

**PRIORITIZING CONTENT OF INTEREST IN MULTIMEDIA DATA
COMPRESSION**

Chong Shao

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in
partial fulfillment of the requirements for the degree of Doctor of Philosophy in the
Department of Computer Science.

Chapel Hill
2018

Approved by:

Shahriar Nirjon

Russell M. Taylor II

David Brady

Ketan Mayer Patel

Marc Niethammer

©2018
Chong Shao
ALL RIGHTS RESERVED

ABSTRACT

CHONG SHAO: Prioritizing Content of Interest in Multimedia Data Compression
(Under the direction of Shahriar Nirjon and Russell M. Taylor II)

Image and video compression techniques make data transmission and storage in digital multimedia systems more efficient and feasible for the system's limited storage and bandwidth. Many generic image and video compression techniques such as JPEG and H.264/AVC have been standardized and are now widely adopted. Despite their great success, we observe that these standard compression techniques are not the best solution for data compression in special types of multimedia systems such as microscopy videos and low-power wireless broadcast systems. In these application-specific systems where the content of interest in the multimedia data is known and well-defined, we should re-think the design of a data compression pipeline. We hypothesize that by identifying and prioritizing multimedia data's content of interest, new compression methods can be invented that are far more effective than standard techniques. In this dissertation, a set of new data compression methods based on the idea of prioritizing the content of interest has been proposed for three different kinds of multimedia systems.

I will show that the key to designing efficient compression techniques in these three cases is to prioritize the content of interest in the data. The definition of the content of interest of multimedia data depends on the application. First, I show that for microscopy videos, the content of interest is defined as the spatial regions in the video frame with pixels that don't only contain noise. Keeping data in those regions with high quality and throwing out other information yields to a novel microscopy video compression technique. Second, I show that for a Bluetooth low energy beacon based system, practical multimedia data storage and transmission is possible by prioritizing content of interest. I designed custom image compression techniques that preserve edges in a binary image, or foreground regions of a color image of indoor or outdoor objects. Last, I present a new

indoor Bluetooth low energy beacon based augmented reality system that integrates a 3D moving object compression method that prioritizes the content of interest.

To Morty.

ACKNOWLEDGEMENTS

I would like to takes this chance to express my thanks to Prof. Shahriar Nirjon and Prof. Russell M. Taylor II, for their constant support and guidance in my research. I would also like to thank Prof. David Brady, Prof. Ketan Mayer Patel, and Prof. Marc Niethammer, for the meaningful discussions and suggestions.

TABLE OF CONTENTS

TABLE OF CONTENTS	vii
LIST OF TABLES	xiii
LIST OF FIGURES	xv
LIST OF ABBREVIATIONS	xx
1 Introduction	1
1.1 Research Questions	3
1.1.1 How to Better Compress Digital Videos with Specific Usages?	3
1.1.2 How to Better Compress Digital Images that Transmitted Through a Bandwidth Constrained System?	3
1.1.3 How to Compress Multiple Types of Application Data in a Bandwidth Constrained System?	4
1.1.4 How to Apply Generative Models in Extreme Compression?	4
1.2 A Brief Outline of the Proposed Methods	4
1.2.1 Content-Prioritizing Correlation-Based Microscopy Video Compression	4
1.2.2 Extreme Image Compression that Enables Image Beacons	5
1.2.3 Feature Selection and Key Point Extraction that Enables Indoor Augmented Reality Data Transmission over BLE Broadcasting	5
1.2.4 Generative Image Compression	5
1.3 Thesis and Contributions	6
1.4 Organization of the Rest of the Dissertation	8
2 Background	9
2.1 Shannon's Source Coding Theorem	9

2.2	Lossless Compression	10
2.2.1	Variable Length Coding	10
2.2.2	Dictionary Transform	11
2.2.3	Contextual Transform	12
2.2.4	Context Adaptive Variable Length Coding and Context Adaptive Variable Binary Arithmetic Coding	13
2.3	Prediction/Residual Framework	14
2.4	Multimedia Compression	15
2.4.1	Discrete Cosine Transform	16
2.4.2	Discrete Wavelet Transform	17
2.5	Previous Work on Domain-Specific Video Multimedia Data Compression	17
2.5.1	Region-Prioritizing Video Compression methods	18
2.5.2	Video Compression Methods That are Built on Existing Standards	19
2.6	Miscellaneous Topics in Modern Multimedia Compression	20
2.6.1	Point Spread Function in Image and Video Acquisition Process	20
2.6.2	Bandwidth Limited Transmission Channel: Bluetooth Low Energy Broad- casting Mode	21
2.6.3	Multimedia Data Quality Evaluation Using Statistical Tests	22
3	Video Compression to Preserve Analysis-Critical Information	24
3.1	Related Work	25
3.2	Overview	27
3.2.1	Analysis-Preserving Compression	27
3.2.2	Analysis-Aware Compression	29
3.2.3	Statistical Tests	30
3.3	Methods	31
3.3.1	Segmentation Stage	33
3.3.2	Compression Stage	35

3.3.3	Post-Processing Stage	37
3.3.4	Analysis-Aware Video Quality Measurement	38
3.4	Results	40
3.4.1	Analysis-Preserving Compression Results	40
3.4.2	Analysis-Aware Compression Results	45
3.5	Summary	55
4	Image Compression to Generate Energy Efficient Broadcast Image Data	58
4.1	Related Work	63
4.2	BLE System Characterization	65
4.3	Image Beacon and Use Cases	66
4.3.1	Long-term Surveillance Systems	67
4.3.2	Navigation Systems	67
4.3.3	Internet of Everything Minus the Internet	67
4.3.4	New Applications	67
4.4	Challenges in Building an Image Beacon	68
4.4.1	Limited BLE Bandwidth	68
4.4.2	The Case for Lossless Image Broadcast	69
4.4.3	The Case for Compressed Image Broadcast	69
4.5	Algorithm Design	71
4.5.1	Patch-Based Binary Image Compression Algorithm	71
4.5.2	Overview of the Color Image Beacon System	75
4.5.3	Multiview Capture and Depth Estimation	78
4.5.4	Color Image Encoding	83
4.6	Empirical Evaluation	88
4.6.1	Image Beacon Implementation Details	88
4.6.2	Binary Image Beacon System Evaluation	89

4.6.3	Color Image Beacon System Evaluation	94
4.7	Real Deployment	101
4.7.1	Write-Read-Recognize	101
4.7.2	Navigation in the Building	104
4.8	Summary	107
5	Extraction and Compression of Augmented Reality Content for Low Power Augmented Reality System	108
5.1	Related Work	111
5.2	Overview of MARBLE	112
5.2.1	Two Phases of MARBLE	114
5.2.2	Internal Modules and Basic Workflow	114
5.2.3	Advantage of MARBLE	115
5.3	Application Content Generation	116
5.3.1	Visual Features	117
5.3.2	Selecting Unique and Useful Features	118
5.3.3	Storing Camera Properties	119
5.3.4	Generating AR Content	119
5.4	Real-Time AR Content Rendering	120
5.4.1	BLE-based Location and Viewing Angle Estimation	121
5.4.2	Camera-based Location and Viewing Angle Estimation	121
5.4.3	Fusion of Multiple Sensor Inputs	122
5.4.4	Rendering Objects	123
5.5	Implementation Notes	123
5.6	Evaluation	124
5.6.1	Microbenchmarks	124
5.6.2	Algorithm Evaluation	127
5.7	Summary	132

6	Generative Compression as an Alternative Approach to Extreme Compression	133
6.1	Introduction	133
6.2	Related Work	135
6.3	System Architecture	136
6.3.1	Offline Training	137
6.3.2	Online Capture, Broadcast, and Render	138
6.4	Algorithm	139
6.4.1	Background on Variational Autoencoder (VAE)	139
6.4.2	Background on Generative Adversarial Networks (GAN)	139
6.4.3	VAE-GAN in Deep Beacon	139
6.4.4	Compressed VAE-GAN Embedding	140
6.5	Embedding Size Reduction Algorithm	140
6.6	Data Packet Format	141
6.7	Evaluation	142
6.7.1	Experimental Setup	142
6.7.2	The Choice of Embedding Size	144
6.7.3	Comparision with JPEG Encoding	145
6.7.4	Performance on Different Image Types	146
6.7.5	Impact of Number of Beacons	146
6.7.6	User Study	147
6.7.7	Comparison with Other Image Beacon Systems	149
6.8	Summary	149
7	Conclusion and Discussion	151
7.1	Summary of Contributions	151
7.1.1	Microscopy Video Compression	151
7.1.2	Image Compression for BLE Beacons	152

7.1.3	MARBLE: Augmented Reality Application Data Compression for Bluetooth Low Energy Devices	153
7.2	Discussion and Future Work	154
7.2.1	Microscopy Video Compression	154
7.2.2	Image Compression for BLE Beacons	154
7.2.3	MARBLE: Augmented Reality Application Data Compression for Bluetooth Low Energy Devices	156
7.3	Remaining Technical Issues	156
7.3.1	MARBLE: Generic Gesture Capture and Rendering	156
7.3.2	Smart Packet Rotation Strategy in Bluetooth Low Energy Broadcasting . .	157
7.3.3	No-Calibration Deployment for MARBLE	157
	BIBLIOGRAPHY	158

LIST OF TABLES

2.1	Comparison between common compression standards.	15
2.2	Comparison between common two-sample statistical tests.	23
3.1	Compression ratio for five real-world videos. The analysis-preserving method outperformed standard H.264 compression by large factors in three out of five test cases. In both cases, the resulting compressed video is in H.264 format so can be easily fed into analysis pipelines.	40
3.2	Comparison of multiple compression methods on four of the videos shown in the main text. The left and right numbers match those in TABLE 1. Bzip2 and lossless JPEG2000 compression on the original file sequence were usually worse and never much better than lossless H.264. Lossless JPEG2000 and Bzip2 on the filtered image set were always worse than lossless H.264.	41
3.3	Maximum tracking error when using lossy compression techniques and using my method in four cases (in units of the experiment noise floor, which is $\frac{1}{10}$ th of a pixel). In one video, a bead track was lost, indicated here as infinite error. When they are forced to achieve the same compression ratios, perceptually-tuned compression techniques have a large maximum impact on analysis results.	42
3.4	Mean and standard deviations of tracking error when using lossy compression techniques and using my method in four cases (in units of experiment noise floor, which is 1/10th of a pixel).	42
3.5	Compression ratio using the sliding window method and using lossless H.264 alone for the Fastbeads video. With longer windows, the beads moved across essentially every pixel in the image, so the method produced almost no improvement over lossless H.264 alone. With very short windows noise suppression was slightly reduced, causing an increase in file size. The optimal window size for this video was 20 frames, resulting in an approximately 2 improvement in file size. As expected, smaller dilation results in better compression (less fore-ground).	44
3.6	Achieved compression ratios for applying analysis-aware methods and lossless compression method on synthetic data and real data and maintain KS test p-score larger than 0.95 in the resulting video. For synthetic data, an improvement of around a factor of 2 was achieved above the earlier lossless method. For real data, which had more noise, the improvement was around a factor of 350.	54
5.1	System lifetime under different settings.	126
5.2	CPU and memory usage of Raspberry Pi	127

5.3	CPU and memory usage of Smartphone	127
6.1	Comparison among three image beacon systems.	149

LIST OF FIGURES

1.1	Comparison between common generic image/video compression (a) and prioritizing content of interest compression (b).	2
2.1	H.264 encoder and decoder as an example of prediction/residual framework.	15
2.2	Sample hand-written digit image (4096 bytes) and the DCT-compressed version of the image with the compressed size in bytes.	16
2.3	A 2D PSF convolved with a sample image, generates a blurred image.	21
3.1	Analysis-preserving video compression process.	28
3.2	Analysis-aware video compression process.	30
3.3	Mathematical morphology portions of the original algorithm: (a) a sub region of the first frame in the original video; (b) the correlation-based segmentation result without refinement; (c) the segmentation result after erosion refinement; (d) the segmentation result after erosion and dilation refinement. Some foreground regions in (d) are due to other beads that move into this region in later frames.	33
3.4	Plots of compressed file size comparing against original file size (in percentage) vs. correlation magnitude threshold on scores for four videos. Four subplots shows the result for four test videos respectively. Four plots share the same vertical axis. The horizontal flat line in every subplot indicates the compression result on that video with H.264 lossless technique. The curves become dashed when the analysis results on compressed videos differs from the analysis result on the original, indicating the limit compression without impacting analysis results. . .	34
3.5	Example frame from each of the videos tested, each named as in the description and tables.	35
3.6	Left to right: a) the starting position of a bead in a video and its moving trajectory; b) the resulting binary foreground/background map; c) illustration of the macroblocks that covers the frame; d) the resulting binary macroblock foreground/background labeling map.	37
3.7	Mean vs. variation intensity plot with centers of the two-means cluster.	39
3.8	Left: one frame of Fastbeads video before beads move; middle: foreground/background segmentation on whole video right: foreground/background segmentation within a 20-frame window.	44

3.9	Left to right: first frame of the original video, edge detection results: original video, video with $3.5\times$ compression, video with $4.6\times$ compression, video with $6.8\times$ compression.	45
3.10	Flow chart of the experiment steps with synthetic data	46
3.11	Sample video frames, left: synthetic video, right: real video	47
3.12	Scaled MSD values vs. compression ratio, for five groups of synthetic videos.	49
3.13	KS test p values vs. compression ratio. The horizontal line shows the KS test p score 0.95. The vertical line in every plot shows the compression ratio for the previous analysis-preserving method.	50
3.14	K-L divergence values vs. compression ratio, for five groups of synthetic videos. The vertical line in every plot shows the compression ratio for the previous analysis-preserving method.	52
3.15	Flow chart of the experiment steps with real data.	53
3.16	Scaled MSD values vs. compression ratio, for five groups of real videos.	54
3.17	KS test p values vs. compression ratio. The two horizontal lines showing the KS test p score 0.95 and 0.99, respectively. The vertical line in every plot shows the compression ratio for the previous analysis-preserving method.	55
3.18	K-L divergence values vs. compression ratio, for five groups of real videos. The vertical line in every plot shows the compression ratio for the previous analysis-preserving method.	56
4.1	An image beacon system.	59
4.2	The lifetime depends on packet transmission rate and signal strength.	69
4.3	A 64×64 resolution image compressed in high/low quality settings using JPEG/JPEG2000: (a) JPEG high quality, 1963 bytes (b) JPEG2000 high quality, 2026 bytes (c) JPEG lowest possible quality, 738 bytes (d) JPEG2000 lowest possible quality, 391 bytes.	70
4.4	Two types of 64×64 resolution image compressed in PNG (a) from natural scene, 12112 bytes (b) JPEG2000 high quality, 1012 bytes. PNG is good for handling images with large uniform color regions.	70
4.5	Beacon image processing pipeline.	71
4.6	(a) single spiral, (b) multiple spirals.	73

4.7	(a) original image; (b) result from patches generated from single spiral image; (c) result image from patches generated from single spiral image after morphology refinement; (d) result image from patches generated from multiple spiral images and k-means; (e) result from patches generated from multiple spiral images and k-means after morphology refinement.	74
4.8	Image processing stages.	77
4.9	Multiple views of a scene are used to estimate the depth map. Combined with standard image segmentation, this can identify the pixels of an image that may be of more interest than the rest, e.g. a foreground object.	78
4.10	Image compression details.	83
4.11	64×64 resolution building image compressed in high/low quality settings using my customized DCT/Wavelet/Triangle encoding: (a) Original image, (b) DCT 342 bytes, (c) Wavelet high 360 bytes, (d) DCT 1114 bytes, (e) Wavelet 1098 bytes, and (f) triangularization 366 bytes. For a similar compressed image size, DCT preserves less details than Wavelet method. But for low quality settings (about 350 bytes), Wavelet-encoded images have strange color block defects. Triangularization failed to preserves the information in the original image.	84
4.12	Original image and the triangularization-compressed image.	86
4.13	The process of Triangularization-based encoding.	87
4.14	Triangle texture averaging process.	88
4.15	Test images used in the empirical evaluation.	90
4.16	Image quality versus image size for different encoding methods.	91
4.17	Image quality versus beacon battery life for different images being approximated.	91
4.18	Image quality versus device lifetime for various number of beacons.	92
4.19	Image quality versus lifetime for different patch generation methods.	93
4.20	Image quality versus device lifetime for various patch set sizes. Note that the patch set with size $k=16$ is not simply a subset of the patch set of size $k=32$. They are individual k -means clustering results with different ks	94
4.21	Image quality versus device lifetime for various grid sizes per patch.	95
4.22	Test images used in the empirical evaluation.	96
4.23	Performance of IMU-guided view capture.	97
4.24	Performance of depth-refined segmentation.	98

4.25	Image quality versus image size for different encoding methods.	98
4.26	Image quality versus beacon battery life for different image types.	100
4.27	Image quality versus device lifetime for various number of beacons.	100
4.28	Subjective and SSIM scores for each image.	102
4.29	Test images compressed in three quality levels.	103
4.30	Photos of the object used in the experiment.	104
4.31	Responses from user study. The dashed line shows 33 percent chance of randomly choosing.	104
4.32	Map of second and third floors showing the navigation path.	105
4.33	Images stored at the six locations.	106
4.34	Subjective scores for each image.	106
5.1	The two phases of MARBLE indoor augmented reality system: capture (once) and render (many times).	109
5.2	System in Action. (a): in capture phase, a person's gesture and movement is captured. (b): in render phase, the virtual object avatar is rendered in the empty environment. (c): screenshot of the viewer's screen in render phase.	113
5.3	A block diagram illustrating the work flow of MARBLE.	113
5.4	The process of feature filtering: hundreds of ORB features are extracted from a reference camera image in (a) and feature entropy θ is computed. The resultant values are weighted by the 2D Gaussian function of (b). Four highest scored features are shown in a zoomed in region in (c). They are successfully matched with features extracted from a different image of the same scene even though some of the objects have been moved, as shown in (d).	117
5.5	The capture unit.	123
5.6	Run-time Analysis	125
5.7	Energy Consumption of every stage of MARBLE. The data for the capture phase is colored in blue. The data for broadcasting phase is colored in green. The data for viewing phase is colored in orange. The power data next to the stage labels are the working power of each stage. The number next to the bar charts are the energy consumption for rendering a frame of the virtual object.	126
5.8	Averaged accuracy of matching between ORB features found in 10 query images and the 4 ORB features from the reference image.	128

5.9	(a): the configuration of measurement points in the lab. (b): heat map showing the number of cameras being considered in visual feature matching.	129
5.10	Heat Map of the errors in cm on location estimation in 16 different measurement points. From left to right: BLE Beacon based estimation, Camera based estimation and Sensor Fusion using two signals.	130
5.11	Heat Map of the errors in angle on pose estimation in 16 different measurement points. From left to right: IMU based estimation, Camera based estimation and Sensor Fusion using two signals.	130
5.12	Human Gesture Sampling Frequency Impact at (a) Low Speed (b) High Speed.	131
6.1	The proposed Deep Beacon system.	133
6.2	The data flow of Deep Beacon system.	137
6.3	VAE-GAN model structure.	137
6.4	The Deep Beacon data packet format.	141
6.5	Samples of training and test image dataset. First 4 rows from top row to bottom: hand-written digits, birds, traffic signs, and flowers. The last row contains sampled images compressed by Deep Beacon system.	143
6.6	Bar chart showing the average MS-SSIM of four types of image data with different embedding sizes.	144
6.7	MS-SSIM vs. Image size for 100 hand-written digit images (upper plot) and 100 traffic sign images (lower plot). Images are compressed using JPEG (red dots) and the encoder module in Deep Beacon system (blue dots).	145
6.8	Average MS-SSIM score vs. expected system lifetime plot. Four curves represent four types of testing data. Each curve represents one type of test image data.	146
6.9	Average MS-SSIM score vs. expected system lifetime plot. Three curves represent the experiment result with using one, two, and three beacons, respectively. . .	147
6.10	The traffic sign types used in user study.	148
6.11	The chance in percentage of a participant making a correct answer in recognizing hand-written digit images and traffic sign images compressed by Deep Beacon system.	148
7.1	A sample football field image taken by a human-scale camera, and processed by the PSF-based segmentation method.	152

LIST OF ABBREVIATIONS

AR	Augmented Reality
ATT	Low energy ATtribute protocol
BLE	Bluetooth Low Energy
CBA	Customer Behavior Analysis
CISMM	Computer Integrated Systems for Microscopy and Manipulation
DCT	Discrete Cosine Transform
DWT	Discrete Wavelet Transform
GAN	Generative Adversarial Network
GATT	Generic ATtribute profile
GFSK	Gaussian Frequency Shift Keying
GPS	Global Positioning System
IMU	Inertial Measurement Unit
IoT	Internet of Things
ISM band	Industrial Scientific Medical band
K-L	KullbackLeibler
L2CAP	Logical Link Control and Adaptation Protocol
LZ	Lempel-Ziv
MARBLE	Mobile Augmented Reality with Bluetooth Low Energy
MSD	Mean Square Displacement
ORB	Oriented FAST and rotated BRIEF
PSF	Point Spread Function
QP	Quantization Parameter
ROI	Region Of Interest
RSSI	Received Signal Strength Indicator
SIFT	Scale-Invariant Feature Transform
SLAM	Simultaneous Localization and Mapping

SURF	Speeded Up Robust Feature
SSIM	Structure SIMilarity
URL	Uniform Resource Locator
UUID	Universally Unique IDentifier
VAE	Variational Autoencoder

CHAPTER 1

INTRODUCTION

Data compression is a process to take input data and to generate a compact representation of the input data with a shorter bit length. It is critical for many multimedia systems. With efficient data compression, transmission and storage of digital multimedia data become more affordable. During decades of development in the multimedia data compression field, a variety of popular multimedia compression methods including image and video compression methods have been standardized into CODEC specifications. Examples of the specifications are PNG, JPEG, and H.264. A multimedia compression specification typically includes a generic data compression component. They sometimes also include a smarter sampling strategy, a region of interest (ROI) data streaming or multimedia data modeling/parameterization and synthesis (Xu et al., 2014; Zhang and Bull, 2011; Balle et al., 2011).

New types of multimedia systems with specific purposes are constantly being built. Examples in recent years of the new systems include virtual reality headset (Oculus, 2015), augmented reality system (hol, 2017), high-throughput camera array system (Cribb et al., 2015), and low-power mobile data broadcasting “beacon” (ble, 2018). These new multimedia systems integrate advanced hardware such as a high-resolution display, low-power consumption chips and all types of sensors, along with new algorithms including high-accuracy object recognition and real-time scene understanding.

New technologies enable new systems. They also draw challenges on designing and building them. Sometimes new systems’ data generation, data storage and transmission, and data consumption components need to be redefined and are sometimes completely different from the ones in traditional systems. Conventional compression methods such as the MPEG-4 Part 10 (H.264) standard (Wiegand et al., 2003) and the H.265 standard (Sullivan et al., 2012) no longer work

efficiently on these new systems. For example, a set of microscopy videos produced by an automatic microscope array system is only used by an analysis program in a cell-mechanics experiment pipeline. Most of the pixels in the video are never touched by the analysis program, but a generic video compression technique such as H.264 cannot accurately separate the useful information from noise in the microscopy videos. Another example is the designing of an “image beacon” system, where image data needs to be transmitted over Bluetooth low energy (BLE) broadcasting channel. The BLE broadcasting channel only has 30 bytes of payload size. However, the standard JPEG compression rarely compresses a 64×64 image smaller than 200 bytes.

During my Ph.D. work, I design compression techniques that are used for new types of multimedia systems. I will show that, *prioritizing content of interest* is one useful approach in creating new compression methods for these multimedia systems (Figure 1.1). This dissertation describes several examples of new multimedia systems and their challenges in data compression, followed by my solution to the problem.

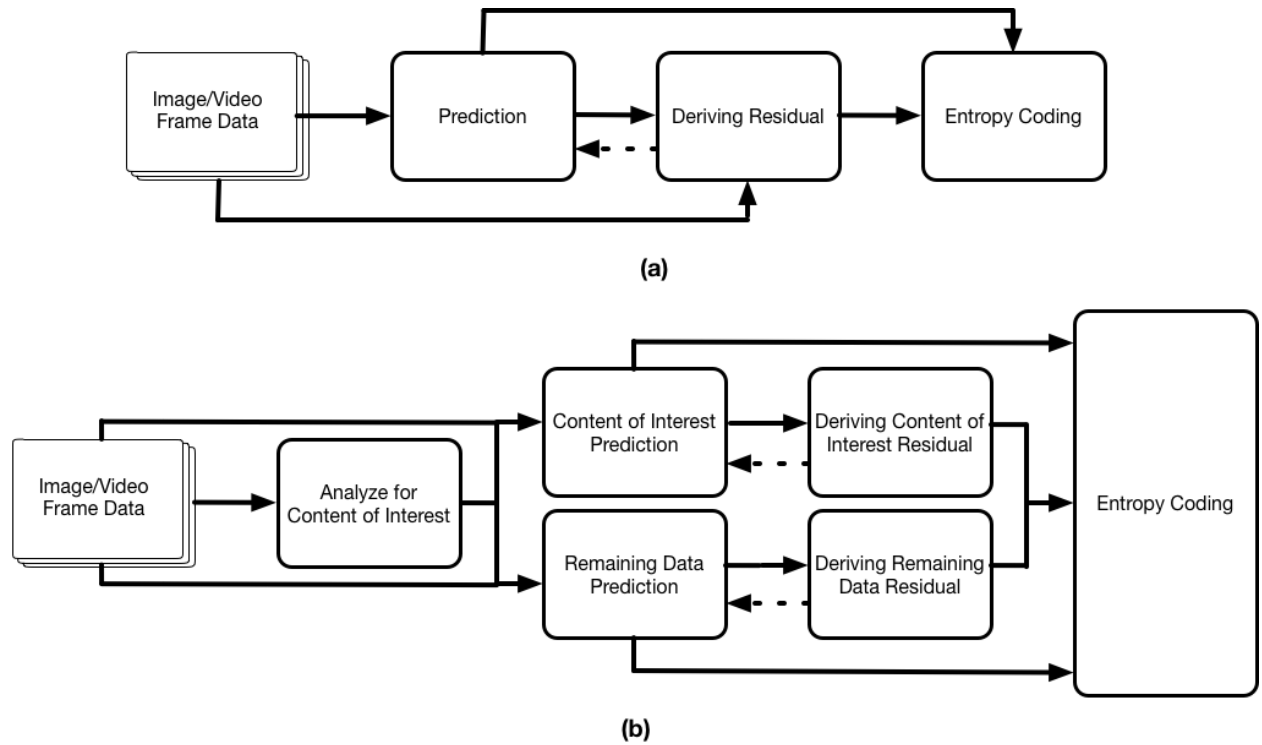


Figure 1.1: Comparison between common generic image/video compression (a) and prioritizing content of interest compression (b).

1.1 Research Questions

In this section, I list the important questions I addressed in this dissertation.

1.1.1 How to Better Compress Digital Videos with Specific Usages?

We observe the increase of generated videos that have specific usages, especially videos that are analyzed by algorithms. Examples are surveillance videos and microscopy videos generated by analytical experiment pipelines. In some use cases such as a high-throughput screening (hig, 2017), there is a requirement to compress the video data to accommodate the limited storage. Standard video compression methods that assign equal importance to every pixel in the frame have difficulties in achieving small enough compressed video size. The question of how to identify the analysis-critical component in a microscopy video and how to generalize the analysis-critical component identification method are addressed in this dissertation.

1.1.2 How to Better Compress Digital Images that Transmitted Through a Bandwidth Constrained System?

The Internet of things (IoT) technology is gradually being introduced into our daily environment. Tiny, special-purpose, and sometimes battery-powered IoT devices bridge our physical world and the Internet. At the same time, billions of digital images are being uploaded to the Internet every day (Ima, 2015). There is a requirement to keep and transmit image data through IoT devices. But many IoT data transmission protocols have extremely small bandwidths. Two examples are the BLE protocol with 30 bytes (it becomes 200 Bytes in a BLE 5.0 packet in the near future) of packet payload size and the LoRa technology (LoR, 2018) with the data rate 38522 bytes per day (lor, 2016). This raises the requirement of developing a technology to compress images into the same order of magnitude as 200 bytes. Standard image compression methods are designed to generate high-quality images that are visually close to the original image, and the compressed image size is rarely below 200 bytes for a 64×64 image. I wanted to build a new image compression method that focuses more on small compressed image size. Specifically, the question is: how to design an

extreme compression method to compress a sufficiently large image (no less than 64×64 in size) into less than 200 bytes, so that the data can be carried by one a BLE 5.0 broadcasting packet?

1.1.3 How to Compress Multiple Types of Application Data in a Bandwidth Constrained System?

Some bandwidth-limited systems handle multiple types of data. It is an open problem to design a mechanism to handle data with different properties in such a system. I explored applying new compression methods in a bandwidth constrained system in compressing multiple types of data.

1.1.4 How to Apply Generative Models in Extreme Compression?

Recently, deep network models have shown strength in generating realistic fake image data. This opens the door to applying such a model in extreme compression. I also explored this direction.

1.2 A Brief Outline of the Proposed Methods

Here I list the methods I proposed in this dissertation, to address the questions in the last section.

1.2.1 Content-Prioritizing Correlation-Based Microscopy Video Compression

We seek a custom video compression method that efficiently compresses a microscopy video to preserve the analysis-critical information so that the compressed video generates an analysis result that matches the one for the original uncompressed video. I propose two new microscopy video compression methods that efficiently compress a microscopy video, so that the compressed video does not change the analysis result (analysis-preserving), or it generates an analysis result with a change within the range of the change caused by noise, comparing to the result from the uncompressed video (analysis-aware). I will present a correlation-based segmentation method to locate the analysis-critical region in every video frame.

1.2.2 Extreme Image Compression that Enables Image Beacons

In building a self-contained image data storage and broadcast system using the BLE broadcasting channel, standard image compression techniques such as JPEG and PNG cannot generate sufficiently small compressed images. I design two new image compression methods that enable 64×64 dimension image storage and broadcast via BLE broadcasting channel. The first method is a dictionary-based patch matching binary image compression method. The second method is a foreground-preserving color image compression method that uses depth information in identifying the foreground in the input image.

1.2.3 Feature Selection and Key Point Extraction that Enables Indoor Augmented Reality Data Transmission over BLE Broadcasting

I demonstrate a data reduction approach for indoor augmented reality systems, which addresses question 1.1.3. Using BLE beacons as the storage and transmission component in an indoor AR system brings many benefits, such as long battery-powered lifetime, easy content data update, internet-free, and low cost. To overcome the challenges of small data bandwidth of Bluetooth Low Energy (BLE) protocol, I designed a 3D object data compression, along with a new entropy and location-based visual feature selection method. They enable indoor vision-based positioning and 3D content broadcasting via BLE broadcasting channel.

1.2.4 Generative Image Compression

I explore an alternative approach for enabling image data storage and broadcast using BLE beacons. I design a Deep Beacon system that integrates a variational autoencoder generative adversarial network, which enables 64×64 color image broadcast via BLE broadcasting channel. Comparing to the previous-explained image beacon system, Deep Beacon supports wider categories of image data. The input image does not have to have the notion of foreground and background.

1.3 Thesis and Contributions

Thesis: Some multimedia systems handle image or video data that has the notion of content of interest. Identifying and prioritizing image and video's content of interest in these systems lead to new image and video compression algorithms including analysis-preserving microscopy video compression, analysis-aware microscopy video compression, patch-based binary image compression, and adaptive color image encoding that enable smaller compressed data size that prediction-residual based generic methods cannot achieve. With the smaller data size, the image or video data compressed by prioritizing content of interest can also have the same quality as the data produced by generic methods, measured with a generic image or video quality metric such as Structural Similarity (SSIM).

In the following paragraphs, I highlight the contributions of this dissertation. Each contribution is labeled by the corresponding chapter number in this dissertation.

- I propose the first video compression technique in the literature that is based on correlation-based segmentation. The segmentation method identifies regions in the video that contain information. The compression method can achieve up to 20x better compression than H.264. I invent a correlation-based video frame segmentation technique based on the point spread function (PSF). I integrate the segmentation technique into the compression process. The PSF is involved in the imaging processes of a wide variety of optical systems. Therefore this segmentation method can be generalized to many video modalities (Chapter 3).
- I propose a new method to evaluate the quality of a microscopy video based on a statistical test. The method gives a video quality estimation based on the information loss of the analysis-critical part of the video. It takes the noise into consideration (Chapter 3).
- I propose a new binary image compression method as a central component of a binary image beacon system. This is a dictionary-based patch matching algorithm. The input image is compressed into a set of patch indices. The result size can be less than 40 bytes. The method can be generalized into other use cases that require binary image compression (Chapter 4).

- I develop and evaluate the first binary image beacon system. The evaluation compares the system performance under two instances of the binary image compression module trained with two different sets of data. The evaluation gives a deep insight of the system performance under different settings and proves its effectiveness in practice (Chapter 4).
- I propose a new color image compression method as a central component of a color image beacon system. This method takes the depth information of the object in the input image to generate an accurate segmentation for compressing the content of interest. It can generate less than 200 bytes of compressed image data. The method can be generalized into other use cases that require color image compression, and depth information of the image can be retrieved (Chapter 4).
- I develop and evaluate the first color image beacon system. The system supports storage and broadcast of color images of objects indoor and outdoors. My evaluation compares the system performance under different system settings and different types of input data. My user study with a real deployment of the system proves the system's effectiveness in practice (Chapter 4).
- I develop and evaluate a new Internet-free beacon AR system that combines camera capture, IMU sensing input, and BLE signal strength in indoor positioning. The system provides a low-cost and extensible solution to indoor AR experience with common mobile devices (Chapter 5).
- I invent an Oriented FAST and rotated BRIEF (ORB) visual feature selection algorithm as a critical component in the BLE-based indoor AR system. The method selects the most useful four visual features among hundreds of candidates. It enables visual feature storage and broadcast via BLE channels for indoor positioning. The method can be generalized to other systems that require visual feature filtering. It can also be extended to other types of visual features (Chapter 5).

- I design and develop the first image beacon system that integrates variational autoencoder generative adversarial network (VAE-GAN). The system supports a wide range of binary and color images. It enables compression of a color 64×64 image into less than 20 bytes. I analyzed the system performance under different settings of the VAE-GAN model (Chapter 6).

1.4 Organization of the Rest of the Dissertation

The background material that is relevant to multiple projects across my research work is explained in Chapter 2. Other project-specific previous work information is distributed into other chapters. The background chapter is followed by the three major parts of my dissertation. Chapter 3 describes my research work on microscopy video compression. Chapter 4 explains the proposed image compression techniques for Bluetooth low energy beacon devices. Chapter 5 describes an indoor augmented reality system based on Bluetooth low energy beacons. It also highlights the proposed 3D object motion data compression method integrated into the system. Chapter 6 discusses using generative models in extreme compression, this research results in a new Deep Beacon system for generic image storage and compression over BLE. Chapter 7 concludes the dissertation and discusses potential future research directions.

CHAPTER 2 BACKGROUND

This chapter contains the background material related to my dissertation. Section 2.1 discusses Shannon’s Source Coding theorem, which builds the foundation of information theory. In Section 2.2, I highlight a set of lossless compression techniques. I give a brief introduction to each of them. Section 2.3 describes the prediction/residual framework that is used in many common multimedia compression systems. After that, in Section 2.4, I explain the important aspects of common multimedia compression standards, including the two widely used signal transform methods. This is followed by a summary of existing domain-specific multimedia compression techniques in Section 2.5. I finish this chapter with Section 2.6. It includes several properties in multimedia systems that I used in building new compression methods and the background knowledge of new methodologies used in my compression algorithm design.

2.1 Shannon’s Source Coding Theorem

In 1948, Shannon formulated the source coding theorem (Shannon, 2001). The theorem gives a theoretical bound of the expected bit length of input codewords. The bound is a function of the entropy of the input symbols.

The mathematical version of the theorem is:

Let X be a triple (x, A_x, P_x) , where the outcome x is the value of a random variable that takes one of the possible values in set $A_x = \{a_1, a_2, \dots, a_I\}$ with probabilities $P_x = \{p_1, p_2, \dots, p_I\}$. $P(x = a_i) = p_i, p_i \geq 0$. And X is with entropy $H(X) = H$ bits. Given $\epsilon > 0$ and $0 < \delta < 1$, there exists a positive integer N_0 such that for $N > N_0$:

$$|\frac{1}{N}H_\delta(X^N) - H| < \epsilon \quad (2.1)$$

Here H_δ is the essential bit content, which is defined as $H_\delta = \log_2|S_\delta|$, in which S_δ is the smallest subset that satisfies $P(x \in S_\delta) \geq 1 - \delta$ for a given δ .

A restatement of the theorem is that: having the input data as N independent and identically distributed (i.i.d.) random variables each with entropy $H(X)$, it can be compressed into more than $NH(X)$ bits with negligible risk of information loss, as $N \rightarrow \infty$. On the other hand, if they are compressed into fewer than $NH(X)$ bits, it is virtually certain that information will be lost.

Shannon's source coding theorem is tightly related to multiple lossless compression algorithms. However, it does not take the dependency between variables into consideration, which is widely used by many standard lossless compression algorithms such as contextual transform.

2.2 Lossless Compression

Lossless compression refers to the category of compression algorithms that generate compressed data that can be used to fully reconstruct the original data. In multimedia compression systems, lossless compression is usually used in an entropy coding stage, where the transformed multimedia data is losslessly encoded. In this section, I review three common lossless compression techniques: variable length coding (VLC), dictionary transform, and contextual transform. At the end of the section, I describe the two lossless compression methods that are customized for entropy encoding in video compression.

2.2.1 Variable Length Coding

Variable length coding (VLC) divides input data into a set of symbols. It then represents each symbol by a variable length of bits.

The idea of variable length coding is to analyze the frequency of every codeword of the input data so that the symbols with shorter lengths are assigned to codewords that appear more frequently. Two most common examples are Huffman coding (Huffman, 1952) and arithmetic coding (Rissanen, 1976; McAnlis and Haecky, 2016). Both Huffman coding and arithmetic coding are used in the entropy coding stage of H.264 encoding process (Richardson, 2011).

The difference between Huffman coding and arithmetic coding is that Huffman coding uses symbols of integer length to represent each input codeword. Therefore, the symbol length may not exactly match the frequency of the corresponding codeword. Arithmetic coding addresses this problem by allowing floating number length of symbols. This is implemented by assigning intervals to each codeword at every encoding and decoding stage recursively. For example, we can assign symbol A to interval $[0, 0.7)$ and assign symbol B to interval $[0.7, 1]$. A is then represented by any floating number in interval $[0, 0.7)$. AB is represented by any floating number in interval $[0.7 * 0.7, 1]$. Compression is achieved from the shorter bit length for representing a floating number comparing to the bit length of the original symbol sequence.

2.2.2 Dictionary Transform

A dictionary transform first builds a dictionary from the input data. It then uses the built dictionary to transform the data. The result is then sent to a statistical encoding routine such as VLC. Using dictionary transform as a pre-processing step before VLC makes the compression more effective. One example of dictionary transform is the Lempel-Ziv (LZ) transform (Ziv and Lempel, 1977).

Given the input data as a sequence of symbols, the LZ transform works by maintaining a search buffer with a limited size. The algorithm scans the input sequence. During the scanning, it fills in the search buffer with visited symbols. The unvisited symbols starting from the current scanning position are encoded by the offset and length of the matched symbol sub-sequences in the current scanning buffer content.

To explain the LZ transform more precisely, I use an example: a current scanning buffer TOBEORNOT, and the unscanned sequence TOBET. So the entire input sequence of symbols in this example is TOBEORNOT | TOBET. The ‘|’ mark shows the current scanning position. The algorithm tries to match the longest sequence from the current scanning position in the current scanning buffer. The result is TOBE. This sequence is encoded as a pair of offset and length. In my example, it is (9, 4). After this, the scanning buffer is shifted to the right by four steps. Now it is ORNOTTOBE, and the current scanning position is also shifted to the right by four steps.

2.2.3 Contextual Transform

Contextual transform makes use of the contextual properties of the data. For example, in some types of data, adjacent codewords are more likely to have the same value. Hence, we can encode the sequence of repetitive codewords by a pair of a symbol and number of runs of that symbol. This transform is called run-length encoding. Run-length encoding is the most basic type of contextual transform. When the data tends to have successive increasing or decreasing values, starting from the second symbol, we can represent each symbol as the difference from the previous symbol. Then the result sequence of differences data will be more compactly represented by another pass of run-length encoding. This process is called delta-encoding.

The Burrows-Wheeler transform is another noteworthy contextual transform. It is effective in compressing data that certain orders of a pair of tokens are more likely to appear in. For example, in the English language, the letter ‘h’ is likely to appear after the letter ‘t.’ The Burrows-Wheeler transform works by rotating the length n data vector $n - 1$ times, followed by sorting by the first token. The transformed data is represented by the last column of the result sorted data matrix plus the index of the location of the original input data vector. When certain orders of symbols are more likely to appear in the data, this representation tends to be more compact than the original. This is because that the last column of the result data matrix tends to contain many repeated characters when the first column of the data matrix is sorted.

I give an example of the process of the Burrows-Wheeler transform. I set the input sequence to be transformed as BANANA. The Burrows-Wheeler transform generates a list of sequences with length equal to the input sequence length by rotating the original sequence, and sorting by the first letter: ABANAN, ANABAN, ANANAB, BANANA, NABANA, NANABA. The transform output is the sequence of the last letter in the list: NNBAAA, and the index of the original sequence in the list: 3. This pair of items is sufficient for recovering the original input sequence. The Burrows-Wheeler transform generates a sequence that is easier to compress because of the property that certain orders of a pair of tokens are more likely to appear in, as described in the last paragraph. Therefore, when the sequence of the first letter in the list is sorted, the corresponding sequence of the last letter is likely to have a repeating pattern.

2.2.4 Context Adaptive Variable Length Coding and Context Adaptive Variable Binary Arithmetic Coding

Context Adaptive Variable Length Coding (Karczewicz and Ridge, 2004) (CAVLC) and Context Adaptive Binary Arithmetic Coding (Marpe et al., 2003) (CABAC) are custom versions of VLCs for video compression. They are used in the entropy coding stage of H.264 and H.265 video compression standards. H.264 standard contains three types of profiles for different use cases. They are baseline profile, main profile, and high profile. In H.264, CAVLC is supported in all profiles. CABAC is only supported in main profile and high profile. CABAC can usually achieve a higher compression ratio. However, it requires more computation resource.

In H.264, one intermediate step is to encode a 2D matrix of numbers representing the frequency component coefficients from a DCT transform. CAVLC uses run-length encoding to encode the trailing zeros in every DCT coefficient block. The non-zero coefficients are encoded using a look-up table. There exists a set of candidate look-up tables. The choice of the look-up table depends on the non-zero coefficients in the neighboring DCT coefficient blocks. Hence, we call the algorithm “context adaptive.”

CABAC is similar to CAVLC because in both algorithms a probability model is selected among a set of candidates according to the current context. Differing from CAVLC, CABAC requires every input data to be *binarized* before compression. This is done by explicitly mapping every input symbol that is non-binary valued onto a sequence of binary decisions. It uses arithmetic coding rather than a look-up table.

2.3 Prediction/Residual Framework

We can extract a common pattern from the workflow from most multimedia compression systems. This pattern is named the *Prediction/Residual Framework*. The prediction/residual framework compresses data by making predictions on a subset of the data and use the prediction to produce a more compact representation of the data. A prediction can be *temporal* or *spatial*. A temporal prediction predicts a subset of the data based on other subsets of the data with a different timestamp. One example is that one frame in a video can be predicted from the previous frames. A spatial prediction predicts a sub-region in an image based other sub-regions in the same image.

Due to the complexity of the actual input data, a predicted result usually is not identical to the actual data. The difference between the predicted result and the actual data is called *residual*. In the framework, the residual is extracted, compressed, and encoded as part of the compressed data.

As a summary, the compressed data from a prediction/residual framework is the set of prediction parameters and the compressed residual. The data is sent to an entropy encoder to achieve a shorter bit length.

The contribution of my research work described in this dissertation includes tweaking the components of this framework to fulfill specific application needs. Therefore, it is useful to understand the general workflow of this framework.

