$ is the timestamp of sampling recorded in the hardware driver in Android, and $t_{\text{app\_callback}}$ is acquired at the invocation of sensor data callback function. To measure the time of (2) (denoted as $t_{\text{App}\rightarrow\text{Screen}}$), we measure $t_{\text{Camera}\rightarrow\text{App}} + t_{\text{App}\rightarrow}$ together using a basic camera application[3], which continuously reads images from the camera to memory and sends them to screen. As shown in Figure 5.7, we use a GoPro Hero 3+ high frequency camera to record at 240 FPS, and manually count the number of frames it takes for the smartphone screen to reflect an LED change. On average, the results count 25 frames delay for Google Pixel and 34 frames delay for Lenovo Phab 2 Pro, which translates to 104.2 milliseconds and 141.7 milliseconds, respectively. We then subtract $t_{\text{Camera}\rightarrow\text{App}}$ from the total time to get $t_{\text{App}\rightarrow\text{Screen}}$. It is

[3] `https://github.com/googlesamples/android-Camera2Basic`

impossible to accurately measure the time to updating an UI component (e.g., drawing a box) from user space memory to screen, we assume its time is not longer than (2) because image data is usually larger.

Table 5.3 shows the end-to-end latency of a baseline MAR and MARVEL running on two different smartphones. The baseline MAR simply offloads image once at a time to a local server continuously for image localization and label identification. We measure the time it takes from serializing an image localization request to the return of identified labels, denoted as $t_{\text{App}\rightarrow\text{Cloud}\rightarrow\text{App}}$. In MARVEL, because IMU-based tracking only depends on the latest calibration offset $\mathbf{P}^{\mathcal{E}}_{\mathcal{I}}(t)$ stored locally, the only additional latency besides data moving is integration of sensor readings, which takes less than 0.4 ms on both smartphones.

Overall, the baseline system has 281.9 ms, and 429.5 ms end-to-end latency on each smartphone with Wi-Fi, whereas MARVEL introduces 45.9 ms and 83.7 ms latency, which are lower than our 100 ms latency goal. Note that this latency is even shorter than the basic camera app, which makes the camera app's latency (which we cannot control) will be the overall latency of MARVEL.

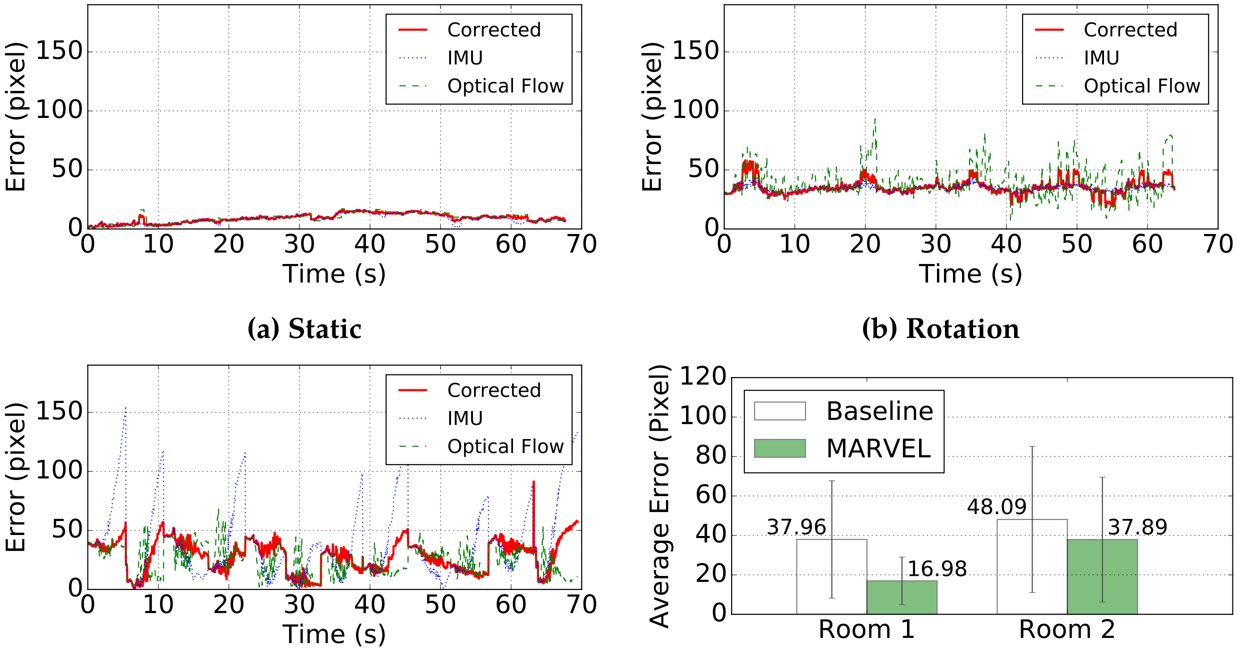
5.6.5 Label Placement Accuracy

Micro-benchmark

MARVEL is designed for high label recognition accuracy. We conduct three micro-benchmarks to show its effectiveness with three different operations of the smartphone: (1) holding still, (2) rotating only at a natural speed, and (3) moving around (e.g., translate) at a natural speed. Figure 5.10a-5.10c shows the pixel errors over a 60 second period of the experiment for the three different operations. For comparison, we plot the errors for three types of results, 'IMU' ($\mathbf{s}^{L}_{imu}(t)$), 'Optical Flow' ($\mathbf{s}^{L}_{of}(t)$), and 'Corrected' ($\mathbf{s}^{L}_{final}(t)$). To clearly visualize the effectiveness of our label placement correction using IMU and optical flow, we use calibration threshold $\phi = 100$ for less image offloading.

