

# Parameterized Data Reduction Framework for IoT Devices

Jyun-Jhe Chou

Graduate Institute of Networking and Multimedia  
Department of Computer Science and Information Engineering  
NTU IoX Research Center  
National Taiwan University

## Abstract

In a IoT environment, many devices will periodically transmit data. However, most of the data are redundant, but sensor itself may not have a good standard to decide transmit or not. Some static rule maybe useful on specific scenario, and become ineffective when we change the usage of the sensor. Hence, we design an algorithm to solve the problem of data redundant. In the algorithm, we iteratively separate an image into some smaller regions. Each round, choose a region with highest variability, and separate it into four regions. Finally, each region has different size and uses its average value to represent itself. If a area is more various, the density of regions will be higher. In this paper, we present a method to reduce the file size of thermal sensor which can sense the temperature of a surface and outputs a two dimension gray scale image. In our evaluation result, we can reduce the file size to 50% less than JPEG when there is 0.5% of distortion, and up to 93% less when there is 2% of distortion.

## I. INTRODUCTION

Walking exercises the nervous, cardiovascular, pulmonary, musculoskeletal and hematologic systems because it requires more oxygen to contract the muscles. Hence, *gait velocity*, or called *walking speed* [Middleton2015] has become a valid and important metric for senior populations [Middleton2015, studenski2011, Studenski03].

In 2011, Studenski et al [studenski2011] published a study that tracked gait velocity of over 34,000 seniors from 6 to 21 years in US. The study found that predicted survival rate based on age, sex, and gait velocity was as accurate as predicted based on age, sex, chronic conditions, smoking history, blood pressure, body mass index, and hospitalization. Consequently, it has motivated the industrial and academia communities to develop the methodology to track and assess the risk based on gait velocity. The following years have led to many papers that point to the importance of gait velocity as a predictor of degradation and exacerbation events associated with various chronic diseases including heart failure, COPD, kidney failure, stroke, etc [Studenski03, pulignano2016, Konthoraxjnl2015, kutner2015].

In the US, there are 13 million seniors who live alone at home [profile2015]. Gait velocity and stride length are particularly important in this case since they provide an assessment of fall risk, the ability to perform daily activities such as bathing and eating, and hence the potential for being independent. Assessment of gait velocity is recommended to instruct the subjects to walk back and forth in a 5, 8 or 10 meter walkway. Similar results were found in a study comparing a 3 meter walk test to the GAITRite electronic walkway in individuals with chronic stroke [Peters2013].

The above approaches are conducted either at the clinical institutes or designated locations. They are recommended by the physicians but are required to be conducted at limited time and location. Consequently, it is difficult to observe the change in long term. It is desirable for the elderly, their family members, and physicians to monitor gait velocity for the elderly all the time at any location. However, the assessment should take into account several factors, including accuracy, privacy, portability, robustness, and applicability.

Shih and his colleagues [Shih17b] proposed a sensing system to be installed at home or nursing institute without revealing privacy and not using wearable devices. Given the proposed method, one may deploy several thermal sensors in his/her apartments as shown in Figure 1. In this example, numbers of thermal sensors are deployed to increase the coverage of the sensing signals. In large spaces such as living room, there will be more than one sensor in one space; in small spaces such as corridor, there can be only one sensor. One fundamental question to ask is how many sensors should be deployed and how these sensors work together seamlessly to provide accurate gait velocity measurement.

In a IoT environment, many devices will periodically transmit data. Some sensor is use for avoid accidents, so they will have very high sensing frequency. However, most of the data are redundant. Like a temperature sensor on a gas stove, the temperature value is the same as the value from air conditioner and does not change very frequently, but it will have dramatically difference when we are cooking. We can simply make a threshold that when temperature is higher or lower than some degrees, the data will be transmitted, and drop the data that we don't interest. This is a very easy solution if we only have a few devices, but when we have hundreds or thousands devices, it is impossible to manually configure all devices, and the setting may need to change in the winter and summer, or different location.

In this paper, we study the data from Panasonic Grid-EYE, a  $8 \times 8$  pixels infrared array sensor, and FLIR ONE PRO, a  $480 \times 640$  pixels thermal camera. Both are setting on ceiling and taking a video of a person walking under the camera.