As an example of the Prediction/Residual Framework, Figure 2.1 illustrates the workflow of an H.264 encoder. Every frame of the input video is passed into the motion estimation module, the motion compensation module and the intra (spatial) prediction module. These three modules work as the prediction stage. The prediction result is compared to the original video to generate

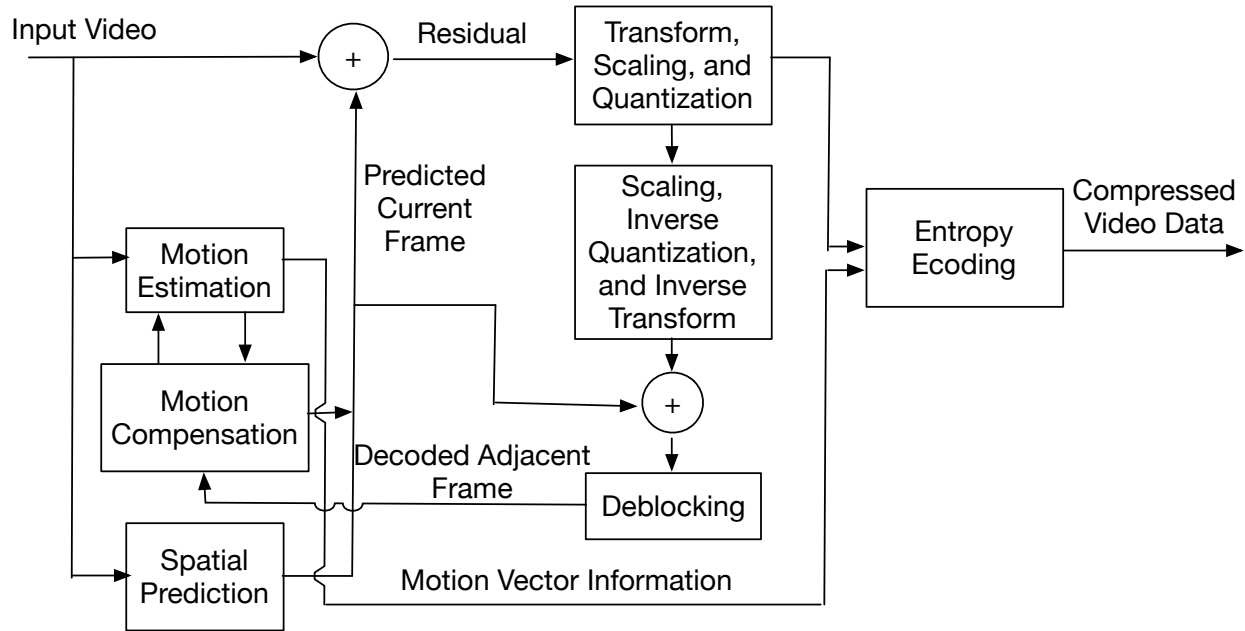


Figure 2.1: H.264 encoder and decoder as an example of prediction/residual framework.

the residual part of the compressed data. The residual data is further sent to the transform/scaling quantization module for compression. In the last step, the entropy coder encodes the data to produce compressed video bits. In H.264, an encoded frame is sent back to the previous motion estimation and motion compensation modules to encode other frames.

2.4 Multimedia Compression

I revisited some common image and video compression standards. The comparison between these standards is formulated in Table 2.1. The table contains each compression standard's year of release, integrated transform algorithms, and the entropy coding algorithms.

	JPEG	JPEG2K	PNG	WebP	H.264	H.265	VP9
Released Year	1992	2000	1996	2010	2003	2013	2012
Support Video?	No	No	No	No	Yes	Yes	Yes
Transform	DCT	DWT	Filtering	Block Prediction + DCT	DCT	DCT/DST	DCT/DST
Entropy Coding	Huffman coding	LZ77	LZ77	Arithmetic Coding	CABAC	CABAC	Arithmetic coding

Table 2.1: Comparison between common compression standards.

In the rest of this section, I briefly explain the two most commonly used data transform algorithms. They are discrete cosine transform (DCT) and discrete wavelet transform (DWT).

2.4.1 Discrete Cosine Transform

Discrete Cosine Transform is widely used in image and video compression including JPEG (Wallace, 1992) and H.264 (Wiegand, 2003). 2D DCT transforms input 2D information into the frequency domain, where the low-frequency component coefficients and high-frequency component coefficients are separated. In a lossy compression process, the high-frequency information is usually reduced by quantization. Eq. 2.2 gives the DCT formula, where X_k is the k th frequency coefficient. The sampled input size in spatial domain is denoted by N . The n th input in spatial domain is denoted by x_n and k ranges from 0 to $N - 1$.

Figure 2.2 gives an example of a test binary image containing hand-written digit ‘2’ and the images compressed by DCT transform, reducing high-frequency component coefficients, and DCT inverse transform. The number on top of each image represents the image data size in bytes.

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-i2\pi kn}{N}} \quad (2.2)$$

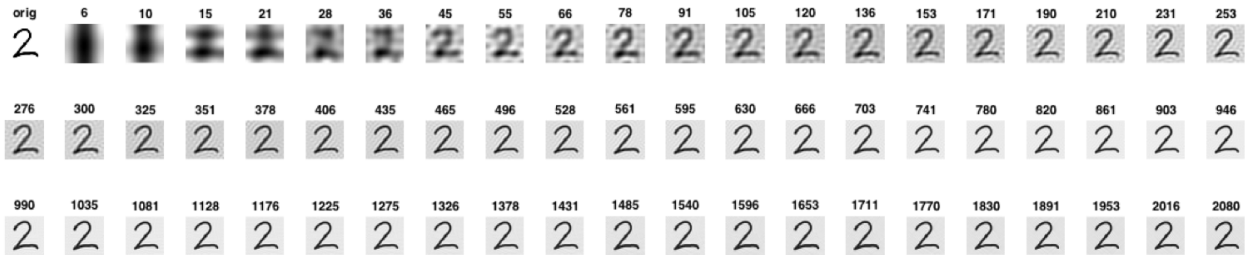


Figure 2.2: Sample hand-written digit image (4096 bytes) and the DCT-compressed version of the image with the compressed size in bytes.

2.4.2 Discrete Wavelet Transform

Discrete Wavelet Transform is a transform that recursively applies a filter to the input data at different scales. Usually, as a result of the process at each stage, the input data is separated into a low-frequency component and a high-frequency component. The low-frequency component is further passed into the next stage of filtering. Data reduction is achieved by lossy compression on the high-frequency components.

The most basic Discrete Wavelet Transform is the Haar transform, which was invented by Alfréd Haar in 1910. The Haar transform at each stage outputs the average and the difference between two adjacent values of the input. This is formulated as:

$$c(n) = 0.5 \times (2n) + 0.5 \times (2n + 1) \quad (2.3)$$

$$d(n) = 0.5 \times (2n) - 0.5 \times (2n + 1) \quad (2.4)$$

In the 2D Haar wavelet transform, at each stage, instead of a pair of filtered outputs, a set of four outputs are produced: three high-pass filtered outputs in with different filter directions and one low-pass filtered output.

The major difference between DWT and DCT is that the high-frequency components of DCT gives a higher frequency resolution, but lower spatial resolution comparing to DWT. As a result, it is more difficult to recognize detailed spatial information in DCT-compressed images (Parmar and Scholar, 2014). Regarding DCT-based image compression standard (JPEG) and DWT-based image compression standard (JPEG2000), JPEG suffers from the blocking artifact due to the block DCT transform it uses.

2.5 Previous Work on Domain-Specific Video Multimedia Data Compression

This section reviews the existing domain-specific multimedia data compression methods that are related to my work. They are grouped into four types: region-prioritizing methods, methods that

optimized for high-resolution cameras, methods that are optimized for decoding time, and methods that are based on existing methods.

2.5.1 Region-Prioritizing Video Compression methods

My microscopy video compression approach is a region-prioritizing method. In this subsection, I present a survey on region-prioritizing video compression methods.

Domain-specific video compression techniques have been developed in many domains, such as video for plant phenotyping (Minervini and Tsafaris, 2013), surveillance (Babu and Makur, 2006) and multi-view video (Martinian et al., 2006). Many of the video analysis routines in these domains belong to computer vision techniques. In applying computer vision methods, visual features in the data are extracted and fed into vision algorithms. Because of this, a good domain-specific compression of this type should do its best job in preserving visual features. Jianshu Chao has made many contributions in developing video compression techniques that preserve visual features in the video data (Chao et al., 2015a; Chao and Steinbach, 2011, 2012; Chao et al., 2013b,a, 2015b). The visual features include frame-based features (SIFT, Speeded Up Robust Features (SURF), Binary Robust Independent Elementary Features (BRIEF) and Oriented FAST and Rotated BRIEF (ORB)) and spatial-temporal features (Harris, Hessian, and Histogram of Gradients (HoG)/Histogram of Optical Flow (HoF)).

Many medical image compression techniques are designed based specifically on the further analysis of the data, and they can be considered as domain-specific compression. Most of these methods used the idea of the region of interest (ROI) or volume of interest (VOI) to separate the analysis-critical part and other parts in the data (Ström and Cosman, 1997; Sanchez et al., 2010; Bai et al., 2004; Signoroni and Leonardi, 1998a,b; Ansari and Anand, 2009; Wakatani, 2002). For the quality measure of the compressed image/video, Simple global metrics such as Peak Signal to Noise Ratio (PSNR) (Sanchez et al., 2010; Ansari and Anand, 2009), Maximum absolute error in ROI (Ström and Cosman, 1997), Correlation coefficient and Mean Squared Error (Ansari and Anand, 2009) are used. Also due to the nature of the medical image, human expert evaluation is

applied in quality measure in (Signoroni and Leonardi, 1998a,b). Another example is a compression of a stack of CT scan medical images, the surrounding region in each image that is not part of the tissue will not be used in the later diagnosis and can be highly lossy compressed (Ström and Cosman, 1997; Sanchez et al., 2010; Bai et al., 2004; Signoroni and Leonardi, 1998a,b). In some situations, a lossy compression that removes part of the unnecessary information in the video is even preferred. One example of this is video artifact removal of microscopy video data described in (Yin and Kanade, 2011). A paper that summarizes the post-processing techniques for artifact removal is (Shen and Kuo, 1998). My methods build on these and add a method for determining the region of interest that is independent of the type of analysis to be performed on the images.

2.5.2 Video Compression Methods That are Built on Existing Standards

In this subsection, I list a set of multimedia compression techniques that are built on existing well-known standards (Chu et al., 1997; Lampert, 2006; Ganesan et al., 2007).

Lampert (Lampert, 2006) presents a set of new video compression techniques that are based on existing XviD MPEG-4 compression. The new techniques are constructed by replacing the macroblock decision module in the existing encoder with a machine learning decision maker. Linear classifier, decision tree, and neural network classifier are used. Chu et al. (Chu et al., 1997) propose a new video compression system with a hybrid frame encoding. It is based on H.263 video compression. The new system divides input video frames into three classes: I frame, P frame, and O frame. The I and P frames are encoded using existing H.263 compression. The O frames are encoded with a segment-based method, where stationary background region in the frames are identified. The background information is represented by the content of the adjacent P frames. Ganesan et al. (Ganesan et al., 2007) discuss a high-performance H.264-based video compression system with the target application of HD video decoding. It integrates H.264 compression with a parallel data processing architecture eXtreme Processing Platform (XPP).

A method that manipulates the existing codec syntax element has been developed in (Bergeron and Lamy-Bergor, 2005), but their application is for encryption instead of compression.

In the analysis-aware microscopy video compression method I invented, there are two variations. One of the two variations is based on existing block-based video transform.

2.6 Miscellaneous Topics in Modern Multimedia Compression

Some of the new compression techniques proposed in this dissertation rely on special properties of a multimedia system. This section describes the point spread function as the key factor in microscopy video compression and BLE broadcasting channel with the low data bandwidth as the major constraint in a BLE-based multimedia system.

My research projects also applied new methodologies in compressing data and evaluating compressed data quality. This section also briefly explains the related background on statistical tests.

2.6.1 Point Spread Function in Image and Video Acquisition Process

The PSF describes the response of an imaging system to a point source or point object. Another way to understand PSF is to consider it as the impulse response of a focused optical system. It can also be defined as the spatial domain version of the optical transfer function (OTF) of the imaging system. A sample PSF is shown in Figure 2.3

The PSF is a useful concept in optics, including Fourier optics, astronomical imaging, medical imaging, electron microscopy and other imaging techniques such as confocal laser scanning microscopy and fluorescence microscopy (psf, 2018).

The degree of spreading (blurring) of the point object is a measure of the quality of an imaging system. The image of a complex object can then be seen as a convolution of the true object and the PSF. (Quammen et al., 2008) et al. developed a visualization system that simulates the effect of PSF in fluorescence microscopy imaging.

Note that in a diffraction-limited optical system, even a “perfect” version of such system’s image formation process still involves PSF blurring. As a result, A “perfect” optical system generates a non-point image from a point source. Such non-point image is also called the “Airy disk”.

In the microscopy videos I use, the PSFs have size less than 2 pixels.

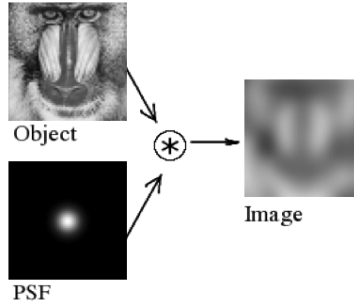


Figure 2.3: A 2D PSF convolved with a sample image, generates a blurred image.

2.6.2 Bandwidth Limited Transmission Channel: Bluetooth Low Energy Broadcasting Mode

Some of my research projects provide solutions to enabling internet-free multimedia data storage and broadcast using Bluetooth Low Energy broadcasting channel. This section gives an introduction to the BLE technology.

Bluetooth Low Energy is a wireless communication technology designed by the Bluetooth Special Interest Group. The aimed applications of BLE technologies fall in the healthcare, fitness, security, and home entertainment industries (ble, 2018). Differing from classic Bluetooth, BLE is designed to provide low power consumption and reduced cost while maintaining a similar communication range.

BLE operates in the 2.4 GHz Industrial Scientific Medical (ISM) band and defines 40 Radio Frequency channels with 2 MHz channel spacing. All physical channels use a Gaussian Frequency Shift Keying (GFSK) modulation. The BLE protocol stack consists of five layers. For the three higher-level layers, the purpose of logical link control and adaptation protocol (L2CAP) layer is to multiplex the data from higher layers protocol on top of Link Layer connections. Low energy attribute protocol (ATT) and generic attribute profile (GATT) are used to manage the roles of two communicating devices: server and client.

Besides connected communication, the BLE standard also defines an advertising mode, where the advertiser role and the scanner role are defined. The advertiser broadcasts 30-byte advertising packets through one of the three pre-defined advertising channels. The scanner detects the advertised packets via scanning. The advertising mode can be used for device discovery and communication before the connection. The advertising packet can also contain a resource locator such as an URL that directs the scanner to a resource on the Internet.

2.6.3 Multimedia Data Quality Evaluation Using Statistical Tests

In Chapter 3, I present an analysis-aware microscopy compression technique that evaluates the compressed video quality using statistical tests. The rationale behind this video quality evaluation is that analysis results on the original video are affected by noise captured as part of the original video. Therefore, they represent only one of a set of possible analysis results and re-taking new uncompressed video of the same specimen would produce slightly different results. Therefore, the compressed videos analysis *does not* have to exactly match that on that particular video, but it should be drawn from a distribution that matches those from multiple runs on the same specimen.

In this section, I list a set of common statistical tests that test the chance of two given sets of samples that following different probability distributions. I formulate a comparison between these standards in Table 2.2.

	Assumes Normal?	Approach	Considerations
KS test	No	Quantifies differences between the empirical distribution functions of two samples	Sensitive to differences in both location and shape of the empirical cumulative distribution function
Mean test	No	Quantifies differences between the means of two samples	Assumes the distributions on means of different samples are normally distributed
Variance F-test	Yes	Compute the ratio between two empirical variances	Only compares variances, does not make assumptions on the equal means
Van der Waerden test	No	Compute the normal scores of two probability distribution functions, make a decision based on the mean and variances of normal scores	Can take $k(k > 2)$ population distribution function at once.
TOST	No	Quantifies the difference between two equal means, test if it's in between the upper and lower equivalence bound	N/A
B-test	No	Test whether distributions P and Q are different on the basis of samples drawn from each of them, by finding a smooth function which is large on the points drawn from P, and small (as negative as possible) on the points from Q	N/A

Table 2.2: Comparison between common two-sample statistical tests.

CHAPTER 3

VIDEO COMPRESSION TO PRESERVE ANALYSIS-CRITICAL INFORMATION

In this chapter, I present two video compression methods that preserve analysis-critical information. This is the first part of my research work on multimedia compression methods that prioritizes content of interest. My target the application in this part is microscopy video compression.

High-speed, high-resolution and high-content microscopy systems are increasing the rate and amount of video data being acquired more rapidly than the rate of increase in affordable data storage (Wollman and Stuurman, 2007). This forces the bench scientist either to be very selective in which data sets they store or to greatly compress their data (Oh and Besar, 2003). At the same time, funding agencies and journals are increasingly requiring all data from published experiments be retained to enable re-analysis by others. One example of high-resolution microscopy system is described in (Cribb et al., 2015).

The goal of my microscopy video compression research is to develop a method that obtains high compression while preserving the information needed to perfectly reproduce analysis results. I propose two novel compression methods for video microscopy data. The methods are based on Pearson's correlation and mathematical morphology. The two methods make use of the PSF (described in Section 2.6.1) in the microscopy video acquisition phase. I designed experiments to compare the proposed methods to other lossless compression methods and to lossy JPEG, JPEG2000 and H.264 compression for various kinds of video microscopy data including fluorescence video and brightfield video. The result shows that for certain data sets, the new methods compress much better than lossless compression with no impact on analysis results (analysis-preserving) or the impact on analysis result is within the range of error introduced by noise (analysis-aware). The first method of the two, analysis-preserving compression achieved a best compressed size of 0.77%

of the original size, $25\times$ smaller than the best lossless technique (which yields 20% for the same video). The second method, analysis-aware compression can achieve 1000x compression on certain test microscopy videos for the same error level in the analysis result. The compressed size scales with the video's scientific data content. Further testing showed that existing lossy algorithms greatly impacted data analysis at similar compression sizes. For the analysis-aware compression method, I propose a new measurement of quality of a microscopy video based on the level of preservation of analysis results. I evaluated the method with a bead tracking analysis program.

3.1 Related Work

There are a number of lossless compression techniques available that reduce the size of a data set while enabling exact reconstruction of the original file (Christopoulos et al., 2000; Wiegand, 2003; Vatolin D, 2007; Burrows and Wheeler, 1994). Some have been developed specifically for use on images (Christopoulos et al., 2000) and video data (Wiegand, 2003; Vatolin D, 2007). Noise in the video images combines with the requirement that every pixel be exactly reproduced in every frame to limit compression rates for these techniques.

Several high-quality image compression techniques are tuned specifically for the human visual system to produce image artifacts that are not easily seen (they are “perceptually lossless”). They achieve far greater compression rates without visible quality loss (Christopoulos et al., 2000). Similarly, various video compression techniques have been invented and standardized in the past decades. Currently, several of the most widely used techniques are H.264, H265 and VP9 (Wiegand, 2003; Sullivan et al., 2012; Mukherjee et al., 2013). Each of these video compression techniques is designed to achieve acceptable compression performance on a wide range of videos while maintaining good visual quality for human observers, sometimes based on popular video quality metrics designed to measure this such as Peak Signal to Noise Ratio (PSNR).

The use of three new compression techniques for single confocal fluorescence microscopy images of cells was explored by Bernas et al. (Bernas et al., 2006) to determine how much compression could be achieved based on the signal to noise ratio (SNR) of the images. They used two

techniques to estimate SNR for the images (Amer and Dubois, 2005; Nowak and Baraniuk, 1999). Their spatial downsampling approach reduced image resolution to match the frequency at which the spatial intensity contrast passed below the estimated noise floor in the images. Their intensity downsampling approach reduced the number of intensities per pixel to the number of distinguishable levels based on the noise floor. Their wavelet compression approach removed wavelet coefficients that were expected to represent only noise. They achieved compression ratios of between 3 and 9 without significant reduction in three quality tests. I seek compression ratios of up to 100 for 96-well experiments.

With the goal of decreasing the transmission bandwidth for time-series of confocal optical microscopy image transmission, Avinash looked at the impact of different quality levels of JPEG compression (compared to lossless compression) on the image intensity variance in single 2D images (Avinash, 1995). He compared this to estimates of image noise based on the variance in visually uniform background regions. He hypothesizes that adding only slight compression noise compared to the already-present background noise may not impede quantitative analysis (20% increase in noise). He compared time-averaged versions of the same region to simulate images with different noise levels. He found that at a JPEG quality setting of 75/100, the noise variance was much higher than the difference variance ($22\text{--}32\times$ greater); at this value, the compression ratio varied between around 3 (noisy image) and 5 (less-noisy image). The compression ratio was never more than 11, even for images with significant degradation.

The idea of evaluating the quality of a video based on analysis algorithms can be found in (Korshunov and Ooi, 2011). In their paper, the video analysis routine is a set of computer vision algorithms: face recognition, face detection, and face tracking. They used H.264 to compress video multiple times with various quality settings to generate a set of the compressed video. They discovered that face recognition and face detection results are not sensitive to compression until they reach a particularly low quality setting. Above that, compression maintains similar face recognition and detection results as the original video. Experiments have also been performed on tracking faces in a set of compressed videos with a certain portion of frames dropped. They proposed that mutual

information and blackness be two metrics as they better correlate with the qualities of these analysis results. These metrics differ from those required by scientific analysis algorithms (Korshunov and Ooi, 2011).

The video compression method described in this chapter can be characterized as Region-of-Interest (ROI) based methods. Previous ROI video encoding methods have been explored in (Van Leuven et al., 2008; Liu et al., 2008; Grois and Hadar, 2012). One application of ROI video coding to face detection and tracking is discussed in (Menser and Brunig, 2000). Application to aerial videos is introduced in (Meuel et al., 2011). Chao et al. discussed the ROI video coding for preserving computer vision visual features in (Chao and Steinbach, 2011, 2012; Chao et al., 2013b). To my knowledge, there is no work done in exploring the use of ROI video coding for microscopy video analysis.

3.2 Overview

3.2.1 Analysis-Preserving Compression

I describe a new method for microscopy video compression that achieves up to $100\times$ compression, enabling high-throughput video-acquisition experiments to be stored in the same space as conventional experiments. The compression has no impact on analysis results. It achieves this by storing only the information in a video that analysis can use and averaging out noise in the background. It first separates every frame into foreground (pixels that carry information) and background (pixels that do not change throughout the video) and then losslessly compresses the foreground regions. This successfully keeps all relevant data while achieving a better compression ratio.

The key problem is to decide whether each pixel in each video frame is foreground or background. For microscopy video, the noise behavior is well understood to be independent between neighboring pixels, whereas blurring—convolution with the point-spread function (PSF) will spread image brightness changes in one pixel to its neighbors (Sheppard et al., 2006). The property of PSF

is detailed in Section 2.6.1 in the Background chapter. Making use of this property, I designed a correlation-based method that separates foreground from background. Figure 3.1 summarizes the steps, which are further detailed below and in the Methods section.

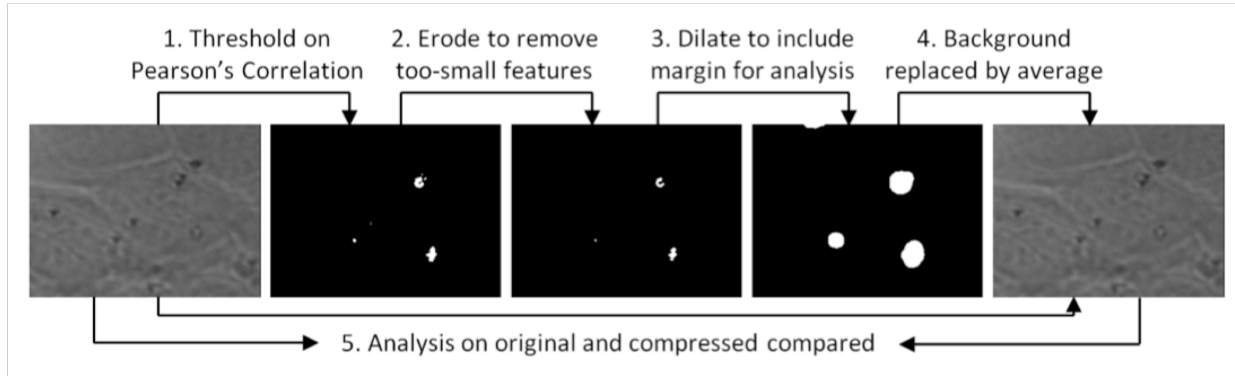


Figure 3.1: Analysis-preserving video compression process.

The method first generates a binary segmentation of each frame into foreground and background pixels by thresholding on the maximum magnitude of the Pearson's correlation coefficient (Stigler, 1989) between each pixel and its eight neighbors. This coefficient is computed over all frames of the video, selecting pixels whose brightness changes are correlated with those of their neighbors.

Because even independent random variables have nonzero correlations with some probability, a number of pixels are falsely labeled as foreground. These pixels are likely to be spread evenly across the image, whereas true foreground pixels will be grouped into clusters that are at least as large as the main lobe of the PSF. To remove these false positives, the binary segmentation is refined by the mathematical morphology erosion operation.

Because analysis methods make use of pixels near the foreground pixels, the resulting set of foreground pixels is dilated to include pixels that are close enough to affect analysis (this radius depends on the parameters of the analysis algorithm, but not on the specific algorithm being used).

Using the refined binary segmentation, the original video has each pixel in its background regions replaced by that pixel's time-averaged value. This removes noise, which makes the video compress more using lossless H.264 compression.

To verify that the compression had no impact on analysis, the compressed video is processed by the same analysis pipeline to make sure the results exactly match those of the original video.

I evaluated the method using Video Spot Tracker (CISMM, 2017) to track moving beads, which showed that my method can get at most 100x compression without any change to analysis results. In comparison, H.264 compression either yielded much smaller compression ratio (lossless) or changed the analysis results (lossy).

3.2.2 Analysis-Aware Compression

The last subsection describes the proposed analysis-preserving compression method that retained identical analysis results after compression. I extend the analysis-preserving method to enable even higher compression while still maintaining results that are statistically indistinguishable from samples of the original video. I observe that microscopy video analysis results are already altered by noise introduced in all stages of the microscopy video acquisition pipeline. The new method does not force the compressed video to have identical analysis result as the original video. Instead, it maintains the original information and replicates noise such that the error introduced by compression is statistically indistinguishable from that introduced by existing noise. This is verified by running multiple different statistical analyses on the original and compressed videos. The statistical analyses are explained in 3.2.3. For the case of analysis of bead-tracking results, this enables a reduction in the number of foreground pixels compared to the prior method, which enables even larger compression ratios without detectable changes in analysis.

The new method is named as analysis-aware compression. Another difference between this method and the analysis-preserving method is that analysis-aware compression includes a post-processing stage that add synthesized noise into the compressed video data to improve the quality of the video. The overall process of the method is shown in Figure 3.2.

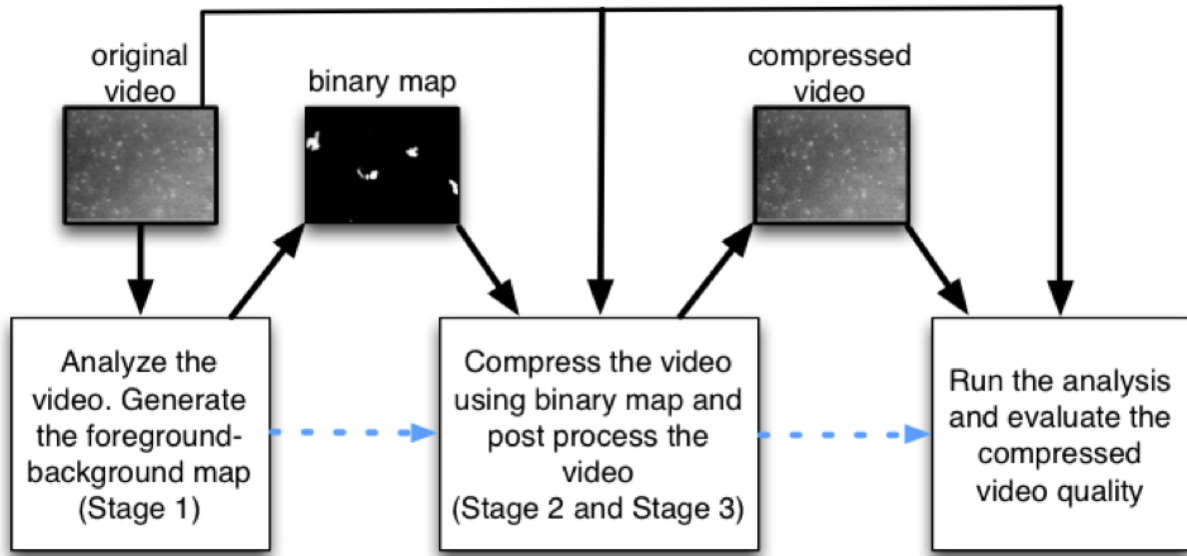


Figure 3.2: Analysis-aware video compression process.

3.2.3 Statistical Tests

Because I judge the quality of a compressed video by comparing its error with that introduced by noise in the original video, the relative distributions should be considered. To provide confidence values, a quantitative approach is preferred. In testing the performance of my methods, I applied two experiments: the two-sample Kolmogorov-Smirnov (KS) test and Kullback-Leibler (K-L) divergence computation.

KS test

In my experiment, the goal is to show that the population of MSD samples from the compressed video group is not different from the population of the samples from the uncompressed video group. This can be verified using the KS test, which is a well-known technique for testing and giving the confidence level that two groups of values drawn from two continuous random distributions are actually drawn from the same distribution. Unlike the t-test, which mainly tests the difference between two populations' means, the KS test takes the shape of the distribution into account and finds the largest vertical distance between two kernel density plots. For this test, the scientist must select a

confidence threshold in order to make the decision on whether the two MSD samples are from the same distribution.

K-L divergence

Computing a K-L divergence can also compare two samples of MSD values from two unknown distributions. K-L divergence is a concept in information theory that measures the difference between two probability distributions. It can be understood as the information lost when probability distributions Q is used to approximated probability distribution P .

In my experiment, P is the sampled population of the MSD values for the original video and Q is the sampled population of the MSD values for a compressed video. The measurement is non-symmetric: $KL-div(P, Q)$ is generally different from $KL-div(Q, P)$. Similar to the KS test, for this test, the scientist also must select a confidence threshold in order to make the decision on whether the two MSD samples are from the same distribution.

3.3 Methods

The basic form of my analysis-preserving method and analysis-aware method both apply a two-step approach.

In the first (segmentation) stage, the analysis-critical regions in every frame in the video are detected. The methods use an approach based on correlation and mathematical morphology to determine the important part of the video in a domain- and analysis-independent manner. Every pixel in every frame is labeled as either foreground or background. This result is stored in a binary map.

After the segmentation stage, the binary map is sent to a compression routine. The compression integrates the segmentation result in its encoding process so that for encoding setting the given fixed resource is allocated in a way to ensure that information in the analysis-critical region is well preserved.

For this stage of the analysis-aware algorithm, I designed and evaluated two different variations. They are detailed in 3.3.2. After the compression is completed, the resulting compressed video has a much smaller size, and it is still useful for analysis.

The extended form of analysis-aware compression includes a third stage: The compression may still introduce changes into the analysis result. To address this problem, I designed a post-processing stage to refine the compressed video. The post-processing stage makes use of the noise statistics in the video and refines the video by reproducing the noise that matches the video system characteristics as explained in Subsection 3.3.3.

The central problem in the proposed methods is to find the separation between foreground pixels (which may carry information) and background pixels (which contain only noise) in microscopy video.

The background noise in the video can be well modeled by a Gaussian distribution plus a Poisson distribution:

$$N(0, \sigma) + P(\lambda) \tag{3.1}$$

Importantly, neither of these terms depends on the values of neighboring pixels: they are identically randomly distributed among the pixels in the image (Sheppard et al., 2006).

Another important property of microscopy video is the mixture of values from neighboring pixels caused by the point-spread function. This causes changes in each pixel's intensity over time to be correlated with those of neighboring pixels. As a result, in a microscopy video, time correlation of the intensity of a pixel and its neighbors tends to become nonzero whenever it is caused by changes in intensity due to specimen motion.

3.3.1 Segmentation Stage

The goal of segmentation is to accurately detect the regions of pixels in a microscopy video frame that might affect analysis. The analysis-independent method for this task made use of the PSF to remove regions containing only noise as detailed in Subsection 2.6.1.

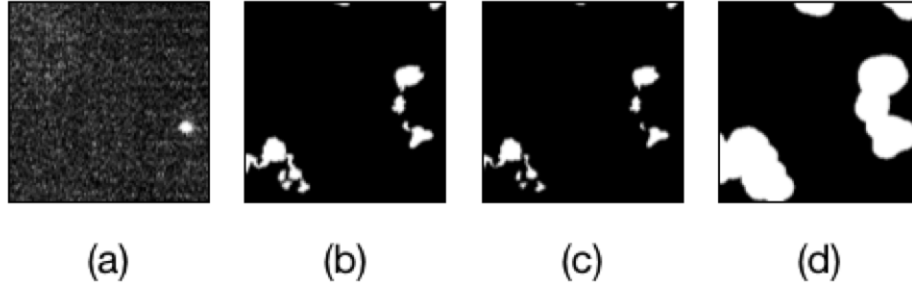


Figure 3.3: Mathematical morphology portions of the original algorithm: (a) a sub region of the first frame in the original video; (b) the correlation-based segmentation result without refinement; (c) the segmentation result after erosion refinement; (d) the segmentation result after erosion and dilation refinement. Some foreground regions in (d) are due to other beads that move into this region in later frames.

The correlation-based segmentation for detecting moving objects in microscopy video is the same in both analysis-preserving and analysis-aware method. Because of the PSF, every pixel is blended slightly with its neighboring pixels. This means that any moving image feature will have a correlated impact on a region of pixels rather than only a single pixel. This does not hold for shot noise and electronic noise, which scale with image brightness but are uncorrelated between pixels. To get a foreground score for every pixel, the method computes the Pearson’s correlation score between it and its neighbors:

$$R_i = \left| \frac{\sum_{j=1}^k (x_j - \bar{x})(y_{ij} - \bar{y}_i)}{k\sigma_x\sigma_{y_i}} \right| \quad (3.2)$$

In the formula, x_j is the pixel intensity value for the center pixel at the j th frame, \bar{x} is the mean pixel intensity value of the center pixel over a time interval, y_{ij} is the pixel intensity value for the neighbor pixel at j th frame, and \bar{y}_i is the mean for the neighbor.

The method computes this value for all eight neighboring pixels for each pixel. It then computes the maximum of all neighbor scores and use a threshold on this to determine which pixels are in the foreground. The threshold was determined by running multiple passes of bead tracking on the compressed video having the same tracking result as the uncompressed video but it can also be determined for a system with known sensor characteristics based on a likelihood threshold based on the system's noise characteristics. Once determined, this threshold can be transferred to videos taken with similar experiment setups. After every pixel has a score assigned to it, all pixels whose score are above the threshold and are marked as potential foreground pixels in a binary map.

Figure 3.4 shows the impact on compression size as this threshold is increased; fewer pixels are selected as foreground and the compression ratio improves. However, at some point this causes foreground pixels in the image to be missed, which impacts the data analysis and the compression begins to change the results of analysis. Because there is no general solution to the question “how much change to analysis values is too much”, I stop at this level and I select the threshold that has the best compression without impacting analysis.

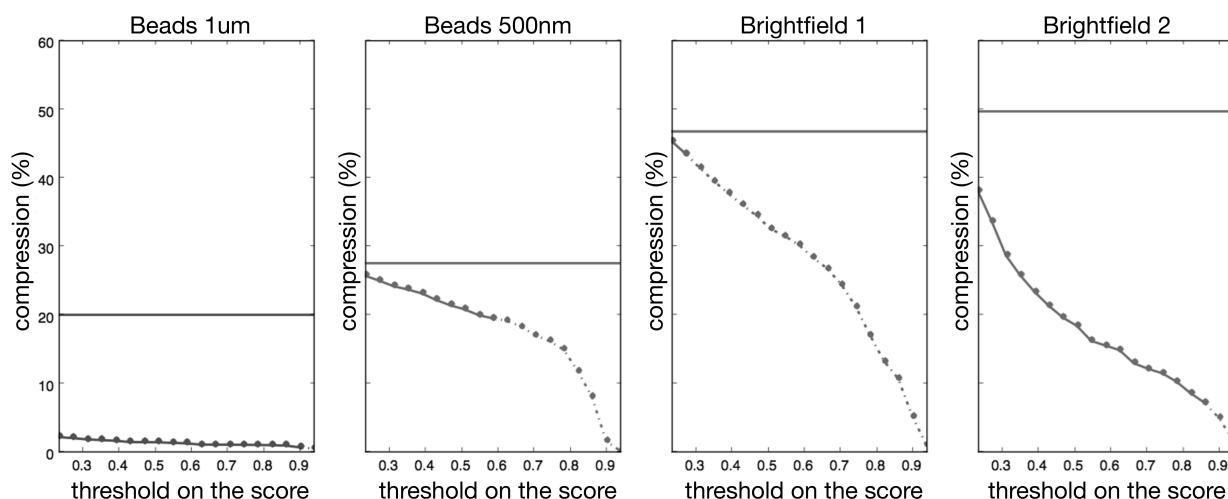


Figure 3.4: Plots of compressed file size comparing against original file size (in percentage) vs. correlation magnitude threshold on scores for four videos. Four subplots shows the result for four test videos respectively. Four plots share the same vertical axis. The horizontal flat line in every subplot indicates the compression result on that video with H.264 lossless technique. The curves become dashed when the analysis results on compressed videos differs from the analysis result on the original, indicating the limit compression without impacting analysis results.

This threshold is set to a liberal value to avoid losing actual features, with the result that the map contains many small false-positive pixel groups whose size is smaller than the PSF for a given microscope. The PSF would spread actual features over larger areas, so the method removes these false positives using the mathematical morphology erosion operation (Serra, 1982). Figure 3.3 shows one example of the test video frame image and the result binary map cleaned up by erosion. The resulting cleaned binary map guides compression.

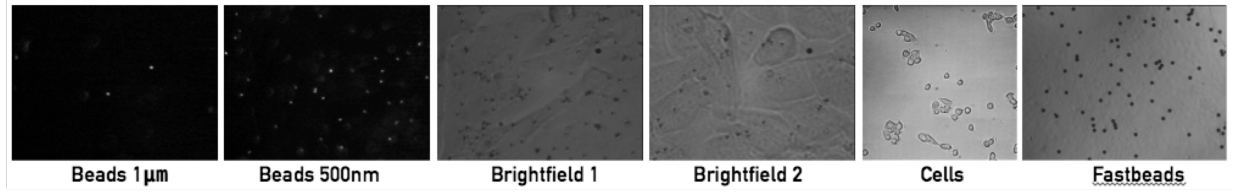


Figure 3.5: Example frame from each of the videos tested, each named as in the description and tables.

In the analysis-preserving method, I perform the dilation size that is equivalent to the erosion size plus additional dilation to expand foreground region in the correlation-based segmentation result. This increased dilation (shown in Figure 3.3) provided a conservative estimation of foreground regions to provide an (analysis-method-dependent) region increase to ensure identical results. In the analysis-aware method, the expanded region is not required, so the additional dilation size is not applied - resulting in a much smaller foreground region and greater compression.

My collaborators evaluate their data using the CISMM Video Spot Tracker (CISMM, 2017). An example of a binary map before refinement and after refinement is given in Figure 3.1.

3.3.2 Compression Stage

For the compression stage, the goal is to make use of the segmentation result to encode the video data so that information in the analysis-critical regions is preserved in a manner that does not affect analysis. There are many options for applying existing well-developed video codecs and integrating the analysis-critical map signals to compress the video data. In developing the system, I explored two paths.

The first approach processes the video frames by averaging background pixel values over time and then losslessly compressing the processed video frames using a standard algorithm. The pre-processed video has many pixel locations with constant value over time, which can be efficiently encoded to provide high compression. Tests were done to compare four standard compression techniques and software: bzip2, jpeg2000, H.264 and H.265. The result showed that the three modern compression routines all give a similar good compression with the processed video frames. From these four methods, H.265 and H.264 achieve the smallest two compressed video file size based on my data set. H.265 is 4% smaller compressed video size than H.264 but the encoding speed of H.265 was much slower than other three techniques. Therefore I choose to use H.264 in my algorithm implementations and experiments. The results should apply to any lossless video compressor.

In an extension of the analysis-aware method, I also evaluated replacing the background averaging with an approach that uses a combination of lossless and lossy compression. This approach works for block-based prediction-residual compression approaches. The implementation used H.264. In H.264, the motion estimation unit is based on 16x16 pixel patch macroblock. The pixel data for each macroblock is transformed into the frequency domain. Data reduction is achieved by reducing the information in the high-frequency components in every macroblock. Specifically, information reduction is done by quantization that collapses a range of close values into one. Quantization level is mostly based on the given bandwidth in compression and it is generally a global property across blocks. But in my compression method, I don't need high quality for blocks that represent background pixels. Therefore my approach assigns different quantization levels to each block based on the segmentation result. I denote qp as the controlled value in quantization. A higher qp results in a wider range of values to be suppressed into one value, which results in shorter encoded bit length and lower video quality. As shown in Figure 3.6, if in one block there are one or more pixels that are classified as foreground in the binary map, I use a better setting ($qp=0$); otherwise, I assign a worse setting ($qp=51$) to the block. This removes the need to calculate running averages across frames at the expense of variable-quantization encoding.

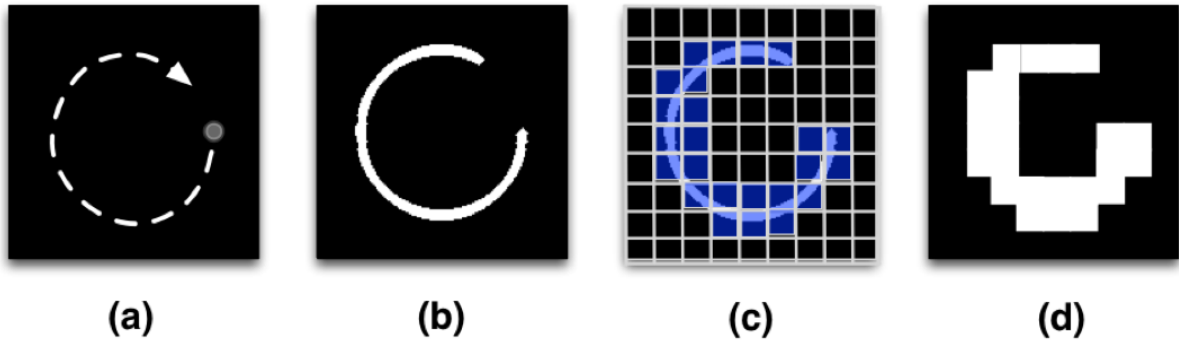


Figure 3.6: Left to right: a) the starting position of a bead in a video and its moving trajectory; b) the resulting binary foreground/background map; c) illustration of the macroblocks that covers the frame; d) the resulting binary macroblock foreground/background labeling map.

I also studied combining the two approaches: averaging the backgrounds and using customized qp assignment in compression. However, the combination yielded larger compressed video sizes than either technique applied by itself. This may be because the artificial edges introduced by the first stage are not usually well aligned with the macroblock boundaries, or it is not well aligned with the prediction model inside H.264.

3.3.3 Post-Processing Stage

By averaging the background pixels over time, the compression is filtering out noise in the original video signal, producing an output video that has less noise than the input video. This modifies the results of analysis routines whose kernels reach beyond the foreground pixels, such as the symmetry-based tracking kernels uses in my analysis.

This can produce more accurate tracking on the compressed video than the original. While more accurate tracking could be considered better, it is also statistically different from the results of tracking in uncompressed video. For cases where different regions of the video have different background fractions this can also produce track-to-track variations in the results. Especially for analysis that looks at random motion distributions (like the mean-squared displacement calculations performed by my collaborators), this means that analysis on compressed video is different from

analysis on uncompressed video. In these cases, the loss of noise in the reconstructed video is a problem.

There are two ways to address this problem. The analysis-preserving method expanding the foreground regions based on knowledge about the spatial extent of the analysis kernels. The analysis-aware method estimates the distribution of background noise in the original video and adds synthesized noise into the compressed video during decompression/analysis. This has the benefit of being independent of the radius of the kernels for analysis performed on the video. This process is the post-processing stage of the analysis-aware method. During analysis, noise is generated and added back into the video in an on-line fashion. To avoid a per-pixel storage cost, the known characteristics of noise in optical microscopy systems can be used to model the entire image with only two parameters.

In estimating the noise parameters, I model noise value probability distribution as a Poisson + Gaussian distribution described in Eq. 3.1. By assuming a large sample size one can further simplify the distribution (speeding reconstruction calculations) into a single Gaussian distribution with non-zero mean. The only parameters are the mean and variance of the distribution. To obtain the parameters for the two distributions (signal and noise), I used k-means clustering method. By finding the two clusters of the pixel intensity over time points in the mean and variances space, I use the cluster with lower mean and variance and use its center as the mean-variance of the noise distribution. One sample plot of the pixel intensity over time's mean vs. variance plot is shown in 3.7 where point A is the chosen cluster center.