There is no ZUPT or image offloading event in Figures 5.10a and 5.10b. This verifies that the IMU's rotation data (from fusing gyroscope and gravity) can be accurate for a long time. Furthermore, Figure 5.10b shows that 'Optical Flow' generates larger errors than 'IMU'. Since optical flow fails with fast image changes or blurred images, the accuracy of 'Optical Flow' is degraded when the mobile device rotates quickly. Note that rotating a mobile device changes images on the screen faster than linearly moving it. As expected, 'Corrected' generates a better performance than 'Optical Flow' when rotating, because it combines IMU and optical flow results by using their own confidence values, inertial tracking compensates optical flow's errors with a higher confidence value.

Figure 5.10c shows that when moving the mobile device with translation, the situation becomes different. Now 'IMU' accumulates errors while 'Optical Flow' works better than Figure 5.10b. Because of cumulative errors of accelerometer, 'IMU' errors increases fast when moving, but are eventually corrected by either image offloading (1st and 4th spikes



(a) Static

(b) Rotation

(c) Translation

(d) Error Comparison

Figure 5.10: (a)-(c) shows label placement error (pixel) while conducting different actions, and (d) shows error comparison between a baseline system and MARVEL.

of IMU errors) or ZUPT (other spikes of IMU errors). We can see these local ZUPT events often occur before MARVEL detects visual-inertial inconsistency and avoid unnecessary offloading event. Compared to previous MAR systems relying on the cloud offloading for every object recognition [57, 95, 97, 107], MARVEL's selective offloading can reduce communication overhead significantly. When 'IMU' accumulates errors, 'Corrected' still maintains good accuracy by combining optical flow-based results with higher confidence values. There are also some points where 'IMU' is better than 'Optical Flow', where 'Corrected' is not affected by optical flow's errors. Overall, these results verify that our idea of using inertial and visual tracking together for annotation placement is valid in various movement scenarios.

End-to-End Accuracy

We also conduct experiments to compare the final accuracy between the baseline system and MARVEL for 2 objects (i.e., a speaker and a fan) in 2 different rooms, as shown in Figure 5.10d with mean and standard deviation. In these experiments, we perform the same natural motions for both systems, which include static periods, rotations, and translations. The baseline system continuously offloads one image to a local server and displays recognized label when server returns (without local optical flow). Note that the

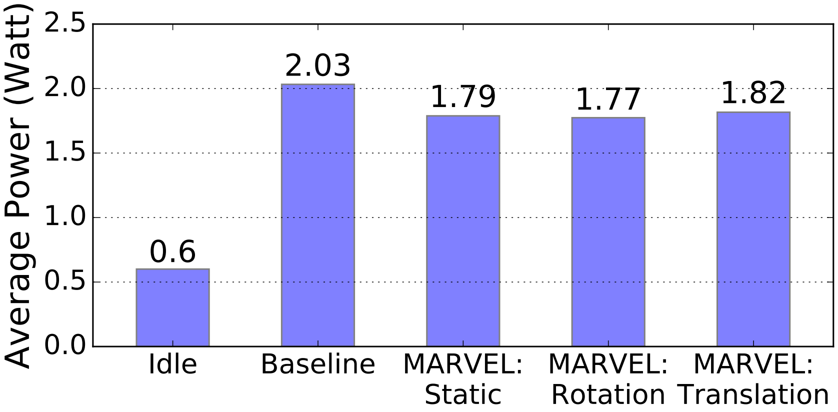


Figure 5.11: MARVEL incurs less optical flow and offloading than baseline (i.e., continuously optical flow and offloading), but also uses energy on IMU.

network overhead can be higher if the server is not deployed locally, which in turn can cause higher errors for the baseline system because of the delay. We use our default calibration threshold $\phi = 25$ for MARVEL.

Figure 5.10d shows that, even though MARVEL incurs less image localization, it provides higher accuracy than the baseline system due to the help of fast IMU processing and selective optical flow. This confirms that local computing is necessary to achieve accurate results in time-sensitive MAR applications.

5.6.6 Power Consumption

We measure MARVEL's power usage while performing different actions. Same as in Section 5.2.3, we use Trepn Profiler to measure the system-wide power consumption on a Lenovo Phab 2 Pro Android phone. To ensure a consistent displaying power consumption [108], we set the screen brightness to the same value, and conduct all energy measurements in the same room with all lights turned on.

We perform the following five actions: (1) idle (only system background tasks without MARVEL, for comparison), (2) baseline (continuously performing optical flow and image offloading, which is what previous work do), (3) static (pointing the smartphone to the object without moving, i.e., only inertial tracking), (4) rotation at a natural speed (inertial tracking, with occasional optical flow and image offloading), and (5) moving (i.e., translate) at a natural speed (inertial tracking, with occasional optical flow and image offloading). Each action is performed for around 90 seconds.

Figure 5.11 shows the average power usage of these actions. As expected, both baseline and MARVEL add significantly more power to the system. Even thought MARVEL uses IMU, it consumes around 0.2 Watt less of power than the baseline. Moreover, because of our selective optical flow and image offloading mechanism, rotation and translation do not introduce more power consumption than static. Since the phone has a 3.8V 2440 mAh battery, MARVEL can run approximately 0.58 hour longer than the baseline system if both run continuously[4]. In comparison, when operating with Wi-Fi connection, Overlay [57] consumes 4.5 Watt, and Glimpse [97] consumes 2.1 Watt.

5.7 Related Work

As a *holistic* MAR system, MARVEL builds on, but is clearly differentiated from a large body of prior work including computer vision, cloud offloading, and indoor localization.

Computer Vision:

Computer vision technology is a key enabler to identify an object on the screen. There are three common methods of object identification: image retrieval, CNN, and image localization. To the best of our knowledge, prior MAR work exploits image retrieval (e.g., Overlay [57], VisualPrint [79], and CloudAR [95]) or CNN (e.g., Glimpse [97]). However, as discussed in [53], image retrieval and CNNs involve a much higher overhead than image localization, which requires a database for accurate instance identification. Furthermore, when these two approaches are applied to MAR, both database and computation are offloaded to the cloud due to the limited capability of mobile devices, making an MAR system unable to identify objects without using the cloud, generating long identification latency.