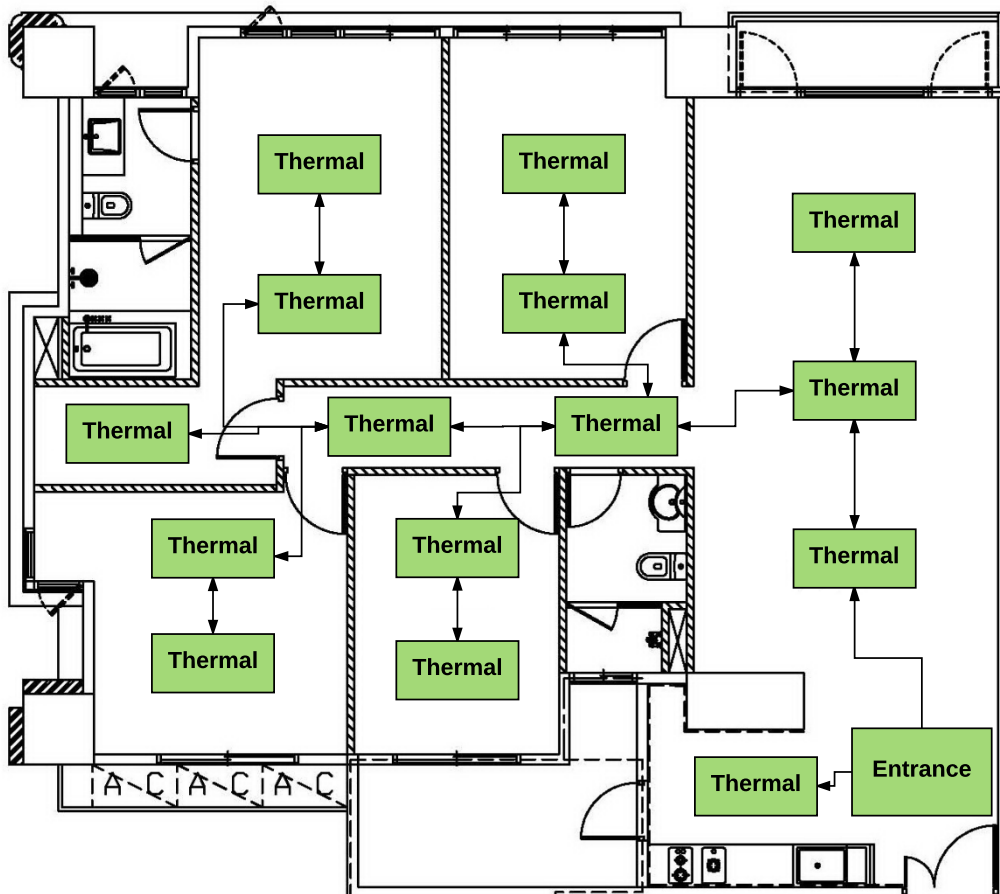


Fig. 1. Gait Velocity Measurement at Smart Homes

In Figure 1, there are fifteen thermal sensor in a house. If they are Panasonic Grid-EYE, it will have 2 bytes per pixel, 64 pixels per frame, 10 frames per second, and total need 1.7GB storage space per day. If they are FLIR ONE PRO, it can only generate 5 frames per second but needs about 45KB per frame, and it will need 291.6GB everyday.

**Contribution** The target of our work is to compress the thermal image retrieved from FLIR ONE PRO to targeted data size and keep the quality of data. Nearby pixels in a thermal image mostly have similar value, so we can easily separate an image into several regions and use its average value to represent it wont cause too much error. By the method we proposed, the size of file can reduce more than 50% compare to JPEG image when both have 0.5%(0.18°C) of root-mean-square error.

The remaining of this paper is organized as follow. Section II presents related works and background for developing the methods. Section III presents the system architecture, challenges, and the developed mechanisms. Section IV presents the evaluation results of proposed mechanism and Section V summaries our works.

## II. BACKGROUND AND RELATED WORKS

### A. Panasonic Grid-EYE Thermal Sensor

First, we study the sensor Panasonic Grid-EYE which is a thermal camera that can output  $8 \times 8$  pixels image with  $2.5^{\circ}\text{C}$  accuracy and  $0.25^{\circ}\text{C}$  resolution at 10 frames per second. In normal mode, the current consumption is 4.5mA. It is a low resolution camera and infrared array sensor, so we install it in our house at ease without some privacy issue that may cause by a surveillance camera.