3.3.4 Analysis-Aware Video Quality Measurement

Because standard video quality metrics such as PSNR and SSIM do not correlate well with analysis such as object tracking (Korshunov and Ooi, 2011). I seek a better metric for evaluation. In analysis-preserving compression, the quality of the video was determined by running the same tracking analysis on the video, and only the video with output exactly matching the original video's analysis result passed the validation. For analysis-aware compression, I consider the fact

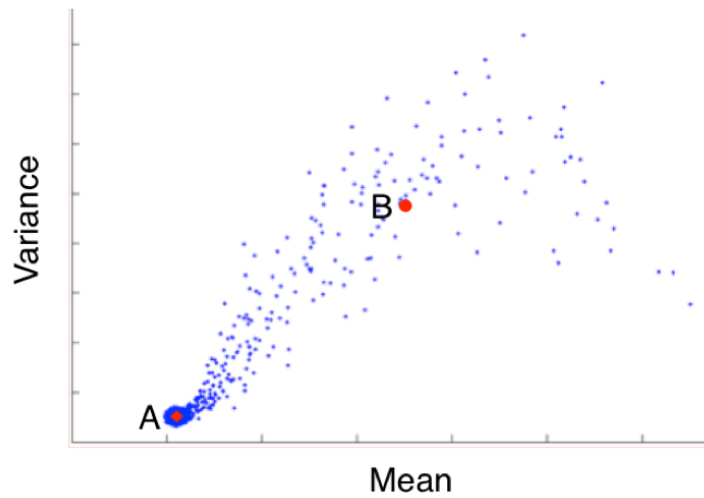


Figure 3.7: Mean vs. variation intensity plot with centers of the two-means cluster.

that analysis results on the original video are affected by noise captured as part of the original video. Therefore they represent only one of a set of possible analysis results, and re-taking new uncompressed video of the same specimen would produce slightly different results. Therefore, the compressed video's analysis *does not* have to exactly match that on that particular video, but *it should be drawn from a distribution that matches those from multiple runs on the same specimen*. I propose robust statistically-based video quality measurements based on the values derived from sets of analysis results.

This statistical approach can be used with any analysis. I demonstrate it using mean square displacement (MSD) curves that are derived from bead tracking results. An MSD value is calculated by averaging the squared displacement over movement measured using a fixed time window (τ). A sequence of MSD values with increasing time windows contains information about the type of cell motion. By characterizing at the shape of the MSD vs. curve, characteristics of the specimen (diffusion coefficient, membrane stiffness) can be classified (Monnier et al., 2012).

In my experiment, I converted the tracking trajectory result into a sequence of MSD values with different time windows. The quality of the video is determined based on the result of the quantitative tests (detailed in 3.2.3) on these MSD values compared to the MSD values from tracking in the original uncompressed video. I would like to verify that the error introduced by the compression

matches that introduced by existing noise such that the two set of measurements are statistically indistinguishable.

3.4 Results

3.4.1 Analysis-Preserving Compression Results

Analysis-preserving compression method achieved much better compression than existing lossless techniques in all cases tested and with identical analysis results in more than half of the cases tested. The data used for testing consists of six cases (Figure 3.5). The first four (two fluorescence and two bright-field imaging) have 1000 frames of moving beads attached to cell membranes. The beads in the fluorescence videos have diameters of 1 μm and 500 nm. (The video with 500 nm beads has many more beads.) The fifth video shows cells moving in bright-field imaging. The sixth video consists of beads that stay mostly still for the first half of the video and then move rapidly in one direction.

H.264 lossless	Beads 1μm	Beads 500nm	Brightfield 1	Brightfield 2	Cells
Without my method	20%	28%	47%	50%	33%
Including my method	0.77%	20%	44%	7.3%	4.8%

Table 3.1: Compression ratio for five real-world videos. The analysis-preserving method outperformed standard H.264 compression by large factors in three out of five test cases. In both cases, the resulting compressed video is in H.264 format so can be easily fed into analysis pipelines.

Table 3.1 compares the performance of my method against lossless H.264 compression. The usage of a more recent video compression standard H.265 was also explored and compared with H.264. In the experiment, the version of the implementation of the standard libx265 is 1.3+861-86ca1de606e3. The H.264 implementation libx264 with version 0.142.50 was used.

Firstly, I used the two compression techniques to compress the data listed in the second column in Table 3.1, namely the fluorescence video with 1 μm -diameter beads. In this test, H.264 outperforms H.265 by having a 0.1MB-smaller compressed video size. The same experiment was done on the cells video (Figure 3.5, fifth from left). This time H.265 has a 0.1MB-smaller compressed data

comparing against H.264. I am interested in the ratio between the size of compressed files with and without my method. I noticed that this ratio stays almost the same for both H.265 and H.264. More precisely, the difference stays within 1% of the range of the data size.

Because H.265 does not always yield a better compression, I did most experiments using the more rapid H.264 implementation. I also compared against lossless JPEG2000 and lossless JPEG, which did not perform as well. For JPEG2000, the libopenjpeg version 1.5.2 was used.

The size achievable before analysis is impacted scales with the information content of the video: videos with information in every pixel see no improvement. For the 500 nm video (which has more foreground) I achieve a compressed size of 19.6%, only slightly better than lossless H.264 alone. For the video with little foreground, the compressed size reaches 0.77%, 25 times better than lossless H.264 alone and an overall reduction factor of 100.

Figure 3.4 plots the compressed size vs. threshold on the scores and the maximum lossless correlation threshold in all five cases. The compression ratios using standard techniques are also displayed as horizontal lines. Table 3.1 compares my best compression ratios against those achieved by standard compression techniques. Table 3.2 shows the compression ratios for different methods for all cases.

Video	Techniques applied with my approach			Techniques applied without my approach		
	H.264 lossless	Bzip2	JPEG2000 lossless	JPEG2000 lossless	Bzip2	H.264 lossless
Bead 1 μ m	0.77%	0.70%	1.3%	24%	19%	20%
Bead 500nm	20%	21%	23%	31%	29%	28%
Brightfield 1	44%	49%	46%	47%	51%	47%
Brightfield 2	7.3%	26%	29%	50%	54%	50%

Table 3.2: Comparison of multiple compression methods on four of the videos shown in the main text. The left and right numbers match those in TABLE 1. Bzip2 and lossless JPEG2000 compression on the original file sequence were usually worse and never much better than lossless H.264. Lossless JPEG2000 and Bzip2 on the filtered image set were always worse than lossless H.264.

In all the experiments described above the erosion diameter was 3 pixels (this depends on the microscope point-spread size). The dilation diameter was set to 51 pixels, which is the size of the search radius for the tracking algorithm when it is testing for beads that disappear during tracking. For experiments without disappearing beads, the diameter can be set to be 24 pixels to achieve

higher compression. Dilation radius depends only on how many pixels the analysis algorithm looks at beyond the pixels that are part of the objects being analyzed.

I investigated the question of how much the quality factor of H.264 lossless compression can be reduced before I saw changes in the analysis. I intended to show the relative sizes of the compressed video at the smallest H.264 lossy-compressed size that did not induce changes in the tracking compared to the size of my analysis-preserving compression. However, in the four test cases (two fluorescence videos and two brightfield videos), the H.264 lossy compression changed the analysis results even at the setting that provided the least compression and gives the best video quality. This indicates that H.264 lossy compression at any level introduces changes in analysis.

I then investigated the question of how much impact the H.264 lossy compression has on tracking. This was done using a lossy compression level that matched the size of the compressed video produced by my analysis-preserving compression with no loss. The metric I use for comparison is specific to my particular analysis mode, and is reported in units of fractions of the noise floor in my instrument.

	Beads 1μm	Beads 500nm	Brightfield 1	Brightfield 2
Lossy H.264	4.6	∞	50.3	2.6
my method	0	0	0	0

Table 3.3: Maximum tracking error when using lossy compression techniques and using my method in four cases (in units of the experiment noise floor, which is $\frac{1}{10}$ th of a pixel). In one video, a bead track was lost, indicated here as infinite error. When they are forced to achieve the same compression ratios, perceptually-tuned compression techniques have a large maximum impact on analysis results.

	Beads 1μm	Beads 500nm	Brightfield 1	Brightfield 2
Lossy H.264	(0.1269, 0.0568)	(∞ , ∞)	(0.0468, 0.0502)	(0.1019, 0.0734)
My method	(0, 0)	(0, 0)	(0, 0)	(0, 0)

Table 3.4: Mean and standard deviations of tracking error when using lossy compression techniques and using my method in four cases (in units of experiment noise floor, which is 1/10th of a pixel).

Tables 3.3 and 3.4 compare the compressed videos from my method to videos compressed using the perceptually-tuned H.264 with its quality parameter set to make the file size match mine.

(JPEG2000 was also tested and had results similar to H.264). It is not possible to determine the impact of individual pixel-brightness values on an arbitrary analysis routine; doing this comparison requires running a particular analysis routine to see the impact. I used the video spot-tracking algorithm employed by my collaborators on both compressed and uncompressed videos and report the difference in centroid locations between the original and the compressed videos. In Table 3.3, the error metric is the squared maximum distance (in units of 0.1 pixels, which is the noise floor of the instrument) between points along bead traces, reported in units of the experiment noise floor. In Table 3.3, the error metric is the per-track mean error along with the standard deviation of the points along bead traces. My method (by design) achieves 0 error, the perceptually-tuned videos had errors ranging from 2-50 times the noise floor and sometimes lost beads entirely.

To test the generality of the method on non-bead-based specimens, I compressed a microscopy video of moving cells. Edge-length analysis was performed on this video as follows: Given a video, every frame in the video is filtered by a Gauss gradient filter with a Gaussian standard deviation 1 pixel to find the high response locations that suggests existence of an edge. After thresholding, connected components analysis is run on the binary map. To remove the noise, the connected components with fewer than 90 pixels were removed.

This process was performed on both original video and compressed video. I compare the edge detection result in all frames. With the exact same edge detection result, the analysis-preserving compression method achieved 3.5 better compression than lossless H.264 encoding alone.

Even non-high-throughput cell-motion videos often use low frame rates to reduce the amount of video storage required. This causes large motion in between frames, which is challenging for vision-based tracking algorithms to handle. An increased frame rate enables storing a much finer time resolution in the same file size, potentially improving the resulting analysis.

Fast-Moving Beads Analysis

In some microscopy videos, foreground objects move quickly over time and can cover most of the frame throughout the course of the entire video. In cases such as this, my basic method does not work well because most pixels are marked as foreground. I applied my method to one such case: a



Figure 3.8: Left: one frame of Fastbeads video before beads move; middle: foreground/background segmentation on whole video right: fore-ground/background segmentation within a 20-frame window.

microscopy video that contains 250 frames. Starting in the 100th frame, the beads in the video start to be move rapidly over the frame. One frame of the video is shown in Figure 3.8. The resulting map has most of the pixels as foreground (as expected).

To achieve a better compression ratio in cases like this, I added a sliding-window extension to the technique. I slide a fixed length (in time) window from the beginning of the video to the end and update the foreground and background pixels using only frames within the window.

For a video of length l and window size s , I get $(l - s)$ different binary maps. By segmenting out only pixels with beads moving over them in a short time period, the sliding-window version of the method marks fewer pixels as foreground in each frame.

Window size	Dilation Diameter 8 Pixels		Dilation Diameter 20 Pixels		Lossless H.264
	With sliding window	No sliding window	With sliding window	No sliding window	
100 frames	16%	35%	21%	39%	39%
50 frames	15%		21%		
20 frames	14%		19%		
10 frames	15%		20%		
5 frames	20%		24%		

Table 3.5: Compression ratio using the sliding window method and using lossless H.264 alone for the Fastbeads video. With longer windows, the beads moved across essentially every pixel in the image, so the method produced almost no improvement over lossless H.264 alone. With very short windows noise suppression was slightly reduced, causing an increase in file size. The optimal window size for this video was 20 frames, resulting in an approximately 2 improvement in file size. As expected, smaller dilation results in better compression (less fore-ground).

Table 3.5 shows the additional compression provided by the sliding-window technique for a video with fast-moving beads. With longer windows, the beads moved across essentially every pixel

in the image, so the method produced almost no improvement over lossless H.264 alone. With very short windows, noise suppression was slightly reduced. The optimal window size for this video was 20 frames, resulting in an approximately $2\times$ improvement in file size. A resulting map is shown in Figure 3.8.

Comparison with Other Techniques

Table 3.2 shows the results of doing compression tests with algorithms other than H.264 lossless. The first and sixth columns of this table match Table 3.1. Column five shows the result of using Bzip2 compression on the original files, which is usually slightly worse and in one case slightly better than lossless H.264. Column four shows the result of using lossless JPEG2000 on the original videos, which is never better than lossless H.264. Columns two and three show the results of using Bzip2 and lossless JPEG2000 on the images after my algorithm has been applied, which were always worse than using H.264 after my algorithm.

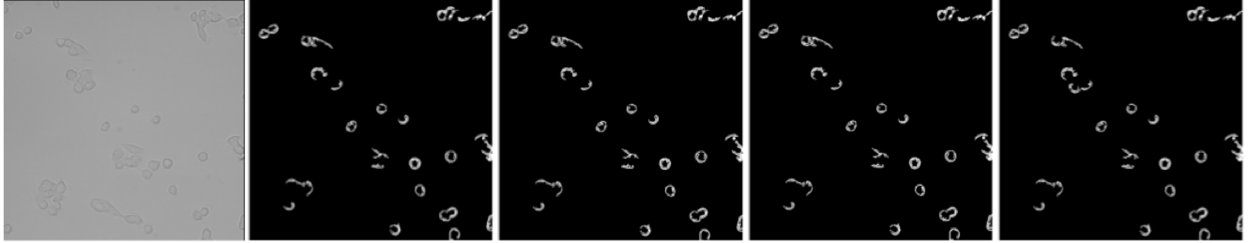


Figure 3.9: Left to right: first frame of the original video, edge detection results: original video, video with $3.5\times$ compression, video with $4.6\times$ compression, video with $6.8\times$ compression.

3.4.2 Analysis-Aware Compression Results

I performed two types of experiments to evaluate the analysis-aware compression methods. The goal is to compare the new statistically-indistinguishable analysis-aware video compression method against the standard video compression technique H.264. I included both variations of the analysis-aware video compression method (V1: per-pixel temporal averaging and V2: custom per-macroblock αp) in the comparisons, and performed the comparisons for each with and without

the noise-addition post-processing. Both synthetic microscopy video data and real microscopy video data are used in the experiments.

For each compression technique, I compare the performance of different compression methods under different bandwidth settings (compression ratios). I ran the tests with various configurations to generate different compressed video sizes. I then plotted the video quality evaluation results versus video data sizes. The experiments on synthetic data and real data are discussed separately.

Experiment on Synthetic Data

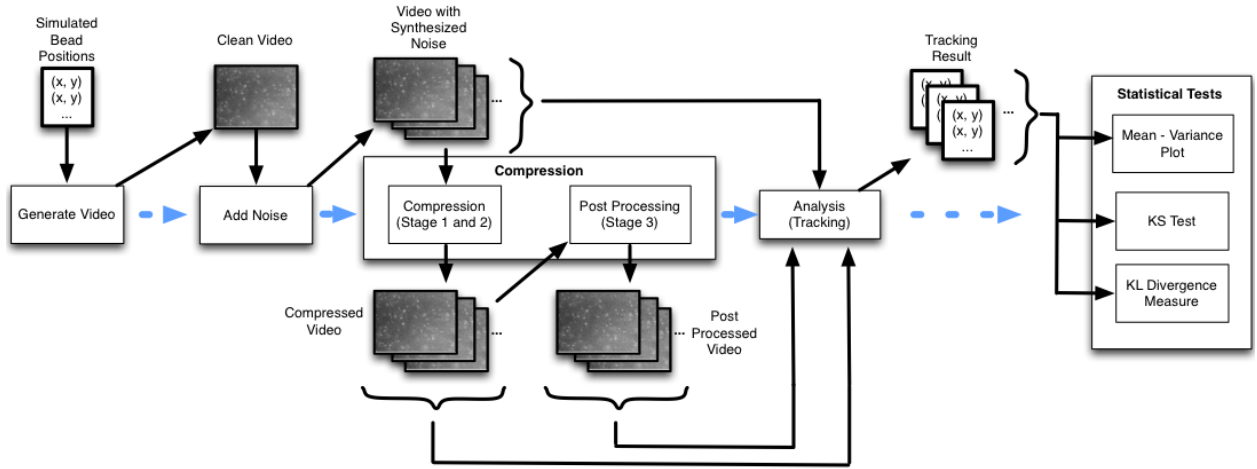


Figure 3.10: Flow chart of the experiment steps with synthetic data

The overall experiment flow on synthetic data is illustrated in Figure 3.10. I wrote a program to generate synthetic microscopy video frames. This data generating process is composed of several stages. First, I use a program to simulate bead trajectories with Gaussian random walks. In this experiment, I generated ten bead trajectories. The data was stored as a list of x-y pairs, describing the sub-pixel bead positions on every frame in the video. For an 1800-frame video with ten beads, I had ten lists with length 1800. With the bead trajectory data, I generated ten videos that contain beads. All ten videos share the same bead trajectories.

In the second stage, for each bead position in every frame, I generate a 2D Gaussian blob with pre-determined mean intensity and standard deviation values. I place it so that it is centered at the given sub-pixel x-y location based on the trajectory data list. The result is a clean video without background noise.

The next step is to add per-pixel noise into the video using Eq. 3.1. I generated the final pixel intensity values with one Gaussian plus Poisson distribution with λ equals to the pixel intensity value and σ equals to 0.01. The values are selected such that the resulting video has the similar characteristics to a real microscopy video. Therefore in every video, the background and foreground pixels values differ, but they are samples from the same distribution.

This results in a set of ten noisy videos. They each share the same bead trajectory, but they have different noise. Every video contains ten beads. Every video has 1800 frames. Figure 3.11 shows 1 frame in one of the ten videos.

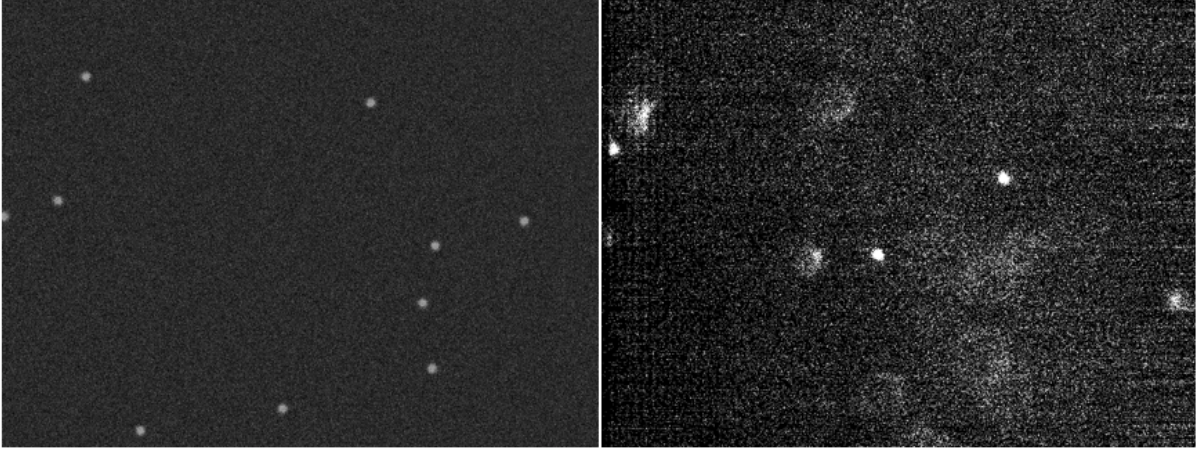


Figure 3.11: Sample video frames, left: synthetic video, right: real video

I tested the compression methods with the noisy videos from the data generation process. For every video, I first identify the foreground using correlation-based segmentation followed by mathematical morphology. The dilation operator size in the refinement is set to 5 pixels, which is smaller than the value I used in the analysis-preserving compression method. This setting does not ensure the exact same analysis results as the original video. I generate a binary map from the first step. Then I process the video with five approaches to generate 5 sets of compressed videos: (a) based on the binary map, I average the background in the video and leave the foreground unchanged. Then I compress the video using H.264. (b) Based on the binary map, I apply a customized H.264 compression by applying a low quality setting (qp=51) for the background pixel

blocks, and applying a high quality setting (lower qp) for the foreground pixel blocks. (c) I further add synthesized noise into the resulting compressed video from (a). (d) I further add synthesized noise into the resulting compressed video from (b). In method (e) I directly compress the original video (ignoring the binary map) using H.264.

To compare the performance of the five compression methods at different quality levels, I adjust the parameters to generate a set of compressed videos with different sizes for each compression method. In methods (a) and (c) I increase the dilation operator size from 5 pixels to a larger value, which produces a larger foreground region (and thus less compression). In methods (b) and (d) I used a list of qp values when compressing the foreground blocks in H.264 compression. In method (e) I used a list of qp values for H.264 compression so that I have different sized videos. I call methods (a) and (c) analysis-aware compression variation 1 (V1). I call (b) and (d) analysis-aware compression variation 2 (V2).

By including analysis-aware compression V1 in the experiment, I also include a generalization of the analysis-preserving method. By increasing morphology dilation size in the refinement stage, I eventually reach a large enough dilation size that makes the analysis result the same as the one for original video.

I plot the sizes of compressed videos from various compression methods so that I can compare their relative effectiveness at a given compressed size. For each method, I ran the compression with all 10 videos to generate a population of compressed videos. The compressed video size for the videos after post-processing is considered the same as before post processing because the post processing can be performed during reconstruction using the compressed video before post-processing as input.

I analyzed the tracking of beads using video spot tracker (CISMM, 2017). I applied the tracking on the original uncompressed video and all compressed videos: before post processing (a and b) and after post-processing (c and d). I then computed the MSD for the tracking results. Figure 3.12 shows the relationship between MSD values and video compression ratios. The MSD values were plotted as a multiple of the MSD values for original video (1.0 means the MSD values are identical

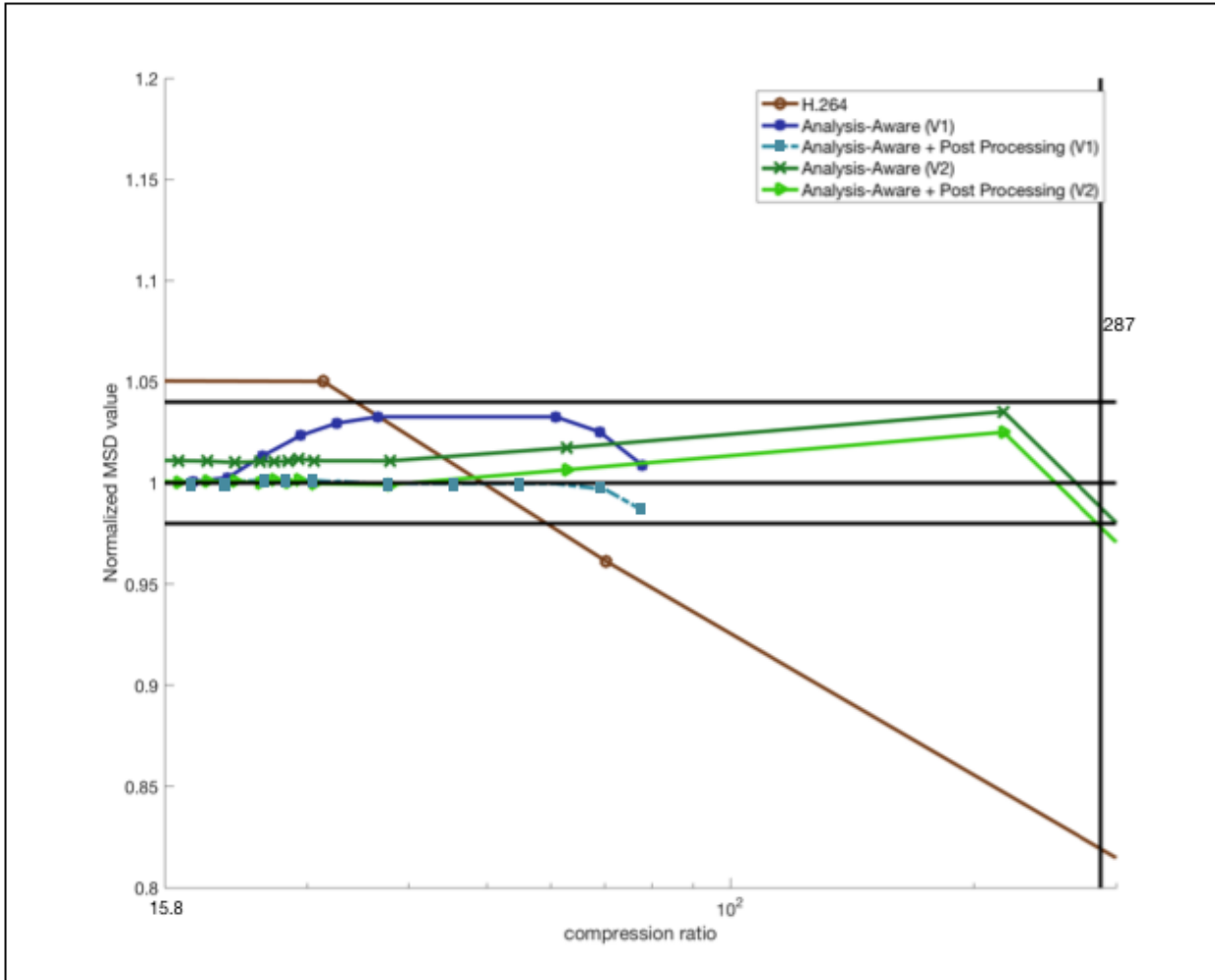


Figure 3.12: Scaled MSD values vs. compression ratio, for five groups of synthetic videos.

to the ones from original videos). Each curve represents the mean MSD values among 10 copies of videos with the same foreground and different instances of sampled noise values from the same noise distribution.

The result suggests that two variations of the analysis-aware compression method both generate a higher quality compressed video because the points on the curves are all close to one (in the range between 0.98 and 1.02) whereas standard H.264 yields a curve far away from 1.0 given the same compression ratio. The curves for the videos after post-processing are closer to the 1.0 horizontal curve, indicating that post-processing of adding back-noise improves the video quality regarding

analysis. V1 with post-processing is very close to the original result, indicating that it may be indistinguishable from it.

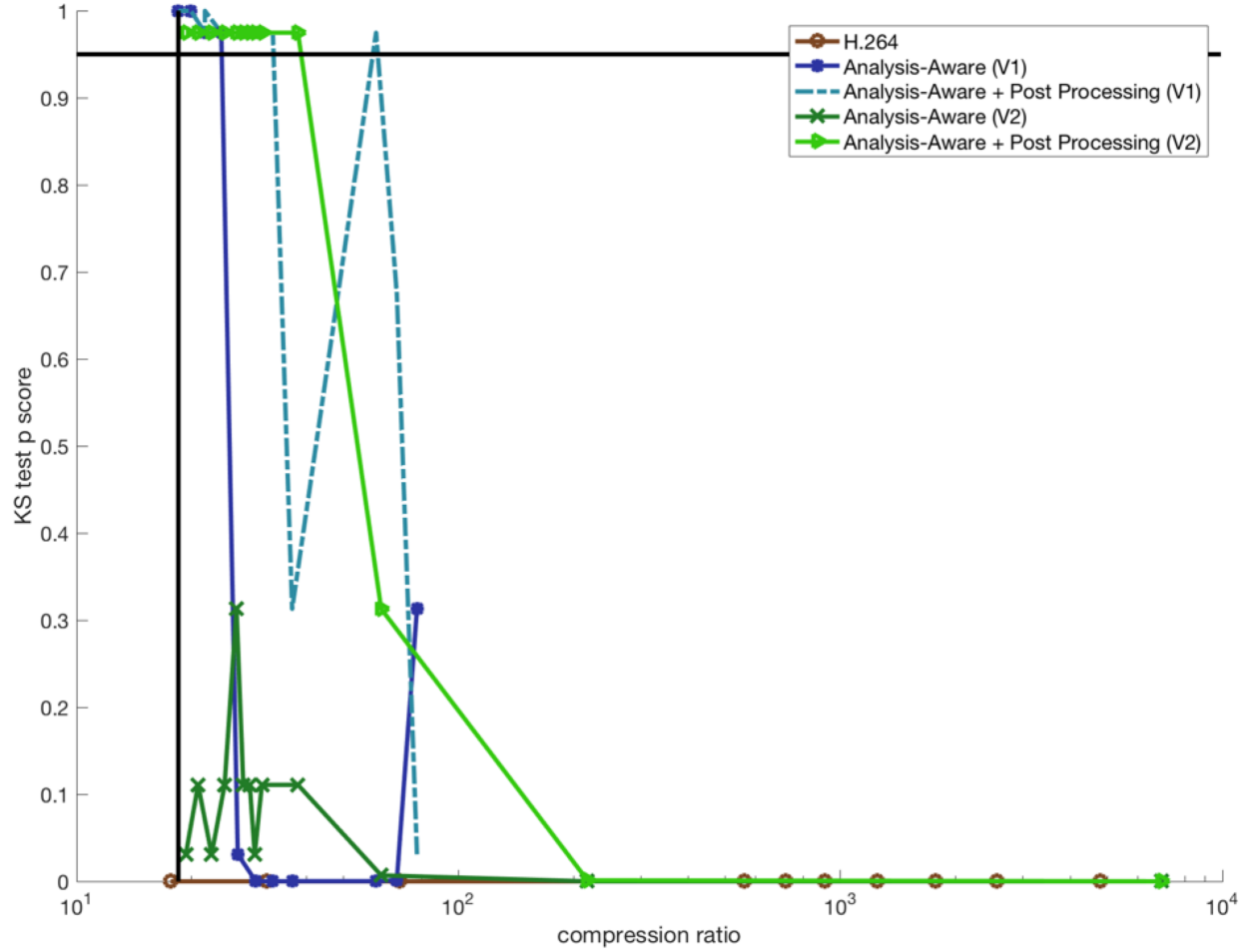


Figure 3.13: KS test p values vs. compression ratio. The horizontal line shows the KS test p score 0.95. The vertical line in every plot shows the compression ratio for the previous analysis-preserving method.

To further probe for potential differences between the original and compressed analysis, I performed the KS test on the MSD values. The KS test process involves defining a null hypothesis and making a decision on whether to reject the null hypothesis. In my case, the null hypothesis states that the two input samples of MSD values are drawn from the same distribution. The KS test process gives a decision on whether the null hypothesis should be rejected. The p value comes out from the KS test result indicates the error rate in rejecting the null hypothesis. For instance, for the KS test that yields $p=0.95$, rejecting the null hypothesis means to conclude that the two input

samples are from the different distribution is very likely to be wrong. Therefore, based on the KS test result we should consider the two input samples are drawn from the same distribution. On the other hand, note that there is no way to prove the null hypothesis, by the definition of the KS test.

Interpretation of the p value

The precise definition of the p value in the KS test is the probability that the two cumulative frequency distributions would be as far apart as observed if the two samples were randomly sampled from identical populations (Kirkman, 1996). For $p=0.95$, there is 95% chance that another round of sampling will generate two cumulative frequency distributions that are at least as far apart as the current samples.

In this experiment, the MSD values were not normalized by the MSD values from the original videos. I used one bead's MSD values across 10 versions of the videos that share the same foreground content. After that, I selected a fixed window size. Figure 3.13 shows the p values output from KS test for MSD values on videos compressed using my compression approach V1, my compression approach V2, and the standard H.264 compression. The curves plot p value vs. compression ratio. The horizontal line indicates the test decision threshold. For all p values above the line, the null hypothesis is not rejected, which means that there is no strong evidence that the MSD values obtained from compressed videos are sampled from a different population than those from the MSD values obtained from the original video (there is a high probability that equals to the y value indicated by the line that another round of sampling will generate two cumulative frequency distributions from the MSD values that are at least as far apart as the current samples).

The KS test for standard H.264 compressed videos always rejects the null hypothesis, indicating that the distributions are statistically distinguishable. For my approach before the post-processing the curve sometimes goes above the threshold, but it also falls below the threshold as compression ratio increases. For the video compressed with my approach after post-processing (V2), the curve is always above the threshold until it reaches compression ratio 38.

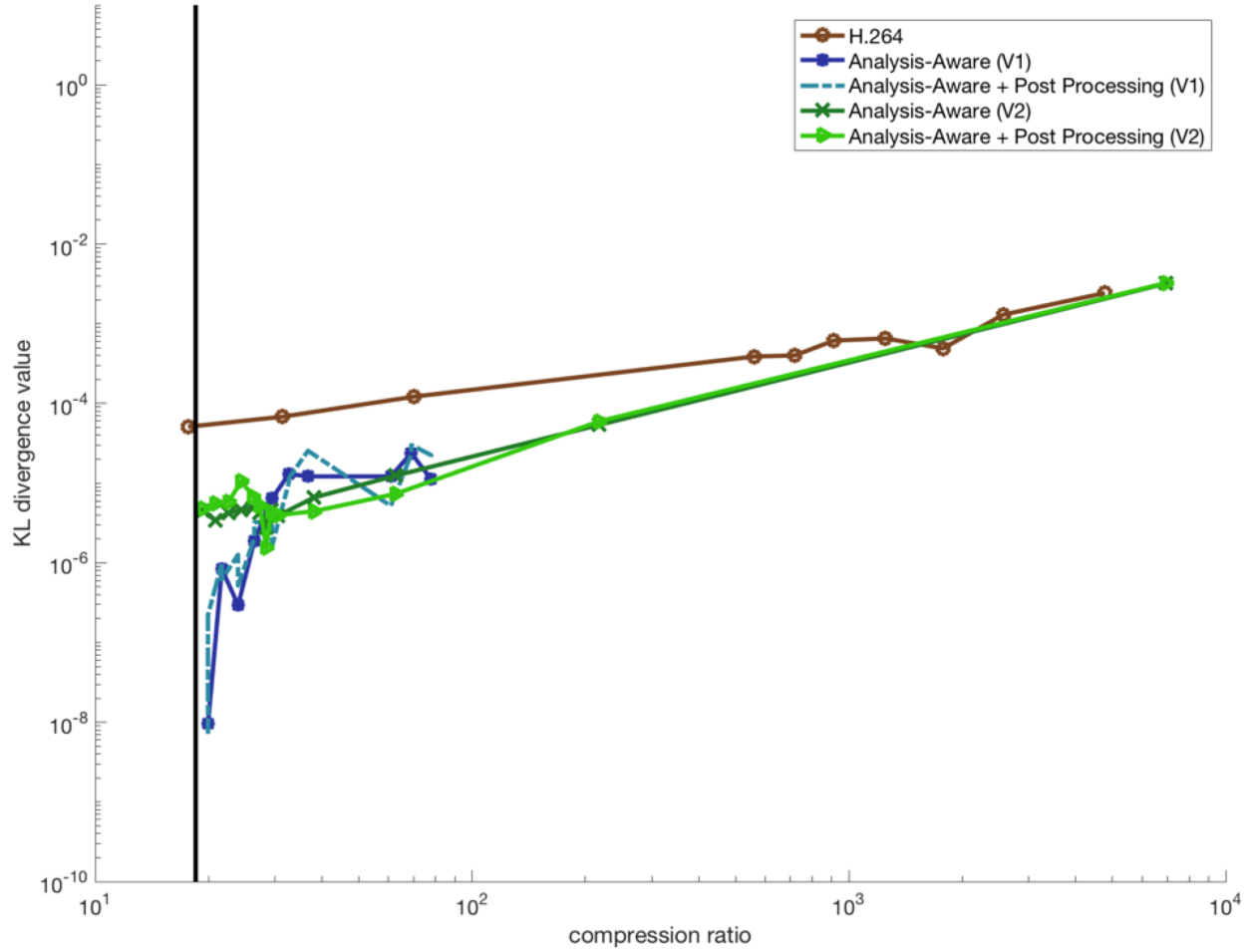


Figure 3.14: K-L divergence values vs. compression ratio, for five groups of synthetic videos. The vertical line in every plot shows the compression ratio for the previous analysis-preserving method.

I also computed K-L divergence values on the same data. The result is shown in Figure 3.14. A lower K-L divergence value suggests a smaller distance between the compressed video's MSD value population and the original video MSD value population.

In this experiment, all compression videos with my method gave similar results for compression ratios smaller than 148, which was better than the result value from standard H.264. My compression approach V2 outperformed standard H.264 up until a compression ratio of 1450.

Experiment on Real Data

I also performed experiments on videos from real experiments. For real data, it is impossible to get the true bead trajectory and generate multiple copies of the same bead trajectory and different

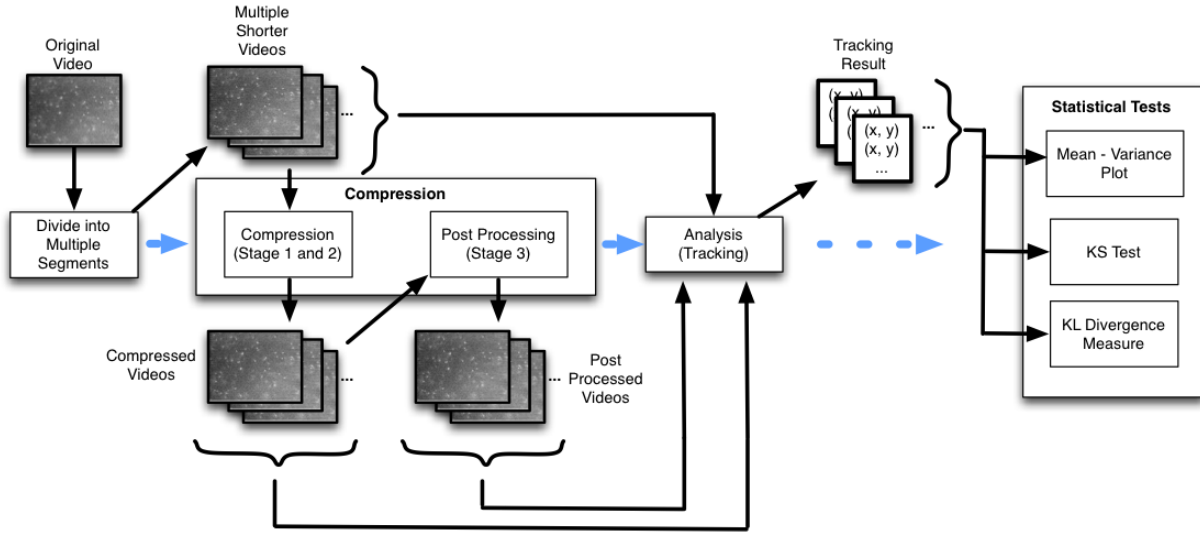


Figure 3.15: Flow chart of the experiment steps with real data.

background noise. I handled this by dividing each video into 10 parts and performing tracking on each of them to produce a population of estimates. It is assumed that there is no significant background noise property change across the videos in single-video the test set. The experiment process is illustrated in Figure 3.15

The scaled MSD values for various compression methods vs. video compression ratio plot for real video data is given in Figure 3.16. For real data, analysis-aware compression approach V1 does not perform better than H.264 compression for compression ratios greater than about 3000, even after post-processing. On the other hand, V2 always gives a better result than H.264 compression.

Figure 3.17 and Figure 3.18 shows the MSD values from multiple compression methods compared using KS test p -value and K-L divergence value. For this data set, K-L divergence does not show a great difference among different compression methods. In the KS test, the p values for the videos compressed by standard H.264 stays below the 0.99 threshold, while my compression method V1 achieves compression ratio larger than 3000 and remains above the 0.99 threshold. My compression method V2 achieves compression ratio 12 before it drops below the 0.99 threshold.

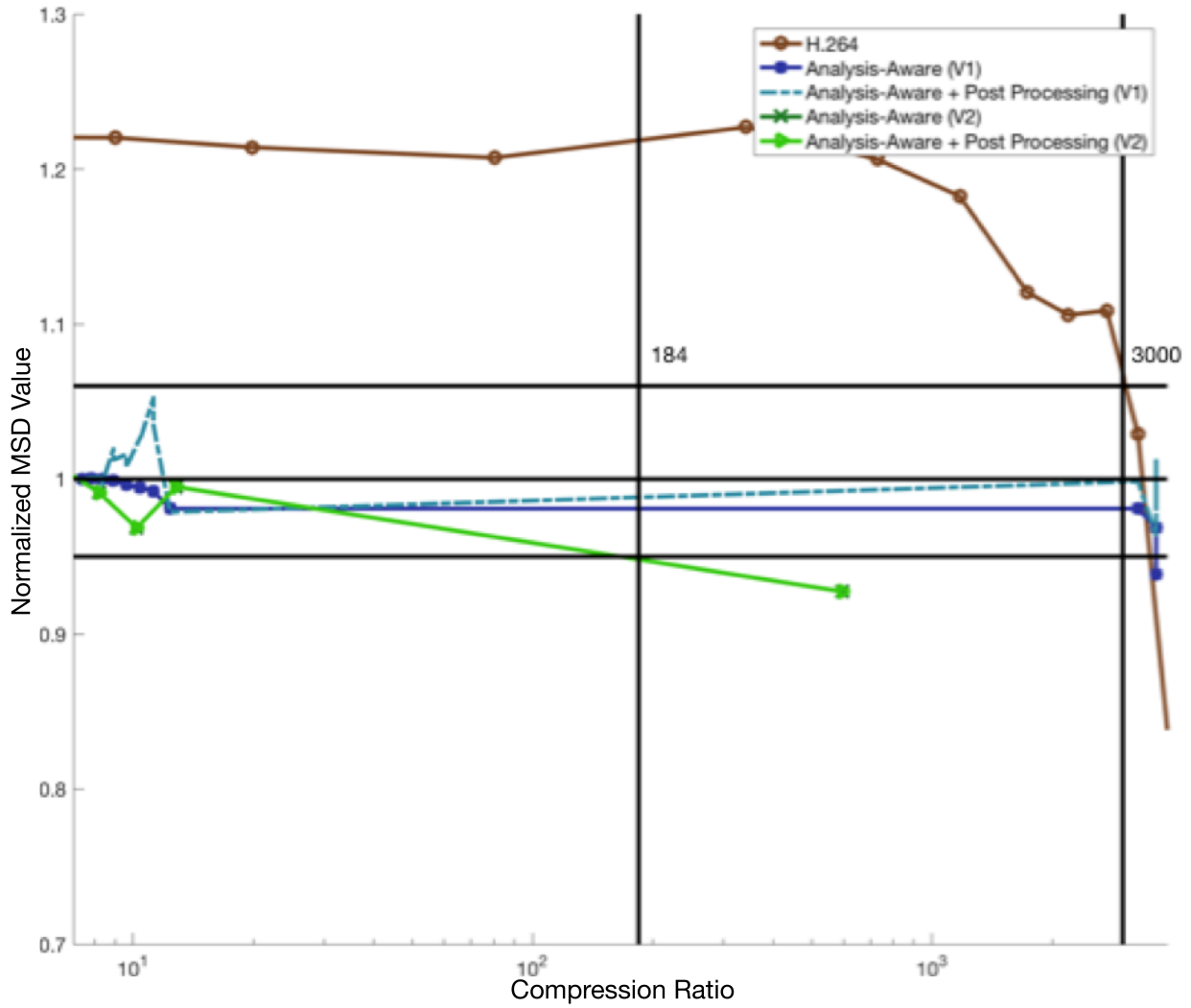


Figure 3.16: Scaled MSD values vs. compression ratio, for five groups of real videos.

	Lossless	V1	V2
Synthetic Data	18.4	32.9	39.3
Real Data	9.7	3580	23.1

Table 3.6: Achieved compression ratios for applying analysis-aware methods and lossless compression method on synthetic data and real data and maintain KS test p-score larger than 0.95 in the resulting video. For synthetic data, an improvement of around a factor of 2 was achieved above the earlier lossless method. For real data, which had more noise, the improvement was around a factor of 350.

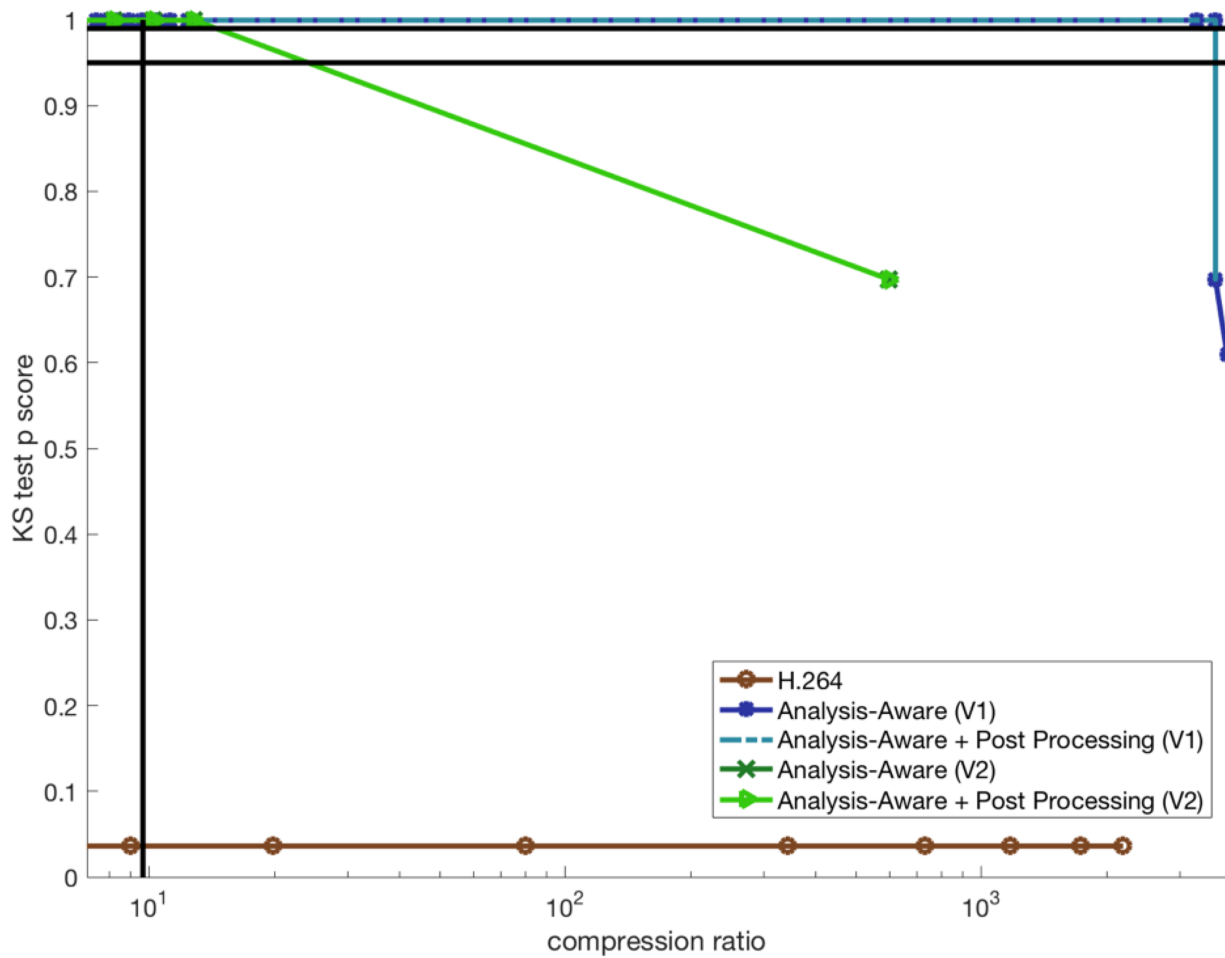


Figure 3.17: KS test p values vs. compression ratio. The two horizontal lines showing the KS test p score 0.95 and 0.99, respectively. The vertical line in every plot shows the compression ratio for the previous analysis-preserving method.