In contrast, since image localization identifies an object through its *location*, it can be easily combined with other localization methods constructively. MARVEL takes this advantage and combines IMU-based localization, which can be quickly perofrmed *locally*, with more expensive image localization. This design choice enables the object identification procedure to be much less dependent on the cloud.

Cloud Offloading:

While cloud offloading has been common in the MAR regime due to the limited capabilities of mobile devices [109], it induces significant latency and energy consumption for communication [110, 111]. A number of studies have tried to use local computing to alleviate the problem. Overlay [57] obtains the mobile device's location from its sensor data, using it for offloading fewer images and reducing the visual search space on the cloud. VisualPrint [79] locally processes an image to extract its most distinct visual features and offloads these features instead of the complete image, which shifts offloading overhead to local image computation overhead. Glimpse [97] and CloudAR [95] rely on the cloud to identify an object but locally keep track of the identified object's location on the screen by using optical flow, enabling the object's annotation to move accordingly in real time,

[4] because $\frac{\text{(2.44 Amp hour)*(3.8 Volt)}}{\text{1.8 Watt}} - \frac{\text{(2.44 Amp hour)*(3.8 Volt)}}{\text{2.03 Watt}} \approx 0.58$ Hour

regardless of offloading latency. However, they still suffer from long latency (dependent on offloading) for identifying a new object on the screen. The prior work has something in common: the cloud takes *the main role* to identify objects and local computing performs *auxiliary work* to help the cloud; these systems cannot identify an object without using the cloud.

More generic cloud offloading work tries to find a sweet spot between offloading overhead and local computation overhead by opportunistic offloading according to network condition; offloading when network conditions are favorable enough to make it more efficient than local computing [112, 113]. These adaptive techniques cannot be applied to the above MAR work (image retrieval- or CNN-based identification) where using the cloud is not an option but a requirement for each object identification.

In contrast to the prior work, MARVEL lets local computing assume the main role for object identification (intertial localization) while the cloud assists it (sporadic calibration based on image localization). This approach decouples identification latency from offloading latency and significantly reduces offloading overhead.

Indoor Localization:

Indoor localization is a very well-researched topic. Along with various other techniques we discussed in Chapter 2, using IMUs on smartphones has also been investigated [48, 100, 114, 115]. While rotation is quite accurate by fusing gyroscope and gravity, a linear accelerometer is still error-prone due to cumulative errors [100], which makes inertial sensor-based localization valid only for a short period of time [48]. Various methods have been explored to calibrate the IMU errors, such as Wi-Fi signal strength [114] and light sensors [115]. Nevertheless, none of them achieve sufficient accuracy to support MAR. In contrast, MARVEL's image-based calibration significantly improves the accuracy of inertial localization.

On the other hand, the robotics community has shown that visual-inertial SLAM can achieve accurate and real-time indoor navigation locally on relatively powerful computers [107, 116, 117]. State-of-the-art visual-inertial SLAM uses both visual and inertial localization and fuses them together using the Extended Kalman Filter [118]. However, these techniques are too heavyweight to operate on ordinary mobile devices due to their limited storage and computation capabilities. More light-weighted software, such as Google ARCore [93] and Apple ARKit [94], can perform VSLAM on the latest smartphones using RGB cameras and IMU sensors, but are still limited by storage and consume considerable energy due to continuous image processing. SnapLink [53] offloads image localization computations to the cloud, which reduces local computation and storage demands but increases communication overhead by completely relying on the cloud (latency and energy consumption).

In contrast, MARVEL mainly uses local inertial information for localization and offloads images only when calibration is necessary. This reduces both computation and communication, improving latency and energy consumption.

5.8 Summary

In this chapter, we presented MARVEL, an MAR system for real-time annotation service, which is designed to run on regular mobile devices while achieving low latency and low energy consumption. Unlike previous MAR systems which suffer from high offloading latency and large energy consumption for image offloading and local image computation, MARVEL mainly uses local inertial data processing which is much faster (from 250 ms to <100 ms) and consumes less energy. While fast placing annotations continuously with local inertial tracking, MARVEL compensates its errors by using optical flow and image offloading. However, in doing so, MARVEL does its best to avoid any redundant image computation and image offloading. In MARVEL, a mobile device maximizes offloading interval, selects only a few query images when offloading, avoids image computation when selecting these query images, and minimizes optical flow computations. Experimental results show that the MARVEL system design enables to achieve low energy consumption without sacrificing accuracy.

Chapter 6

Conclusion

In this chapter, we draw lessons learned in this series of work, discuss other applications that can potentially benefit from our results and lessons, and talk about several directions of future work.

6.1 Lessons Learned

During the exploration of the solution to achieve accurate, intuitive, and fast appliance identification, we summarize several lessons learned, which we hope can help other people tackling the same set of problems.

Image localization is not perfect. Even though image localization has improved significantly with the advances in computer vision and robotics in recent years, its accuracy still depends on many factors, such as the quality of camera calibration, feature selection, and feature uniqueness in the environment [79]. The 3D model constructed in advance must cover features extracted from different viewing angles. Errors caused by those factors are not always easy to eliminate. Therefore it would be helpful to be able to detect those errors and react on them.

Data fusion is helpful. As we discussed in this dissertation, data fusion is used extensively in indoor localization. For our application, it will be helpful if more sensors and context information can be used together. For example, security cameras installed in a building can be used to detect or confirm user's location and orientation.