When someone walks under a Grid-EYE sensor, we will see some pixels with higher temperature than others. Figure 2 shows an example of image from Grid-EYE sensor. The sensor value will look like a cone shape. The pixel with our head will have the highest temperature, body is lower, and leg is the lowest except background because when the distance from camera to our body is longer, the area cover by the camera will be wider and the ratio of background temperature in the pixel will increase, also our head do not cover by cloth, so the surface temperature will higher than other place. while we are walking in a area, the temperature of air in the area will become warmer, and the shape of human will be harder to recognize.

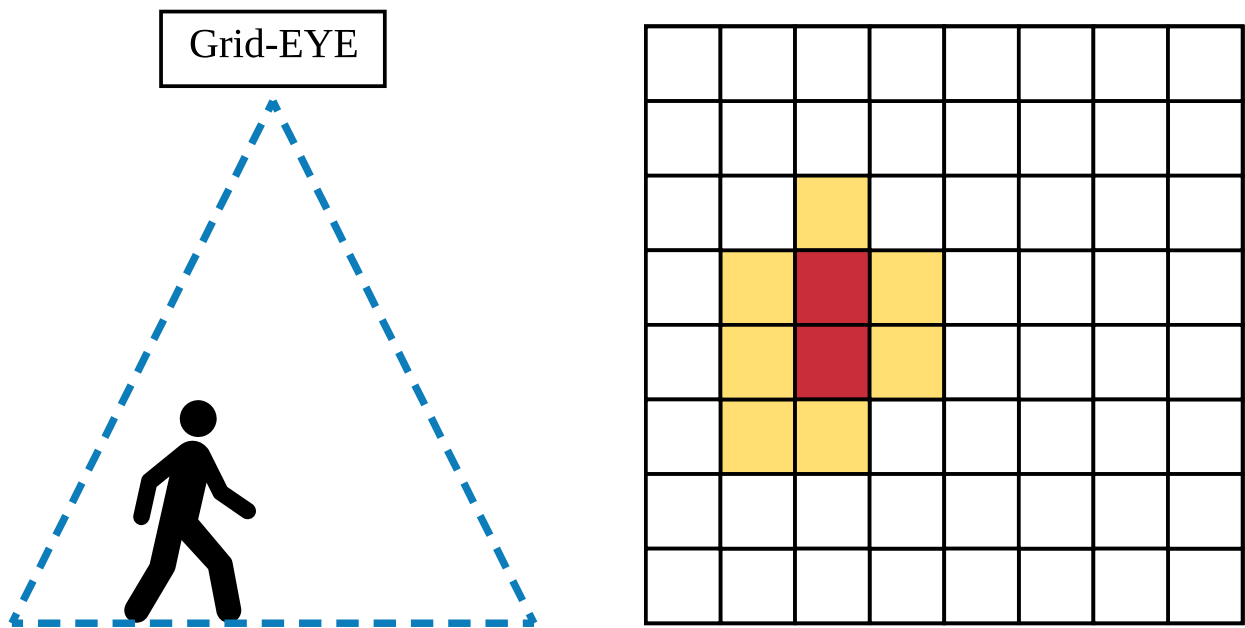


Fig. 2. Walking under a Grid-EYE sensor

The data we used is from a solitary elder's home. We deployed four Grid-EYE sensor at the corner of her living room, and recorded the thermal video for three weeks at 10 frames per second data rate.

## B. FLIR ONE PRO

FLIR ONE PRO can output a  $480 \times 640$  pixels image with  $3^\circ C$  accuracy and  $0.01^\circ C$  resolution, and capture video at about 5 FPS. In picture taking mode, it can retrieve the precise data from the header of picture file. However, in the video taking mode, it only store a gray scale video and show the range of temperature on the monitor. Hence, we use  $^\circ C$  in picture mode, and gray scale value as the unit to analyze error rate. Since FLIR ONE PRO can offer a image with about 5000 times number of pixels compare to Grid-EYE. It cannot simply use a Gaussian function to fit it. Hence, we developed a method to compress FLIR images. It can also treat as a normal image and be stored as jpeg, png, etc.

## C. Raspberry Pi 3

We use Raspberry Pi 3 as our testing environment. It has a 1.2 GHz 64-bit quad-core ARM Cortex-A53 CPU, 1 GB memory, and 802.11n wireless network. We run a Debian-based Linux operating system on it. While it is idling and turning off WiFi, it will consume 240mA and while uploading data at 24Mbit/s, it will consume 400mA.