3.5 Summary

In summary, I have described an analysis-preserving method and an analysis-aware method for microscopy video compression based on correlation and mathematical morphology. Experiments on several real video data sets show that the analysis-preserving compression method can achieve compression ratios of >100 , reducing file size by 99+%. For analysis-preserving compression these compression ratios correspond to file sizes that are $25\times$ smaller than those generated by lossless compression techniques.

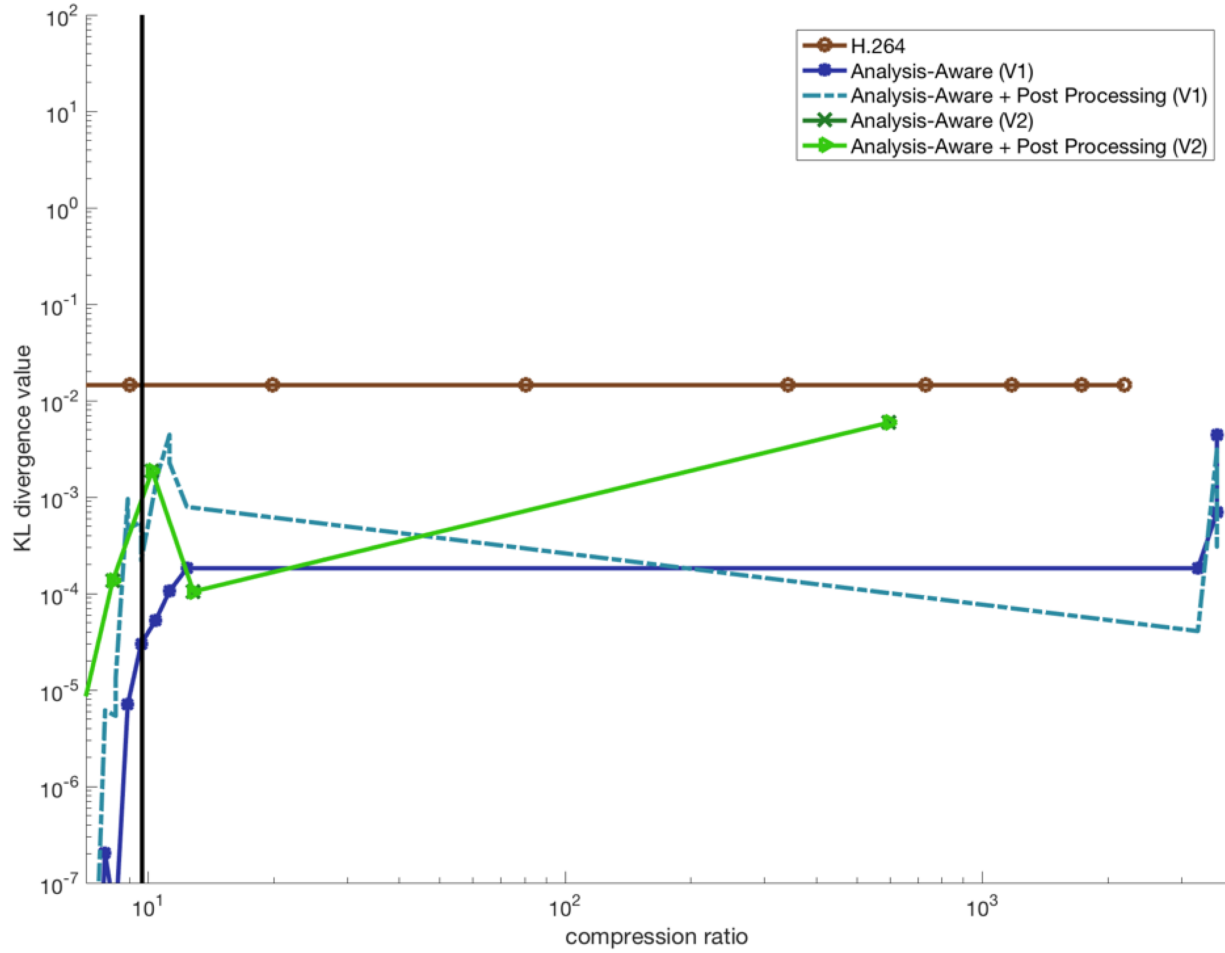


Figure 3.18: K-L divergence values vs. compression ratio, for five groups of real videos. The vertical line in every plot shows the compression ratio for the previous analysis-preserving method.

I show that the analysis-aware compression method preserves scientific analysis by running statistical tests on it; the resulting probability distribution of analysis results is not statistically distinguishable from the analysis result probability distribution from the original video up to a certainty of 95-99% while halving the size of the compressed data compared to lossless compression (Table 3.6). The experiment for measuring compressed video quality was based on MSD values from tracking diffusing beads. The result suggested that comparing against standard video compression technique H.264, for most compression ratio values, analysis-aware compression method gives a better quality video in terms of the analysis results.

The analysis-aware method extends to other types of microscopy video analysis besides object tracking. The statistical validation method can be modified to apply to each type of analysis. I evaluated quality based on KS test and K-L divergence. If a different metric is desired, the statistical tests can be replaced with the new technique applied to the same population of data.

The correlation-based segmentation method used in my compression technique was verified by analysis of tracking in a fluorescence microscopy videos in my experiment. But in the experiments for analysis-preserving compression, I showed that this segmentation method works for a variety of microscopy video types including fluorescence video, bright field video, fast moving beads video and cell video and associated analysis routines including segmentation.

CHAPTER 4

IMAGE COMPRESSION TO GENERATE ENERGY EFFICIENT BROADCAST IMAGE DATA

In this chapter, I switch my application domain from video compression to image compression. I present two new systems I built that integrate a bunch of new image processing techniques I invented, to support extreme image compression. The two systems make use of properties in an input image to identify the content of interest in the input image. The systems can compress a digital color image with size 64×64 pixels small enough so that image broadcasting via a Bluetooth Low Energy broadcasting channel becomes practical.

In this modern age of the Internet of Things (IoT), it is now possible to literally glue tiny computers to everyday objects, so that they can sense, react, and tell their own stories. The IoT community has embraced wireless standards such as Bluetooth Low Energy (BLE) (Gomez et al., 2012) and developed protocols such as iBeacon (Newman, 2014), in order to create programmable ‘beacon’ devices that periodically broadcast a small amount of preloaded data, and last for multiple years (Siekkinen et al., 2012) on a coin-cell battery. Broadcast messages from beacon devices typically contain information about an object, a location, a web-resource, or just an arbitrary string. This connectionless mode of BLE does not require a receiver to pair/bond or connect to a sender, and hence, there is no overhead of connection setup and no inconvenience of requiring a user to enter pins and passwords. These broadcast messages are received by a BLE capable mobile device to obtain relevant information just-in-time and on-the-spot.

Emerging applications of beacon devices include advertising merchandise in retail stores (Pierdicca et al., 2015), identifying late passengers at airports, authorizing people at hospitals,

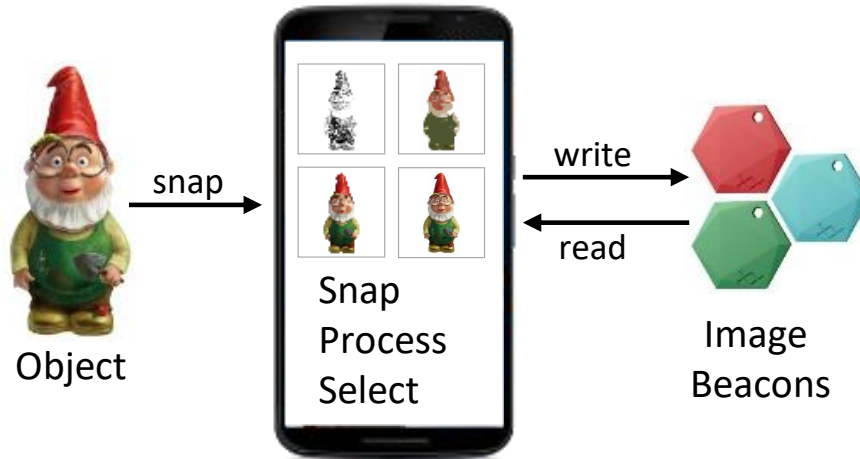


Figure 4.1: An image beacon system.

smarter signage, indoor navigation (Martin et al., 2014), and tracking moving platforms like airline cargo containers, computers on wheels, museum artworks, or even people (Conte et al., 2014).

The enabling technology behind these applications is the ability of a beacon to simply broadcast a few bytes of data (usually a URL or a UUID) as BLE 4.0 advertisement packets. The bound in data rate comes from the lifetime requirement of these devices. Such a tight budget on payload size and the maximum data rate have limited a beacon’s capability to only be able to broadcast an identifier or a small amount of text (effectively ~ 30 bytes). The next generation BLE 5.0 beacon is expected to have a significant increase in broadcasting capacity (~ 200 bytes). Such an increase opens up the possibility to design beacons that can serve larger assets, e.g., an image, carried by connectionless BLE advertisement packets. However, even a simple 72×72 PNG image, such as the Android launcher icon, has a size of over 3KB. To store and broadcast this image, either I require to use a dozen BLE 5.0 beacons, or I will have to accept a very long image transmission and loading time.

Image compression is a natural way to deal with this problem. Existing image compression algorithms, however, fail to achieve the desired compression ratio for an image to be broadcast over BLE. Hence, a fundamental challenge toward realizing an image beacon is to devise an algorithm that efficiently represents an image using as few bits as possible, while taking into account the

application-driven limits on the number of usable beacons per image, broadcast message size, data rate, latency, and lifetime. I proposed solutions to this challenge and devised image beacon systems that broadcast binary images and color images.

Being able to broadcast images from beacons enables more powerful and feature rich applications than the ones supported by today's beacons. I envision that like the web has evolved from serving hypertexts to streaming multimedia contents, the natural successor of today's beacon devices would be the ones that broadcast images. Applications of image beacons would be in scenarios where there is no Internet connectivity but there is a need for storing and broadcasting information that can be best described by an image. For example, coordinating rescue workers in disaster areas, creating a bread-crumbs system for adventurous hikers and mountaineers, remote surveillance (when coupled with a camera), or even a simple system just to let someone know that 'We were here'. Recently, Google started to experiment with an idea called '*Fat Beacons*' (Hardill, 2016), where they are looking into broadcasting html pages over BLE. However, for lack of a suitable image compression technique, the pages do not support images.

As a first step toward realizing an image beacon, I explore the challenges to broadcasting binary images of different categories (e.g., alpha-numeric characters, basic shapes, and arbitrary binary images), and design algorithms to efficiently store contents of an image inside a set of beacon devices (Shao et al., 2016b). The set of beacons simultaneously broadcasts chunks of an image over BLE, which are captured by a mobile device to reconstruct the image. Standard image compression algorithms are not good enough to archive the required compression ratio so that an image can be stored inside a beacon. I investigate image approximation/coding techniques that take into account the limits on number of beacon devices, number of bits available in a beacon device, data rate, latency, and lifetime. Based on empirical analysis, I devise a patch-based image approximation algorithm which greatly reduces the image data while keeping the image distortion under a threshold. I investigate the trade-offs between the image quality and the power consumption to determine the best set of parameters for the system under user-specified constraints.

I further developed a system that store and broadcasts color images with BLE beacons (Shao and Nirjon, 2017). The crux of the system is an algorithm that analyzes an image to identify its ‘important’ semantic regions (as defined by the user or the use case) and then encodes them differently than the rest of the image to reduce the overall image size. The image data are written to and read from the image beacon system using a smartphone application, which runs the proposed compression and rendering algorithms. I use the term ‘beacon system’ instead of ‘a beacon’, since a compressed image may still require more than one physical beacons to ensure its acceptable quality. Allowing multiple beacons per image makes the system flexible. It widens the scope for optimization and helps satisfy users who are willing to dedicate more beacons for better results. Besides, until BLE 5.0 is available, I need to simulate its broadcast capacity with multiple BLE 4.0 devices anyways.

I have developed prototypes of both binary and color image beacon system using a set of commercially available Estimote beacons (Estimote, 2017), and developed an Android application that takes images of an object of interest along with user-specified requirements and constraints on broadcasting the image as inputs, generates previews of the image to be written, writes the image representation into a set of beacons, and reads the broadcasted image back. Figure 4.1 shows an example scenario where a user snaps photos of a gnome statue which he is interested in broadcasting. The smartphone application performs image processing on the phone to produce multiple versions of broadcast image. The user selects one of these compressed images that satisfies his requirements (e.g. available beacons, image quality, lifetime, and image loading latency). The user is allowed to change his requirements and the app immediately shows options for the best possible compressed images under those constraints. The application writes the image data into the beacon system and the image is broadcasted by the beacons. A reader application reads the broadcasted image and displays it on the phone.

I perform an in-depth evaluation of the beacon system. I describe a set of results showing the trade-offs between system lifetime and image quality, when the image type and the number of beacons are varied. I also deploy an image beacon system indoors, and perform a user study in

a real-world scenario in order to have a subjective measure of the quality of the received images, where a group of 20 participants are asked to identify the objects from their beacons images of various resolutions, and locate them among a set of similar looking objects in the real world.

The main contributions of my research work in this chapter are as follows:

- To the best of my knowledge, I am the first to propose a binary image beacon system that uses multiple BLE beacons to broadcast binary images over the BLE advertisement messages.
- I have devised a patch-based image approximation algorithm that greatly reduces the image sizes. I quantify the tradeoffs between the image quality and the device lifetime, and determine the best set of parameters, under the user-specified constraints on the number of beacons, latency, and expected system lifetime.
- I have developed and evaluated a prototype of a binary image beacon system that broadcasts binary images of various types (e.g., alpha-numeric characters, basic shapes, and arbitrary binary images). The evaluation shows that a set of 2–3 beacons is capable of broadcasting high-quality images (75%–90% structurally similar to original images) for a year-long continuous broadcasting, and both the lifetime and the image quality improve when more beacons are used.
- I propose a new color image beacon system that uses multiple BLE beacons to broadcast color images over the BLE advertisement messages.
- I have devised an image approximation algorithm that is tailored to the needs of an image beacon system. I quantify the tradeoffs between the image quality and the device lifetime, and determine the best set of parameters, under the user-specified constraints on the number of beacons, latency, and expected system lifetime.
- I have developed and evaluated a prototype of an image beacon system that broadcasts color images of various types (e.g., near-distance indoor and outdoor objects, road signs, and buildings). The evaluation shows that one BLE 5.0 beacon would be capable of broadcasting

good-quality images (70% structurally similar to original images) for a year-long continuous broadcasting, and both the lifetime and the image quality improve when more beacons are used.

4.1 Related Work

In (Pierdicca et al., 2015), the author discussed an intelligent system involving beacon devices for Customer Behavior Analysis (CBA). The goal is to show how beacon technology could help gather and classify customer behavior data in retail stores. They deployed the system into a real retailing scenario and collected the data from mobile devices interacting with Beacon devices. They also proposed the further data analysis process on the collected data. (Martin et al., 2014) talks about an indoor localization system constructed with beacon devices. With multiple beacons places in the environment, the author showed that the error of the estimated location of the object could be as small as 0.53 meters in average. (Conte et al., 2014) discusses an Beacon occupancy detection system deployed in smart buildings. In the implementation, they modified Apple's iBeacon protocol to better fit their requirements. They showed that with the BLE technology such system could be more energy and cost efficient than previous solutions.

Previously people have developed various image compression techniques. Many of these techniques have been standardized and widely used, including JPEG that is based on discrete cosine transform, and JPEG2000 that is based on wavelet transform. If the images are limited to binary, a compression method designed for binary images is expected to show a better performance (smaller compressed image but equivalent image quality) than JPEG or JPEG2000. One approach to compress binary images is to partition the shape in the image into rectangles and record the upper-left and bottom-right pixel location of every rectangle (Mohamed and Fahmy, 1995; Zahir and Naqvi, 2005). This works great on images that contain shapes that can naturally be divided into rectangles. But in my binary image beacon system, many images contain curves in different directions and different curvatures. For binary images containing curves, rectangle-based approach do not give a desired performance within the limited storage constraint. Given the fact that many

of my binary images contains curves, chain coding based compression (Kim et al., 1988) would be preferred. (Zahir et al., 2007) proposed another chain-coding based compression method. They showed that their coding method could losslessly compress a complex contour shape into around 900 bytes. And their result outperforms previous image coding techniques including JBIG1, JBIG2 and data compression library WinZip. Their method yields compressed image sizes of at least 200 bytes for my data. This still does not satisfy the storage requirement of my systems, and at the same time I do not need the very-high compressed image quality that their method offers. Therefore, I designed a binary image compression method that better fits my requirement.

Shapiro (Shapiro, 1993) developed an image encoding technique named embedded zerotree wavelet (EZW) encoding, which is computationally expensive and slow. Said and Pearlman (Said and Pearlman, 1996) developed a better wavelet-based image encoding method based on set partitioning in hierarchical trees. This method gives similar image compression performance regarding quality and size, and the same time it achieves a faster computation. I have used this method in my color image beacon system.

Lu et al. (Lu et al., 2000) introduced a piece-wise linear image encoding method using surface triangularization. Their triangularization algorithm fits the image surface in a top-down manner. The idea is to apply a constrained resource planning to allocate the least amount of triangles while achieving a small image approximation error. Their experiment result shows that the triangularization method compresses images with a compact code length with a guaranteed error bound. Their method's target is to achieve near lossless compression. With that target, their method is designed to usually generate a large number of triangles (order of magnitude 10000) for an input image, which requires data size limit much larger than 200 bytes. Since my color image beacon system does not require lossless compression, we cannot directly apply their method.

A rich set of image segmentation methods exists in the literature. In the past decade, more modern techniques involving machine learning have been invented. State of the art neural-network based method (Zheng et al., 2015) can achieve a very accurate result (highest score 90.4 on IoU evaluation on airplane type testing data). The method is based on neural-network and conditional

random field. However, this method is computationally expensive to run on a cellphone. Otsu's method (Otsu, 1975) is based on finding a separation on image pixel intensity histograms, which does not take care of local image structure. Gabor filter segmentation is based on finding edges in an image using Gabor filtering. Marker-controlled watershed methods (Parvati et al., 2009) use mathematical morphology to pre-process the data, followed by a watershed segmentation. This avoids over segmentation, which is a weakness of traditional watershed method.

The notion of foreground and background information can be obtained by disparity estimation with semi-global matching (Hirschmuller, 2005). The method enforces smoothness in the neighbor matching process to reduce matching errors. The semi-global matching method is integrated in my image beacon system as one step in image segmentation.

4.2 BLE System Characterization

An image beacon system should be capable of storing and broadcasting contents of an input image without the aid of any additional sources of information about the image. The image is either a photo taken with a camera, an image containing basic shapes, or a hand-drawn image by the user on his smartphone's touchscreen. The image data will be written to and read from the image beacon system using a smartphone application. I assume that the system is self-contained, i.e., no additional information about the broadcasted image is available from any other sources globally (on the web) or locally (on the smartphone).

The problem is formally stated as: given a binary image x (i.e. each pixel is represented by one bit) having the dimensions of $N \times M$ bits, the number of available beacon devices K , the payload size of each beacon packet C bytes, the maximum allowable broadcast rate of R packets/sec, and the maximum allowable latency for an image T , the objective is to find an approximate representation of the image \hat{x} so that the lifetime τ of the beacon system is maximized while the approximation ratio $\lambda(x, \hat{x}) \in [0, 1]$ of the image is high ($\lambda = 1$ means no distortion).

Now, for a single beacon, the broadcast rate:

$$R = \left(\frac{NM}{8C} \right) \frac{1}{T} \quad (4.1)$$

For K beacons, considering $\log K$ overhead bits for addressing the beacons, and K times more payload capacity:

$$R = \left(\frac{NM + \log K}{8CK} \right) \frac{1}{T} \quad (4.2)$$

Both Eq. (4.1) and Eq. (4.2) are for undistorted images.

The lifetime τ of a BLE device depends on its inter packet interval and in general, $\tau \propto \frac{1}{R}$. Replacing R and incorporating approximation ratio λ into (4.2):

$$\frac{1}{\tau} \propto \left(\frac{\lambda NM + \log K}{8CK} \right) \frac{1}{T} \quad (4.3)$$

The above equation relates the lifetime of an image beacon system and the approximation ratio of any image compression algorithm. In the binary image beacon system, I devise a patch-based image approximation algorithm that achieves a sufficiently large λ for a reasonably high lifetime of the system.

4.3 Image Beacon and Use Cases

An image beacon is a battery powered system that supports years-long image data storage and broadcast over BLE advertisement packets without requiring the Internet connection. An image beacon system enables many applications, particularly in scenarios when there is a need to store and broadcast image data for a long period but the Internet is unavailable. Examples of such systems include:

4.3.1 Long-term Surveillance Systems

I can deploy an image beacon system to monitor an environment where the Internet connection is not always available. The image beacon system can be attached to a camera that sleeps for most of the time and takes pictures only when triggered by some other sensor (e.g. movements). The captured image is then stored into an image beacon. Since data is broadcasted by the BLE beacons, a surveillance drone can fly over the environment to collect the image data (embedding). Later the embedding can be used to generate the actual images.

4.3.2 Navigation Systems

Graphical navigation information can be stored and deployed with an image beacon to provide travellers in remote areas with route guidance. Since an image beacon enables storage of arbitrary images, the navigation information does not have to be selected from a set of pre-determined road signs and can be highly customized. A similar application scenario is navigation in mining tunnels where the environment may be constantly changing and a reliable Internet connection is unavailable to provide Internet-based navigation services.

4.3.3 Internet of Everything Minus the Internet

There are ongoing efforts in developing infrastructures that can map every item in the physical world to a virtual object to the web. An example is Google's attempt in designing a 'fat beacon' that can store and broadcast web pages (Hardill, 2016). An image beacon can be a key component in building an 'Internet of everything' to help support thumbnail images on a fat beacon web page.

4.3.4 New Applications

An image beacon can enhance user experience in entertainment applications involving interactions between IoT devices and physical world. Examples include the game "Geocaching" (GEO, 2018) and BLE beacon powered indoor augmented reality described in Chapter 5. In the original

Geocaching, players use a smartphone app to locate the approximate location of a hidden “treasure box”. Then the player needs to find the box with the location hint. After the box has been successfully found, the player can take out the item previously stored in the box and replace it by any new item. Using a graphical “virtual item” stored in an image beacon instead of a physical object, the game experience can be improved by avoiding encounter of unwanted physical items. For indoor augmented reality using BLE beacons, texture data can be stored into image beacon to power a more realistic 3D object rendering.

4.4 Challenges in Building an Image Beacon

The major challenges in designing an image beacon system are listed below. The system suffers from the limited payload size of beacon packets, and the limited bandwidth of BLE. On the other hand, typical image data compressed by standard image compression techniques requires much more data storage space than that a BLE system can afford.

4.4.1 Limited BLE Bandwidth

The main advantage of BLE over any other wireless protocols is its extremely low-energy packet transmission capability. This is achieved by aggressively maximizing the sleeping interval and sending data packets at a much lower rate than classic Bluetooth’s.

According to the BLE 4.0 specification, the maximum payload size C available in beacons is 30 bytes. The expected lifetime of BLE beacons depends on the inter-packet interval (Dementyev et al., 2013). For example, a BLE 4.0 beacon would last up to 3.5 years, if a packet is sent at every second (i.e. $R = 1$). Therefore, for a beacon system to last for 3.5 years, its broadcast bandwidth cannot exceed 30 bytes/sec. Recently, BLE 5.0 has been announced (BLE, 2016) to offer a ~ 200 bytes broadcast capacity and a 2X increase in transmission speed.

Figure 4.2 shows the expected lifetime of Estimote BLE beacons for various inter-packet interval. For example, the beacon would last up to 3.5 years if a packet is sent at every second (i.e. $R = 1$). Therefore, for a beacon system that lasts for 3.5 years, its broadcast bandwidth is bounded to

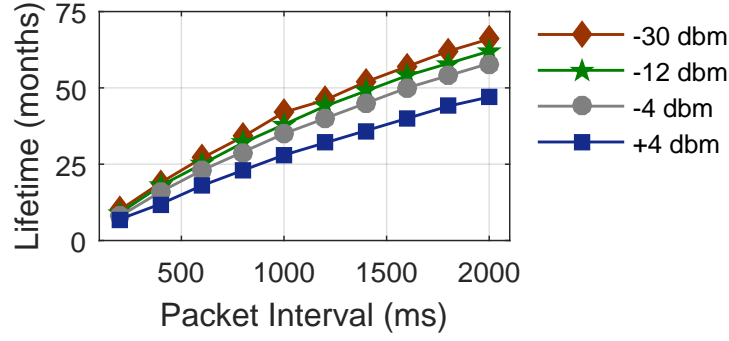


Figure 4.2: The lifetime depends on packet transmission rate and signal strength.

the maximum limit of 16 bytes/sec, and the latency of a complete image transmission cycle would be 125 seconds for a single beacon, or 1 second for a set of 125 beacons. Again, by using an image approximation algorithm having a sufficiently high compression ratio, this latency can be reduced significantly.

4.4.2 The Case for Lossless Image Broadcast

The size of a typical 72×72 PNG image can be anywhere between 3 – 13 KB. Therefore, to transmit such an image, a BLE 4.0 beacon would require 191 – 832 broadcast packets, or alternatively, it would require up to $K = 832$ beacons to simultaneously broadcast different slices of an image. The latency of a complete image transmission cycle would be up to $T = 13.9$ minutes for a single beacon, or 1 second for a set of 832 beacons.

When BLE 5.0 beacons replaces 4.0, the transmission latency will drop to 52 seconds for one beacon, or 1 second 52 of them. Therefore, without compressing the image content, even the new BLE 5.0 beacons will not be able to support a fast image beacon system with a reasonably small number of beacons.

4.4.3 The Case for Compressed Image Broadcast

If standard image compression algorithms could generate compressed images that meet the size and quality requirements of an image beacon system, the problem would have been already solved.

But the fact is, even the best of existing image compression methods, such as JPEG/JPEG2000 and PNG, are not capable of the desired data size for BLE broadcast. Figure 4.3 illustrates that JPEG/JPEG2000 generates extremely poor quality images given a size requirement of 300 bytes even for a very low-resolution (64×64 pixels) image. On the other hand, to have a compressed image of acceptable quality (having a minimal useful visual information to the viewer), JPEG/JPEG2000 takes about 2K bytes.

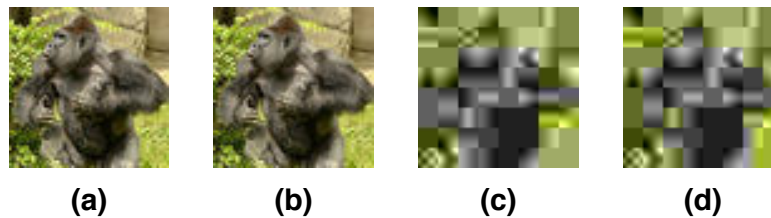


Figure 4.3: A 64×64 resolution image compressed in high/low quality settings using JPEG/JPEG2000: (a) JPEG high quality, 1963 bytes (b) JPEG2000 high quality, 2026 bytes (c) JPEG lowest possible quality, 738 bytes (d) JPEG2000 lowest possible quality, 391 bytes.



Figure 4.4: Two types of 64×64 resolution image compressed in PNG (a) from natural scene, 12112 bytes (b) JPEG2000 high quality, 1012 bytes. PNG is good for handling images with large uniform color regions.

PNG and Vector Graphics image, on the other hand, have the potential to generate a smaller compressed image that *may* fit the constraints. However, these codecs generate smaller images only if the input image is of a specific type – such as an image containing a few regions of uniform colors like a cartoon drawing, or when the shape is not complicated. This is illustrated in Figure 4.4. In general, PNG and Vector Graphics image encoding do not meet the requirements of an image beacon system that broadcasts color images taken by a smartphone user.

4.5 Algorithm Design

4.5.1 Patch-Based Binary Image Compression Algorithm

The process of converting an input binary image to its approximate equivalent is described in this section. The process has an one-time, offline phase where an ‘alphabet’ of carefully designed ‘patches’ are generated. During the on-line phase, the input image is encoded using these patches in order to generate a reduced version of it, which is suitable for writing into the beacons. Figure 4.5 illustrates the overall process.

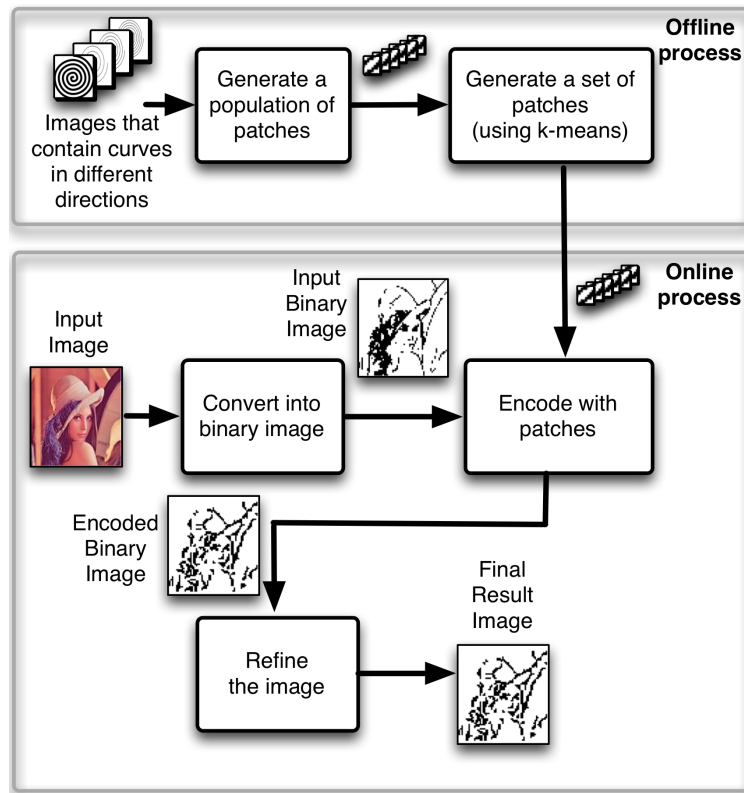


Figure 4.5: Beacon image processing pipeline.

Offline Processing: Patch Set Generation

In order to store an image into a limited amount of storage, one has to make a tradeoff between image variety and the quality of the compressed image. By limiting the image type to binary images that contain mostly curves, I show that it is possible to design a predetermined set of patches to

represent the image in a way that the encoded image is extremely compressed with high quality. The encoding of an image is based on an agreement on the set of patches between the encoder and the decoder. Hence, only the indices of the patches are required to be stored/broadcasted in order to encode/decode an image. An immediate question is therefore: *how do we design a suitable set of patches?*

There are two main considerations in designing the set of patches. First, determining the number of patches to use, and second, determining the content of each patch. The set of patches should be general enough to be able to represent a wide variety of binary images. On the other hand, the size of the set should not be too large, otherwise, the number of bits required to encode the patch indices will be large, resulting in larger images. A rule of thumb in designing base elements for images is to let the set of patches be rich enough for an input image so that given any sub-region within the image that has the same size as that of a patch, there is always patch in the set whose texture is roughly the same as the sub-region's texture, with a very high probability.

Since I have limited the image type to binary images, I seek to design a set of patches that contains binary textures of varies types and curves in different directions. The two-step process of generating the set of patches are as follows.

Step 1. Generating Patch Population

A simple approach to generating a population of patches is to divide a spiral image using a $g \times g$ grid to obtain a total of g^2 patches. I use spiral images since they are easy to parameterize. By having a set of parameterized spirals I can easily control the curvature and the direction of the curves in the patches. Figure 4.6 (a) shows an example of a spiral in a 3×3 grid. This approach, in general, produces a diverse set of patches containing curves of different orientations and directions. However, the patches are biased by the choice of the original spiral. Because of this, I choose to start from a set of binary images, where each image contains a parameterized spiral whose parameters are different from other spiral images. Figure 4.6 (b) shows an example where I have m spiral images. The parametric equation of the n^{th} spiral is: $x = t \sin(t)$, $y = t \cos(t)$, $0 \leq t \leq n\pi$. Each

of these n spirals is divided using a $g \times g$ grid to obtain a population of $g^2 n$ patches. Let us denote the population of patches as $\{P_{i,j}\}$, where $1 \leq i \leq n, 1 \leq j \leq g^2$.

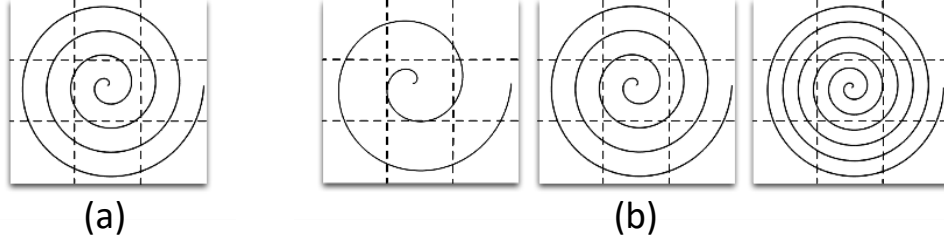


Figure 4.6: (a) single spiral, (b) multiple spirals.

Step 2. Selecting Patches Using k-means

Given a set of test images $\{X_i\}$, $1 \leq i \leq l$, the goal is to find a K -sized optimal subset $S \subset \{P_{i,j}\}$, so that the sum of distances of the fitted image to the original image is minimized, i.e.–

$$\arg \min_S \sum_{i=1}^l \sum_{j=1}^{g^2} \min_{s_k \in S} \left\{ d(x_{i,j}, s_k) \right\} \quad (4.4)$$

where, $s_k \in S$ denotes a selected patch, $x_{i,j}$ is the j^{th} patch-sized subregion of image X_i , and $d(\cdot)$ is a distance function that measures the fitness of a patch to a subregion of an image.

Finding an optimal subset of S for any arbitrary distance function is in general an NP-complete problem. Hence, I employ a clustering-based approach where I use k-means algorithm to cluster the population of patches into K clusters. I use the structural similarity metric (SSIM) (Wang et al., 2004) as the distance function. The intuition behind this approach is that, given the distance metric, as long as the distribution of patches in S resembles that of the unknown images, k-means will select a near optimal subset which minimizes Equation 4.4. More specifically, had I used patches from actual test images, k-means would be highly likely to find an optimal subset of patches that best fits the images.

Online Processing: Encoding and Refinement

During the online phase, every new image at first is converted to binary images based on a color threshold, and then it is encoded (using the selected patches from the offline phase) and is refined to improve the quality of encoding, prior to writing it into the beacons.

I use a simple, fixed-length encoding scheme to describe each input image as a sequence of patch identifiers. Non-overlapping, patch-sized regions of the input image are sequentially accessed, and for each region, the patch (within the set of selected patches) that has the maximum structural similarity to the region is noted, and its index is stored in a queue. When all the regions of the input image have been processed, the queue contains the encoded image. This simple encoding scheme can be further improved if I have prior knowledge of each patch's probability of occurrence. For example, by assigning short-length codes to more frequent patches, the overall bit length of the encoded image could be reduced. To keep things simple and generic, I do not employ such a variable-length encoding approach in this method.

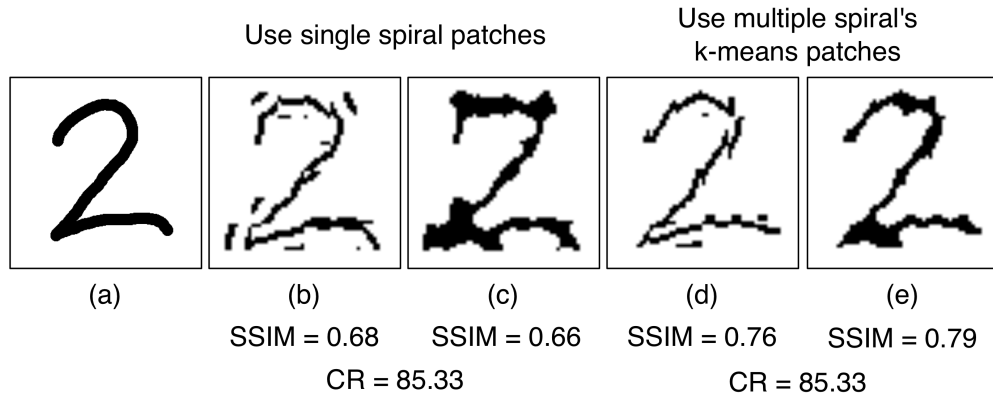


Figure 4.7: (a) original image; (b) result from patches generated from single spiral image; (c) result image from patches generated from single spiral image after morphology refinement; (d) result image from patches generated from multiple spiral images and k-means; (e) result from patches generated from multiple spiral images and k-means after morphology refinement.

After encoding an image, two standard mathematical morphology operations (Schalkoff, 1989) –dilation and erosion– are applied to enhance the quality of the resultant image. Examples of images before and after these refinements are shown in 4.7. The parameters of these operations are their ‘operator sizes’ (in pixels), which depend on the curve-width of the input image and the patch set.

In my experiments, I found that the resultant image has its best quality when the erosion/dilation operator sizes are 3 pixels.

Image Decoding

To decode an image, the broadcasted patch indices from all the beacons are received and serialized by the decoder application. The image is reconstructed by arranging the patches in the correct order as dictated by the index sequence. The refinement process is applied in the decoder as well.

4.5.2 Overview of the Color Image Beacon System

In a typical usage scenario of the proposed color image beacon system, a user at first takes pictures of an object with his smartphone's camera. A smartphone application analyzes the image (which may contain objects, portraits, scenes, shapes, signs, and/or text), identifies semantic regions on it, and processes each region differently to produce a compressed version that satisfies the beacon system's requirements such as the number of available beacon devices, maximum allowable loading time, and lifetime. The user is also shown an interactive preview of the image so that he can verify it, as well as relax/constrain the system requirements. Finally, when he is satisfied with the preview, the image is written into the image beacon system. The beacon system would then broadcast the image periodically over BLE, and any other smartphone user would be able to receive that broadcast and see the image on their phones.

The design choices I made in developing an image processing algorithm for the proposed image beacon system are as follows:

First, the custom image compression technique is designed to work for images taken with a smartphone. I assume that the phone has an on-board IMU in it. The final compressed image will be a color image with a lower resolution, such as 64×64 pixels.

Second, I make a reasonable assumption that the image to be compressed is linked to a real-world "thing" like a near-distance object, a road sign, or a building – which has one or more regions of interest that a person who took the picture wants to preserve with a higher priority than the rest.

By exploiting this, I design an image encoding technique that prioritizes foreground information preservation during image compression, so that the most important information in the image is delivered under a given size constraint.

Third, under a very tight budget for the final image size, any image compression algorithm would distort the original image – which is reflected in different ways such as lacking boundary details, increased noise, changed colors, or removal of texture. I introduce the concept of *adaptive encoding* that applies different encodings to different regions of an image based on the image content (e.g., a road sign vs. a t-shirt), image regions (foreground vs. background), and what a user would prefer to preserve (e.g., texture or true color). The proposed compression algorithm should employ an adaptive approach that applies the most suitable encoding technique for different image types and region types, so that an optimal compression strategy is chosen for a given image based on its content.

Fourth, since both capturing an image and writing the compressed version into the beacon devices involve the smartphone user in the loop, I provide an interactive user interface in order to guide the user in taking pictures and to preview and select the desired image under a given set of system constraints.

For a given set of user-defined beacon system requirements, the overall image processing and compression pipeline (Figure 4.8) consists of four basic stages: multiple view capture, depth estimation, depth-refined segmentation, and image compression. These steps are briefly described in this subsection, and elaborated in detail in the subsequent subsections.

Multiple View Capture

The proposed system requires a user to capture two or more views of an object – which helps at a later stage when the depth map is generated. Estimating pixel depths using a pair of images takes about 2 seconds on a mobile device. Because processing too many images would be time consuming, a careful selection of views (e.g. images having adequate overlaps) makes a difference.

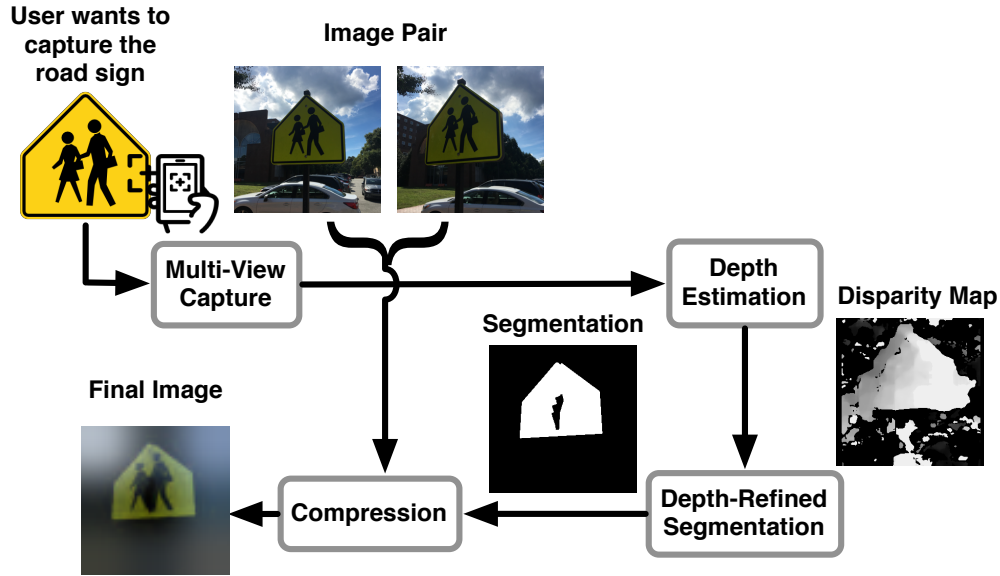


Figure 4.8: Image processing stages.

Figure 4.9 (a) shows two views of a mug that have enough overlap to create a depth map. To guide the user and to select the best pair of images for depth estimation, I leverage IMUs of the smartphone.

Depth Estimation

Depth of each pixel is estimated by finding and matching corresponding ‘feature points’ (e.g., corners and edges) in two or more images. The matched points are then used to generate the camera relative geometry, so that the depth of every pixel can be estimated. Figure 4.9 (b) shows two depth maps of the same image. The left one is the computed depth map, and the right one is thresholded to separate the background from the foreground pixels. However, due to lack of enough views, low resolution, and inaccuracies in the estimation, a depth map alone is not sufficient to segment semantic regions in an image.

Depth-Refined Segmentation

Like the depth map, color/texture-based image segmentation algorithms often fail to identify semantically different/similar regions in an image. For example, the left image in Figure 4.9 (c) is the result of applying marker-controlled watershed segmentation (Parvati et al., 2009) on the original image. When I overlay this with the depth map, I obtain a better segmentation, which

performs a much better job in isolating the mug from the rest. This step is inspired by (Nirjon and Stankovic, 2012) that used RGB and depth images from Kinect sensors. In my work, I use only images to estimate the depth (previous step) and then apply this step to get the final segmentation.

Image Compression

The image compression stage takes both an image (for texture and content information) and its segmentation map (for semantic region information), and produces the best quality image under the user-specific constraints of the beacon system. Until the resultant image size satisfies the system requirements, the algorithm gracefully degrades the quality of different semantic regions, starting from the least important one (e.g., the background).

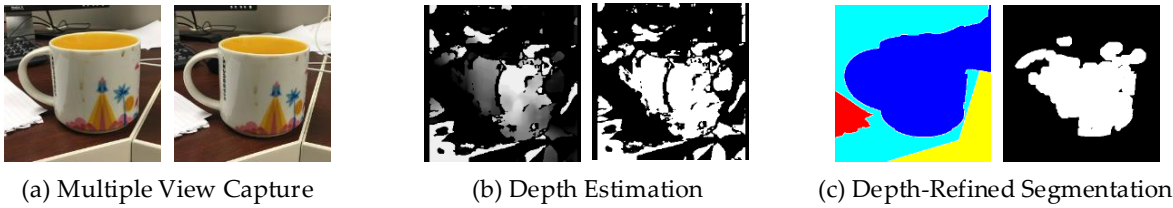


Figure 4.9: Multiple views of a scene are used to estimate the depth map. Combined with standard image segmentation, this can identify the pixels of an image that may be of more interest than the rest, e.g. a foreground object.