More powerful smartphones are on the market now. For example, many smartphones are equipped with a separate GPU, which can be used to perform simple computer vision algorithms in an energy efficient way. In fact, companies have been working on using them [93, 94]. We can definitely look into how local computer vision computation can help us rule out more image offloading.

User experience is important. Accurate, fast, and continuous image localization is crucial to our appliance identification application. However, when localization fails, a good design is also important to provide a smooth user experience. We can indicate that our localization failed and direct the user to point her camera to another location, which potentially has more unique visual features, before proceeding. Moreover, given a better understanding of the localization failure, a natural user experience can help us correct errors. For example, if a user's camera is too close to an appliance, we can ask her to move a bit further away to provide more visual context information.

6.2 Broader Impacts

In addition to appliance identification, we can imagine our work to inspire many other applications. Examples are included but not limited to the following applications.

Location-based Authorization. We demonstrated the robustness and ease of use of image localization inside buildings. This inspires us to build a location-based authorization service by asking a user to capture and upload an arbitrary video of her current room. The video can be localized by our system to validate the location, and the randomness of the video can be utilized to avoid replay attacks. This is useful in scenarios where only occupants in a room are allowed to control appliances in that room.

Indoor Navigation. We can use continuous image localization to localize people and display navigation guidance. For example, we can build an app in a museum that shows information about nearby art and navigate people to certain locations.

Building Chronology. Inspired by Scene Chronology [119], we can collected users' query images over time and construct a visual chronology of a building model. This not only visualizes the history of building change, but also can potentially help find lost items.

6.3 Future Work

With SnapLink and MARVEL developed, we can still envision more work to be done for better reliability and performance.

Integrate with category recognition. Although we have shown that image localization is robust to daily changes in the environment for months after the initial 3D model is created, image localization may fail to provide robust instance recognition when appliances themselves are moved, or enough of the environment is changed. We plan to add category recognition to validate localization results and detect environmental changes.

For example, if an appliance is removed, category recognition can detect that and issue an alert for label updates.

Crowd-source data. A 3D model can also be built from crowdsourced images and videos from building occupants. We plan to add this feature to reduce deployment overhead for building manager. Moreover, crowdsourced data can be used to detect changes and update 3D models over time.

Improve the labeling process. Besides data collection, labeling is the only aspect of SnapLink that requires human intervention. To minimize human effort, we have a labeling tool that enables one simple click-and-type interaction to label each appliance. We plan to push this further with more intuitive and automatic mechanisms. For example, we can allow users to point their smartphone camera at an appliance and label it by speaking its ID aloud. We can also use category recognition to provide an initial guess for appliance IDs.

Extract more from images. Apart from SURF, many other features can be utilized to help identify instances, including SIFT [39] and BRISK [120]. other information can be also extracted from the image. For example, Optical Character Recognition (OCR) [121], which recognizes text from images, can be used for places where text is present, such as room number plates.

Use visual markers. Our experiments show that image localization accuracy can be low when the viewing distance is small because these kinds of images cannot capture sufficient unique visual features. We have shown how QR codes can be used to mitigate this problem, although with some deployment overhead. We plan to integrate SnapLink with various visual markers, such as QR codes and AprilTags [89], and therefore improve its accuracy on appliances that are usually controlled within arm's range.

Handle moved objects. Localization-based instance recognition has a strong assumption that objects do not move. We recognize that is a limitation, but argue that many objects in our usage scenario are not usually moved. For example, exhibited items in a museum do not change often. Big appliances in a building, such as projectors and printers, do not move frequently either. Even if they are moved, obtaining an updated 3D model of a room only involves capturing a short video as well as clicking and typing several annotations [53]. Integrating MARVEL with neural network-based systems, such as Glimpse [97], can also help recognizing mobile objects.

6.4 Final Remarks

With the explosive number of smart appliances, our work made a step forward towards better interactions with them. Many existing technologies, such as image localization and indoor localization, are utilized in our systems. However, recognizing the advantages and disadvantages of each technology in our context, and learning the best ways to integrate them together for our purpose, has imposed many challenges along the way. We hope and believe our work and lessons can be a stepping stone for all future explorations in this area.

Bibliography

[1] Stephen Dawson-Haggerty, Andrew Krioukov, Jay Taneja, Sagar Karandikar, Gabe Fierro, Nikita Kitaev, and David Culler. "BOSS: building operating system services". In: *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. 2013, pp. 443–457.

[2] *Market size of the global smart home market from 2013 to 2025 (in million units).* 2018. URL: `https://www.statista.com/statistics/562298/smart-home-market-by-region/`.

[3] Bharathan Balaji, Jian Xu, Anthony Nwokafor, Rajesh Gupta, and Yuvraj Agarwal. "Sentinel: occupancy based HVAC actuation using existing WiFi infrastructure within commercial buildings". In: *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. ACM. 2013, p. 17.

[4] Romain Fontugne, Jorge Ortiz, Nicolas Tremblay, Pierre Borgnat, Patrick Flandrin, Kensuke Fukuda, David Culler, and Hiroshi Esaki. "Strip, bind, and search: a method for identifying abnormal energy consumption in buildings". In: *Proceedings of the 12th international conference on Information processing in sensor networks*. ACM. 2013, pp. 129–140.

[5] Varick L Erickson and Alberto E Cerpa. "Occupancy based demand response HVAC control strategy". In: *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*. ACM. 2010, pp. 7–12.

[6] *August Smart Lock*. `https://august.com/`. 2018.

[7] *Nest Hello Video Doorbell*. `https://nest.com/doorbell/nest-hello/overview/`. 2018.

[8] Bharathan Balaji, Arka Bhattacharya, Gabriel Fierro, Jingkun Gao, Joshua Gluck, Dezhi Hong, Aslak Johansen, Jason Koh, Joern Ploennigs, Yuvraj Agarwal, et al. "Brick: Towards a Unified Metadata Schema For Buildings". In: *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*. ACM. 2016.