## D. Simple Data Compressing

If we save a frame in a readable format, it will take about 380 bytes storage. However, the temperature range of our scenario mostly from  $5^\circ C$  to  $40^\circ C$  and the resolution is  $0.25^\circ C$ , so we can easily represent each temperature by one byte. Hence, we only need 64 bytes to store a frame. We have try several ways to compress the frame.

1) *Huffman Coding*: Huffman coding is a lossless data compressing. In average, it can reduce the frame size from 64 bytes to 40.7 bytes with 6 bytes standard deviation.

2) *Z-score Threshold*: We can only transmit the pixels with higher temperature since thermal sensors are mostly used for detect heat source. Z-score is define as  $z = \frac{\chi - \mu}{\sigma}$ , where  $\chi$  is the value of the temperature,  $\mu$  is the average of the temperature and  $\sigma$  is the standard deviation of the temperature. In our earlier work [Shih17b], we use Z-score instead of a static threshold to detect human because the background temperature may have a  $10^\circ C$  difference between day and night, and when people walk through the sensing area the Grid-EYE, the temperature reading will only increase  $2^\circ C$  to  $3^\circ C$ . Hence, it is impossible to use a static threshold to detect human. In [Shih17b], we only use the pixels with the Z-score higher than 2, so we can reduce the frame size from 64 bytes to 12.6 bytes with 2.9 bytes standard deviation by Z-score threshold 2 and compress by Huffman coding.

3) *Gaussian Function Fitting*: Since the shape of human in a thermal image looks like a cone, we may use a gaussian function to fit the image. A Gaussian function  $y = Ae^{-(x-B)^2/2C^2}$  has three parameter  $A, B$  and  $C$ . The parameter  $A$  is the height of the cone,  $B$  is the position of the cone's peak and  $C$  controls the width of the cone. We let the pixel with highest temperature be the peak of the cone, so we only need to adjust  $A$  and  $C$  to fit the image. Guo [guo2011simple] provide a fast way to get the fitting Gaussian function. In our testing, it will be about  $0.5^\circ C$  root-mean-square error, and only needs 5 bytes to store the position of peak and two parameters.

### III. DATA SIZE DECISION FRAMEWORK

This section presents the proposed method to outcome a data array than have less size compare to jpeg image when we can tolerate some error of data. We use the image captured by FLIR ONE PRO. In a thermal image, the temperature variation between nearby pixels are very small except the edge of objects. Hence, we can separate an image into several regions, and the pixels in a same region has similar value so we can use the average value to represent it and do not cause too much error. However, precisely separate an image into some polygon region takes a lot of computation time and hard to describe the edge of each region. Also, decide the number of region also a problem. Hence, to effectively describe regions we design that every region must be a rectangle, and every region can only separate into 4 regions by cut in half at the middle of horizontal and vertical. The image will start from only contains one region, and 3 regions will be added per round since we cut a region into 4 pieces.

#### A. Region Represent Grammar

For each frame, we can use a context-free language to represent it.

$$\begin{aligned} S &\rightarrow R \\ R &\rightarrow \alpha \\ R &\rightarrow \beta RRRR \end{aligned}$$

$R$  means a region of image, and it can either use the average  $\alpha$  of the pixels in the region to represent whole region or separate into four regions and left a remainder  $\beta$ . Dependence on the image size we desired, we can choose the amount of separating regions. The context-free grammar start from a region contain whole image. For each  $R$  we calculate a score which is based on the quality of data we can improve by separate it in to smaller regions. After some operation, we can encode the image into a string  $\omega$ . Figure 3 shows an example of image which was took by FLIR ONE PRO. One of the possible outcome

is Figure 4 if we separate the image 6 times and it will have 19 regions. By this method, we can iteratively separate an image until the number of regions reach our file size requirement.