4.5.3 Multiview Capture and Depth Estimation

The first step of the proposed color image processing pipeline is to guide the user in capturing two or more views of an object of interest. Further down the pipeline, these images are used to estimate the depth information of each pixel, so that an image can be segmented into background and foreground regions, prior to applying appropriate region-specific encodings.

An alternative to using multiple views is to apply standard image segmentation algorithms (Parvati et al., 2009; Otsu, 1975; Long et al., 2015; Zheng et al., 2015) on a single image. These algorithms group adjacent pixels of an image based on information derived from pixel intensity in various ways. However, in order to obtain a sufficiently accurate segmentation for the proposed system, I require computationally expensive algorithms, such as convolutional/recurrent neural

networks (Zheng et al., 2015), which are not suitable for running on smartphones and do not produce results in real time.

Challenges with Multiple Views

Even though smartphones cannot run state-of-the-art image segmentation algorithms in real time, many other computer vision techniques, including depth estimation from two views, can be implemented on them. In order to get a sense of their real-time performance, I used OpenCV library for Android to compute the depth map for a pair of 400×400 pixel images. It took about two seconds for the algorithm to finish on a Nexus 5 phone. This gives us the lower limit for depth map computation, which may only happen if the user is well-trained and knowledgeable to know which views or camera poses would produce the most effective depth map.

A good pair of images is critical in generating a good depth map. However, the finding of a good pair of images depends on many factors. The most important of which is a suitable difference in view angles. It also depends on the distance between the object/scene and the smartphone. Furthermore, there are other factors such as lighting, texture and shapes of the image.

If a real-time depth estimation system could display the current depth map as the user takes images, it would be easier for him to generate a good pair of images for depth estimation. However, depth estimation does not run in true real time on most smartphones. In absence of a real-time feedback, a user has to take the trial-and-error approach, i.e., he has to take two images, look at the result after two seconds, and then repeat the entire process until the result looks good. This may lead to a very long time in just taking the right photos, and result in a non-smooth user experience.

IMU Assisted View Capture

To address this problem, I designed a method to make use of the inertial measurement unit (IMU) of the phone to shorten the image capture time and to improve the user experience. I adopt a machine-learning based approach.

In the offline training phase, I use a smartphone to capture video/image of multiple indoor and outdoor objects. I also keep record of the IMU values for each captured image. The IMU data consists of δx_r , δy_r and δz_r , which represent the components of the difference vector between the

rotation vectors between a pair of images. The IMU data also contains x_a , y_a and z_a components of acceleration when taking an image. After this, I run depth estimation for all pairs of images and estimate its accuracy by comparing the result against a manually generated ground-truth segmentation.

The segmentation accuracy is measured in terms of *intersection over union* (IoU), where *intersection* is defined as the area of intersections between foreground regions of two segmentations, and *union* is defined as the set of pixels either marked as foreground in the testing segmentation or in the ground truth segmentation. For a segmentation that is identical to the ground truth, IoU equals to 1.

Using IoU values as the variable Y , and the IMU data for the corresponding pair of images as the variable X , where $X = [\delta x_r, \delta y_r, \delta z_r, x_a, y_a, z_a]$, I create a data set for many pairs of images, and then train a regression tree model to learn the relationship between the change in IMU values between a pair of images and an expected quality of depth segmentation.

During the online phase, when the user is taking images for depth estimation, the trained regression tree model keeps track of current IMU readings and gives hints about if current view is a good choice, given the already taken photo(s).

Depth Estimation

In this step, the depth of each pixel is estimated from a pair of images. I use a standard algorithm (Hirschmuller, 2005) that at first estimates the ‘disparity’ between the corresponding points on two images, and then estimated depth from disparity. For example, if a point P_1 on the first image and a point P_2 on the second image correspond to the same point P on the actual physical world object, then $(P_1 - P_2)$ is called the disparity between them. Depth of a pixel is, in general, inversely proportional to its disparity. This is based on the principle that points in the scene that are closer to the camera will have larger disparity, and points that are very far away will be effectively at the same or very close location on both images. Hence, finding the depth map is essentially equivalent to finding the disparity map.

The disparity map is generated by the *stereo matching* algorithm described in (Hirschmuller, 2005). The goal of the algorithm is to find matching pixel blocks in a pair of images. This is also called the *correspondence problem* as it looks for the pixel coordinates on the image pair that corresponds to the same world point. The disparity map is computed based on the matching result. The disparity map is a gray-scale map, where the intensity directly corresponds to depth.

Depth-Refined Segmentation

The depth estimation algorithm groups pixels purely based on depth. It may not group pixel regions even if the regions share a common appearance pattern. As a result, even an accurate depth map tends to contain holes in foreground regions and isolated, incorrectly marked, small, bright regions in the background. Therefore, it is necessary to introduce other types of information derived from the image to obtain a cleaner and better segmentation. The refinement process is described as follows:

Step 1. Thresholding: At first, a binary segmentation of the depth map is obtained by applying a threshold on depth values.

Step 2. Combining with the Watershed Segmentation: I combine the depth-based segmentation map with another image segmentation method which groups pixels into several connected large regions and is computationally inexpensive to run on a mobile device. The watershed segmentation algorithm fulfills these requirements. However, the traditional watershed segmentation tends to generate an over-segmented result. Hence, I adopt the marker-controlled watershed segmentation, which uses mathematical morphology operations to pre-process an input image to avoid over-segmentation (Parvati et al., 2009).

At first, the input image is converted into grayscale. Then I run the marker-controlled watershed segmentation on the grayscale image. A successful segmentation contains more than one segmented region to separate foreground from background in the image. To determine which region belongs to the foreground, I apply a voting approach: the region that includes the highest number of common pixels with the foreground region in the depth-based segmentation map is labeled as foreground.

Here I denote the number of common pixels as N , where N is defined as:

$$N = \max_i C(W_i, D) \quad (4.5)$$

Here, i is the index of a region, and W_i is the corresponding region. D represents the foreground region in the depth-based segmentation. $C(,)$ computes the number of common pixels between two regions. If there is another region W_j for which, $C(W_j, D) \geq 0.8 \times N$, then it is also considered as foreground. This procedure iterates until no more regions can be added to the foreground. Having two result segmentation maps, I produce a final map by labeling pixels that are considered foreground in both maps as foreground.

Step 3. Final Refinement: Finally, I perform a pixel-level refinement process. I remove all connected foreground pixel regions with size less than 1000 pixels because regions of this size tend to be a background region. Then I apply two common mathematical morphology operations erosion and dilation, to clean out any remaining bright pixel islands in the background and to expand the foreground regions, respectively. Lastly, I enforce the accuracy of the segmentation boundary by combining the result with the labeled watershed foreground region.

The last stage of the proposed image processing pipeline is the image compression step. Using the segmentation information from the previous stage, this step encodes different segments of an image using different encoding techniques. The overall goal is to make sure that the resultant image fits the storage requirement of a beacon system, while making sure that the foreground regions are the least affected by during the compression process.

I propose three encoding options for image compression – *discrete cosine transform* (DCT) and coefficient reduction, *wavelet transform* with coefficient reduction, and *foreground texture triangularization*. All three are applied on the input image and finally the one that produces the best quality image is chosen as the output. The effect of applying different encodings is illustrated in Figure 4.11.

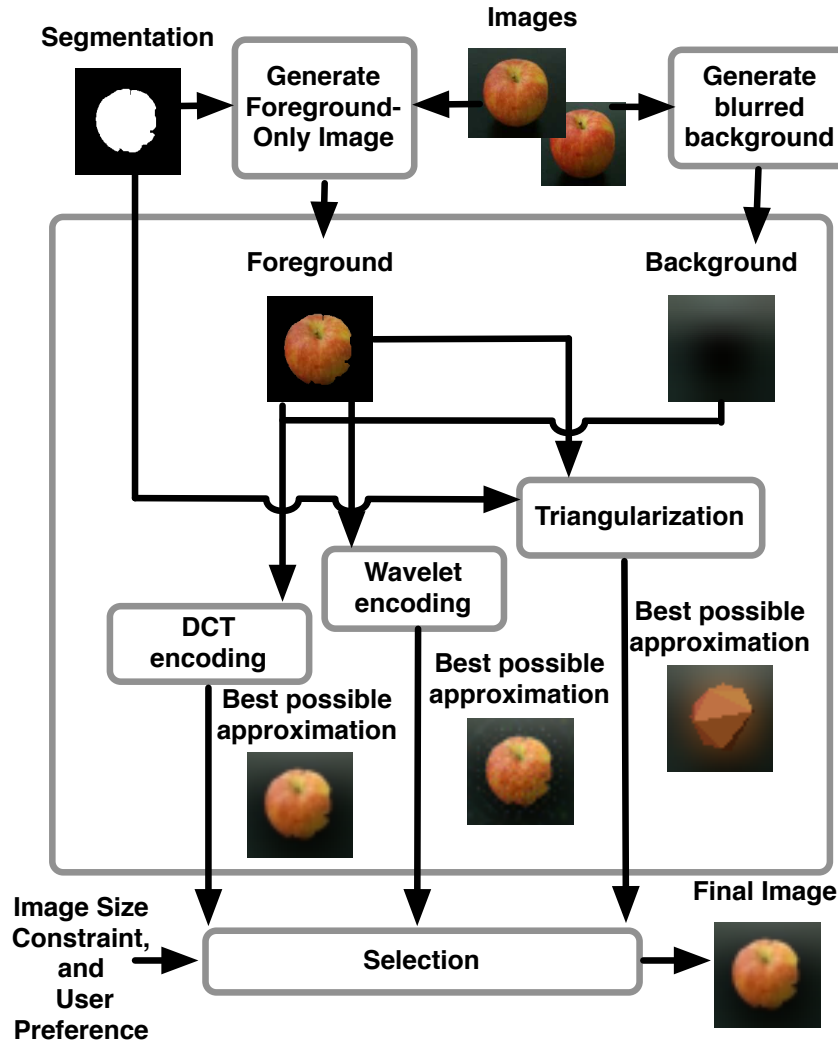


Figure 4.10: Image compression details.

4.5.4 Color Image Encoding

Discrete Cosine Transform Encoding

Discrete Cosine Transform (DCT) is a widely adopted image encoding technique. I integrate DCT encoding into my compression system as a baseline encoding option. The benefit of using DCT is that – by reducing low frequency coefficients of an image (in the DCT transformed space), the resultant compressed image’s *appearance* details is removed first, while its *global shape* is preserved. This is useful in the usage scenario when a user wants to preserve the shape of an object in the image more than its detailed appearance.

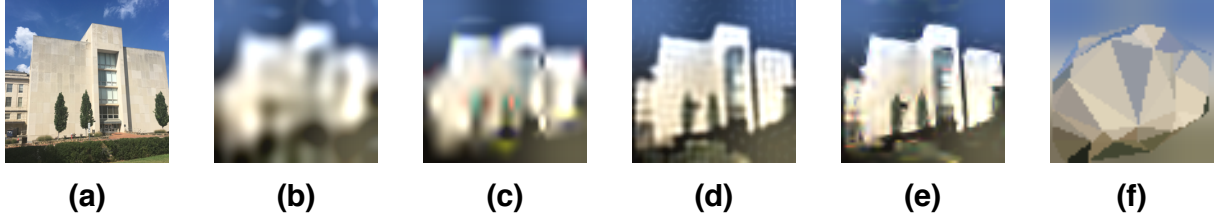


Figure 4.11: 64×64 resolution building image compressed in high/low quality settings using my customized DCT/Wavelet/Triangle encoding: (a) Original image, (b) DCT 342 bytes, (c) Wavelet high 360 bytes, (d) DCT 1114 bytes, (e) Wavelet 1098 bytes, and (f) triangularization 366 bytes. For a similar compressed image size, DCT preserves less details than Wavelet method. But for low quality settings (about 350 bytes), Wavelet-encoded images have strange color block defects. Triangularization failed to preserve the information in the original image.

At the beginning of encoding, I generate a foreground image by using the segmentation map to set the background pixels' intensity of the input image to zero. The image is then down sampled to 64×64 pixels. To further reduce the size, I make use of the fact that the quality of an image depends more on its brightness information than its color information. During the encoding, at first, an image is transformed into the YUV space, where Y represents the brightness information and U/V represents the image color information. I further down sample U and V channels into 32×32 pixels, while keeping the resolution of Y channel intact. Then DCT is applied on all three channels, followed by a data-reduction step that sets high frequency components to zero. Finally, the resultant frequency coefficients are compressed using gzip. The data is sent to the beacon along with the blurred background image, which is also encoded using DCT. The size of a DCT compressed image, S_{DCT} can be expressed as:

$$S_{\text{DCT}} = B(g(d)) + B(g(b)) \quad (4.6)$$

Where, $B()$ denotes the bit length, $g()$ denotes the gzip encoded data size, d is the DCT transformed (reduced coefficient version) foreground information, and b is the coefficient of the DCT transformed (blurry) background image data.

At the receiving end, a broadcast image is recovered by superimposing the foreground image and the background image. Note that, the background image needs to have the foreground pixel intensities set to zero, before it is down sampled. This is done to make sure that the foreground pixel intensities are not added twice.

A limitation of this above approach is that, for an image with a uniform dark background and a foreground having more details, DCT may yield a ringing artifact close to the sharp edges, especially in a low quality setting. This problem can be addressed by switching to using wavelet.

Wavelet Encoding

Wavelet is the second image encoding method that I integrate in the adaptive image encoding process. When compared to DCT, wavelet tends to better handle images whose backgrounds have a uniform intensity. Similar to DCT, the best information reduction parameters are sent to the wavelet encoding module in order to generate the highest quality image under a given storage limit. I adopt global thresholding of the wavelet coefficients and Huffman encoding, based on the method described in (Said and Pearlman, 1996).

Similar to DCT, prior to encoding the foreground image, its background pixel intensities are set to zero to obtain the wavelet data. The down sampled background image (with zero intensity foreground) is also sent along with the wavelet data. At the receiving end, wavelet coefficients are inverse-transformed to generate the foreground image and then superimposed on the background image to render the final image.

The size of a wavelet compressed image S_W is as follows:

$$S_W = B(H(w)) + B(g(b)) \quad (4.7)$$

where, $H()$ denotes the Huffman encoded data, and w denotes the reduced wavelet coefficients on the wavelet transformed foreground image. All other symbols carry the same meaning as discussed in the previous section.

The weakness of wavelet encoding is that, for limited storage requirements (200 bytes), a wavelet encoded image may have unrealistic texture patches after decoding.

Triangularization-Based Encoding

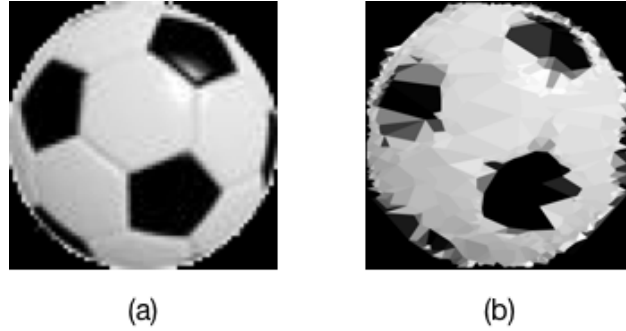


Figure 4.12: Original image and the triangularization-compressed image.

Both DCT and wavelet encodings blend the geometric information and texture information of the foreground image. However, for some cases, an accurate texture information and a fine-grained boundary representation are not necessary. For example, a “soccer” image’s foreground texture and shape could be decoupled. For a viewer to understand that the image is a “soccer”, a repeated patch of a soccer’s surface texture along with an approximate “soccer” shape information would suffice (Figure 4.12). Both DCT and wavelet would encode too much redundant information for such an image. To address this, I designed a triangularization-based image encoding method, which consists of three stages: triangularization, triangle reduction, and color/texture filling. These are illustrated in Figure 4.13, and are described as follows:

Step 1. Triangularization: Given an input image with the foreground/background segmentation, the first step is to generate a binary *boundary map* from the segmentation map, in which, only the pixels on the segmentation boundary have an intensity of 1. A Delaunay triangularization (De Berg et al., 2000) is performed on the boundary map. The parameters are chosen to produce a high number of triangles to capture boundary details.

Step 2. Triangle Reduction: Having a set of triangles, the next step is to reduce the number of vertices, iteratively, one vertex at a time. For this, I compute the sum of distances for every

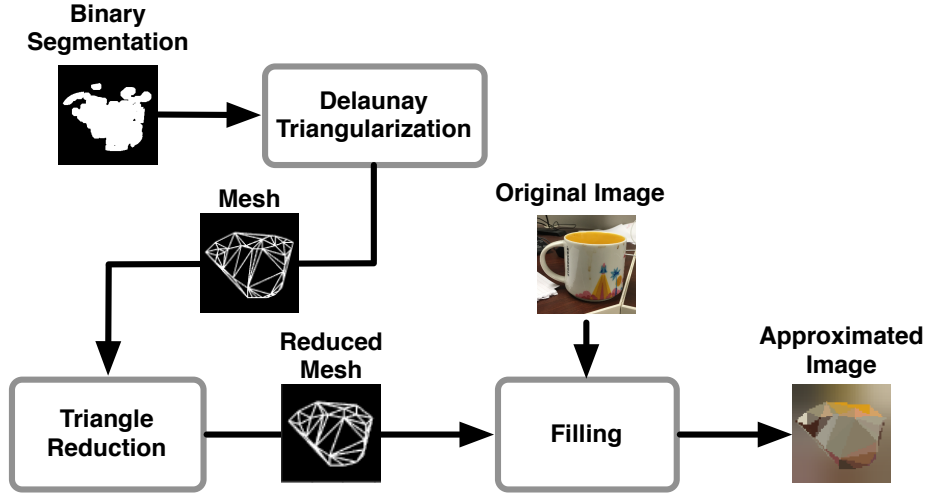


Figure 4.13: The process of Triangularization-based encoding.

vertex from its nearest 3 neighbors, and then remove the one with the minimum sum of distances. The intuition behind this process is that a region of vertices group together densely because of the non-smooth boundary in the boundary image. Since the goal of removing vertices is to reduce the details and preserve the general shape information, I should pick a vertex from dense regions.

Step 3. Color/Texture Filling: I provide two options for filling a triangle – with texture or with a single color. For texture, I choose to fill all triangles with a limited set of textures which are derived from regions surrounded by each triangle. To reduce the number of textures, I take an average of textures from different triangles. Fig. 4.14 shows the process. For each triangle, I transform it to a fixed-size triangle by an affine transform, and then compute one or two average textures for all triangles. For the case of two textures, I apply k-means algorithm.

The size of the compressed image using triangularization encoding with color filling S_{Tc} , and with texture filling S_{Tt} are as follows:

$$S_{Tc} = B(g([v, c, f]) + B(g(b)) \quad (4.8)$$

$$S_{Tt} = B(g([v, c, i]) + B(g(t)) + B(g(b)) \quad (4.9)$$

Here, v denotes the location of the vertices, c denotes the connectivity list, f denotes RGB color values, i denotes the texture index, and t denotes the reduced DCT coefficients on the texture patch transformed using DCT. All other symbols carry the same definition as in previous sections.

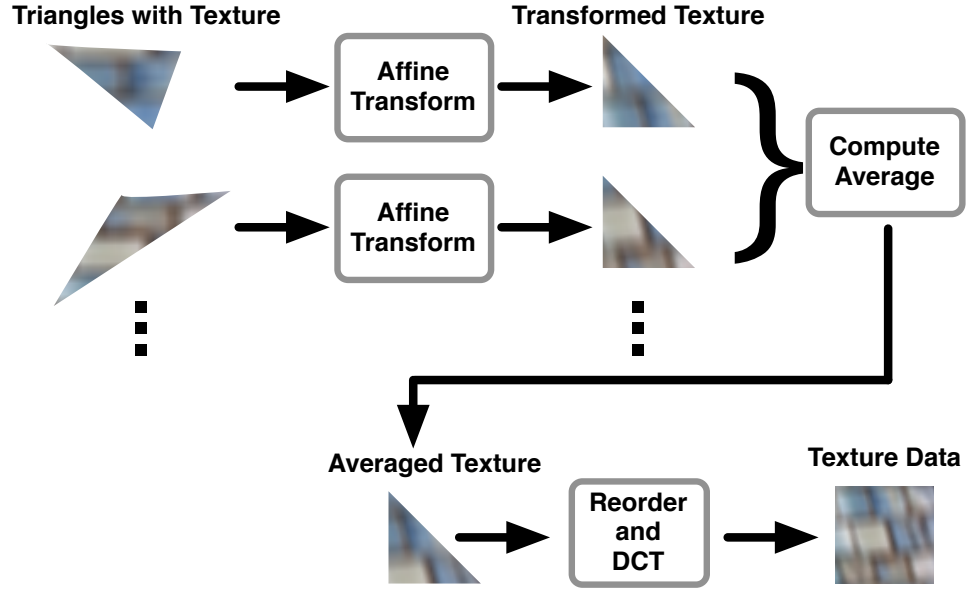


Figure 4.14: Triangle texture averaging process.

4.6 Empirical Evaluation

4.6.1 Image Beacon Implementation Details

In all of my experiments in this chapter, I have used Estimote model Rev.D3.4 and model REV.F2.3 Radio Beacons (Estimote, 2017) having a 32-bit ARM Cortex M0 CPU, 256 KB flash memory, 4 dBm output power, 40 channels (3 for advertising), and 2.4–2.4835 GHz operating frequency. I vary the BLE broadcast interval for a beacon between 100 ms to 2,000 ms. However, an encoded image (broadcasted from multiple beacons) reaches a user’s device in less than 1 second. The transmission power is set to -12 dBm, which limits the range of each beacon to about 30 meters. The image writing and reading application runs on two platforms: an iPhone 5s having an ARM v8 based dual-core 1.3 GHz Cyclone CPU, Apple A7 chipset, 1 GB DDR3 RAM, BLE v4.0, and

runs iOS 9.2, and a Nexus 5 smartphone having a 2.26GHz quad-core Qualcomm Snapdragon 800 processor, 2 GB RAM, BLE v4, and runs Android 6.

I mimic BLE 5.0 broadcast packets by a set of rolling BLE 4.0 packets. The rolling mechanism is implemented by configuring the Estimote Location beacons to broadcast customized advertising packets. The customized data is received from an Android compatible LightBlue Bean device (LBB, 2017) via the beacon's GPIO, configured as an UART interface.

4.6.2 Binary Image Beacon System Evaluation

In this subsection, I describe a series of empirical evaluations on the binary image beacon system. At first, the patch-based binary image compression approach is compared with JPEG encoding. Then I describe a set of results that quantifies the tradeoffs between the device lifetime and the image quality, when the type of images, number of beacons, patch function generation method, number of patches, and the grid or patch size are varied.

Experimental Setup

I use three types of images in my experiments: images containing alpha-numeric characters, basic shapes, and arbitrary binary images. Examples of these images are shown in Figure 4.15. Some of these images are directly drawn on the phone by a user (e.g., letters, numbers, and free hand drawings), while some are real pictures that have been converted to binary format by my writer application. All images are down-sampled to 64×64 pixels prior to writing.

The two main metrics that are used in the experiments are structural similarity (SSIM) scores, and device lifetime in months. I measure these two under different conditions and show their tradeoffs. The structural similarity scores are used to measure the quality of the produced images when compared to the original ones. The device lifetime is estimated from its relation to a beacon's transmission frequency. Before each experiment, I program the beacons to set a transmission frequency and use the corresponding estimated device lifetime (as reported by the Estimote beacon API) in my experiments.

Comparison with JPEG

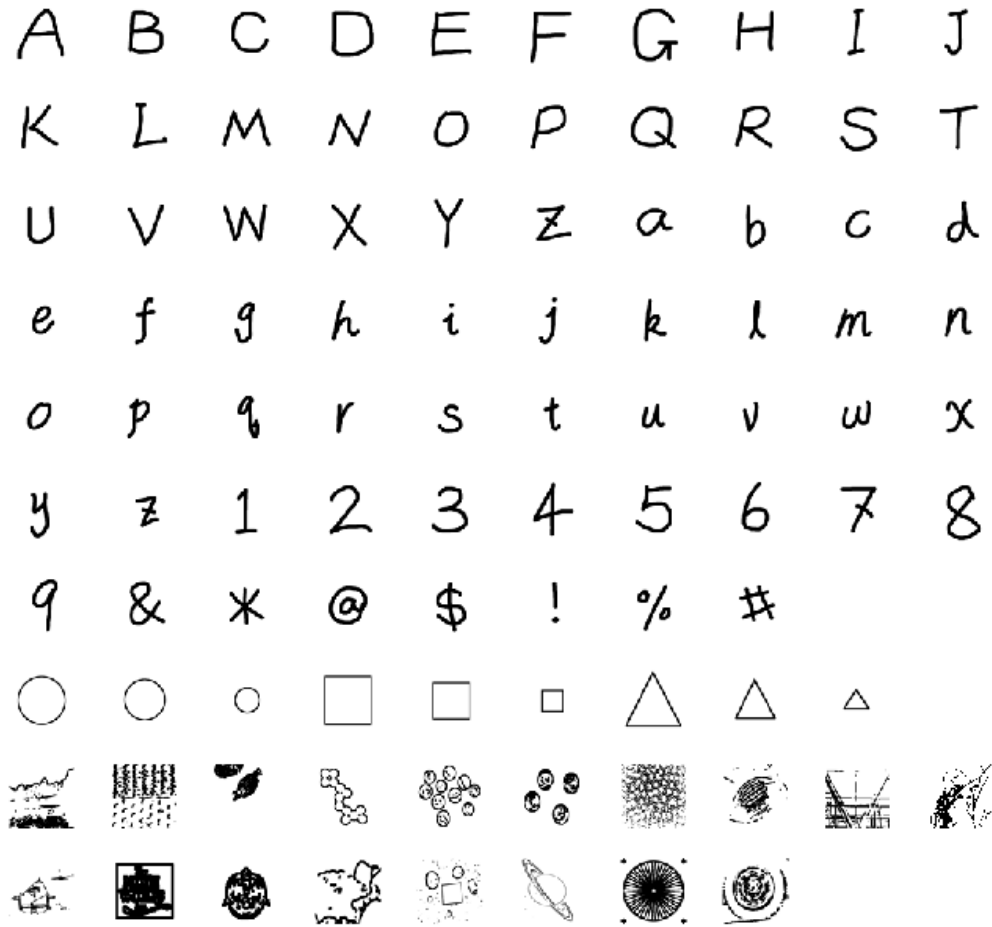


Figure 4.15: Test images used in the empirical evaluation.

In this experiment, I compare the proposed patch-based image encoding algorithm with JPEG. I pick the image of a handwritten ‘2’ as the test image. In Figure 4.16, I plot the quality of encoded images obtained from three methods – two versions of the k-means patch-based approach (4 and 8 pixels per patch) and JPEG, for various sizes of images. The result suggests that– JPEG generates better quality images in general, however, it also requires larger sized images. For a 64×64 binary image, JPEG encoded image is about 464 bytes, even with the lowest quality settings. Compared to that, my patch-based method generates images in smaller sizes (30–200 bytes), making it possible to store an image inside a small number of beacon devices.

Besides JPEG, I also studied several other image compression techniques (Mohamed and Fahmy, 1995; Zahir et al., 2007; Zahir and Naqvi, 2005). But none of these yield suitably small sized images.

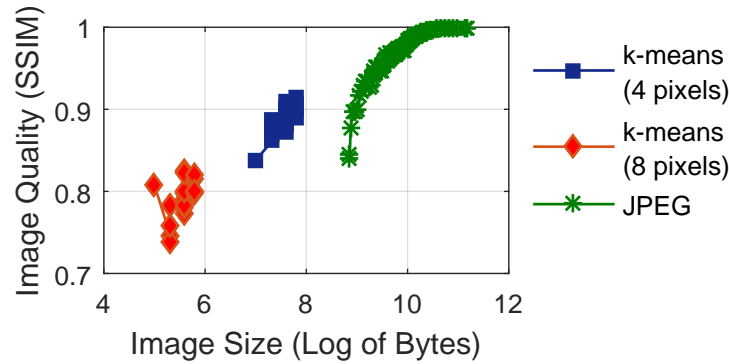


Figure 4.16: Image quality versus image size for different encoding methods.

Effect of Changing on Different Parameters

1. Effect on Image Type. I conduct an experiment to measure how the patch-based image encoding method's performance (in terms of quality) changes when the type of input images is varied. The three types of images I use include hand-written alpha-numeric characters, basic geometric shapes, and arbitrary binary images containing complicated shapes and curves. For each type, I compute the average image quality for a given lifetime. I use three beacons in this experiment to store and read images.

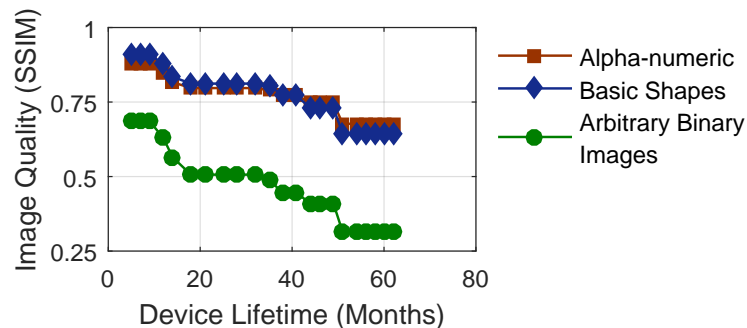


Figure 4.17: Image quality versus beacon battery life for different images being approximated.

Figure 4.17 shows that the image quality drops when the beacon’s power consumption is reduced to target a prolonged device lifetime. A longer lifetime limits the broadcasting frequency, and as a result, the amount of data the system can transmit within a fixed period (1 second in my setting) is reduced—resulting in a poor quality encoding that uses less amount of bytes. This can however be fixed by allowing a longer image transmission delay (i.e. more than 1 second wait-time for the reader application). I also notice that the basic geometric shapes and alpha-numeric characters achieve very high structural similarity scores under all power settings. This happens since their sub images are very similar to the patches. However, the system does not perform its best when tested with arbitrary, complex binary images (specially ones with a lot of dark regions).

Effect of Number of Beacon Devices

In this experiment, I investigate the impact of the number of beacons on image quality. Recall that I have used the the Eddystone URL beacon packet format to store the custom image data where each beacon can contain at most 16 bytes of image data. Hence, the more beacons I use, the more bytes I have available to store the same image. Figure 4.18 shows that as the number of beacons is increased from 1 to 6, the quality of produced images also increases from 0.72 to 0.84.

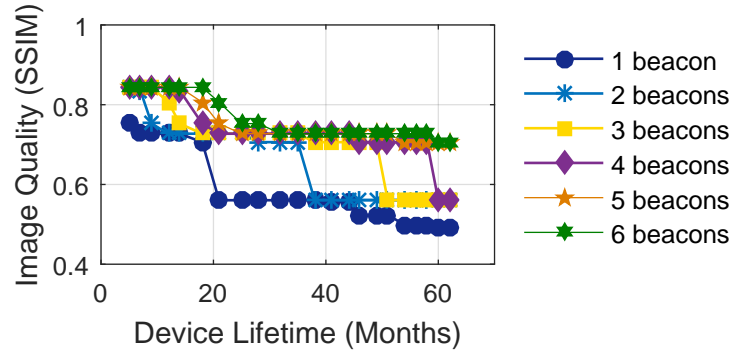


Figure 4.18: Image quality versus device lifetime for various number of beacons.

Effect of Patch Function Type

In this experiment I compare three patch generation methods. The first one is the proposed k -means based approach that considers multiple spirals. The second one uses only a single spiral, and the third one generates patches from a standard test dataset for hand-written characters

MNIST (LeCun et al., 1998). To generate patches, I up-scale the images to 64×64 , adjust intensity, and dilate the background. Then I divide each image into patches and run k-means on the patches.

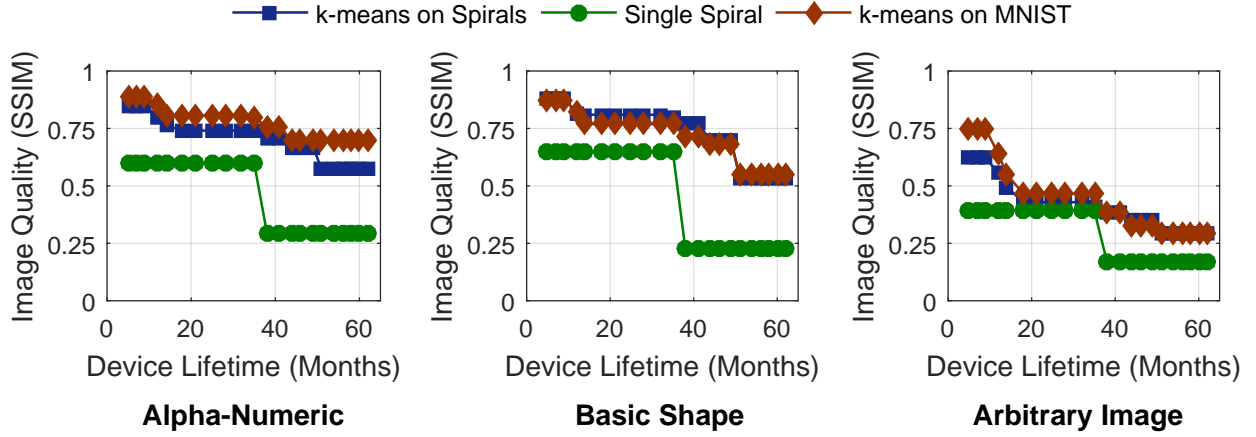


Figure 4.19: Image quality versus lifetime for different patch generation methods.

From Figure 4.19 I observe that patches generated from MNIST dataset performs the best for alpha-numeric characters. This is expected since MNIST is designed specifically for handwritten digits. The multiple spiral-based method performs better for basic shapes, and both algorithms perform similar when tested with arbitrary images. This shows that having prior knowledge helps, but even if I do not have it, my method performs reasonably well.

Effect of Patch Set's Size

An important parameter of the beacon system is the size of the patch set, which is same as the number of clusters k in k-means clustering. A larger patch set is more capable in representing an input image, but requires more bits to encode the image. This leads to a higher broadcasting frequency (given the 1 second bound on the maximum transmission latency), and more power consumption. So, it is important to find a good value for k . Figure 4.20 shows image quality versus device lifetime for various k . The plot shows that $k = 64$ almost always outperforms others as expected. The other two values of k also perform reasonably well (0.65–0.7 when expected lifetime is between 20–40 months). Note that $k = 128$ is not applicable to my setup (i.e., 3 beacons and 1 second latency) since this would require an excessive amount of space to store an image.

Effect of Grid Size

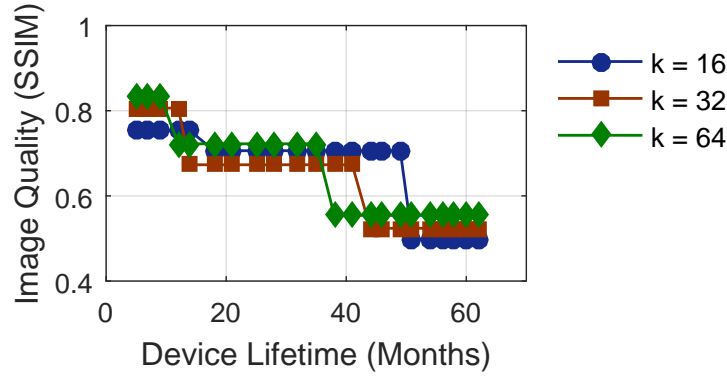


Figure 4.20: Image quality versus device lifetime for various patch set sizes. Note that the patch set with size $k=16$ is not simply a subset of the patch set of size $k=32$. They are individual k -means clustering results with different k s.

Similar to the patch set's size, the grid size used to divide a spiral image to produce different sizes of patches also has impacts on the image quality and the encoded image's size. Using smaller patches results in high resolution image encoding, but the size of the encoded image will be larger. For example, a grid size of 1 pixel results in an exact replica of the original image. Figure 4.21 shows the system's performance for different grid sizes. Using 3 beacons and a grid size of 4 pixels, the system cannot encode images when the desired device lifetime is more than 20 months, but produces best quality images for shorter expected lifetimes. With a grid size of 8 pixels, the system would last for about 50 months and will perform consistently well to produce images having about 0.75 structural similarity scores. Using larger grids (16 and 32 pixels) results in even longer device lifetime, but the quality of images degrades to 0.5.

4.6.3 Color Image Beacon System Evaluation

In this section, I describe a series of empirical evaluations on the color image beacon system. First, I evaluate the performance of IMU-guided multiple view capture and depth-refine segmentation. Then my image compression approach is compared with JPEG encoding. After that, I describe a set of results that quantifies the trade-offs between the beacon system lifetime and image quality,

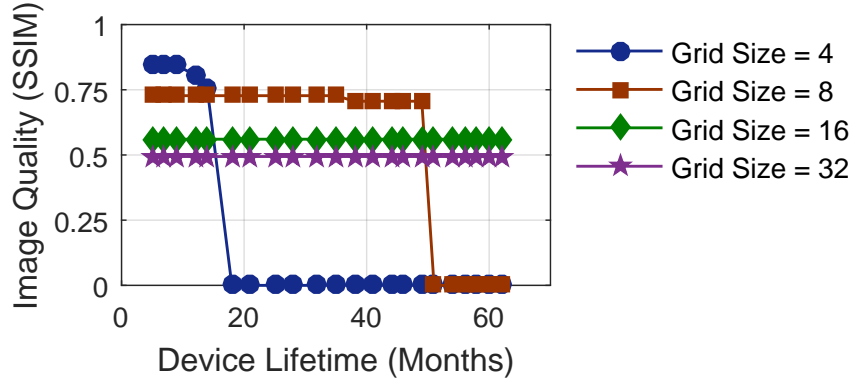


Figure 4.21: Image quality versus device lifetime for various grid sizes per patch.

when the image type and number of beacons are varied. I also perform a full system evaluation involving real users, which is described in Section 4.7.

Experimental Setup

I use four types of images in my experiments: images containing road signs, common indoor and outdoor objects, and buildings. Examples of these images are shown in Figure 4.22. Indoor object images are taken from a 50cm - 150cm distance. Outdoor objects and signs are taken from 2m to 5m distance. Building images are taken from far. All images are cropped into square shaped images, and are down-sampled to 288×288 pixels prior to compression for a fast disparity map and watershed result computation. Each image is down sampled to 64×64 pixels before writing into the beacon system.

The two main metrics that are used in the experiments are structural similarity (SSIM) scores, and device lifetime in months. I measure these two under different conditions and show their tradeoffs. The device lifetime is estimated in the same way as described in the last subsection.

Performance of IMU-Guided Multiple View Capture

I evaluate the accuracy and time to capture multiple views with and without the guidance of IMU. Recall that, the regression tree model takes IMU data and predicts if the current smartphone positioning is good for taking an image for depth estimation, given an already taken image. In the evaluation, for each test, a set of images and their corresponding IMU readings are recorded. If

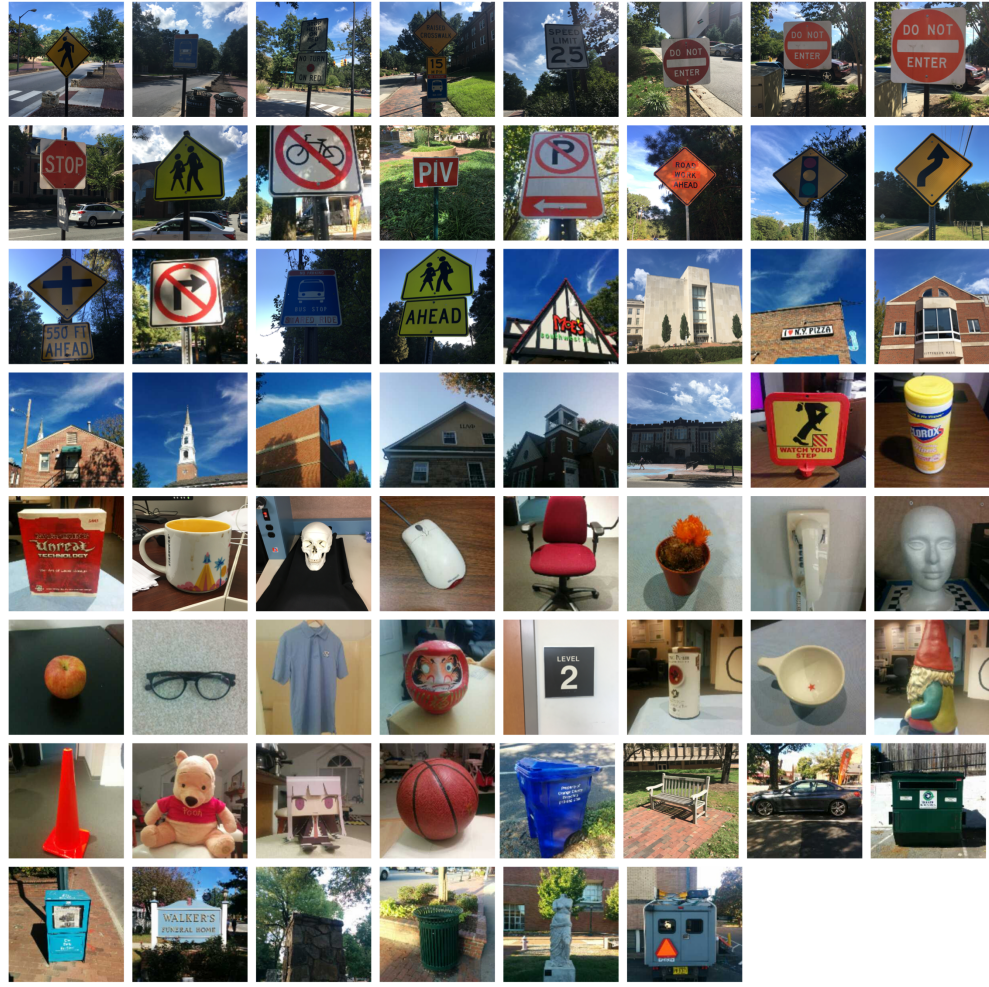


Figure 4.22: Test images used in the empirical evaluation.

the predicted image produces the best segmentation result when compared to other images in the set, I record this test result as a ‘hit’, otherwise, a ‘miss’. The accuracy of prediction is determined by the ratio of ‘hits’ to total tests. A total of four tests are performed separately for indoor and outdoor objects. For indoors, there is 1 miss over 4 tests, whereas for outdoors, the model correctly identifies the best image pair for all tests. The model for outdoor objects is more robust since the smartphone’s rotation angle shows smaller variations and usually the phone is held vertical. Indoors, users often take pictures of an object from above, from below, or from the same height, which results in a large variation in angles.

Figure 4.23 shows the expected time for finding a good pair of images with and without the assistance from the IMU. I observe that the average time to obtain the depth map from a pair of images takes about 2 seconds, which includes the time to focus, raw image processing, and disparity map computation. Since the IMU-guided system predicts a good pair in real time, it significantly decreases the time from 8 seconds to 2-2.5 seconds.

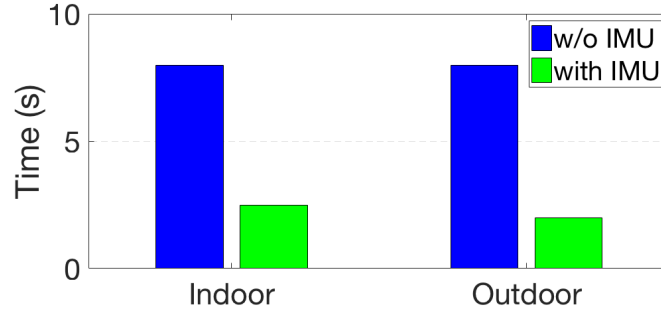


Figure 4.23: Performance of IMU-guided view capture.

Performance of Depth-Refined Segmentation

I compare my depth-refined segmentation method against a segmentation approach that is purely based on the depth map. I use intersection over union (IoU) as the evaluation metric. I tested my segmentation method on 20 images of four types: road signs, buildings, indoor and outdoor objects. For each set, I manually generate the ground truth segmentation. Figure 4.24 shows the result. I observe that my segmentation technique, which combines depth information and watershed segmentation information, outperforms depth-only approach by up to 26%. My method performs better in cases where the object and the background has larger difference in depths, e.g., signs and indoor objects.