[9] *Amazon Echo*. 2016. URL: `https://www.amazon.com/Amazon-Echo-Bluetooth-Speaker-with-WiFi-Alexa/dp/B00X4WHP5E`.

[10] *Google Home*. 2018. URL: `https://store.google.com/us/product/google_home`.

[11] Stephen Dawson-Haggerty, Xiaofan Jiang, Gilman Tolle, Jorge Ortiz, and David Culler. "sMAP: a simple measurement and actuation profile for physical information". In: *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*. ACM. 2010, pp. 197–210.

[12] Shahriar Nirjon, Jie Liu, Gerald DeJean, Bodhi Priyantha, Yuzhe Jin, and Ted Hart. "COIN-GPS: indoor localization from direct GPS receiving". In: *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. ACM. 2014, pp. 301–314.

[13] Paramvir Bahl and Venkata N Padmanabhan. "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. Ieee. 2000, pp. 775–784.

[14] S.P. Tarzia, P.A. Dinda, R.P. Dick, and G. Memik. "Indoor localization without infrastructure using the acoustic background spectrum". In: *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM. 2011, pp. 155–168.

[15] Jonathan Fürst, Kaifei Chen, Mohammed Aljarrah, and Philippe Bonnet. "Leveraging physical locality to integrate smart appliances in non-residential buildings with ultrasound and Bluetooth low energy". In: *Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on*. IEEE. 2016, pp. 199–210.

[16] Paul Martin, Bo-Jhang Ho, Nicholas Grupen, Samuel Munoz, and Mani Srivastava. "An ibeacon primer for indoor localization: demo abstract". In: *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*. ACM. 2014, pp. 190–191.

[17] X. Jiang, C.J.M. Liang, K. Chen, B. Zhang, J. Hsu, J. Liu, B. Cao, and F. Zhao. "Design and evaluation of a wireless magnetic-based proximity detection platform for indoor applications". In: *Proceedings of the 11th international conference on Information Processing in Sensor Networks*. ACM. 2012, pp. 221–232.

[18] A. Rai, K.K. Chintalapudi, V.N. Padmanabhan, and R. Sen. "Zee: Zero-effort crowdsourcing for indoor localization". In: *Proceedings of the 18th annual international conference on Mobile computing and networking*. ACM. 2012, pp. 293–304.

[19] Zheng Yang, Chenshu Wu, and Yunhao Liu. "Locating in fingerprint space: wireless indoor localization with little human intervention". In: *Proceedings of the 18th annual international conference on Mobile computing and networking*. ACM. 2012, pp. 269–280.

[20] Zheng Sun, Rick Farley, Telis Kaleas, Judy Ellis, and Kiran Chikkappa. "Cortina: Collaborative context-aware indoor positioning employing rss and rtof techniques". In: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*. IEEE. 2011, pp. 340–343.

[21] N.B. Priyantha, A. Chakraborty, and H. Balakrishnan. "The cricket location-support system". In: *Proceedings of the 6th annual international conference on Mobile computing and networking*. ACM. 2000, pp. 32–43.

[22] Chunyi Peng, Guobin Shen, Yongguang Zhang, Yanlin Li, and Kun Tan. "Beepbeep: a high accuracy acoustic ranging system using cots mobile devices". In: *Proceedings of the 5th international conference on Embedded networked sensor systems*. ACM. 2007, pp. 1–14.

[23] Jonathan Fürst, Kaifei Chen, Hyung-Sin Kim, and Philippe Bonnet. "Evaluating Bluetooth Low Energy for IoT". In: *Proceedings of the IEEE 1st Workshop on Benchmarking Cyber-Physical Networks and Systems*. IEEE. 2018.

[24] Joseph M Kahn and John R Barry. "Wireless infrared communications". In: *Proceedings of the IEEE* 85.2 (1997), pp. 265–298.

[25] Liqun Li, Pan Hu, Chunyi Peng, Guobin Shen, and Feng Zhao. "Epsilon: A Visible Light Based Positioning System." In: *NSDI*. Vol. 14. 2014, pp. 331–343.

[26] Jasurbek Khodjaev, Yongwan Park, and Aamir Saeed Malik. "Survey of NLOS identification and error mitigation problems in UWB-based positioning algorithms for dense environments". In: *annals of telecommunications-annales des télécommunications* 65.5-6 (2010), pp. 301–311.

[27] Zheng Sun, Aveek Purohit, Kaifei Chen, Shijia Pan, Trevor Pering, and Pei Zhang. "PANDAA: physical arrangement detection of networked devices through ambient-sound awareness". In: *Proceedings of the 13th international conference on Ubiquitous computing*. ACM. 2011, pp. 425–434.

[28] Patrick Lazik and Anthony Rowe. "Indoor pseudo-ranging of mobile devices using ultrasonic chirps". In: *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*. ACM. 2012, pp. 99–112.

[29] Ionut Constandache, Sharad Agarwal, Ivan Tashev, and Romit Roy Choudhury. "Daredevil: indoor location using sound". In: *ACM SIGMOBILE Mobile Computing and Communications Review* 18.2 (2014), pp. 9–19.

[30] Kiran Raj Joshi, Dinesh Bharadia, Manikanta Kotaru, and Sachin Katti. "WiDeo: Fine-grained Device-free Motion Tracing using RF Backscatter." In: *NSDI*. 2015, pp. 189–204.

[31] V Fox, Jeffrey Hightower, Lin Liao, Dirk Schulz, and Gaetano Borriello. "Bayesian filtering for location estimation". In: *IEEE pervasive computing* 2.3 (2003), pp. 24–33.

[32] Shwetak N Patel and Gregory D Abowd. "A 2-way laser-assisted selection scheme for handhelds in a physical environment". In: *UbiComp 2003: Ubiquitous Computing*. Springer. 2003, pp. 200–207.

[33] Ben Zhang, Yu-Hsiang Chen, Claire Tuna, Achal Dave, Yang Li, Edward Lee, and Björn Hartmann. "HOBS: head orientation-based selection in physical spaces". In: *Proceedings of the 2nd ACM symposium on Spatial user interaction*. ACM. 2014, pp. 17–25.

[34] Evan Welbourne, Leilani Battle, Garrett Cole, Kayla Gould, Kyle Rector, Samuel Raymer, Magdalena Balazinska, and Gaetano Borriello. "Building the internet of things using RFID: the RFID ecosystem experience". In: *IEEE Internet computing* 13.3 (2009).

[35] Jonathan Fürst, Gabe Fierro, Philippe Bonnet, and David E Culler. "BUSICO 3D: building simulation and control in unity 3D". In: *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*. ACM. 2014, pp. 326–327.

[36] Andrew Krioukov, Gabe Fierro, Nikita Kitaev, and David Culler. "Building application stack (BAS)". In: *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. ACM. 2012, pp. 72–79.

[37] Michel Vacher, Dan Istrate, François Portet, Thierry Joubert, Thierry Chevalier, Serge Smidtas, Brigitte Meillon, Benjamin Lecouteux, Mohamed Sehili, Pedro Chahuara, et al. "The sweet-home project: Audio technology in smart homes to improve well-being and reliance". In: *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE. 2011, pp. 5291–5294.

[38] Qifan Pu, Sidhant Gupta, Shyamnath Gollakota, and Shwetak Patel. "Whole-home gesture recognition using wireless signals". In: *Proceedings of the 19th annual international conference on Mobile computing & networking*. ACM. 2013, pp. 27–38.

[39] Nabeel Younus Khan, Brendan McCane, and Geoff Wyvill. "SIFT and SURF performance evaluation against various image deformations on benchmark dataset". In: *Digital Image Computing Techniques and Applications (DICTA), 2011 International Conference on*. IEEE. 2011, pp. 501–506.

[40] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features". In: *European conference on computer vision*. Springer. 2006, pp. 404–417.

[41] Kaifei Chen and Karthik Vadde. *Design and Evaluation of an Indoor Positioning System Framework*. Tech. rep. UCB/EECS-2016-16. EECS Department, University of California, Berkeley, Apr. 2016. URL: `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-16.html`.

[42] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins. "Global positioning System. Theory and Practice." In: *Global Positioning System. Theory and practice. J.. Springer, Wien (Austria), 1993, 347 p.* 1 (1993).

[43] Y. Chen, D. Lymberopoulos, J. Liu, and B. Priyantha. "Fm-based indoor localization". In: *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM. 2012, pp. 169–182.

[44] A. Ward, A. Jones, and A. Hopper. “A new location technique for the active office”. In: *Personal Communications, IEEE* 4.5 (1997), pp. 42–47.

[45] R. Want, A. Hopper, V. Falcão, and J. Gibbons. “The active badge location system”. In: *ACM Transactions on Information Systems (TOIS)* 10.1 (1992), pp. 91–102.

[46] L.M. Ni, Y. Liu, Y.C. Lau, and A.P. Patil. “LANDMARC: indoor location sensing using active RFID”. In: *Wireless networks* 10.6 (2004), pp. 701–710.

[47] H. Chang, J. Tian, T.T. Lai, H.H. Chu, and P. Huang. “Spinning beacons for precise indoor localization”. In: *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM. 2008, pp. 127–140.

[48] Fan Li, Chunshui Zhao, Guanzhong Ding, Jian Gong, Chenxing Liu, and Feng Zhao. “A reliable and accurate indoor localization method using phone inertial sensors”. In: *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. ACM. 2012, pp. 421–430.

[49] M. Azizyan, I. Constandache, and R. Roy Choudhury. “SurroundSense: mobile phone localization via ambience fingerprinting”. In: *Proceedings of the 15th annual international conference on Mobile computing and networking*. ACM. 2009, pp. 261–272.

[50] Kaifei Chen, Siyuan He, Beidi Chen, John Kolb, Randy H Katz, and David E Culler. “BearLoc: A Composable Distributed Framework for Indoor Localization Systems”. In: *Proceedings of the 2015 Workshop on IoT challenges in Mobile and Industrial Systems*. ACM. 2015, pp. 7–12.

[51] D. Pandya, R. Jain, and E. Lupu. “Indoor location estimation using multiple wireless technologies”. In: *Personal, Indoor and Mobile Radio Communications, 2003. PIMRC 2003. 14th IEEE Proceedings on*. Vol. 3. IEEE. 2003, pp. 2208–2212.

[52] Y. Gwon, R. Jain, and T. Kawahara. “Robust indoor location estimation of stationary and mobile users”. In: *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*. Vol. 2. IEEE. 2004, pp. 1032–1043.

[53] Kaifei Chen, Jonathan Fürst, John Kolb, Hyung-Sin Kim, Xin Jin, David E Culler, and Randy H Katz. “SnapLink: Fast and Accurate Vision-Based Appliance Control in Large Commercial Buildings”. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1.4 (2017), 129:1–129:27.

[54] Jian-tung Wang, Chia-Nian Shyi, T-W Hou, and CP Fong. “Design and implementation of augmented reality system collaborating with QR code”. In: *Computer Symposium (ICS), 2010 International*. IEEE. 2010, pp. 414–418.

[55] A de Freitas, Michael Nebeling, Xiang ‘Anthony’ Chen, Junrui Yang, ASKK Ranithangam, and Anind K Dey. “Snap-to-it: a user-inspired platform for opportunistic device interactions”. In: *Proceedings of the 34th Annual ACM Conference on Human Factors in Computing Systems (CHI 2016)*. 2016.