Comparison with JPEG

In this experiment, I compare my proposed color image encoding method with JPEG. For an in-depth illustration, I use the picture of an apple as our test image. I compare four different options of my adaptive image encoding methods in the comparison, i.e. DCT-based, wavelet-based, triangularization with color filling, and texture filling. I measure the quality of a compressed image using Structure Similarity (SSIM) (Wang et al., 2004). A SSIM score ranges from 0 to 1, and

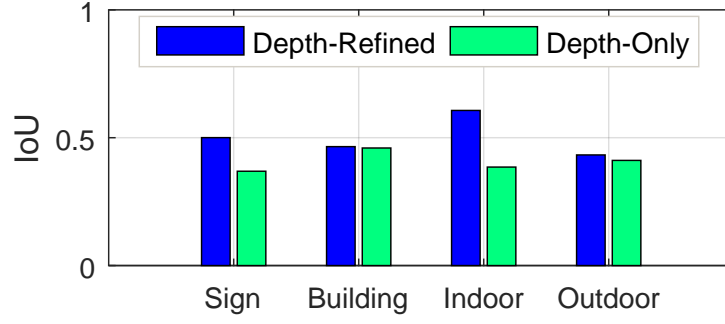


Figure 4.24: Performance of depth-refined segmentation.

two identical images have the best SSIM score of 1. I compute the SSIM value between every compressed image and the original image. I plot the SSIM versus compressed image size in bytes in Figure 4.25.

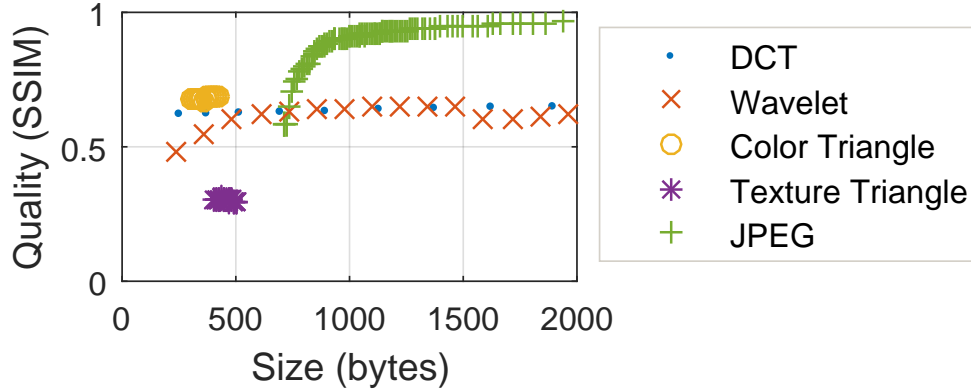


Figure 4.25: Image quality versus image size for different encoding methods.

The result suggests that JPEG is able to generate a compressed image with a higher quality (SSIM close to 1). However, such a high quality image has size larger than 2 KB. When JPEG is set to compress an image into lower than 1KB, the image quality drops sharply. On the other hand, my image compression method is based on foreground/background separation. The background in a compressed image is always blurred. This makes it impossible for my method to get a high SSIM score larger than 0.8. But my method is able to allocate bits more efficiently under a tight space constraint. This makes my method (when using wavelet/DCT encoding) outperform JPEG when the

compressed image size is about 800 bytes. The plot also shows that my method can compress an image into as low as 240 bytes (left most point on the wavelet method curve), which is impossible with JPEG, even in its lowest quality setting.

Triangularization with color filling option performs the best for this test case, and the compressed image size stays less than 500 bytes, while triangularization with texture filling encoding fails to generate a good compressed image.

Besides JPEG, I also studied several other image compression techniques (Mohamed and Fahmy, 1995)(Zahir et al., 2007)(Zahir and Naqvi, 2005). But none of these yield suitably small sized images.

Effect of Image Type

In this experiment, I test how the color image encoding method's performance changes as I vary the type of input images. I consider four categories of images, i.e., road signs, buildings, indoor objects, and outdoor objects. For each image, my algorithm selects the best compressed image from different versions of the adaptive encoded images based on SSIM score. Then I compute the average image quality for a given lifetime for each category of images. I simulate a BLE 5.0 beacon in this experiment to store and read the images. I limit my system to deliver the image data within 0.5 second from one beacon.

The result shown in Figure 4.26 suggests that, on average, indoor object images achieves the best quality over all types. The building images are best approximated if the beacon system broadcasts packets at a higher frequency, sacrificing the system life time. The road sign images have relatively lower quality than other images, but they tend to last for up to 28 months on a single battery.

Effect of Number of Beacon Devices

I explore the impact of the number of beacons on image quality. Since each beacon in a system of beacons can be set to broadcast different parts of an image, the more beacons I have, the better quality images I can generate by utilizing the additional space. This is based on the assumption that the image loading time and the device lifetime requirements are fixed.

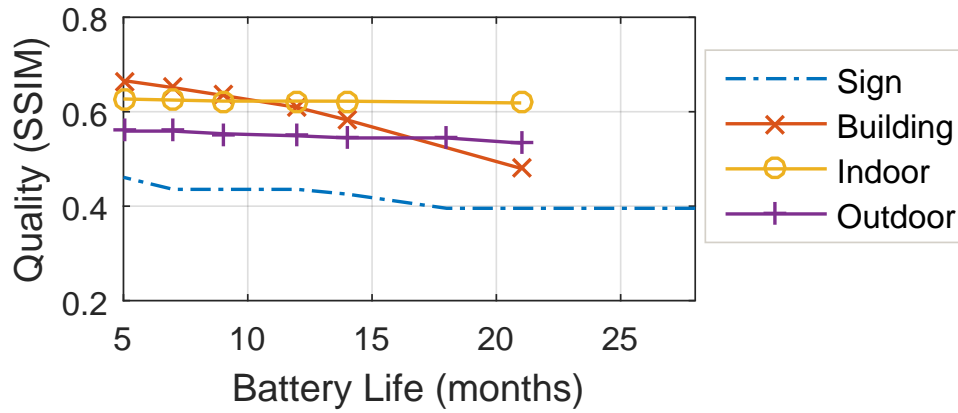


Figure 4.26: Image quality versus beacon battery life for different image types.

In this experiment, I vary the number of beacons and record the SSIM of the best quality compressed image generated by my system. The best image is chosen from the adaptive-encoding results with the highest SSIM score.

I plot the SSIM scores for various expected system lifetime in Figure 4.27. The experiment results suggest that as the number of beacons is increased from 1 to 3, for the beacon system to have an expected lifetime of e.g., 32 months, the quality of produced images also increases from 0.45 to 0.69.

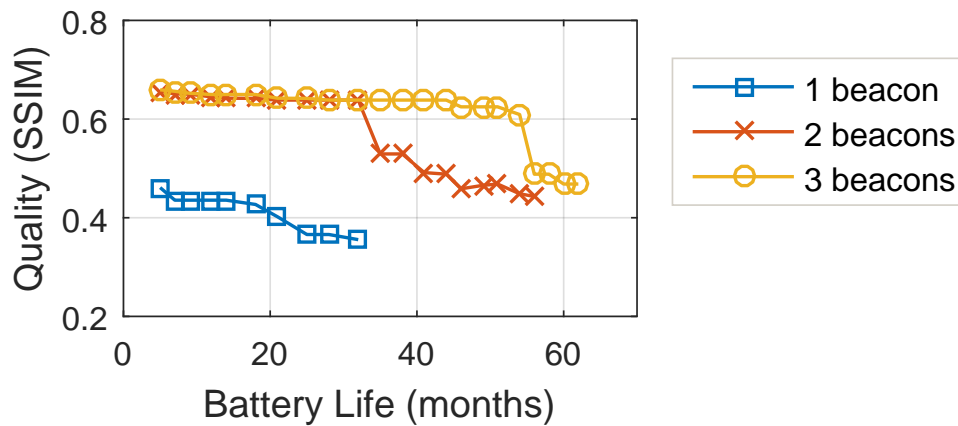


Figure 4.27: Image quality versus device lifetime for various number of beacons.

4.7 Real Deployment

I implemented the binary image beacon system and the color image system. I performed user studies of the systems that involve a set of participants. There are two types of experiments: “write-read-recognize” and “navigation in the building”.

4.7.1 Write-Read-Recognize

Binary Image Beacon Real Deployment

I deploy the binary image beacon system in a scenario where a group of 12 participants uses the smartphone application to draw, preview, write into, and read from a set of image beacons. All the participants are graduate students in my department.

I ask every participant to use the ‘previewer’ app to draw images on the phone using its touchscreen. These images are converted to binary and saved into the phone’s internal storage. Each participant draws random images containing digits, letters, numbers, and recognizable symbols and shapes. They also label their drawings with meaningful tags. These images are then encoded so that they can be stored in 4 beacons, their transmission delay is 1 second, and the lifetime of the system is over two years. A subset of these encoded images are then shown to other participants who are asked to recognize them and then provide subjective scores for them. The score denotes how a participant feels about the overall quality of the shown image and his difficulty in recognizing the symbols and objects in that image. The score ranges from 1 to 10, where ‘1’ stands for the lowest perceived quality and the hardest to recognize. Seven of these images are scored by all participants.

I gather user scores and normalize them to [0, 1]. To compare these subjective scores with structural similarity (SSIM) based objective scores, I computed the mean of the user scores for each image and show them in the same plot with their SSIM scores. SSIM scores are also scaled to [0, 1]. Figure 4.28 shows that the trend in the SSIM curve and the averaged subjective quality scores are about the same except for the last image of ‘pi’. The averaged subjective scores for different images vary more than the SSIM scores. The reason for this is that the participants tend to give a

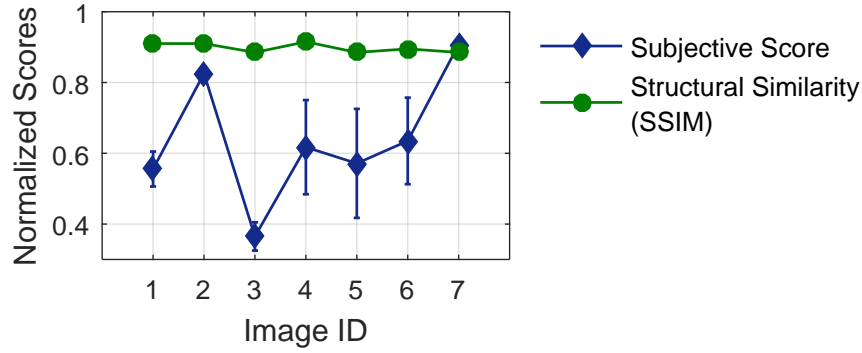


Figure 4.28: Subjective and SSIM scores for each image.

relatively low score (around 1 to 3) for the images they fail to recognize. This has happened for the ‘@’ symbol. Many participants tend to recognize this as a compressed version of double ‘O’. The ‘pi’ image has a very high average subjective score but a relatively low SSIM score. This ‘outlier’ may be due to the fact that unlike other images, this image takes the whole image display area, and therefore, it is easier for human to recognize.

Color Image Beacon Real Deployment

I deploy the color image beacon system in an indoor environment to evaluate the ability of color image beacons in delivering visual information of various real-world objects. An approximated image stored in the beacon system contains an object’s shape as well as its texture information. The goal of this deployment experiment was to understand how an image beacon system delivers these two kinds of information of an image.

To have a subjective measure of the performance of an image beacon system, I conduct a user study involving 20 participants. Each participant is given a smartphone that receives image broadcasts from four different beacons placed inside a room. The goal of the user is to identify the objects in all four images, and then locate them in the room.

The four broadcasted images are of an apple, a chair, a text book and a computer mouse. For each image, I compress it under three size constraints: 256 bytes, 512 bytes, and 768 bytes, and obtain the best quality image produced by my compression algorithm under the constraint. The images compressed in three levels along with the original image are shown in Figure 4.29. I choose

these sizes to mimic 1, 2, and 3 BLE 5.0 beacons, respectively; but my actual implementation used a rolling mechanism with BLE 4.0 (connected with an Arduino), as described earlier. Each user is progressively shown a better quality image until he is able to identify the object in the room.

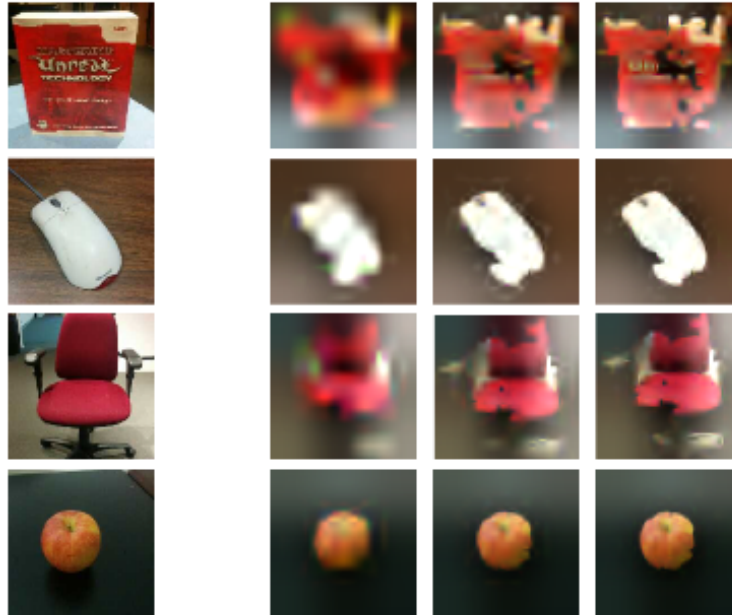


Figure 4.29: Test images compressed in three quality levels.

I make sure that there are at least 3 objects in the room that are similar to the one that the user is looking at on his phone. To test if an object's texture details are preserved, for the apple test case, I put another apple having a green/red mixed color and an orange next to it. For the book test case, I put two other similar sized books next to it that have different covers. To test if an object's shape details are preserved, for the chair test case, I add another two chairs of the same color. For the mouse test case, I add an iPhone and a mac mouse. Figure 4.30 shows photos of the objects along with the objects that I added to introduce confusion.

The experiment results are shown in Figure 4.31. For each size limit, I plot the number of correct guesses by the participants for various categories of images. I observe that, as expected, when the image size limit is larger, participants tend to perform better. Even with the lowest size, at about 50% images were always guessed correctly by the participants. Therefore, with 3 or more

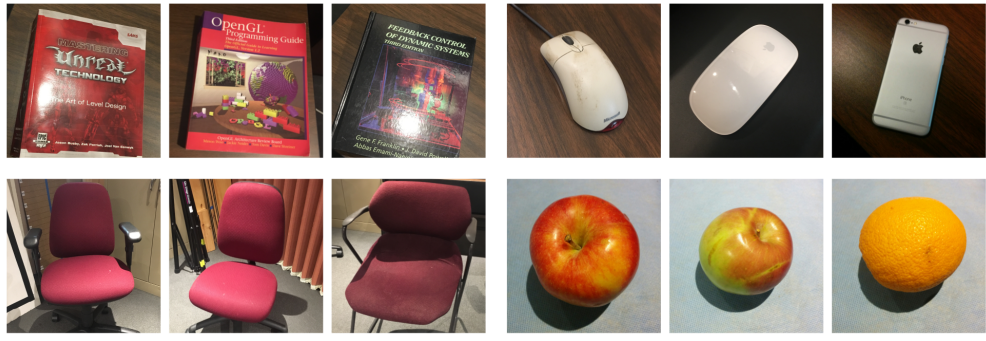


Figure 4.30: Photos of the object used in the experiment.

beacons, the image quality of my system is high enough to let people distinguish very similar objects.

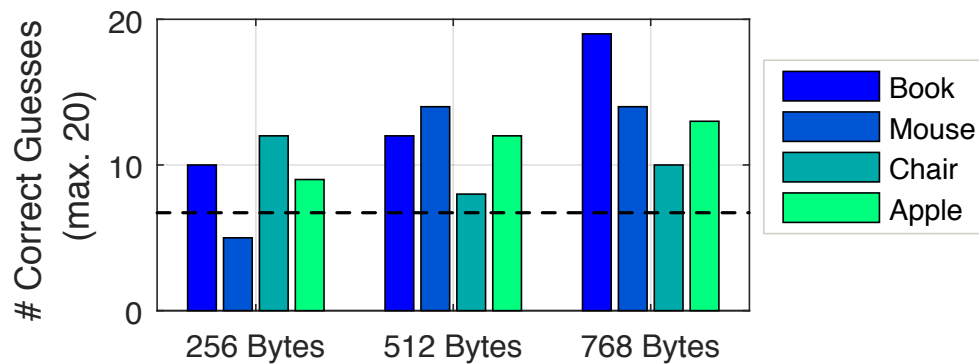


Figure 4.31: Responses from user study. The dashed line shows 33 percent chance of randomly choosing.

4.7.2 Navigation in the Building

In this experiment, I choose six locations inside the UNC Computer Science building, and at each location I place a binary image beacon that broadcasts a compressed image. These compressed images serve as ‘guides’ for navigating inside the building. Each participant is asked to follow the navigation information displayed on his phone screen. The participant has been instructed that ‘walking forward’ is the default direction, and if he sees, for example, a ‘turn right’ image at a turning point, he should make a right turn and keep walking straight until the current image

disappears and the next image is displayed. This experiment is designed to simulate a real world use case. In a place where the Internet is not accessible one can write arbitrary graphical information into a set of beacons for others to read at a later point. Therefore, it is critical to support arbitrary binary image writing functionality in my system, instead of having a predefined set of symbols.



Figure 4.32: Map of second and third floors showing the navigation path.

The navigation path was designed so that it starts at a corner on the second floor, goes across half of the floor, continues to the third floor, goes across to the half of the third floor, and then stops. Figure 4.32 shows the path. Each beacon's broadcasting interval is set to 1 second and the broadcasting strength is set to -20 dBm so that it's range does not overlap with its neighboring

beacon's. The experiment finishes when a participant successfully reaches his final destination (the last beacon showing the 'STOP' image) or the participant feels totally lost and cannot continue. After the experiment, the participants are asked a set of questions to answer: (1) overall difficulty of going from the starting point to the third floor, (2) overall difficulty of going from the starting point of the third floor to the destination, (3)–(8): how clear each image looked, (9) usability of the app (image reader). The score ranges from 1 to 10, where '1' stands for the least perceived quality and the hardest to recognize.



Figure 4.33: Images stored at the six locations.

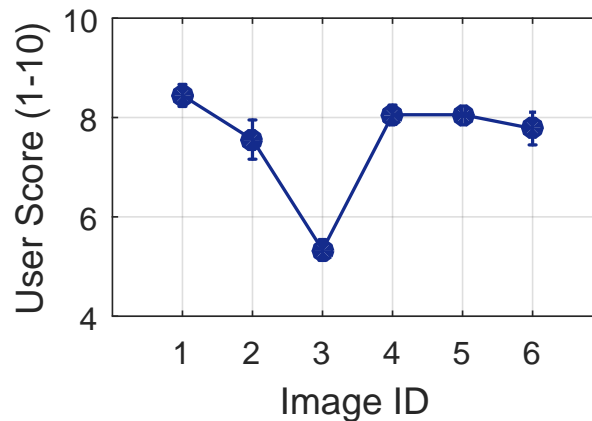


Figure 4.34: Subjective scores for each image.

Of the 5 participants who completed the user study, 4 successfully researched the destination point. 1 participant was lost after he reached the third floor. The images used in the study are shown in Figure 4.33. For each of these six images, I plot the average subjective scores in Figure 4.34. It is interesting to note that, although the second and the fifth image are the same, the fifth image has got a higher average score. I believe, this is because when a person sees the same direction for the second time, he tends to be more confident in recognizing the image. The third image has got the

lowest score. This is due to the fact that the instruction at that location asks the user to go upstairs which was confusing to many participants who did not expect a multi-floor travel path.

I also asked each user about his difficulty in following the navigation signs in floor 2 and floor 3. The average scores for these questions are 0.74 and 0.88, which means the users thought that the directions were easier to follow during the later stage of the experiment when they became used to the system. The overall satisfaction of the application was very high. All participants commented that the app is ‘easy to use’.

4.8 Summary

In this chapter, I described two systems involving beacon devices and smartphones with BLE receiving functionality. The first system allows a user to generate binary images, write binary images into the limited beacon device storage, and receive compressed binary images from beacon devices’ broadcasting packets. The main contribution of this work is the overall system construction, the patch-based binary image compression method, the evaluation of various system parameters in the system, and the evaluation of the trade-off between image quality and beacon battery lifetime based on my patch-based image compression method.

The second system I described allows a user to generate an approximation of an input color image, write the approximated image into the limited beacon device storage, and receive compressed images from beacon devices’ broadcast packets. The main contribution of this work is the overall system construction, the adaptive encoding image compression method, the evaluation of various parameters of the system, and quantifying the trade-off between image quality and beacon battery lifetime for my image compression method. My work widens the usage of the energy efficient, long-lasting beacon devices by allowing easy storage and access of custom image data in scenarios where there is no Internet connection.

CHAPTER 5

EXTRACTION AND COMPRESSION OF AUGMENTED REALITY CONTENT FOR LOW POWER AUGMENTED REALITY SYSTEM

Chapter 3 and Chapter 4 present the research projects I worked on for video and image compression, respectively. This chapter expands the domain of multimedia compression into another type of data: 3D object motion data. In this chapter, I present a new mobile augmented reality system that is based on Bluetooth Low Energy. The system integrates a content-prioritizing process to extract the most useful information from a captured object's motion so that the resulting 3D object motion information can be broadcasted by the small BLE packets.

I am particularly interested to know whether BLE beacons are capable of storing and broadcasting data structures describing 3D objects so that nearby mobile devices are able to receive and render those virtual objects onto the real world and to having a seamless mobile augmented reality experience. It turns out that the answer is *yes*, but in designing such a system, I need to deal with at least two fundamental challenges. Before I dig into their details, let us look at the benefits of a beacon-based mobile augmented reality system.

There are several advantages of having an augmented reality system that consists primarily of a set of BLE beacons. First, the system will be **low cost**. Second, beacons would store 3D objects locally and broadcast them over connection-less advertising channels **without requiring any Internet connectivity**. Third, beacons are designed to **last for a long time with battery power**, which makes these systems easier to setup and maintain. Fourth, in many modern buildings, **beacons are already in place**. Setting up a mobile augmented reality system in those buildings would practically cost nothing.

The main challenges in any augmented reality systems are: 1) communication of 3D object data that includes visual features, textures, rendering information, and a time series of these objects in case of video, and 2) the location and pose of the viewer so that the object can be overlaid on the real-world at the correct location and orientation. Furthermore, in case of mobile augmented reality, as the user moves, the object needs to be reoriented and redrawn based on his current position. For a seamless experience, all these have to happen in real time as well. BLE beacons, to some extents, provide support for both storage and localization. In the last chapter, I explored BLE’s capabilities to store images. Recently, Google started to experiment with an idea called ‘*Fat Beacons*’, where they are looking into broadcasting html pages over BLE. Indoor localization and navigation using BLE signals (Zhuang et al., 2016; Martin et al., 2014) is another active area of research. In this chapter, I combine these two promising aspects of BLE to enable more than what we have achieved with beacons so far.

In this Chapter, I present the first mobile augmented reality infrastructure that is based primarily on a set of low-cost BLE beacon devices. The target application is a motion capture scenario where a user (e.g. an actor, a doctor, or a lecturer) would enter into an area being monitored and make natural gestures while a distributed camera system would capture his motions. Later these captured movements can be replayed and viewed in 3D for various types of post-facto analysis purposes such as training and skill improvement.

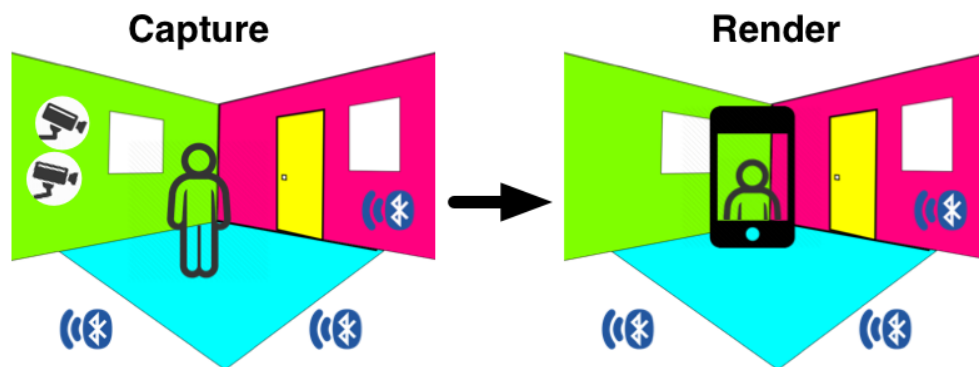


Figure 5.1: The two phases of MARBLE indoor augmented reality system: capture (once) and render (many times).

I propose a self-contained system (as shown in Figure 5.1) that consists of a cluster of BLE beacon units that are connected to an embedded micro-controller and a low-cost stereo camera. Once the system is installed and set up, it operates in two phases. The first phase is dedicated to detecting dynamic events in the area being monitored and to capture and store sufficient information about moving objects or subjects in the scene. This information is compressed, stored, and broadcast while meeting the storage and expected lifetime requirement of the system. The second phase is dedicated to receiving and rendering the 3D virtual objects and placing them at the right location and at right scale as a viewer moves and looks at the scene through his smartphone.

The crux of the system are two algorithms that are central to the two phases of the system. The first of which intelligently determines 1) a least number of most informative and useful visual features, and 2) a minimal amount of information about the moving parts of a 3D object, and store this extremely compressed information into the limited storage of the beacons. The second algorithm utilizes the BLE signal strength and combines it with the user's smartphone's IMU and camera images to accurately estimate his position and orientation in real time.

I have developed a prototype of the system, called the Mobile Augmented Reality with Bluetooth Low Energy (MARBLE), that consists of eight Estimote beacons (Estimote, 2017) connected to eight Raspberry Pis (RPI, 2017), each having two Arducam (Arducam, 2017) cameras attached to it. The prototype has been thoroughly tested to quantify its CPU and memory usage, as well as the accuracy of feature selection and localization algorithms. I demonstrate MARBLE by setting up an indoor motion capture scenario where 10 volunteers make five types of gestures for about three seconds while the system captures and stores their motions. Later, they enter the scene, walk around, and view the captured actions in 3D through their mobile phones.

The main contributions of this research work are as follows:

- To the best of my knowledge, I am the first to propose a BLE beacon-based mobile augmented reality system. This is a drastically different approach than existing AR technologies where I take the sensing, storage, and computation out of the mobile device, distribute them in low cost, low power, long lasting, and shared infrastructure.

- I devise two algorithms: (a) determining the least number of most informative and useful visual features, and a minimal amount of information about the moving parts of a 3D object, and (b) utilizing the BLE signal strength and combining it with the user's smartphone's IMU and camera images to accurately estimate his position and the orientation in real time.
- I have developed and evaluated a prototype of the proposed system. My evaluation shows that the system takes about 170 ms to capture an object's motion. In the rendering phase, the system renders the captured content in real-time. The selected features are 95%-100% accurate in determining the reference view, and the mean localization error is 14.5 cm.
- I conduct a user study involving 10 participants and demonstrate that the system is capable of capturing freehand gestures and when replayed back, the users were able to view and experience them in real-time.

5.1 Related Work

Many research projects have been done in exploring the indoor localization using Bluetooth signals. In (Cheung et al., 2006), the authors describe an indoor positioning system with Bluetooth beacons for a low cost (\$20). The approach is to place a beacon with a unique signature inside every room so that the smartphone can identify the signature to tell which room it is in. Their system achieves room-level precision. (Kajioka et al., 2014) talks about an experiment of evaluating the accuracy of indoor positioning using BLE. This work uses 22 BLE beacons inside and outside of a 10.5m x 15.6m classroom. The localization granularity is less than 1m. They divide the space in the room and outside of the room into 56 regions. They generate 5800 training examples for later position identifying. In determining the user's location in the experiment, the current received signal strength indicator (RSSI) signal is matched against pre-generated data using template matching based on Sum of Squared Difference (SSD). The accuracy is as high as 95%. (Inoue et al., 2009) describes a system similar to (Kajioka et al., 2014), but the system estimates a state transition using a different movement model. It is then followed by a probability distribution calculation to

better determine the location. (Faragher and Harle, 2014, 2015) contains analysis of the accuracy of BLE indoor positioning. It discusses the fast fading effects in BLE indoor positioning and its impact on the positioning accuracy, it also mentions that the existence of the WiFi signal in the space also impacts the positioning accuracy. The solution they proposed is to migrating a batch of RSSI readings (multipath mitigation) by taking the median of a batch of values. They showed that this improves the RSSI fingerprinting performance. They also further proposed the optimal BLE deployment density and argue that more dense beacon deployment does not improve the system performance after a certain threshold is reached. The BLE-based localization in MARBLE uses an approach similar to (Kajioka et al., 2014), besides that MARBLE also combines other sensor inputs in localization.

Research has been done in using one or multiple sensor inputs in mobile devices for augmented reality applications, including (Ababsa, 2009; Al Delail et al., 2013; Deniz et al., 2014; Guan et al., 2016; Dissanayake et al., 2001; Aulinas et al., 2008). There is a category of technologies named simultaneous localization and mapping (SLAM) that explores the environment and locates the viewer simultaneously (Dissanayake et al., 2001; Aulinas et al., 2008), which can be considered as an alternative solution to my localization and pose estimation problem. (Deniz et al., 2014; Guan et al., 2016) only uses visual information in determine the viewer's pose and location. Instead of using ORB features, (Guan et al., 2016) used Speeded Up Robust Feature (SURF) for image matching. In (Ababsa, 2009) the author combined multiple sensor inputs using Kalman filter, but their work is to build an outdoor localization system, where Global Positioning System (GPS) signal is used as one sensor input. My system works indoor. Differ from the previous approach, MARBLE fuses visual feature signal, BLE signal, and IMU signal.

5.2 Overview of MARBLE

The MARBLE is a mobile augmented reality system that uses a cluster of off-the-shelf, low power, storage and bandwidth constrained BLE beacon units as an infrastructure. The system operates in two phases: 1) the capture and store phase, and 2) the read and render phase.

Figure 5.1 (left) shows that a person enters an area being monitored with MARBLE. A set of cameras capture his actions and stores them inside beacons. Later (right), his activity during the first phase is viewed through a 3D augmented reality application on a mobile phone. Due to the low energy consumption feature of BLE protocol, the broadcasting phase is extremely energy efficient. In most application scenarios of MARBLE, capture phase is only performed once, or the capture is triggered by a rare event that only happens every few hours or few days. Therefore, most of the time MARBLE is working in the broadcasting phase and the expected system lifetime can be more than 6 years.

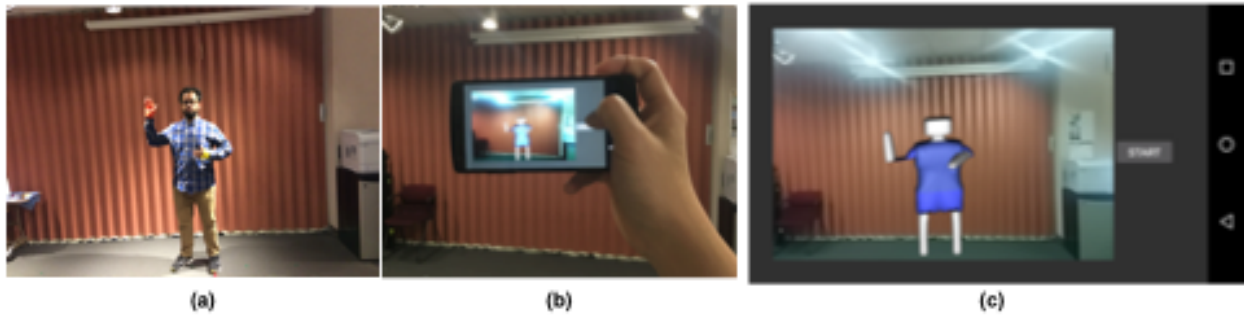


Figure 5.2: System in Action. (a): in capture phase, a person's gesture and movement is captured. (b): in render phase, the virtual object avatar is rendered in the empty environment. (c): screenshot of the viewer's screen in render phase.

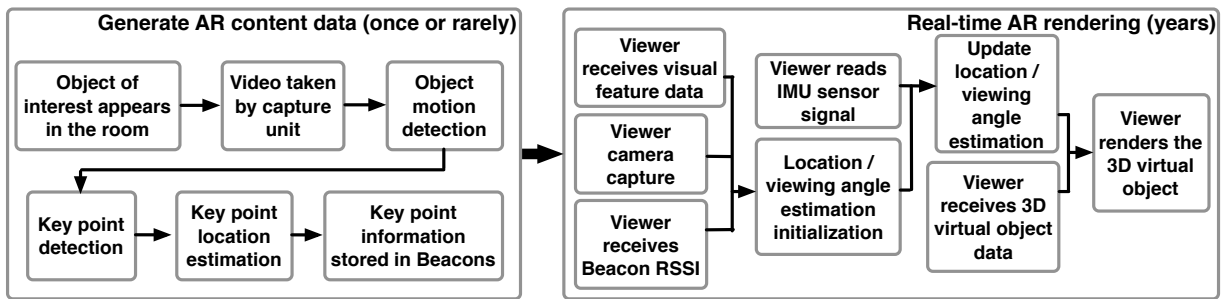


Figure 5.3: A block diagram illustrating the work flow of MARBLE.

5.2.1 Two Phases of MARBLE

During the *capture and store* phase, a set of low-power cameras are used to capture the dynamics of a scene. To simplify the design of each unit, I attach a stereo camera to each BLE beacon via a low-power embedded microcontroller. However, this is not a requirement in MARBLE for every BLE beacon to have its own camera, i.e. a camera can be shared by multiple beacons. I assume that the duration of dynamic events during this phase is relatively shorter, and hence, the energy consumption due to active cameras is not excessively high. Once the microcontroller computes the minimal necessary information about the scene, and the 3D objects and their motions to store, data is sent to the beacon for broadcasting over the connectionless BLE advertisement packets.

During the *read and render* phase, a mobile device such as a smartphone or a augmented reality headset with a BLE capability, receives the broadcasts and renders the previously captured scene in 3D. A major task in this phase is to determine the position and orientation of the mobile device, so that the objects are rendered at the right position and orientation in the superimposed space created by the virtual and the real environment as the users walk in the space. To achieve this, the received signal strengths of the BLE beacons, in combination with the IMU and camera of the viewer's mobile device, is used.

5.2.2 Internal Modules and Basic Workflow

During the set up of MARBLE, a number of beacon units are placed at fixed positions in an indoor environment surrounding the area to be monitored. Assuming a camera is attached to each beacon unit, each beacon takes pictures of the empty space – capturing only the environmental artifacts such as interesting points on the wall, furniture, and portions of the floor and the ceiling it sees. Each beacon then computes and stores a unique set of visual features (ORB features (Rublee et al., 2011)) of the view it observes. This view serves as the baseline view for a beacon, and is later used during the view tracking of the mobile user.

The workflow of my system is illustrated in Figure 5.3 along with the internal modules for each of the two phases of MARBLE.

The capture and store phase starts with the detection of any kind of change in the scene being monitored, e.g., when a subject enters into the area or an already present object or the subject moves. This awakens the cameras attached to the beacons, which starts capturing and processing the image. Processing the image stream involves simple predefined tasks such as detecting predefined interesting artifacts like a human body. If an object of interest is detected, key points corresponding to its moving parts (e.g., head and hands in case of a person) are recorded over time inside the beacon storage.

During the rendering phase, each beacon broadcasts its unique set of visual features and a subset of the captured 3D point time series data. At the receiving end, the viewer's smartphone receives two types of information from all beacons: 1) broadcasted visual features, and 2) received signal strengths. The smartphone combines this information with its own view as seen from its own camera to obtain a BLE-aided image-based location and pose estimate of its own. This is further combined with the smartphone's IMU data to accurately determine the position and orientation of the smartphone. Finally, location data and 3D object data are used to render the 3D object on his smartphone in real-time.

5.2.3 Advantage of MARBLE

There are several advantages of using MARBLE compared to existing mobile augmented reality systems:

- **Low Cost:** MARBLE is built upon low-cost technologies. The current prototype, consisting of eight Estimote BLE beacons, two Raspberry Pis with Arducam cameras, cost about \$150 and is sufficient to monitor an indoor space of $6.5\text{m} \times 5.3\text{m}$.
- **Internet Free:** MARBLE relies only on Bluetooth connectivity. It works in indoor environments where Internet access is unavailable or an inconvenience. Being Internet-free has some advantages; e.g., monitored environments such as retail stores and museums may already have WiFi access for customers who have bad cellular coverage, such WiFi accesses typically

require additional log in steps or going through confirmation pop-ups which is an inconvenience to the user. Compared to this, my proposed system brings a seamless experience to the user.

- **Very Long Lifetime:** Thanks to the promised multi-year broadcast lifetime of the BLE standard, which enables MARBLE to last for a very long time (months, if not years) over battery power. In scenarios where the dynamics of the scene is lower (e.g., monitoring a deserted area over a very long period) or motion is captured once but viewed many times (e.g., for post-fact analysis purposes), MARBLE may remain vigilant for multiple years without a battery replacement.
- **External Power Requirement:** Because of the phased operation, MARBLE does not require External power supply during the read and render phase. This makes this system convenient and deployable in scenarios where supplying external power supply is a practical problem.

5.3 Application Content Generation

In order to create a realistic 3D AR experience, the rendered 3D objects need to have the correct appearance in accordance with the viewer's position and orientation. A key challenge in MARBLE is to accurately estimate the viewer's position and viewing angle, which is discussed in Section 5.4. This section describes how MARBLE prepares necessary information to enable proper rendering of captured 3D content on a viewer's device.

Beacons are required to provide visual features of the scene (as seen from each beacon) so that these features can be matched with the viewer's own view to determine where he is looking at. I describe what are these features, and how they are generated, compressed, and stored inside beacons.

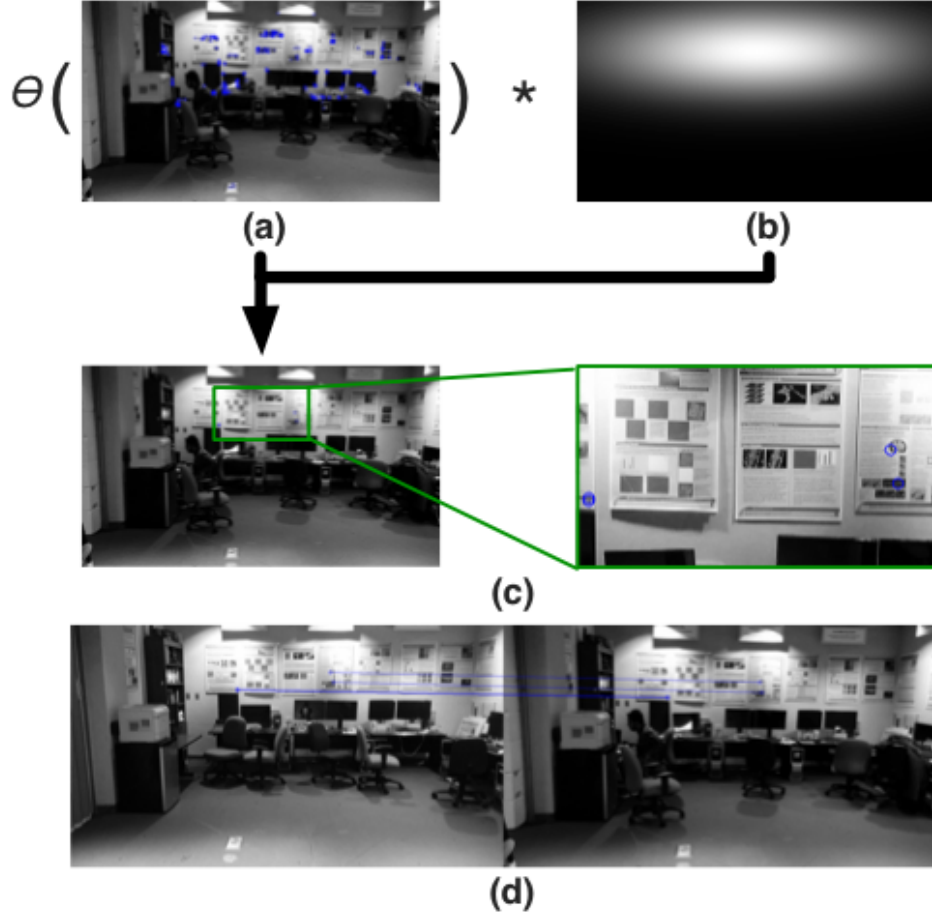


Figure 5.4: The process of feature filtering: hundreds of ORB features are extracted from a reference camera image in (a) and feature entropy θ is computed. The resultant values are weighted by the 2D Gaussian function of (b). Four highest scored features are shown in a zoomed in region in (c). They are successfully matched with features extracted from a different image of the same scene even though some of the objects have been moved, as shown in (d).

5.3.1 Visual Features

MARBLE extracts visual features from images taken by the reference cameras deployed in the environment. I use ORB features (Rublee et al., 2011) that are faster to compute than commonly used scale-invariant feature transform (SIFT) (Lowe, 1999) and SURF (Bay et al., 2006) features. Each ORB feature describes a small local region in the image. ORB features are robust to rotation, scaling, and translation due to changes in viewing angles. This makes them ideal for registering two images taken with slightly different viewing positions and angles. Once I find matches between ORB feature locations on an image from a reference camera to another image from the viewer’s camera, I adopt the *homography* method (Malis and Vargas, 2007) to estimate the viewer’s camera

relative rotation and translation to the reference camera. I deploy reference cameras surrounding the environment such that they cover the whole scene. In practice, I don't need multiple reference cameras. Only one camera can be placed at different locations to generate visual features to be stored in beacons. The storage requirement in bytes B_f for n ORB features is:

$$B_f = n * (k + \lceil \frac{w}{4} \rceil) \quad (5.1)$$

where, k is the length of a descriptor in bytes, w is the bit length for describing the horizontal/vertical offset of the location of a feature on the image, k for ORB feature is 32, and w in my case is 10.

An indoor photo taken by a smartphone usually contains hundreds of ORB features. But a BLE 5.0 beacon is only able to broadcast seven features in one broadcasting period. This capacity mismatch motivates us to design a visual feature selection algorithm that selects only four (among hundreds) features that are necessary for the homography algorithm. This results in a storage requirement of $4*(32+3) = 140$ bytes, which fits in a single BLE 5.0 packet.

5.3.2 Selecting Unique and Useful Features

During feature selection, I consider two factors: 1) the uniqueness of the descriptor, and 2) the chance of finding this feature in the second view during matching.

To determine the uniqueness of the descriptor, I compute feature entropy (Cao et al., 2014):

$$\theta_D = - \sum_i P(d_i) \log_2 P(d_i) \quad (5.2)$$

Here, feature descriptor D is a vector $\{d_1, \dots, d_n\}$, and $P(d_i)$ is the probability of d_i in a feature. $P(d_i)$ is estimated from all the features extracted from the reference cameras.

In my application, the difference in viewing angles between a viewer and a reference camera is usually within 45 degrees. Within this range, there is a high chance of having overlapping regions between two images near the center of the image taken by the reference camera. Furthermore, due

to the co-planar constraint in homography method, I select the feature points that share a plane. I observe that the feature points on the upper half of the image from the first view are more likely to be co-planar since they tend to be extracted from the far background (the wall).

In summary, if I denote w and h to be the width and height of the reference camera image, I would prefer the feature points near the upper half center ($w/2, h/4$) on the frame. I implement this by multiplying every θ_D computed on every feature descriptor by the 2D Gaussian function value centered on ($w/2, h/4$) on that location to determine the final feature weight. This is illustrated in Fig. 5.4 (a) and Fig. 5.4 (b).

Finally, I select n features with the highest weight among all candidates and store them in the beacon. The entire process is illustrated in Fig. 5.4.

5.3.3 Storing Camera Properties

Besides visual features, I also need to obtain the *intrinsic parameters* of each reference camera to compute accurate estimation of viewer's location and viewing angle in the rendering phase. The camera intrinsic parameters contain information about the focal length, aspect ratio, and principal points. These parameters along with the camera's location and orientation information are measured and stored inside a beacon only once.

5.3.4 Generating AR Content

MARBLE is designed to support *animated* 3D content, i.e. a time series of changing 3D contents. This section describes the 3D content representation of MARBLE.

MARBLE's 3D content consists of one or more 3D objects. A 3D object in digital systems is commonly represented by a set of surface meshes or a skeleton model of its internal structure, plus a distance function to its surface (Siddiqi and Pizer, 2008). Both representations require shape-related data points to be densely sampled and stored. Typically, hundreds of 3D points and their connection information are used to describe the surface of a non-trivial 3D object that is not simply a cube or a sphere. Since I am targeting animated 3D contents, the storage requirement will further increase

by an order of magnitude. In total, this may exceed 1MB, even after applying state-of-the-art compression techniques. This data size does not match MARBLE's storage limit.

To address this problem, I design my data retrieval component to make use of some prior knowledge about captured objects. For example, in case of a human body as the captured object, my system divides a human body into several major components such as two hands, a torso, and a head. 3D representations of these parts are pre-loaded in the viewer's app. In the AR content generation phase, the 3D position and orientation of these major components in the environment are detected and stored. This requires less than 100 bytes of storage. In the rendering phase, these components' 3D position and orientation information is received by the viewer's device. The viewer's device then combines this information with pre-stored components' 3D models to render the full 3D object. Using eight BLE 5.0 beacons, MARBLE can transmit 112 frames of human gestures in one broadcast packet.

In addition to captured data, MARBLE has the flexibility to store and broadcast synthesized virtual 3D object data which can either be pre-written into BLE beacons, or if the Internet connectivity is available, they can also be downloaded from the cloud. This enables MARBLE to render virtual objects with more details or objects that never appeared in the scene.