[56] Torsten Sattler, Bastian Leibe, and Leif Kobbelt. “Fast image-based localization using direct 2d-to-3d matching”. In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 667–674.

[57] Puneet Jain, Justin Manweiler, and Romit Roy Choudhury. “OverLay: Practical Mobile Augmented Reality”. In: *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. ACM. 2015, pp. 331–344.

[58] Quan Kong, Takuya Maekawa, Taiki Miyanishi, and Takayuki Suyama. “Selecting home appliances with smart glass based on contextual information”. In: *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM. 2016, pp. 97–108.

[59] Michael Deru, Kristin Field, Daniel Studer, Kyle Benne, Brent Griffith, Paul Torcellini, Bing Liu, Mark Halverson, Dave Winiarski, Michael Rosenberg, et al. “US Department of Energy commercial reference building models of the national building stock”. In: (2011).

[60] Jakob Nielsen. *Usability engineering*. Elsevier, 1994.

[61] Simon Mayer, Markus Schalch, Marian George, and Gábor Sörös. “Device recognition for intuitive interaction with the web of things”. In: *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*. ACM. 2013, pp. 239–242.

[62] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. “Object retrieval with large vocabularies and fast spatial matching”. In: *Computer Vision and Pattern Recognition, 2007. CVPR’07. IEEE Conference on*. IEEE. 2007, pp. 1–8.

[63] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[64] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. “Imagenet: A large-scale hierarchical image database”. In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 248–255.

[65] Liang Zheng, Yi Yang, and Qi Tian. “SIFT meets CNN: A decade survey of instance retrieval”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017).

[66] Herve Jegou, Matthijs Douze, and Cordelia Schmid. “Hamming embedding and weak geometric consistency for large scale image search”. In: *Computer Vision–ECCV 2008* (2008), pp. 304–317.

[67] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. “Lost in quantization: Improving particular object retrieval in large scale image databases”. In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE. 2008, pp. 1–8.

[68] *Project Tango*. 2016. URL: https://get.google.com/tango/.

[69] Jorge Fuentes-Pacheco, José Ruiz-Ascencio, and Juan Manuel Rendón-Mancha. "Visual simultaneous localization and mapping: a survey". In: *Artificial Intelligence Review* 43.1 (2015), pp. 55–81.

[70] Mathieu Labbe and Francois Michaud. "Appearance-based loop closure detection for online large-scale and long-term operation". In: *Robotics, IEEE Transactions on* 29.3 (2013), pp. 734–745.

[71] Jianxiong Xiao, Andrew Owens, and Antonio Torralba. "SUN3D: A database of big spaces reconstructed using sfm and object labels". In: *Computer Vision (ICCV), 2013 IEEE International Conference on*. IEEE. 2013, pp. 1625–1632.

[72] Josef Sivic and Andrew Zisserman. "Video Google: A text retrieval approach to object matching in videos". In: *null*. IEEE. 2003, p. 1470.

[73] Marius Muja and David G Lowe. "Scalable nearest neighbor algorithms for high dimensional data". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.11 (2014), pp. 2227–2240.

[74] Jason Zhi Liang, Nicholas Corso, Eric Turner, and Avideh Zakhor. "Image-based positioning of mobile devices in indoor environments". In: *Multimodal Location Estimation of Videos and Images*. Springer, 2015, pp. 85–99.

[75] Long Quan and Zhongdan Lan. "Linear n-point camera pose determination". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 21.8 (1999), pp. 774–780.

[76] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. "O'Reilly Media, Inc.", 2008.

[77] Robert B Miller. "Response time in man-computer conversational transactions". In: *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*. ACM. 1968, pp. 267–277.

[78] Fabian Gieseke, Justin Heinermann, Cosmin E Oancea, and Christian Igel. "Buffer kd Trees: Processing Massive Nearest Neighbor Queries on GPUs." In: *ICML*. 2014, pp. 172–180.

[79] Puneet Jain, Justin Manweiler, and Romit Roy Choudhury. "Low Bandwidth Offload for Mobile AR". In: *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. ACM. 2016, pp. 237–251.

[80] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. "Building rome in a day". In: *Communications of the ACM* 54.10 (2011), pp. 105–112.

[81] Radu Bogdan Rusu and Steve Cousins. "3d is here: Point cloud library (pcl)". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 1–4.

[82] Wenlu Hu, Brandon Amos, Zhuo Chen, Kiryong Ha, Wolfgang Richter, Padmanabhan Pillai, Benjamin Gilbert, Jan Harkes, and Mahadev Satyanarayanan. "The case for offload shaping". In: *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*. ACM. 2015, pp. 51–56.

[83] Michael P Andersen, Gabe Fierro, and David E Culler. "Enabling synergy in iot: Platform to service and beyond". In: *Journal of Network and Computer Applications* (2016).

[84] Helke Gabele and D Wüller. "The usage of digital cameras as luminance meters". In: *Germany, Cologne: University of Applied Sciences Cologne, Diploma Thesis* (2006).

[85] Peter Kieseberg, Manuel Leithner, Martin Mulazzani, Lindsay Munroe, Sebastian Schrittwieser, Mayank Sinha, and Edgar Weippl. "QR code security". In: *Proceedings of the 8th International Conference on Advances in Mobile Computing and Multimedia*. ACM. 2010, pp. 430–435.

[86] *ZBar bar code reader*. 2017. URL: `http://zbar.sourceforge.net/`.

[87] *Trepn Power Profiler*. 2018. URL: `https://developer.qualcomm.com/software/trepn-power-profiler`.

[88] I Poupyrev H Kato, Mark Billinghurst, and Ivan Poupyrev. "Artoolkit user manual, version 2.33". In: *Human Interface Technology Lab, University of Washington* 2 (2000).

[89] Edwin Olson. "AprilTag: A robust and flexible visual fiducial system". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 3400–3407.