5.4 Real-Time AR Content Rendering

MARBLE stays in the broadcasting mode for most of its lifetime. In this mode, when a viewer enters the environment, he starts receiving 3D contents from the BLE beacons without requiring him to connect/pair with the beacons, and the virtual objects are rendered on his mobile device by an application that I developed. To generate a realistic AR experience, the application updates the rendered scene in real-time in accordance with the viewer's current viewing angle and position. This section describes my approach to estimating a viewer's smartphone's camera pose and location in the world coordinate.

5.4.1 BLE-based Location and Viewing Angle Estimation

A viewer's location information is represented by the camera position $p = (x, y, z)$ in the world coordinate. Similarly, the viewing angle is represented by a set of Euler angles $a = (a_0, a_1, a_2)$ in the world coordinate.

In BLE-based localization, multiple beacons are deployed on the boundary of the environment at known fixed locations. A viewer's mobile device receives the broadcasted BLE signal from all BLE beacons which results in a list of received signal strength indicator (RSSI) readings, along with the transmission power information A from the Bluetooth packets. The distance between the viewer's mobile device and each beacon is estimated by:

$$\text{RSSI} = -20 \log(d) + A \quad (5.3)$$

Here, d is the distance being estimated. To increase the robustness of the algorithm, I take the median of the last three RSSIs.

In order to estimate the current location of the phone, the 3D space is divided into grids. I estimate the theoretical distance between each beacon and the center of each grid and compare those with the RSSI-based distance estimates. The viewer's mobile device's location is at the center of the grid that has the smallest error E_{min} :

$$E_{min} = \min_i \sum_j \|\hat{d}_{ij} - d_{ij}\|_2 \quad (5.4)$$

where, \hat{d}_{ij} and d_{ij} denote the theoretical and RSSI-based distances between the i th grid and j th beacon, respectively. I use a square grid of dimensions $0.1\text{m} \times 0.1\text{m} \times 0.1\text{m}$.

5.4.2 Camera-based Location and Viewing Angle Estimation

Since Bluetooth's signal strength is sensitive to changes in the environment (Faragher and Harle, 2014), beacon-based localization has a poor accuracy, with a mean error of 30cm. Hence, I look

for an alternative method that uses the camera on the user’s phone and use it in combination with RSSI-based method.

MARBLE’s camera-based location and viewing angle estimation method uses the reference visual features broadcasted from the beacons. Recall that these features are generated by a reference camera, placed at a known location in the environment (Section 5.3). When camera-based localization is enabled, the viewer’s camera takes an image, extracts ORB features, and matches these features with those obtained from the beacons using (Hamming, 1950).

After the matching process, a homography (Agarwal et al., 2005) matrix corresponding to one reference camera is generated. The decomposition of this matrix provides us with the rotation and translation of the viewer’s camera, relative to the reference camera (Malis and Vargas, 2007). The translation is scaled by the depth of the scene, which is estimated from the location information of the reference camera. The rotation/viewing angle is further combined with the IMU readings of the phone to obtain a better viewing angle estimate.

5.4.3 Fusion of Multiple Sensor Inputs

I apply a Kalman filter (Kalman and Bucy, 1961) to fuse BLE-based and camera-based location and viewing angle estimation methods to achieve a better accuracy. I also incorporate with viewer device’s IMU readings in viewing angle estimation. I denote the state of a viewer as a six-element vector $S = \{a_0, a_1, a_2, x, y, z\}$. Whenever there is an update on any of the three sources of information, i.e. camera-based method’s feature matching (1Hz), IMU reading (100Hz), and beacon signal scanning (8Hz), the current state is updated based on the input signals and the previous state. For example, a_0 , a_1 and a_2 are updated whenever there is a new IMU (rotation) reading is available. When a beacon signal is received, x , y and z are updated based on the new location estimate. When camera-based location and pose estimates are available, all the entries in S are updated. My experimental results (Section 5.6) show that the use of a Kalman filter stabilizes the location and pose estimates, and results in a better estimation accuracy. By combining a high-frequency but low-accuracy method (BLE and IMU-based) with a low-frequency but high-accuracy method

(camera-based), MARBLE achieves the best of both worlds — real-time and accurate location and pose estimation. This makes MARBLE suitable for less powerful mobile devices where a purely camera-based method takes a very long time (e.g. 0.6 seconds on Nexus5) and a purely IMU/RSSI-based method is extremely inaccurate.

5.4.4 Rendering Objects

As the final AR content rendering, MARBLE uses the estimated viewer's location and viewing angle to transform the virtual object. The transformed virtual object is rendered in real-time on the viewer device's screen, superimposed on the environment.

5.5 Implementation Notes

I developed each unit of MARBLE using commercial, off-the-shelf components. Figure 5.5 shows one such unit.

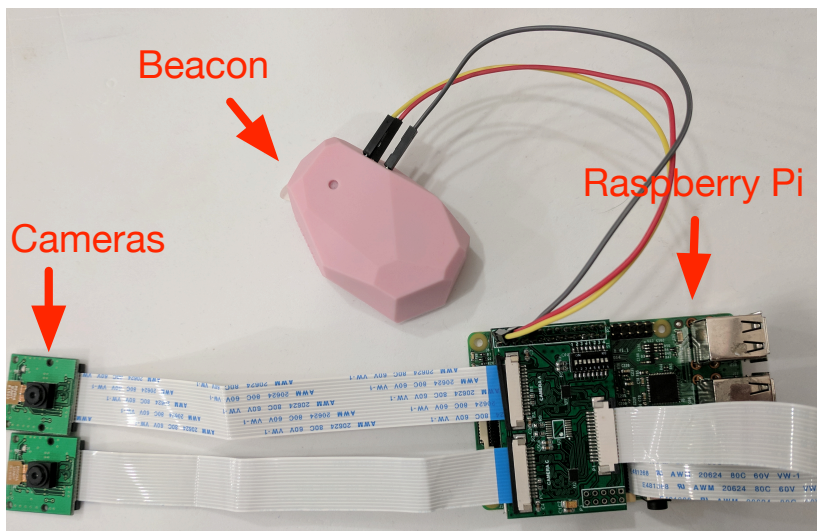


Figure 5.5: The capture unit.

Raspberry Pi is a card-sized single board low cost computer. I am using the Raspberry Pi 3 model, which has Quad Core Broadcom BCM2837 64-bit ARMv8 processor with up to 1.2GHz, 1 GB RAM and 40 GPIO pins. The dimension and weight of Raspberry Pi 3 is 86.9mm x 58.5mm

x 19.1mm and 1.5oz. I use Arducam OV5647 Mini Camera Module for Raspberry Pi. As Pi 3 is equipped with one Camera interface (CSI), I use Arducam Multi Camera Adapter Module as the interface to the stereo camera.

In all of my experiments, I use Estimote model REV.F2.3 Radio Beacon (Estimote, 2017) with a 32-bit ARM Cortex M0 CPU, 256 KB flash memory, 4 dBm output power, 40 channels (3 for advertising), and 2.4-2.4835 GHz operating frequency. The rendering application is written in Java/C++ and runs in a Nexus 5 smartphone with a 2.26GHz quad-core Qualcomm Snapdragon 800 processor, 2 GB RAM, BLE v4, and Android 6. I use OpenGL ES and ARToolkit library in rendering the virtual object. I use OpenCV library in camera based pose estimation and localization. I mimic BLE 5.0 broadcast packets by a set of rolling BLE 4.0 packets. The rolling mechanism is implemented by configuring the Estimote Location beacons to broadcast customized advertising packets.

5.6 Evaluation

In this section, I describe a series of empirical evaluations. I designed experiments to cover the evaluations for both the capture and rendering phases. I set up an experimental environment in the lab with size $6.5\text{m} \times 5.3\text{m}$. This is a typical use case scenario for my indoor AR system. For the capture phase, I evaluated the performance of visual feature selection. For the rendering phase, I evaluated the performance of reference camera matching in initialization. I also evaluated the system localization and pose estimation accuracy. In the end, I performed a user study to discover the trade off between the size of 3D object of interest data and the level of preservation of the object motion information captured by the system.

5.6.1 Microbenchmarks

Figure 5.6 shows the execution times of different components of the MARBLE system. The capture phase of the system runs on a Raspberry Pi. The render phase executes on a Nexus 5 Android phone. Detecting moving objects takes 65 ms and identifying point of interest takes 103

ms. Obtaining 3D location takes only 1 ms on average. Both rendering, BLE ranging and IMU data processing takes less than 1 ms on average. The most time expensive process is camera data processing, which takes 611 ms.

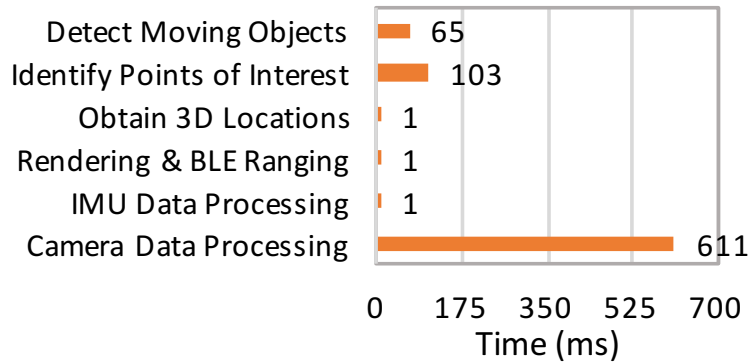


Figure 5.6: Run-time Analysis

I also perform an energy analysis of the system. The result is shown in Figure 5.7. The energy consumption of different stages of the system is listed. The analysis is based on the setting of 1s object of interest capture time, 1-year data broadcasting and 15-second viewing. The data consists of 15 frames of a virtual object motion. For the data broadcasting stage, the Beacon is configured with the broadcasting power -10 dBm and advertising interval 500 ms.

I consider the fact that different components in MARBLE operate in different time periods. To better understand the impacts of the energy consumption in each stage on the overall system, I compute the average energy cost in each component for delivering one frame (one set of key points positions) of virtual object data to the viewer. Even though BLE Beacons (green bar) operates at a much lower power than Raspberry Pi capture unit and Android viewer device because it is expected to stay in broadcasting mode for a year, it consumes the most energy in MARBLE.

I would also like to know the impact of changing the frequency of MARBLE's capturing operation on the system's lifetime. By fixing the amount of energy the system can consume, I compute the expected lifetime of the system under a variety of capture settings. The result is given in Table 5.1. I set the total consumable energy to be 25000 mJ. Because the viewer's device is

usually powered by a separate battery, and the number of viewers in the environment is not fixed, I don't include the viewer's power consumption in this analysis.

The result suggests that when my system works in an active surveillance mode that captures the object of interest in the environment every minute the system is expected to last for 30 days. When I only capture once at the beginning and then keep broadcasting without re-capturing, the system can work nearly 7 years.

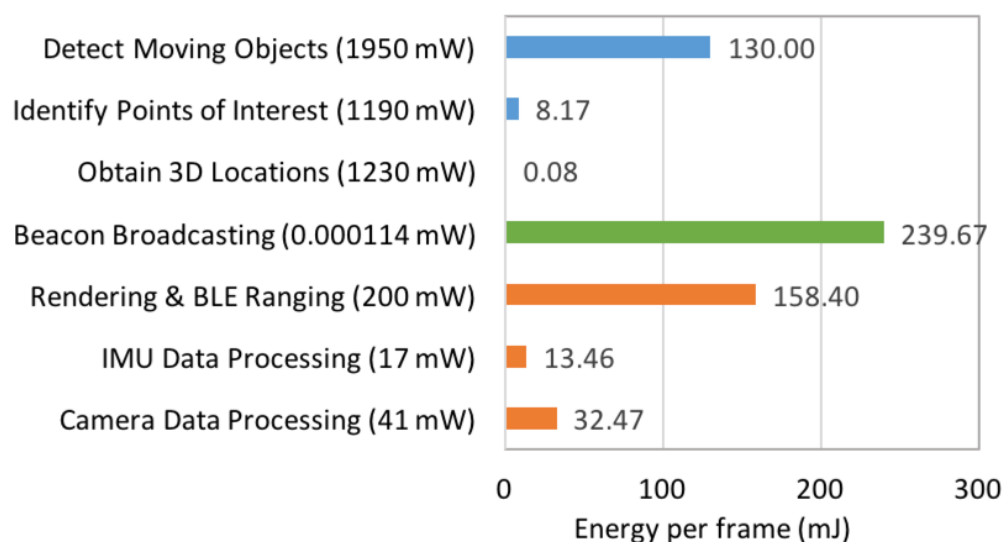


Figure 5.7: Energy Consumption of every stage of MARBLE. The data for the capture phase is colored in blue. The data for broadcasting phase is colored in green. The data for viewing phase is colored in orange. The power data next to the stage labels are the working power of each stage. The number next to the bar charts are the energy consumption for rendering a frame of the virtual object.

Capture Frequency	Expected Lifetime
No capture (Only Broadcasting)	6 years and 348 days
Once in a lifetime	6 years and 348 days
Once every day	6 years and 207 days
Once every hour	2 years and 325 days
Once every minute	30 days

Table 5.1: System lifetime under different settings.

Table 5.2 lists the CPU and memory usage for the corresponding process in the pipeline in capture phase. For this phase the most expensive process is the detection of moving objects which

takes 55.6% CPU and 5.4% memory. Identifying points of interest and obtaining 3D location takes 17.6% and 1.3% CPU, respectively. Table 5.3 gives an overview of the CPU and memory usage for the components of second phase. Here, IMU data processing consumes most user CPU, which is 16.33%. Though camera data processing requires lowest user CPU (4.22%), it uses the most memory (52.11 MB) among these three.

	CPU (%)	Memory (%)
Detect Moving Objects	55.6	5.4
Identify Points of Interest	17.6	5.4
Obtain 3D Locations	1.3	2.5

Table 5.2: CPU and memory usage of Raspberry Pi

	User CPU (%)	Kernel CPU (%)	Memory (MB)
Rendering & BLE Ranging	6.55	3.29	65.3
IMU Data Processing	16.33	0.35	5.27
Camera Data Processing	4.22	2.02	52.11

Table 5.3: CPU and memory usage of Smartphone

5.6.2 Algorithm Evaluation

Visual Feature Selection Performance

I evaluate my feature selection approach described in section 3.1. I compare my method with the approach of picking features randomly. I define the metric of this evaluation by dividing the number of correctly matched features with the total number of candidate features from the viewer's cameras.

In the experiment, I fix the number of features from the reference camera to be 4. I use 9 images taken by the viewer's camera. For each image taken by the viewer's camera, I record the number of correctly matched feature points with the reference camera's features selected using my method and reference camera's features selected randomly. I performed the same experiment twice with two sets of images taken in two lab spaces with different lighting conditions. I plot the result in Fig. 5.8

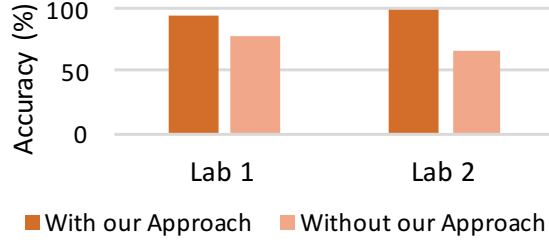


Figure 5.8: Averaged accuracy of matching between ORB features found in 10 query images and the 4 ORB features from the reference image.

The result shows that my method can significantly improve the feature matching performance, given the smallest possible number of features to be stored, my method gives 95% matching accuracy in lab 1 (darker room) and 100% accuracy in lab 2 (brighter room). As a comparison, randomly picking features gives 77.5% and 65% chance of generating a correct result in two rooms, respectively.

Beacon-Guided Camera Matching Performance

I evaluate the performance of Beacon-Guided Camera Matching in the initialization stage. In setting up the experiment, I placed 64 measurement spots on the floor of the environment, as illustrated in Fig. 5.9 (a) After that, the 8 beacon's Bluetooth signals' strength are measured and the BLE beacon based location estimation is performed. Then the reference cameras that are far away from the viewer are filtered out. The number of reference cameras being considered at each measurement spot is shown in Fig. 5.9 (b). I can observe that in all measurement points, at least one reference camera is excluded. In half of the cases, more than 3 reference cameras are excluded. This means reduction of computation time by almost 40% and increase of potential matching success rate.

Localization Accuracy

I conduct an experiment to measure the localization and pose estimation accuracy of MARBLE system in the render phase. I design a viewer trajectory that has a length about half of the room size. I set 16 measurement points along the trajectory. I fix the viewer camera's angle and location at every measurement point. I record the angle and location of every measurement point as the ground

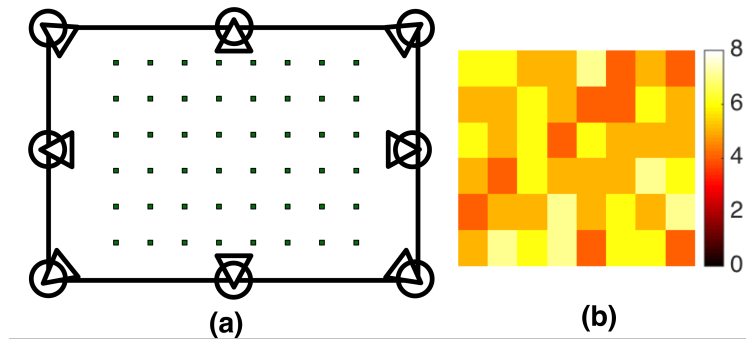


Figure 5.9: (a): the configuration of measurement points in the lab. (b): heat map showing the number of cameras being considered in visual feature matching.

truth. In the experiment, I move the viewer device along the trajectory, at each measurement point. I carefully move the viewer's device so the device's location and viewing angle match the designed value at that point. At every measurement point, I record the camera state estimation (IMU-only, BLE-only, camera-only and the Kalman filter joint estimation). I design the experiment to have two measurement points with viewer's device not pointing to the direction of the virtual object. At these two points, the system turns off the camera based estimation. So the estimation error from these two points for the camera-only method is not available (marked as crossed in Figure 5.10 and Figure 5.11). In the analysis, I compute the localization error based on square root of the sum of squared differences in x , y and z between estimate and ground truth.

I plot the localization result error in Figure 5.10. The result suggests that the camera based localization has a higher accuracy than the BLE based localization in most cases. The sensor fusion using both BLE signal and camera signal provides an even more accurate result, except for the first measurement point, where the camera based method has the highest accuracy, and the last measurement point, where the BLE only method has the highest accuracy.

Pose Estimation Accuracy

I also plot the accuracy of pose estimation for the experiment described in the previous subsection. I derive the error by computing the square root of the sum of squared difference on all rotation angles between the ground truth and the estimated values.

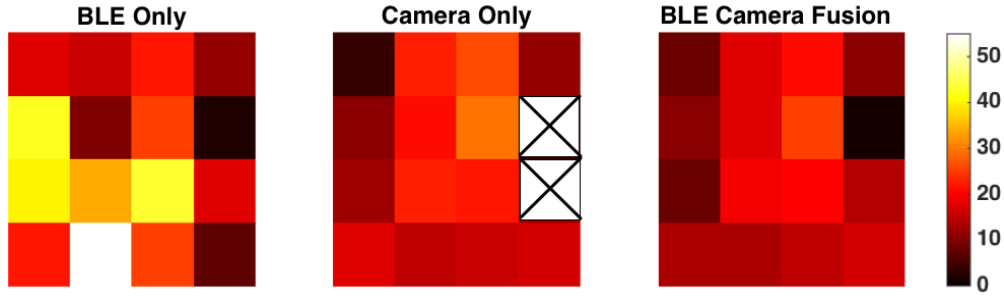


Figure 5.10: Heat Map of the errors in cm on location estimation in 16 different measurement points. From left to right: BLE Beacon based estimation, Camera based estimation and Sensor Fusion using two signals.

The result suggests that the IMU based pose estimation has a slightly higher accuracy than camera based estimation. The error stays less than 20 degrees.

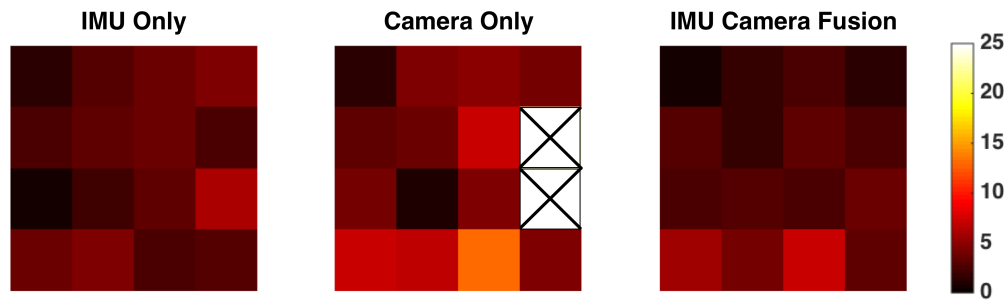


Figure 5.11: Heat Map of the errors in angle on pose estimation in 16 different measurement points. From left to right: IMU based estimation, Camera based estimation and Sensor Fusion using two signals.

Real Deployment and User Study

I perform a real user study to explore the performance of the entire system. I design the experiment with 10 participants entering the room one by one. I capture videos of a person doing five gestures at two different speeds in the room. The five gestures are: left hand wave, right hand wave, clap, hand roll, and both hands wave. The gesture is completed within 1.5 seconds and 0.75 seconds for low speed and high speed respectively. I then detect the head and hands locations in 3D to generate the key points over time. I sampled the points at 3Hz, 6Hz, 9Hz, 12Hz and 15Hz to generate data with different sizes.

In the render phase I use these data to create the moving virtual object (avatar). Once a user enters the room I show him/her the avatar performing one of the five gestures on the phone running the render app. The user then guesses which gesture is being performed. I vary the avatar data being used with different sampling frequency and speed. In total I gather 50 answers. Based on the correctness of the answers I derive the quality of the avatar at every sampling frequency and speed combination.

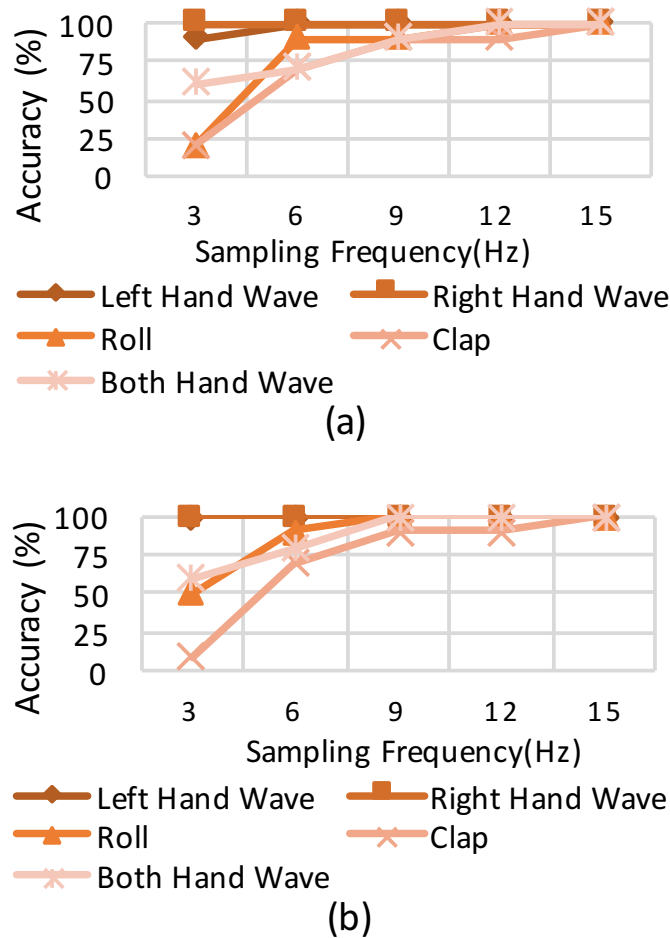


Figure 5.12: Human Gesture Sampling Frequency Impact at (a) Low Speed (b) High Speed.

In Figure 5.12 (a), I can see that for low speed gesture, 6Hz sampling frequency is enough (100% accuracy) for users to recognize the easy gestures like right hand wave and left hand wave. Although for more difficult gestures like both hand wave, it gives about 75% accuracy. In Figure 5.12 (b), I

can see that higher gesture speed has a higher recognition accuracy (more than 90%) for difficult gestures at 9Hz, it gives lower accuracy (70%) of easy gestures at 6Hz. Overall, I argue that 9Hz sampling frequency is good enough to capture the meaning of a motion around 1s. This requires 162 bytes data, which can be broadcasted by one BLE 5 packet.

5.7 Summary

In this chapter, I described MARBLE, an Internet-free augmented reality system that uses beacon devices. The system captures the object of interest in the environment, finds the 3D key points that best describes the object of interest given limited storage, and transmits it to the viewer for rendering. The viewer uses multiple sensor information, including the beacon BLE signal strength, to determine its location and pose in the environment. In the broadcasting phase, the MARBLE system works without external power support and battery replacement for years.

The main contribution of my work described in this chapter is the overall system design and construction, the method of selecting the best visual features for camera-based localization and pose estimation and quantifying the trade-off between object key point data size and the quality of the object approximation. My work widens the usage of the energy efficient, long-lasting beacon device by enabling the application of this type of devices for indoor AR application. My system can be easily extended for building other types of AR applications including real world object tagging.

CHAPTER 6

GENERATIVE COMPRESSION AS AN ALTERNATIVE APPROACH TO EXTREME COMPRESSION

6.1 Introduction

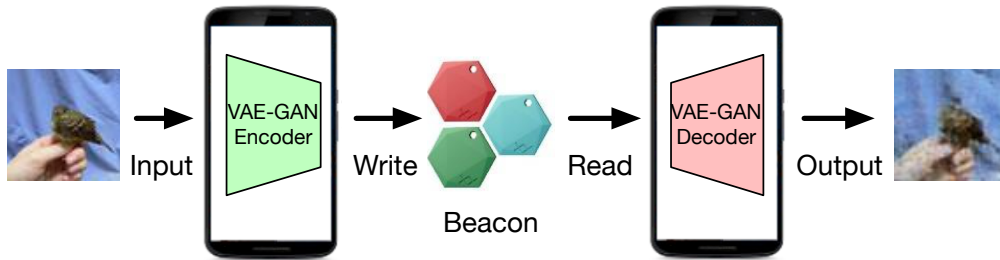


Figure 6.1: The proposed Deep Beacon system.

In Chapter 4 and Chapter 5, I devised image beacon systems that broadcast images of a few limited categories such as binary images and images containing common indoor and outdoor objects. I designed custom image compression modules that are applicable to only a fixed set of images. These limitations prohibit the usages of image beacons in many scenarios. For example, the color image beacon relies upon depth information of an image which is obtained by engaging a user in taking multiple images of the target object. The system does not work for a regular image whose depth information is missing. Furthermore, when color images were considered, they required multiple advertisement packets for a full image to be broadcasted. This often resulted in minutes of waiting time as a receiver has to wait for a full cycle of broadcast in case of packet drops. Hence, a new generation of image beacon system needs to be developed which is capable of efficiently storing and retrieving a wide variety of high-quality images.

Recently, the machine learning community has developed *autoencoders* which demonstrate strong performance in finding the best low dimensional representation of a certain type of data. A specific type of autoencoder called the *Variational Autoencoder* (VAE) (Kingma and Welling, 2013) consists of an encoder network and a decoder network. The encoder network converts an input image to an *embedding* vector, which is a representation of the image in a low dimension and is several orders of magnitude smaller in size than the original image. The decoder network takes the embedding vector and recovers the input image. VAE is useful in applications such as ‘fake’ content generation. For example, smiling faces and sad faces have different embedding vectors. One can take these two embedding vectors, interpolate them, execute the decoder network on each interpolated vector, and be able to generate intermediate faces to show a realistic illusion of a person becoming sad from smiling.

I am inspired by VAE’s power of extracting a low dimensional representation of an input image and explore the potential in developing an image beacon system that employs an autoencoder to enable high-quality, arbitrary, color image storage and broadcast in a BLE beacon system. The system (as shown in 6.1) consists of one or more BLE beacons, a *writer* app equipped with a trained encoder that runs on a mobile device, and a *reader* app equipped with a trained decoder which also runs on the mobile device. I assume that no additional information on the broadcasted image from any other sources is available (e.g. from the web or the user’s smartphone).

I have developed a prototype of a Deep Beacon system using a set of commercially available Estimote beacons (Estimote, 2017), and developed an Android application that takes images along with user-specified requirements and constraints on broadcasting the image as inputs, generates previews of the image to be written, and writes the image representation into a set of beacons. I developed a reader application that reads the broadcasted image data, decodes the image, and displays it on the phone.

I perform an in-depth evaluation of the beacon system. I describe a set of results showing the tradeoffs between system lifetime and image quality when the image type and the number of beacons are varied. I also deploy a Deep Beacon system indoors and perform a user study in a

real-world scenario in order to have a subjective measure of the quality of the received images. In this study, a group of 15 participants are asked to identify traffic signs and hand-written digits from their beacons of various qualities.

The main contributions of this project are as follows:

- To the best of my knowledge, I am the first to propose an image beacon system that incorporates a deep neural network generative model to compress, store, and broadcast color images over BLE advertisement messages.
- I have devised a bit length reduction algorithm that is tailored to the needs of an image beacon system. The algorithm reduces the bit length of the image encoding down to 10 bytes. I vary the bit length reduction setting and quantify the tradeoffs between image quality and device lifetime, and determine the best set of parameters, under the user-specified constraints on the number of beacons, latency, and expected system lifetime.
- I have developed and evaluated a prototype of a Deep Beacon system that broadcasts images of various types e.g., birds, traffic signs, flowers, and hand-written digits. My evaluation shows that one BLE 4.0 beacon is capable of broadcasting images (90% multi-scaled structurally similar to original images) for 4 year-long continuous broadcasting, and both the lifetime and the image quality improve when more beacons are used.

6.2 Related Work

My Deep Beacon system uses a generative model for image compression. The concept of using generative model for compression is discussed in (Santurkar et al., 2017). During recent years, many useful generative models have been developed in the machine learning community. Examples include (Theis and Bethge, 2015; Goodfellow et al., 2014). Among those, generative adversarial networks (GAN) (Goodfellow et al., 2014; Goodfellow, 2016) is especially powerful. This approach iteratively trains a generator model and classifier model in an adversarial way. Experiments show that GAN is able to generate realistic, visually plausible images (Zhang et al., 2016). People have

extended GAN and developed a various of enhanced generative models. Examples include (Ledig et al., 2016; Isola et al., 2016; Denton et al., 2015; Zhang et al., 2016; Reed et al., 2016). My system uses the original version GAN (Goodfellow et al., 2014).

Another potential direction of using deep neural network model for image compression involves “style transfer” using a trained convolutional networks (Gatys et al., 2015; Champandard, 2016; Gatys et al., 2016). The style transfer model transfers the style of texture from one input image onto another input image. This can be used to enhance the compressed image, in which the texture information is lost during compression. The original style transfer algorithm is designed to transfer oil painting stroke texture onto real photographic images. The transfer process is computationally intensive. Later, researchers have developed faster versions of the algorithm that can run on a smartphone in real time (Ulyanov et al., 2016; Johnson et al., 2016). On the other hand, models that transfer high quality real photos have been developed (Luan et al., 2017). Researchers also combined style transfer model with U-net (Ronneberger et al., 2015) to guide the style transfer, so that the model performs automatic coloring on gray scale images (Liu et al., 2017). This could potentially turn the binary image beacon system into a color image beacon system. However, this type of automatic coloring algorithm tends to have good performance only on sketched images. It has difficulties in handling complex variations o textures and colors of natural images. Therefore, I decide to not apply this approach for my system.

6.3 System Architecture

Deep Beacon consists of a set of BLE beacons which store and broadcast images. The system comes with a mobile application that is used to (1) capture, compress, and write an image into the beacons, as well as (2) to receive and render a broadcast image on the screen of the smartphone.

Figure 6.2 shows these two operations. A writer app takes a picture and uses a pre-trained encoder to generate an extremely compressed version of the image which is written into the beacons. Later, a reader app captures these compressed image broadcasts and uses a pre-trained decoder to render the image on the smartphone’s screen.

The following two subsections describe the training of the encoder and decoder that happens offline, and operations that execute on the mobile application at run-time, respectively.

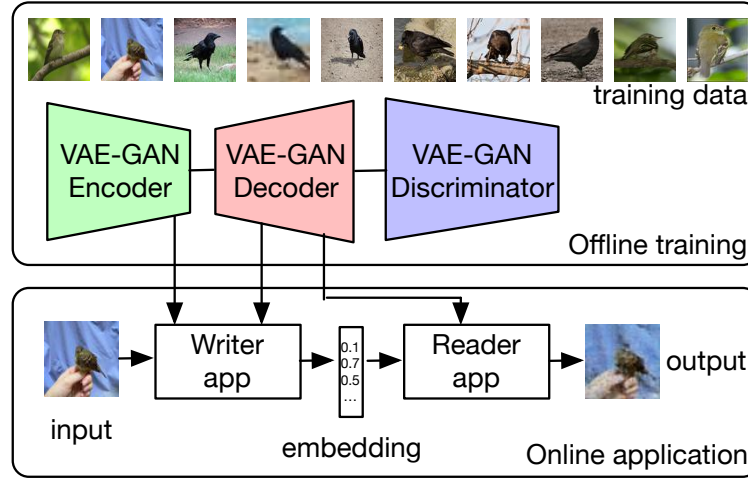


Figure 6.2: The data flow of Deep Beacon system.

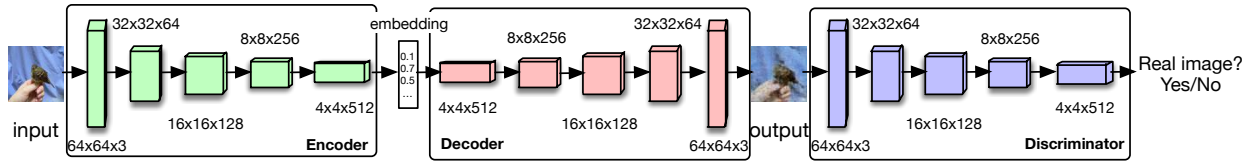


Figure 6.3: VAE-GAN model structure.

6.3.1 Offline Training

Before the deployment of the Deep Beacon system, the VAE-GAN model is trained using a large collection of images. The training process iteratively updates the model parameters to minimize a loss function. After training, the model represents the data space of a given image type. A compact representation (embedding) is generated from the encoder of the model given an input image.

The VAE-GAN model and the optimization function of the training process are explained in Section 6.4. The dataset and the details of the training process is described in Section 6.7.

6.3.2 Online Capture, Broadcast, and Render

Once the VAE-GAN model is trained and deployed, the Deep Beacon system becomes ready for image storage and broadcast. The workflow of the online phase of Deep Beacon consists of three stages: 1) image capture and write, 2) image broadcast, 3) image read and decode.

In the image capture phase, a user who wants to write an image into Deep Beacon uses the writer app on his mobile phone to compress the image. The input image can either be an existing image or one taken with the phone camera. The writer app scales and crops the image into 64×64 pixels. Then it uses the trained Deep Beacon encoder to compress the processed image. The compression output or the *embedding* is a list of floating point numbers.

The embedding has between 10 to 100 floating point numbers. The precision of these floating point numbers is configurable in Deep Beacon which provides flexibility to a user in making a trade-off between data size and image quality. The details of this trade-off is presented in Section 6.7.

Once the embedding data is stored in BLE beacons, it is broadcasted by them over their advertisement packets. For a very large image, it may require more than one advertisement packet to carry the embedding data. Based on the number of available beacons, two approaches are used to handle such cases. First, a single beacon broadcasts embedding data (whose size is larger than a single packet) by iterating over packets during broadcasting. Second, when multiple BLE beacons are available, data are distributively stored and broadcasted from multiple beacons.

Each time a user (in the role of an image data viewer) enters the environment, the reader app on his mobile phone starts BLE scanning to collect BLE advertisement packets containing the embedding data. Once the entire embedding is received, data are passed to the Deep Beacon decoder to recover the input image.

6.4 Algorithm

6.4.1 Background on Variational Autoencoder (VAE)

A Variational Autoencoder (VAE) contains a pair of image encoder and decoder. Specifically, an image encoder takes a 64×64 RGB image and converts it to a sequence of floating point numbers. It does so by passing an image through the encoder in VAE and then taking the computed values of the final layer nodes as the embedding output. The encoder is composed of four convolution layers. Each layer contains a set of convolution filters that generate a smaller patch from the output patch of the previous layer. The decoder, on the other hand, takes an embedding and generates a 64×64 RGB image. When a VAE model is properly trained, the result image is expected to be visually similar to the original one. Similar to an encoder, a decoder also has four layers of filters but they perform the opposite (deconvolution) operation.

6.4.2 Background on Generative Adversarial Networks (GAN)

The VAE-GAN used in Deep Beacon is an enhancement on the VAE with the Generative Adversarial Networks (GAN) (Goodfellow et al., 2014) model. GAN shows strong performance in terms of learning an unknown distribution of given image dataset, and generating ‘fake’ images that are visually plausible. GAN combines a *generator* that generates fake images and a *discriminator* that recognizes real images from fake images. The training process involves optimizing the generator to generate realistic images as well as optimizing the discriminator to better classify generated images versus real images.

6.4.3 VAE-GAN in Deep Beacon

A traditional VAE has a weakness that it tends to generate blurry images as the training process penalizes sharp edge differences between an input image and an encoded image. A blurry image is likely to be classified as a false image by the GAN discriminator, resulting in a larger loss. By

incorporating the loss term derived in GAN training process, the VAE is trained to generate sharper images. The structure of the VAE-GAN model used in Deep Beacon system is shown in Figure 6.3.

The overall VAE-GAN structure is shown in the offline training block in Figure 6.2. The decoder in VAE is shared with GAN as the generator in GAN. The training of VAE-GAN involves optimizing a loss function with three terms: a *prior* term that enforces the embedding to follow standard Gaussian distribution, a *log likelihood* term that measures reconstruction error, and a *style error* term from the GAN training. The details of the training process can be found in (Larsen et al., 2015).

6.4.4 Compressed VAE-GAN Embedding

The embedding of an image is a list of n digit single-precision floating point numbers. Storing each number requires four bytes of memory, resulting in a $4n$ bytes storage space requirement for the embedding. I optimize the storage by further reducing the bit length of the embedding data. I design Deep Beacon system such that it supports a variable number of bytes in representing a floating point number (with up to 4 bytes) N and a half byte size is also supported. This is done by quantizing the last byte in the representation. The steps of reducing the bit length of an embedding is given in Section 6.5. It assumes little-endian byte ordering. The impact of a shorter byte representation of an embedding on the result image quality is discussed in the evaluation.

6.5 Embedding Size Reduction Algorithm

The embedding size reduction algorithm takes the targeting bytes per-element k , and preserves the highest k significant bytes for each digit in the input. For a n -digit input, the algorithm outputs nk bytes.

6.6 Data Packet Format

I design a custom data packet format on top of the BLE broadcasting packet to carry the image embedding information. The package structure is shown in Figure 6.4. The packet contains a 2-byte header, which contains three syntax elements:

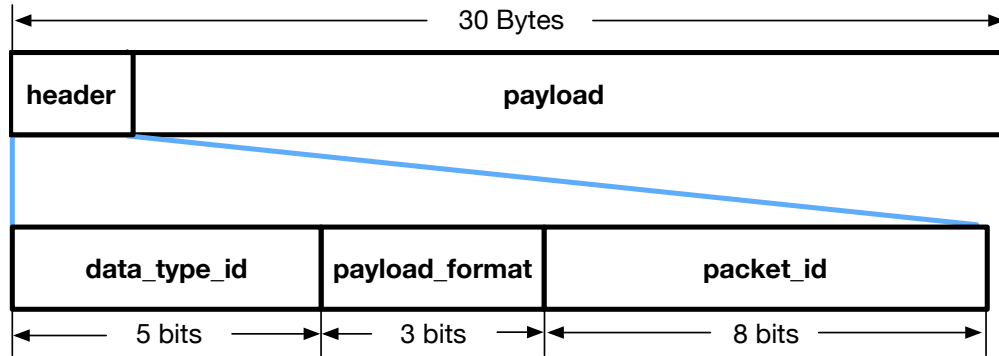


Figure 6.4: The Deep Beacon data packet format.

- `data_type_id`: the first 5 bits of the header carries the image data type information. The Deep Beacon system VAE-GAN model deployer, the writer, and the reader should agree on the meaning of the data type id to data type mapping. Essentially, each data type maps to a specific instance of VAE-GAN model that handles a certain type of image encoding and decoding.
- `payload_format`: the second part of the header represents the payload format. This information indicates how many bytes are used to represent a single floating point number in the embedding data. This ensures a correct decoding of the embedding data on the image reader.
- `packet_id`: when the payload embedding data cannot fit into one BLE broadcasting packet, the packet id information is required so that the order of the set of received data packets can be determined.

The remaining 28 bytes in the packet is used as a payload to carry the embedding data.

6.7 Evaluation

In this section, I describe a set of empirical evaluations. Firstly, I show the quality of compressed images for different embedding sizes. Then I compared my system’s image compression performance with JPEG. I then perform experiments to quantify the tradeoffs between image quality and expected battery life when the system is configured for different types of images or different number of beacons. I also conduct a user study that involves multiple use cases including hand-written digit and traffic sign recognition tasks. Finally, I compare my Deep Beacon system with two image beacon systems I developed previously.

6.7.1 Experimental Setup

I use Estimote model Rev.D3.4 Radio Beacons (Estimote, 2017) having a 32-bit ARM Cortex M0 CPU, 256 KB flash memory, 4 dBm output power. I vary the BLE broadcast interval for a beacon between 100 ms to 2,000 ms. However, an encoded image (broadcasted from multiple beacons) reaches a user’s device in less than 0.5 second. The transmission power is set to -12 dBm, which limits the range of each beacon to about 30 m. The writer and reader application runs in a Pixel 2 smartphone having an Qualcomm Snapdragon 835 processor, 4 GB RAM, BLE v4.0, and runs Android 8.1.0 Operating System.

I use four image datasets containing hand-written digits (LeCun et al., 1998), birds (Welinder et al., 2010), flowers (Nilsback and Zisserman, 2008), and traffic signs (Stallkamp et al., 2011). They are selected from the MNIST dataset (LeCun et al., 1998), the Caltech-UCSD Birds dataset (Welinder et al., 2010), the 102 Category flower dataset (Nilsback and Zisserman, 2008). and the GSTRB dataset (Stallkamp et al., 2011), respectively. Examples of images are shown in the first four rows in Figure 6.5. All images are center-clipped and down-sampled (or up-scaled in case of MNIST) to 64×64 pixels. In the training, I use 1000–2000 images from each dataset. As test images, I use 100 images from each dataset. There is no overlapping between training images and test images.

The VAE-GAN model and the training program is implemented in TensorFlow 1.1.0. I use Google Cloud Machine Learning Engine (goo, 2017) for training.



Figure 6.5: Samples of training and test image dataset. First 4 rows from top row to bottom: hand-written digits, birds, traffic signs, and flowers. The last row contains sampled images compressed by Deep Beacon system.

The metrics I use to measure image quality are multi-scale structural similarity (MS-SSIM) (Wang et al., 2003) and device lifetime in months. I measure these two under different conditions and show their tradeoffs. The MS-SSIM scores are used to measure the quality of the produced images when compared to the original ones. Compared to single scale SSIM metric, MS-SSIM gives more flexibility in image quality assessment with respect to different viewing conditions. The formula for computing the MS-SSIM score is given in Eq. 6.1, where l_M measures the illumination (pixel intensity mean) difference between image x and image y , c_j measures the two images' contrast (pixel intensity variance) difference, s_j measures the two images' structural difference, computed by the covariance between pixel values. The c_j and s_j are computed over different scales and the results are scaled by the constants α_M , β_j , and γ_j and multiplied. In my experiments, I use 5 scales. I apply the constant values with the same ratios suggested in (Wang et al., 2003): $\alpha_M = \beta_5 = \gamma_5 = 0.0267$, $\beta_4 = \gamma_4 = 0.0473$, $\beta_3 = \gamma_3 = 0.06$, $\beta_2 = \gamma_2 = 0.0571$, and $\beta_1 = \gamma_1 = 0.09$. The device lifetime is estimated from its relation to a beacon's advertising interval. Before each experiment, I program the beacons to set a beacon interval and use the corresponding estimated device lifetime (as reported by the Estimote API) in my experiments.

$$\text{MS-SSIM}(x, y) = [l_M(x, y)]^{\alpha_M} \prod_{j=1}^M [c_j(x, y)]^{\beta_j} [s_j(x, y)]^{\gamma_j} \quad (6.1)$$

6.7.2 The Choice of Embedding Size

My goal in this experiment is to determine an optimal embedding size for the VAE-GAN model. I train 12 different VAE-GAN models that are arranged in four groups. Each group is trained with one image dataset: hand-written digit, bird, flower, or traffic sign. Within each group, three embedding lengths are applied: 10, 50, and 100.

With the 12 trained models, I apply the test images (described in the previous subsection) on each model to generate compressed versions of the test images. Each type of test image is only applied to the corresponding model that has been trained with the same type of training images. I then compute the MS-SSIM scores for every compressed image. Figure 6.6 shows the average MS-SSIM score for all 12 models. Each color represents one embedding length.

The result suggests that the models having an embedding length of 10 does not show a significant weakness compared to the models having an embedding length of 50 or 100. For the hand-written digit images, the model having an embedding length of 10 achieves similar performance as the model with embedding length 100. Considering the fact that the most important factor in the design of a BLE-based system is the limited data bandwidth, I choose an embedding length of 10 in Deep Beacon system since this configuration requires much shorter bit length to represent an image than the embedding length 50 and 100. I use this value in all other experiments.