[90] Kaifei Chen, Tong Li, Hyung-Sin Kim, David Culler, and Randy Katz. "MARVEL: Enabling Mobile Augmented Reality with Low Energy and Low Latency". In: *To appear in the Proceedings of the 16th ACM Conference on Embedded Network Sensor Systems*. ACM. 2018.

[91] *Microsoft HoloLens*. `https://www.microsoft.com/en-us/hololens`. 2017.

[92] Dimitris Chatzopoulos, Carlos Bermejo, Zhanpeng Huang, and Pan Hui. "Mobile Augmented Reality Survey: From Where We Are to Where We Go". In: *IEEE Access* (2017).

[93] *Google ARCore*. `https://developers.google.com/ar/`. 2017.

[94] *Google ARKit*. `https://developer.apple.com/arkit/`. 2017.

[95] Wenxiao Zhang, Sikun Lin, Farshid Hassani Bijarbooneh, Hao Fei Cheng, and Pan Hui. "CloudAR: A Cloud-based Framework for Mobile Augmented Reality". In: *Proceedings of the on Thematic Workshops of ACM Multimedia 2017*. Thematic Workshops '17. New York, NY, USA: ACM, 2017, pp. 194–200.

[96] Wenxiao Zhang, Bo Han, and Pan Hui. "On the Networking Challenges of Mobile Augmented Reality". In: *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*. ACM. 2017, pp. 24–29.

[97] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. "Glimpse: Continuous, real-time object recognition on mobile devices". In: *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. ACM. 2015, pp. 155–168.

[98] Karthik Kumar, Yamini Nimmagadda, and Yung-Hsiang Lu. "Energy conservation for image retrieval on mobile systems". In: *ACM Transactions on Embedded Computing Systems (TECS)* 11.3 (2012), p. 66.

[99] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints". In: *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM. 2016, pp. 123–136.

[100] Isaac Skog, Peter Handel, John-Olof Nilsson, and Jouni Rantakokko. "Zero-velocity detection - An algorithm evaluation". In: *IEEE Transactions on Biomedical Engineering* 57.11 (2010), pp. 2657–2666.

[101] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. "Fog computing and its role in the internet of things". In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM. 2012, pp. 13–16.

[102] Aaron Carroll and Gernot Heiser. "The systems hacker's guide to the galaxy energy usage in a modern smartphone". In: *Proceedings of the 4th Asia-Pacific Workshop on Systems*. ACM. 2013, p. 5.

[103] Berthold KP Horn and Brian G Schunck. "Determining optical flow". In: *Artificial intelligence* 17.1-3 (1981), pp. 185–203.

[104] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[105] Irwin Sobel and Gary Feldman. "A 3x3 isotropic gradient operator for image processing". In: *a talk at the Stanford Artificial Project in* (1968), pp. 271–272.

[106] Bruce D Lucas, Takeo Kanade, et al. "An iterative image registration technique with an application to stereo vision". In: (1981).

[107] Alejo Concha, Giuseppe Loianno, Vijay Kumar, and Javier Civera. "Visual-inertial direct SLAM". In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 1331–1338.

[108] Aaron Carroll and Gernot Heiser. "An Analysis of Power Consumption in a Smartphone". In: *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 21–21. URL: `http://dl.acm.org/citation.cfm?id=1855840.1855861`.

[109] Zhanpeng Huang, Weikai Li, Pan Hui, and Christoph Peylo. "CloudRidAR: A cloud-based architecture for mobile augmented reality". In: *Proceedings of the 2014 workshop on Mobile augmented reality and robotic technology-based systems*. ACM. 2014, pp. 29–34.

[110] Nayyab Zia Naqvi, Karel Moens, Arun Ramakrishnan, Davy Preuveneers, Danny Hughes, and Yolande Berbers. "To cloud or not to cloud: a context-aware deployment perspective of augmented reality mobile applications". In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM. 2015, pp. 555–562.

[111] Tristan Camille Braud, Dimitrios Chatzopoulos, Pan Hui, et al. "Future Networking Challenges: the Case of Mobile Augmented Reality". In: *Proceedings-International Conference on Distributed Computing Systems*. 2017.

[112] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. "MAUI: making smartphones last longer with code offload". In: *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM. 2010, pp. 49–62.

[113] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. "Clonecloud: elastic execution between mobile device and cloud". In: *Proceedings of the sixth conference on Computer systems*. ACM. 2011, pp. 301–314.

[114] Guobin Shen, Zhuo Chen, Peichao Zhang, Thomas Moscibroda, and Yongguang Zhang. "Walkie-markie: indoor pathway mapping made easy". In: *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*. USENIX Association. 2013, pp. 85–98.

[115] Qiang Xu, Rong Zheng, and Steve Hranilovic. "IDyLL: Indoor localization using inertial and light sensors on smartphones". In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM. 2015, pp. 307–318.

[116] Gabriele Bleser. *Towards visual-inertial slam for mobile augmented reality*. Verlag Dr. Hut, 2009.

[117] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. "Keyframe-based visual–inertial odometry using nonlinear optimization". In: *The International Journal of Robotics Research* 34.3 (2015), pp. 314–334.

[118] João Luís Marins, Xiaoping Yun, Eric R Bachmann, Robert B McGhee, and Michael J Zyda. "An extended Kalman filter for quaternion-based orientation estimation using MARG sensors". In: *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*. Vol. 4. IEEE. 2001, pp. 2003–2011.

[119] Kevin Matzen and Noah Snavely. "Scene Chronology". In: *Proc. European Conf. on Computer Vision*. 2014.

[120] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. “BRISK: Binary robust invariant scalable keypoints”. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2548–2555.

[121] Shunji Mori, Hirobumi Nishida, and Hiromitsu Yamada. *Optical character recognition*. John Wiley & Sons, Inc., 1999.