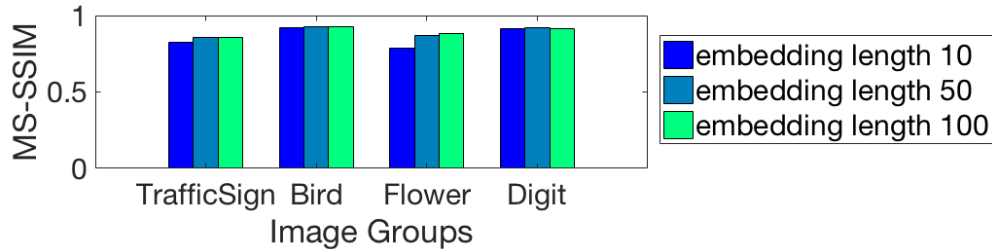


Figure 6.6: Bar chart showing the average MS-SSIM of four types of image data with different embedding sizes.

6.7.3 Comparison with JPEG Encoding

The goal of this experiment is to get an insight on Deep Beacon’s image compression performance compared to standard JPEG (Wallace, 1992). I use two test data sets: hand-written digits and traffic signs. Each test data set contains 100 images. I compress each image using JPEG in its lowest quality setting. This generates the smallest possible JPEG compressed image. I also compress every image with the encoder in Deep Beacon. For a compression with Deep Beacon, I use the embedding length 10, and 4 bytes for each number in the embedding.

I compute the MS-SSIM between every compressed image and the original uncompressed image. I then scatter plot the MS-SSIM values vs. the compressed image size in bytes in Figure 6.7. The result in each plot clearly shows two clusters: one for images compressed with Deep Beacon encoder, one for images compressed with JPEG. For black-white hand written digit images, JPEG generates compressed image of sizes between 400 to 500 bytes. For color traffic sign images, the JPEG compressed size ranges between 700 to 900 bytes. As a comparison, my Deep Beacon encoder always generates 44 byte compressed data size, including the data packet header. I also observe that the compressed images’ MS-SSIM quality scores from Deep Beacon encoder can reach the same level as the images from JPEG compression.

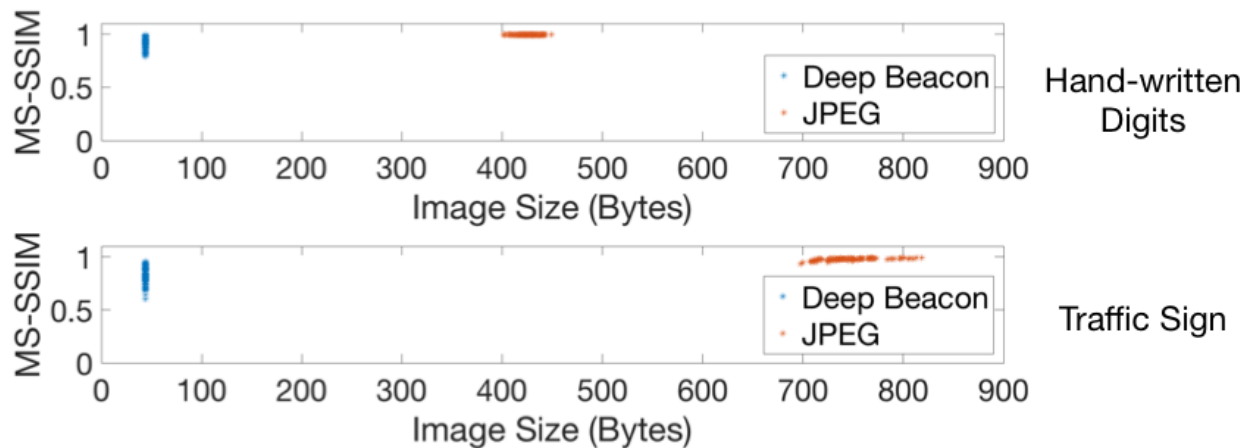


Figure 6.7: MS-SSIM vs. Image size for 100 hand-written digit images (upper plot) and 100 traffic sign images (lower plot). Images are compressed using JPEG (red dots) and the encoder module in Deep Beacon system (blue dots).

6.7.4 Performance on Different Image Types

It is useful to understand the performance of the Deep Beacon system for different types of image data. To derive the relation between the average image quality and the expected system lifetime for a certain type of image data, I fix several parameters in the system. I set the number of beacons in the system to be 1. I fix the image data delivery latency to the reader to be 0.5 second. For the system to fulfill this latency limit requirement, the BLE broadcasting frequency will be adjusted accordingly, results different expected system lifetime.

I plot the average MS-SSIM scores between the original uncompressed image and the compressed image vs. the expected system lifetime in months in Figure 6.8. As expected, the higher the expected system lifetime, the less data the system is able to broadcast within a fixed time window, results a lower quality image. Note that the average image quality does not change significantly until the system lifetime requirement goes above 37 months. On average, the system performs the best on simple hand-written digit images, having average MS-SSIM scores larger than 0.9. When I apply RGB images with complicated textures and structures, the distortion introduced in the compression yields a quality drop on the compressed image.

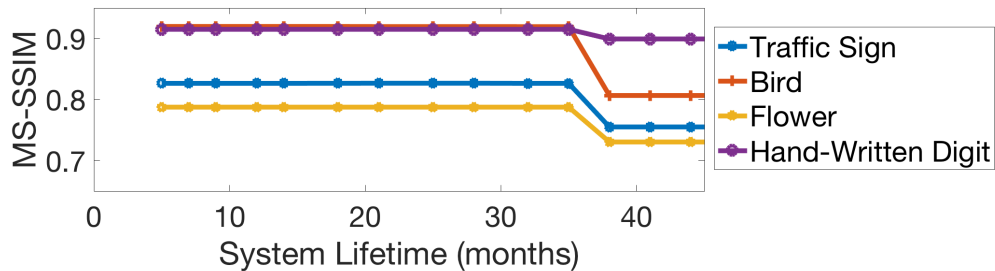


Figure 6.8: Average MS-SSIM score vs. expected system lifetime plot. Four curves represent four types of testing data. Each curve represents one type of test image data.

6.7.5 Impact of Number of Beacons

I also explore the case when multiple beacons are used together as a single storage and broadcast unit. In this case, each beacon's broadcasting frequency can be lower and the Deep Beacon system can still achieve the same broadcast bandwidth as a single-beacon Deep Beacon system does.

I test the cases when 1, 2, and 3 beacons are used, with the traffic sign image set. I plot the average MS-SSIM score vs. the expected system lifetime in Figure 6.9. The three curves represent the case when 1, 2, or 3 beacons are used, respectively. I mark the MS-SSIM score zero when the system is not able to broadcast the image data within the given fixed time limit for that system lifetime requirement. From the result I observe that when switching the number of beacons from 1 to 2, the system lasts much longer, from less than 4 years to more than 5 years and still broadcasts high quality images (average MS-SSIM score higher than 0.8). But adding another beacon to construct a 3-beacon system does not further improve the system performance. This is because the Deep Beacon system encodes an image into an embedding vector that can always be carried by up to 2 BLE broadcast packets.

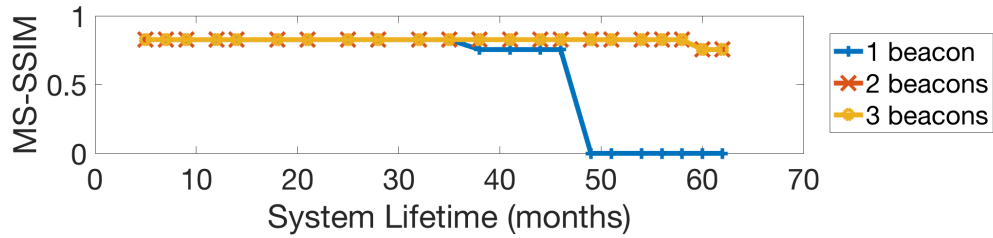


Figure 6.9: Average MS-SSIM score vs. expected system lifetime plot. Three curves represent the experiment result with using one, two, and three beacons, respectively.

6.7.6 User Study

I conduct a user study to evaluate the usefulness of Deep Beacon as a traffic sign storage and broadcast system and a hand-written digit storage and broadcast system. I store 18 images randomly picked from the traffic sign test image dataset and the hand-written digit test image dataset. The images are encoded and stored in one BLE 4 beacon, read back from the beacon, and decoded. Samples of compressed images are shown as the first eight images (starting from the left) on the last row of Figure 6.5. Each decoded image is shown to a participant. For traffic sign images, the participant is asked to identify the traffic sign labels by choosing one of the ten reference traffic sign labels shown in Figure 6.10. The ten reference traffic sign labels cover all 18 test traffic sign images. For hand-written digit images, the participant is asked to write down the number he or she thinks

that is in the image. The Deep Beacon is configured to use length 10 embedding, with 4 bytes for every digit in the embedding.

I collect result data from 15 participants. I group the result data into lowest $\frac{1}{3}$, medium $\frac{1}{3}$, and highest $\frac{1}{3}$ quality measured by MS-SSIM scores. I then compute the average chance of one participant making a correct guess in each group. The result is shown in Figure 6.11. The result suggests that Deep Beacon greatly preserves the information in hand-written digit images so that the correctness ratio is close to 1. For traffic sign images, the performance varies more. For high-quality images, almost every participant correctly identifies the meaning of the traffic sign. For the low-quality group, the chance of success drops significantly. This is caused by the higher appearance variation in the training image data that contains images in different lighting conditions and viewing angles (as shown in the 3rd row in Figure 6.5).



Figure 6.10: The traffic sign types used in user study.

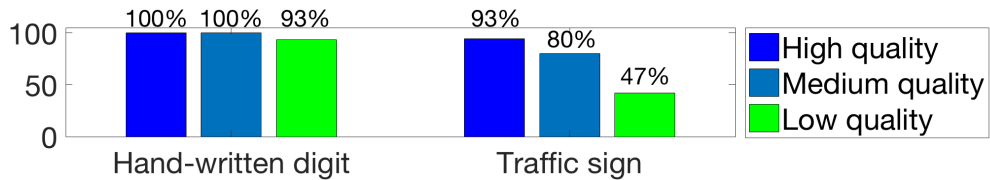


Figure 6.11: The chance in percentage of a participant making a correct answer in recognizing hand-written digit images and traffic sign images compressed by Deep Beacon system.

6.7.7 Comparison with Other Image Beacon Systems

I compare the proposed Deep Beacon with the binary image beacon system and the color image beacon system described in Chapter 4. I look at the following characteristics:

- *Does the system support color images?*
- *Is an offline training with an image dataset required?*
- *Is the writer required to take multiple images for a write?*
- *The typical latency for a reader to receive the image.*

I list the comparison result in TABLE 6.1. The advantage of Deep Beacon over the previous color image beacon system is that the image writer does not need to take multiple images in preparing the image write. This is a necessary step in the color image beacon system described in Chapter 4 because the depth information needs to be extracted from multiple input images.

The Deep Beacon system has the lowest latency ($\approx 0.5s$) in delivering an image to the image reader. The latency corresponds to the use of 4 BLE beacons and an expected lifetime of 4 years. Like the binary image beacon system, the Deep Beacon system also requires offline training for each image type.

Table 6.1: Comparison among three image beacon systems.

	Binary Image Beacon	Color Image Beacon	Deep Beacon
Color Image Support	No	Yes	Yes
Require Offline Training	Yes	No	Yes
Require Taking Multiple Images	No	Yes	No
Typical Latency	$\approx 1s$	$\approx 8s$	$\approx 0.5s$

6.8 Summary

In this chapter, I present the third type of image beacon system called the Deep Beacon. It uses the state-of-the-art deep learning models composed by an autoencoder VAE-GAN. I show that by applying VAE-GAN as the compressor and decompressor, it is possible to save and broadcast one

RGB image using a single BLE broadcasting packet. The resultant image is of high quality and carries meaningful information. I perform quantitative analysis on my Deep Beacon, and I compare its performance with two previously developed image beacon systems.

CHAPTER 7

CONCLUSION AND DISCUSSION

This chapter concludes the dissertation. Section 7.1 reviews the contributions of this dissertation. Section 7.2 discusses potential future research directions. Some remaining technical issues are explained in Section 7.3.

7.1 Summary of Contributions

7.1.1 Microscopy Video Compression

For the projects that I worked on with the application of microscopy compression, I made three major contributions. First, I invented a new video compression technique that is based on adjacent-pixel over time correlation scores (Shao et al., 2015). To the best of my knowledge, this is the first video compression technique in the literature of this kind. The compression technique integrates a correlation-based video frame segmentation technique that makes use of the PSF. Second, I proposed a new way to evaluate the quality of a compressed microscopy video based on statistical tests (Shao et al., 2018). Third, the two new video compression methods I invented achieve better performance comparing to H.264 video compression standard. Having the same compressed video quality, the analysis-preserving compression achieves up to 20x better compression than H.264. The analysis-aware compression achieves up to 1000x compression. The work in microscopy compression also results in a patent: (Russell et al., 2017).

My segmentation technique can be applied to a wide variety of videos, as long as the video is taken by an optical system that involves the PSF. I tested applying my PSF-based video frame segmentation method on a video taken by a human-scale camera. Figure 7.1 shows one frame of

a football field video, and the segmentation on that frame with my method. My method correctly identifies the moving part of the video. The background regions in the segmentation are the stationary football field. During a compression, these regions can be represented by a fixed football field texture.

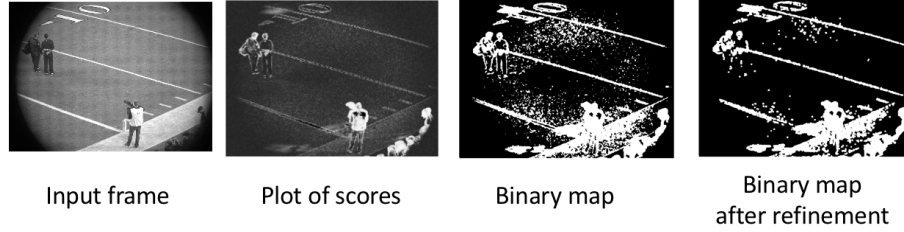


Figure 7.1: A sample football field image taken by a human-scale camera, and processed by the PSF-based segmentation method.

7.1.2 Image Compression for BLE Beacons

I worked on three projects on compressing image data to enable image storage and broadcast via BLE broadcasting channels. My major contribution on this topic is three-fold: first, I invented the first binary image beacon system that enables 64×64 binary image storage and broadcast with BLE beacons (Shao et al., 2016a). Second, I designed and built the first color image beacon system that takes an RGB input image and compresses it into less than 200 bytes (Shao and Nirjon, 2017). This enables affordable RGB image data broadcasting via BLE. Third, I applied the VAE-GAN model to encode a 64×64 RGB image into as small as 10 bytes. This makes it possible to carry an RGB image data with one BLE broadcasting packet.

In building these image beacon systems, I designed new algorithms for my custom image compression. They include a method to reduce patch dictionary size for binary image compression, an adaptive image encoding method for color image compression, and an IMU-guided image capture process that shortens the average image capture time.

These new algorithms can be generalized and be used in other systems. The adaptive image encoding algorithm can be applied in other systems where there is a strict limitation on storage

space and broadcasting capacity. The IMU-guided capture technique can be applied to other systems that involve multi-view depth estimation.

Moreover, I performed a comprehensive study to evaluate the image beacon systems with a set of criteria including system life, image quality, and the number of beacons. The study includes both theoretical derivation and empirical experiments. The experiments I conducted include: measuring the patch matching performance with different test image types (hand-written image, simple and complex geometric shapes) in binary image compression, measuring the adaptive image encoding performance with different types of color images taken indoors and outdoors, evaluating IMU-guided image capture performance on average time savings on capturing a good pair of images for depth estimation in two indoor environments with different lighting conditions, and measuring the compressed image quality under a set of embedding length settings in the VAE-GAN encoder.

7.1.3 MARBLE: Augmented Reality Application Data Compression for Bluetooth Low Energy Devices

The contributions from the project MARBLE include overcoming the challenge in building a robust indoor localization using BLE RSSI and the design and implementation of the first indoor sensor fusion system that uses BLE signals. The system presents a novel use of BLE beacons for both content broadcasting and localization. The system has the benefit of low power consumption. The battery-powered MARBLE system lasts seven years. The system also has low cost. The cost of building MARBLE is around \$200, which is 20 times cheaper than some of the state-of-art indoor AR solutions such as Microsoft HoloLens.

I designed an ORB visual feature selection algorithm to support ORB feature broadcasting via BLE broadcasting channels for indoor localization. My feature selection algorithm outperforms existing techniques in selecting visual features for camera localization and pose estimation. This is because it takes both the uniqueness and spatial location of a visual feature into consideration. The algorithm can be generalized for selecting other types of visual features. The indoor localization technique I invented that combines BLE, IMU and camera is the first indoor localization technique

that combines these three sensor signals. It has a higher accuracy comparing to existing techniques that uses two or one of the three sensor signals. This technique can be applied to other tasks including indoor motion planning.

7.2 Discussion and Future Work

7.2.1 Microscopy Video Compression

For the videos that were analyzed in the microscopy video compression projects, I selected the largest threshold value that still produced analysis output identical to that from the original videos (the compression included all of the steps described below). This requires running the compression and analysis pipelines several times for each video, so is not efficient.

In future work, I seek a closed-form approach to the selection of the threshold value. Otsu's method (Otsu, 1975) fails in the case where there is either no foreground or no background in a given video. Because it is forced to generate two classes, it wrongly classifies a certain portion of background and foreground in all cases. I am investigating camera-noise-estimation methods to automatically determine an appropriate threshold.

7.2.2 Image Compression for BLE Beacons

The two image beacon systems are most useful when the application scenario requires years-long working time without maintenance. In the future, it is meaningful to design a long timescale experiment to analyze the data write-read pattern within a long period.

The color image beacon system will perform at its best with beacons that adopt the upcoming Bluetooth 5.0 standard. Future work is to evaluate the different aspects of the system performance as Bluetooth 5.0 is released and gets popular. Moreover, a 'fat-beacon' standard is under development at Google (Hardill, 2016), which allows an even higher broadcast transmission capacity for BLE beacons. The goal of that standard is to equip beacon devices with the ability to broadcast basic

web contents to smartphones in the absence of the Internet connectivity. It will be meaningful to study the application of the image beacon system combined with a fat beacon.

The proposed image beacon system only considers stationary objects. This is an inherent problem of any depth estimation technique. In such case, we have to resort to texture or color-based segmentation.

The IMU-based prediction algorithm uses a regression tree model. Its prediction accuracy is lower indoors than outdoors. A robust model may train a separate regression tree for different cases, such as one model for taking images on objects below the phone, one model for objects of the same height as the phone.

A property of marker-controlled watershed segmentation algorithms is that they always generate a clean segmentation result. The combined segmentation method does not fully exploit this feature. I could further enhance the power of combining depth estimation and watershed results by deploying a smarter foreground region growing method.

Potential future work on the deep beacon system includes adding support for a larger image size by redesigning the VAE-GAN model. Also, when many types of image data need to be supported simultaneously, the cost of storing a broad set of models could bring issues in deploying the Deep Beacon system on a storage-constrained mobile device. Yao et al. (Yao et al., 2017) proposed a generic deep network model compression technique for mobile applications. Raghavan et al. (Raghavan et al., 2017) explored a bit-regularizing approach to the model compression problem. It is useful to explore the possibility of adopting a model compression technique to my system. Finally, note that the GAN model can be used to generate other types of multimedia data including speech data (Kaneko et al., 2017). It would be useful if the Deep Beacon system can be extended to support other types of multimedia data.

Adding the support for animated images (GIFs, etc.) into an image beacon increases the usefulness of the system. Common inter-picture image prediction techniques need to be re-designed to fit in the low-bandwidth constraint from BLE broadcasting protocol.

7.2.3 MARBLE: Augmented Reality Application Data Compression for Bluetooth Low Energy Devices

The MARBLE system is a realization of using BLE beacons for indoor augmented reality. Starting from MARBLE, several possible extensions can be developed, and research can be conducted.

In the capture phase, if the object of interest is a human body, with face recognition and face tracking techniques, the accurate orientation of the person's face can be estimated. The face details can be captured. Therefore, the additional face information can be stored and broadcasted in the system. This will enable visualization of the face as a part of the avatar. Besides that, the entire shape information may be captured and represented as a 3D mesh, to replace the key points information. A mesh compression method such as (Brettle and Galligan, 2017) or (Valette and Prost, 2004) that reduces the 3D mesh data size can be applied.

To enhance the user experience with an avatar that contains more details of the object of interest, another approach is to store both the texture information and shape information. The challenge for this is that the texture information usually requires thousands of bytes of storage space, which exceeds the system's storage capacity.

Depending on the viewer's device, more types of input signals including GPS or infrared camera can be added into the sensor fusion model. MARBLE should make use of the three input signals described in Chapter 5 as the basis for the localization and pose estimation task, and it should be able to accommodate new types of sensor inputs when they are available.

7.3 Remaining Technical Issues

I would like to list a set of remaining technical issues for the systems I built.

7.3.1 MARBLE: Generic Gesture Capture and Rendering

In demonstrating the MARBLE system, I applied the most basic setting: a color marker guided head and hand location detection, followed by rendering an avatar composed of simple geometric shapes. I would like to replace the marker-guided system with a more generic head and hand

detection with state-of-the-art computer vision techniques. I would also integrate a more realistic avatar model into the rendering component. An analysis can be performed to evaluate the user experience improvement after the change.

7.3.2 Smart Packet Rotation Strategy in Bluetooth Low Energy Broadcasting

From the development of an image beacon demo that divides image data into multiple data packets, I realized that a simple round-robin strategy for packet switching is suboptimal when the scanner has a scanning rate lower than the packet switching rate. When one packet is not received by the scanner, another rotation round is required for the scanner to get that packet.

A smart packet rotation strategy can be used in this case to decrease the average time for a scanner to receive all the packets. One observation on the BLE packet loss pattern is packet loss burstness (Wei et al., 2007): when one packet loss happens, the successive packets are more likely to be lost. This can be used in designing the smart packet rotation strategy.

7.3.3 No-Calibration Deployment for MARBLE

The MARBLE project has the goal of enabling a new indoor AR system with BLE infrastructure. However, the current implementation of MARBLE may not work in certain types of indoor environments. Indoor spaces with irregular boundary shapes may bring trouble for the BLE-based indoor localization. More tests need to be done to verify MARBLE's limitation. Noisy indoor spaces with Wi-Fi signals may also impact the BLE-based localization accuracy. In that case, the noise parameter for BLE-based estimation in the Kalman filter needs to be adjusted accordingly.

BIBLIOGRAPHY

- How many photographs of you are out there in the world? <https://www.theatlantic.com/technology/archive/2015/11/how-many-photographs-of-you-are-out-there-in-the-world/413389/>, 2015.
- Bluetooth 5.0 Press Release. <http://tinyurl.com/hvrosf5>, 2016.
- Send picture via lorawan? <https://www.thethingsnetwork.org/forum/t/send-picture-via-lorawan/2407/5>, 2016.
- Lightblue bean. punchthrough.com/bean, 2017.
- Raspberry pi. <https://www.raspberrypi.org/>, 2017.
- Google cloud machine learning engine. <https://cloud.google.com/ml-engine/>, 2017.
- High-throughput screening. https://en.wikipedia.org/wiki/High-throughput_screening, 2017.
- Microsoft hololens. <https://www.microsoft.com/hololens>, 2017.
- Geocaching. <https://www.geocaching.com/play>, 2018.
- Lpwan. https://en.wikipedia.org/wiki/LPWAN#LoRa_based, 2018.
- Bluetooth low energy. https://en.wikipedia.org/wiki/Bluetooth_Low_Energy, 2018.
- Point spread function. https://en.wikipedia.org/wiki/Point_spread_function, 2018.
- F. Ababsa. Advanced 3d localization by fusing measurements from gps, inertial and vision sensors. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 871–875. IEEE, 2009.
- A. Agarwal, C. Jawahar, and P. Narayanan. A survey of planar homography estimation techniques. *Centre for Visual Information Technology, Tech. Rep. IIIT/TR/2005/12*, 2005.
- B. Al Delail, L. Weruaga, M. J. Zemerly, and J. W. Ng. Indoor localization and navigation using smartphones augmented reality and inertial tracking. In *Electronics, Circuits, and Systems (ICECS), 2013 IEEE 20th International Conference on*, pages 929–932. IEEE, 2013.
- A. Amer and E. Dubois. Fast and reliable structure-oriented video noise estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(1):113–118, 2005.
- M. Ansari and R. Anand. Context based medical image compression for ultrasound images with contextual set partitioning in hierarchical trees algorithm. *Advances in Engineering Software*, 40(7):487–496, 2009.

- Arducam. Arducam camera. arducam.com, 2017.
- J. Aulinas, Y. R. Petillot, J. Salvi, and X. Lladó. The slam problem: a survey. In *CCIA*, pages 363–371. Citeseer, 2008.
- G. B. Avinash. Image compression and data integrity in confocal microscopy. *Scanning*, 17(3): 156–160, 1995.
- R. V. Babu and A. Makur. Object-based surveillance video compression using foreground motion compensation. In *Control, Automation, Robotics and Vision, 2006. ICARCV'06. 9th International Conference on*, pages 1–6. IEEE, 2006.
- X. Bai, J. S. Jin, and D. Feng. Segmentation-based multilayer diagnosis lossless medical image compression. In *Proceedings of the Pan-Sydney area workshop on Visual information processing*, pages 9–14. Australian Computer Society, Inc., 2004.
- J. Balle, A. Stojanovic, and J.-R. Ohm. Models for static and dynamic texture synthesis in image and video compression. *IEEE Journal of Selected Topics in Signal Processing*, 5(7):1353–1365, 2011.
- H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.
- C. Bergeron and C. Lamy-Bergor. Complaint selective encryption for h. 264/avc video streams. In *Multimedia Signal Processing, 2005 IEEE 7th Workshop on*, pages 1–4. IEEE, 2005.
- T. Bernas, E. K. Asem, J. P. Robinson, and B. Rajwa. Compression of fluorescence microscopy images based on the signal-to-noise estimation. *Microscopy research and technique*, 69(1):1–9, 2006.
- J. Brettle and F. Galligan. Introducing Draco: compression for 3D graphics. <https://opensource.googleblog.com/2017/01/introducing-draco-compression-for-3d.html>, 2017.
- M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital Equipment Corporation, 1994.
- Y. Cao, C. Ritz, and R. Raad. Adaptive and robust feature selection for low bitrate mobile augmented reality applications. In *Signal Processing and Communication Systems (ICSPCS), 2014*. IEEE, 2014.
- A. J. Champandard. Semantic style transfer and turning two-bit doodles into fine artworks. *arXiv preprint arXiv:1603.01768*, 2016.
- J. Chao and E. Steinbach. Preserving sift features in jpeg-encoded images. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, pages 301–304. IEEE, 2011.
- J. Chao and E. Steinbach. Sift feature-preserving bit allocation for h. 264/avc video compression. In *Image Processing (ICIP), 2012 19th IEEE International Conference on*, pages 709–712. IEEE, 2012.

- J. Chao, A. Al-Nuaimi, G. Schroth, and E. Steinbach. Performance comparison of various feature detector-descriptor combinations for content-based image retrieval with jpeg-encoded query images. In *Multimedia Signal Processing (MMSP), 2013 IEEE 15th International Workshop on*, pages 029–034. IEEE, 2013a.
- J. Chao, H. Chen, and E. Steinbach. On the design of a novel jpeg quantization table for improved feature detection performance. In *Image Processing (ICIP), 2013 20th IEEE International Conference on*, pages 1675–1679. IEEE, 2013b.
- J. Chao, R. Huitl, E. Steinbach, and D. Schroeder. A novel rate control framework for sift/surf feature preservation in h. 264/avc video compression. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(6):958–972, 2015a.
- J. Chao, E. Steinbach, and L. Xie. Keypoint encoding and transmission for improved feature extraction from compressed images. In *Multimedia and Expo (ICME), 2015 IEEE International Conference on*, pages 1–6. IEEE, 2015b.
- K. C. Cheung, S. S. Intille, and K. Larson. An inexpensive bluetooth-based indoor positioning hack. In *Proceedings of UbiComp*, volume 6, 2006.
- C. Christopoulos, A. Skodras, and T. Ebrahimi. The jpeg2000 still image coding system: an overview. *IEEE transactions on consumer electronics*, 46(4):1103–1127, 2000.
- C.-T. Chu, D. Anastassiou, and S.-F. Chang. Hybrid object-based/block-based coding in video compression at very low bit-rate. *Signal Processing: Image Communication*, 10(1-3):159–171, 1997.
- CISMM. Video spot tracker. <http://cismm.cs.unc.edu/resources/software-manuals/video-spot-tracker-manual/>, 2017.
- G. Conte, M. De Marchi, A. A. Nacci, V. Rana, and D. Sciuto. Bluesentinel: a first approach using ibeacon for an energy efficient occupancy detection system. In *1st ACM International Conference on Embedded Systems For Energy-Efficient Buildings (BuildSys)*, 2014.
- J. Cribb, L. D. Osborne, J. P.-L. Hsiao, L. Vicci, A. Meshram, E. T. OBrien III, R. C. Spero, R. Taylor, and R. Superfine. A high throughput array microscope for the mechanical characterization of biomaterials. *Review of Scientific Instruments*, 86(2):023711, 2015.
- M. De Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 2000.
- A. Dementyev, S. Hodges, S. Taylor, and J. Smith. Power consumption analysis of bluetooth low energy, zigbee and ant sensor nodes in a cyclic sleep scenario. In *Wireless Symposium (IWS), 2013 IEEE International*, pages 1–4. IEEE, 2013.
- O. Deniz, J. Paton, J. Salido, G. Bueno, and J. Ramanan. A vision-based localization algorithm for an indoor navigation app. In *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, pages 7–12. IEEE, 2014.

- E. L. Denton, S. Chintala, R. Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.
- M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on robotics and automation*, 17(3):229–241, 2001.
- Estimote. Estimote Beacons, 2017. <https://estimote.com/>.
- R. Faragher and R. Harle. An analysis of the accuracy of bluetooth low energy for indoor positioning applications. In *Proceedings of the 27th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2014), Tampa, FL, USA*, volume 812, 2014.
- R. Faragher and R. Harle. Location fingerprinting with bluetooth low energy beacons. *IEEE Journal on Selected Areas in Communications*, 33(11):2418–2428, 2015.
- M. K. A. Ganesan, S. Singh, F. May, and J. Becker. H. 264 decoder at hd resolution on a coarse grain dynamically reconfigurable architecture. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 467–471. IEEE, 2007.
- L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.
- L. A. Gatys, M. Bethge, A. Hertzmann, and E. Shechtman. Preserving color in neural artistic style transfer. *arXiv preprint arXiv:1606.05897*, 2016.
- C. Gomez, J. Oller, and J. Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9):11734–11753, 2012.
- I. Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.
- I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.
- D. Grois and O. Hadar. Advances in region-of-interest video and image processing. *Multimedia Networking and Coding*, pages 76–123, 2012.
- K. Guan, L. Ma, X. Tan, and S. Guo. Vision-based indoor localization approach based on surf and landmark. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2016 International*, pages 655–659. IEEE, 2016.
- R. W. Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2): 147–160, 1950.
- B. Hardill. Physical web fatbeacons. <https://www.hardill.me.uk/wordpress/2016/09/06/physical-web-fat-beacons/>, 2016.

- H. Hirschmuller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 807–814. IEEE, 2005.
- D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.
- Y. Inoue, A. Sashima, and K. Kurumatani. Indoor positioning system using beacon devices for practical pedestrian navigation on mobile phone. In *International Conference on Ubiquitous Intelligence and Computing*, pages 251–265. Springer, 2009.
- P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*, 2016.
- J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.
- S. Kajioka, T. Mori, T. Uchiya, I. Takumi, and H. Matsuo. Experiment of indoor position presumption based on rssi of bluetooth le beacon. In *2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE)*, pages 337–339. IEEE, 2014.
- R. E. Kalman and R. S. Bucy. New results in linear filtering and prediction theory. *Journal of basic engineering*, 1961.
- T. Kaneko, H. Kameoka, N. Hojo, Y. Ijima, K. Hiramatsu, and K. Kashino. Generative adversarial network-based postfilter for statistical parametric speech synthesis. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 4910–4914. IEEE, 2017.
- M. Karczewicz and J. Ridge. Context-based adaptive variable length coding for adaptive block transforms, Sept. 21 2004. US Patent 6,795,584.
- S.-D. Kim, J.-H. Lee, and J.-K. Kim. A new chain-coding algorithm for binary images using run-length codes. *Computer Vision, Graphics, and Image Processing*, 41(1):114–128, 1988.
- D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- T. Kirkman. Statistics to use: Kolmogorov-smirnov test. <http://www.physics.csbsju.edu/stats/KS-test.html>, 1996.
- P. Korshunov and W. T. Ooi. Video quality for face detection, recognition, and tracking. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 7(3):14, 2011.
- C. H. Lampert. Machine learning for video compression: Macroblock mode decision. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 1, pages 936–940. IEEE, 2006.

- A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015.
- Y. LeCun, C. Cortes, and C. J. Burges. The mnist database of handwritten digits, 1998.
- C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint arXiv:1609.04802*, 2016.
- Y. Liu, Z. G. Li, and Y. C. Soh. Region-of-interest based resource allocation for conversational video communication of h. 264/avc. *IEEE transactions on circuits and systems for video technology*, 18(1):134–139, 2008.
- Y. Liu, Z. Qin, Z. Luo, and H. Wang. Auto-painter: Cartoon image generation from sketch by using conditional generative adversarial networks. *arXiv preprint arXiv:1705.01908*, 2017.
- J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.
- D. G. Lowe. Object recognition from local scale-invariant features. In *The proceedings of the seventh IEEE international conference on Computer vision*. Ieee, 1999.
- T. Lu, Z. Le, and D. Yun. Piecewise linear image coding using surface triangulation and geometric compression. In *Data Compression Conference, 2000. Proceedings. DCC 2000*, pages 410–419. IEEE, 2000.
- F. Luan, S. Paris, E. Shechtman, and K. Bala. Deep photo style transfer. *arXiv preprint arXiv:1703.07511*, 2017.
- E. Malis and M. Vargas. *Deeper understanding of the homography decomposition for vision-based control*. PhD thesis, INRIA, 2007.
- D. Marpe, H. Schwarz, and T. Wiegand. Context-based adaptive binary arithmetic coding in the h. 264/avc video compression standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):620–636, 2003.
- P. Martin, B.-J. Ho, N. Grupen, S. Muñoz, and M. Srivastava. An ibeacon primer for indoor localization: demo abstract. In *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*, pages 190–191. ACM, 2014.
- E. Martinian, A. Behrens, J. Xin, A. Vetro, and H. Sun. Extensions of h. 264/avc for multiview video compression. In *Image Processing, 2006 IEEE International Conference on*, pages 2981–2984. IEEE, 2006.
- C. McAnlis and A. Haecky. *Understanding Compression: Data Compression for Modern Developers*. ” O’Reilly Media, Inc.”, 2016.

- B. Menser and M. Brunig. Face detection and tracking for video coding applications. In *Signals, Systems and Computers, 2000. Conference Record of the Thirty-Fourth Asilomar Conference on*, volume 1, pages 49–53. IEEE, 2000.
- H. Meuel, M. Munderloh, and J. Ostermann. Low bit rate roi based video coding for hdtv aerial surveillance video sequences. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 13–20. IEEE, 2011.
- M. Minervini and S. A. Tsaftaris. Application-aware image compression for low cost and distributed plant phenotyping. In *Digital Signal Processing (DSP), 2013 18th International Conference on*, pages 1–6. IEEE, 2013.
- S. A. Mohamed and M. M. Fahmy. Binary image compression using efficient partitioning into rectangular regions. *Communications, IEEE Transactions on*, 43(5):1888–1893, 1995.
- N. Monnier, S.-M. Guo, M. Mori, J. He, P. Lénárt, and M. Bathe. Bayesian approach to msd-based analysis of particle motion in live cells. *Biophysical journal*, 103(3):616–626, 2012.
- D. Mukherjee, J. Bankoski, A. Grange, J. Han, J. Koleszar, P. Wilkins, Y. Xu, and R. Bultje. The latest open-source video codec vp9-an overview and preliminary results. In *Picture Coding Symposium (PCS), 2013*, pages 390–393. IEEE, 2013.
- N. Newman. Apple ibeacon technology briefing. *Journal of Direct, Data and Digital Marketing Practice*, 15(3):222–225, 2014.
- M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.
- S. Nirjon and J. A. Stankovic. Kinsight: Localizing and tracking household objects using depth-camera sensors. In *2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems*, pages 67–74. IEEE, 2012.
- R. D. Nowak and R. G. Baraniuk. Wavelet-domain filtering for photon imaging systems. *IEEE Transactions on Image Processing*, 8(5):666–678, 1999.
- V. Oculus. Oculus rift. Available from WWW; <http://www.oculusvr.com/rift>, 2015.
- T. Oh and R. Besar. Jpeg2000 and jpeg: image quality measures of compressed medical images. In *Telecommunication Technology, 2003. NCTT 2003 Proceedings. 4th National Conference on*, pages 31–35. IEEE, 2003.
- N. Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.
- H. M. Parmar and P. Scholar. Comparison of dct and wavelet based image compression techniques. *International Journal Engineering Development and Research*, 2(1):664–669, 2014.

- K. Parvati, P. Rao, and M. Mariya Das. Image segmentation using gray-scale morphology and marker-controlled watershed transformation. *Discrete Dynamics in Nature and Society*, 2008, 2009.
- R. Pierdicca, D. Liciotti, M. Contigiani, E. Frontoni, A. Mancini, and P. Zingaretti. Low cost embedded system for increasing retail environment intelligence. In *Multimedia & Expo Workshops (ICMEW), 2015 IEEE International Conference on*, pages 1–6. IEEE, 2015.
- C. W. Quammen, A. C. Richardson, J. Haase, B. D. Harrison, R. M. Taylor, et al. Fluorosim: a visual problem-solving environment for fluorescence microscopy. In *Eurographics Workshop on Visual Computing for Biomedicine*, volume 2008, page 151. NIH Public Access, 2008.
- A. Raghavan, M. Amer, and S. Chai. Bitnet: Bit-regularized deep neural networks. *arXiv preprint arXiv:1708.04788*, 2017.
- S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.
- I. E. Richardson. *The H. 264 advanced video compression standard*. John Wiley & Sons, 2011.
- J. J. Rissanen. Generalized kraft inequality and arithmetic coding. *IBM Journal of research and development*, 20(3):198–203, 1976.
- O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
- E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. IEEE, 2011.
- M. T. I. Russell, C. Shao, Z. Zhong, and K. D. Mayer-Patel. Methods, systems, and computer readable media for compressing video images, Aug. 24 2017. US Patent App. 15/506,725.
- A. Said and W. A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on circuits and systems for video technology*, 6(3):243–250, 1996.
- V. Sanchez, R. Abugharbieh, and P. Nasiopoulos. 3-d scalable medical image compression with optimized volume of interest coding. *IEEE Transactions on Medical Imaging*, 29(10):1808–1820, 2010.
- S. Santurkar, D. Budden, and N. Shavit. Generative compression. *arXiv preprint arXiv:1703.01467*, 2017.
- R. J. Schalkoff. *Digital image processing and computer vision*, volume 286. Wiley New York, 1989.
- J. Serra. *Image analysis and mathematical morphology, v. 1*. Academic press, 1982.

- C. E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.
- C. Shao and S. Nirjon. Imagebeacon: Broadcasting color images over connectionless bluetooth le packets. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, pages 121–132. ACM, 2017.
- C. Shao, A. Zhong, J. Cribb, L. D. Osborne, E. T. O’Brien, R. Superfine, K. Mayer-Patel, and R. M. Taylor. Analysis-preserving video microscopy compression via correlation and mathematical morphology. *Microscopy research and technique*, 78(12):1055–1061, 2015.
- C. Shao, S. Nirjon, and J.-M. Frahm. Years-long binary image broadcast using bluetooth low energy beacons. In *Distributed Computing in Sensor Systems (DCOSS), 2016 International Conference on*, pages 225–232. IEEE, 2016a.
- C. Shao, S. Nirjon, and J.-M. Frahm. Years-long binary image broadcast using bluetooth low energy beacons. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems (DCOSS 2016)*, 2016b.
- C. Shao, J. Cribb, L. D. Osborne, E. T. O’Brien III, R. Superfine, K. Mayer-Patel, and R. M. Taylor. Analysis-aware microscopy video compression. *Microscopy research and technique*, 2018.
- J. M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on signal processing*, 41(12):3445–3462, 1993.
- M.-Y. Shen and C.-C. J. Kuo. Review of postprocessing techniques for compression artifact removal. *Journal of visual communication and image representation*, 9(1):2–14, 1998.
- C. J. Sheppard, X. Gan, M. Gu, and M. Roy. Signal-to-noise ratio in confocal microscopes. *Handbook of biological confocal microscopy*, pages 442–452, 2006.
- K. Siddiqi and S. Pizer. *Medial representations: mathematics, algorithms and applications*, volume 37. Springer Science & Business Media, 2008.
- M. Siekkinen, M. Hienkari, J. K. Nurminen, and J. Nieminen. How low energy is bluetooth low energy? comparative measurements with zigbee/802.15. 4. In *Wireless Communications and Networking Conference Workshops (WCNCW), 2012 IEEE*, pages 232–237. IEEE, 2012.
- A. Signoroni and R. Leonardi. Progressive roi coding and diagnostic quality for medical image compression. In *Visual Communications and Image Processing’98*, volume 3309, pages 674–686. International Society for Optics and Photonics, 1998a.
- A. Signoroni and R. Leonardi. Progressive medical image compression using a diagnostic quality measure on regions-of-interest. In *Signal Processing Conference (EUSIPCO 1998), 9th European*, pages 1–4. IEEE, 1998b.
- J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, pages 1453–1460, 2011.

- S. M. Stigler. Francis galton's account of the invention of correlation. *Statistical Science*, pages 73–79, 1989.
- J. Ström and P. C. Cosman. Medical image compression with lossless regions of interest. *Signal processing*, 59(2):155–171, 1997.
- G. J. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012.
- L. Theis and M. Bethge. Generative image modeling using spatial lstms. In *Advances in Neural Information Processing Systems*, pages 1927–1935, 2015.
- D. Ulyanov, V. Lebedev, A. Vedaldi, and V. S. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, pages 1349–1357, 2016.
- S. Valette and R. Prost. Wavelet-based progressive compression scheme for triangle meshes: Wavemesh. *IEEE Transactions on Visualization and Computer Graphics*, 10(2):123–129, 2004.
- S. Van Leuven, K. Van Schevensteen, T. Dams, and P. Schelkens. An implementation of multiple region-of-interest models in h. 264/avc. In *Signal Processing for Image Enhancement and Multimedia Processing*, pages 215–225. Springer, 2008.
- K. D. S. S. Vatolin D, Grishin S. Lossless video codecs comparison. http://compression.ru/video/codec_comparison/lossless_codecs_2007_en.html, 2007.
- A. Wakatani. Digital watermarking for roi medical images by using compressed signature image. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 2043–2048. IEEE, 2002.
- G. K. Wallace. The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv, 1992.
- Z. Wang, E. P. Simoncelli, and A. C. Bovik. Multiscale structural similarity for image quality assessment. In *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, volume 2, pages 1398–1402. Ieee, 2003.
- Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612, 2004.
- D. X. Wei, P. Cao, and S. H. Low. Packet loss burstiness: measurements and implications for distributed applications. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8. IEEE, 2007.
- P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.
- T. Wiegand. Draft itu-t recommendation and final draft international standard of joint video specification. *ITU-T rec. H. 264—ISO/IEC 14496-10 AVC*, 2003.

- T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h. 264/avc video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003.
- R. Wollman and N. Stuurman. High throughput microscopy: from raw images to discoveries. *Journal of cell science*, 120(21):3715–3722, 2007.
- H. Xu, H. Zha, and M. A. Davenport. Manifold based dynamic texture synthesis from extremely few samples. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3019–3026, 2014.
- S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher. Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. ACM, 2017.
- Z. Yin and T. Kanade. Restoring artifact-free microscopy image sequences. In *Biomedical Imaging: From Nano to Macro, 2011 IEEE International Symposium on*, pages 909–913. IEEE, 2011.
- S. Zahir and M. Naqvi. A new rectangular partitioning based lossless binary image compression scheme. In *Electrical and Computer Engineering, 2005. Canadian Conference on*, pages 281–285. IEEE, 2005.
- S. Zahir, K. Dhou, and B. Prince George. A new chain coding based method for binary image compression and reconstruction. *PCS, Lisbon, Portugal*, pages 1321–1324, 2007.
- F. Zhang and D. R. Bull. Advances in region-based texture modeling for video compression. In *Proc. SPIE*, volume 8135, page 81350N, 2011.
- H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *arXiv preprint arXiv:1612.03242*, 2016.
- S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1529–1537, 2015.
- Y. Zhuang, J. Yang, Y. Li, L. Qi, and N. El-Sheimy. Smartphone-based indoor localization with bluetooth low energy beacons. *Sensors*, 16(5):596, 2016.
- J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.