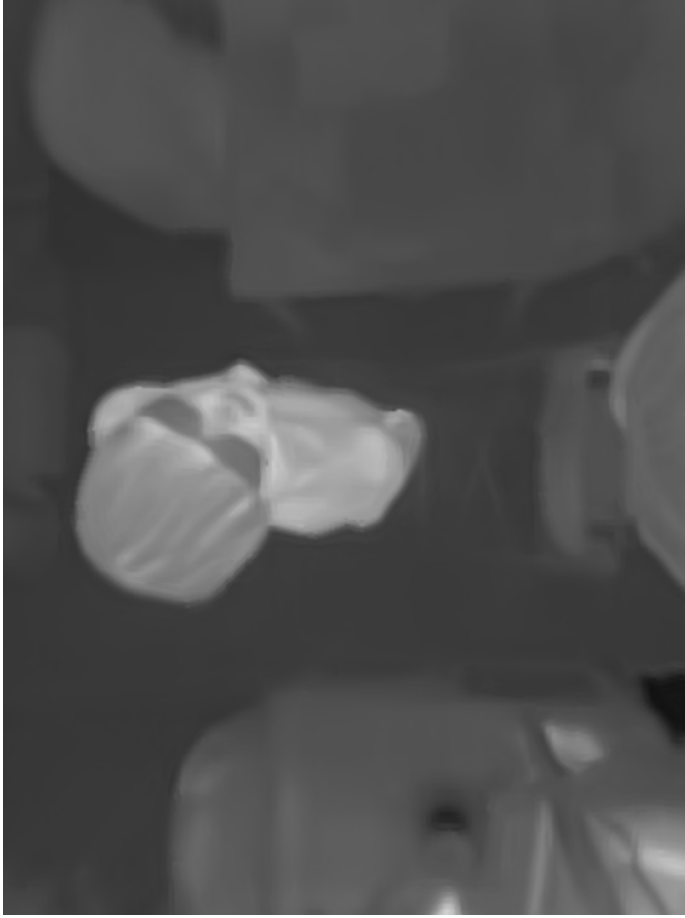


Fig. 3. PNG image, size = 46KB

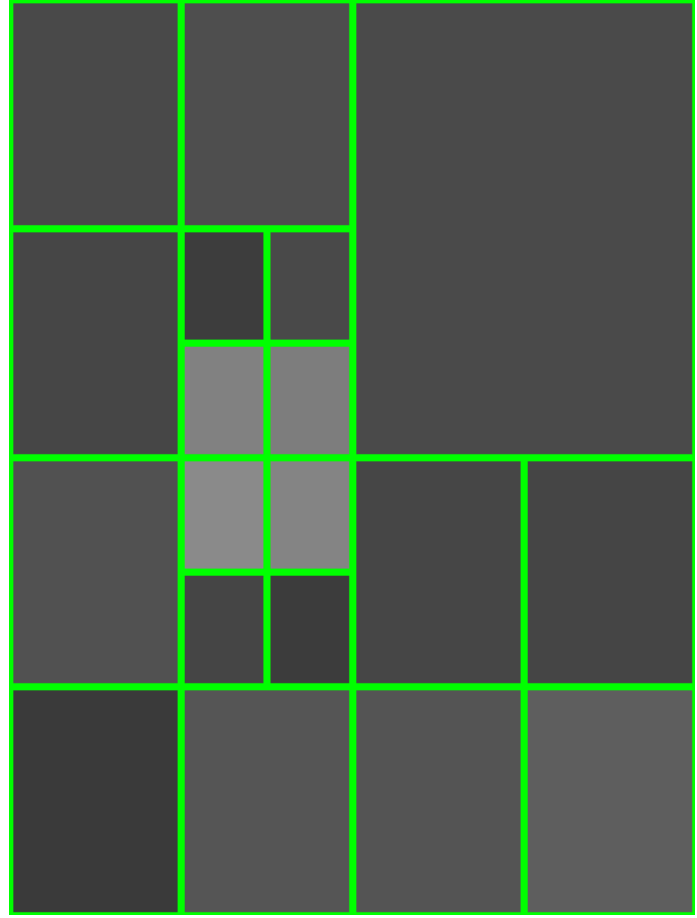


Fig. 4. Region separate by CFG

### B. Data Structure and Region Selection Algorithm

To help us choose which region to be separated, we give every region a score, and put them into a heap. For each round, we pick the region with the highest score, separate it into four subregions, calculate the score of subregions, and put them into the heap. We use the sum of square error of pixels in the region  $R$  as the score of the region.

$$\begin{aligned}\mu &= E(R) \\ \text{Score} &= \sum_{X \in R} (X - \mu)^2 \\ &= \sum_{X \in R} X^2 - |R|\mu^2\end{aligned}$$

By the equation shows above, we just need to know the sum of squared and the mean of a region, we can get its score. we can use a segment tree to store all possible regions and its scores. For each node, it

store the range on both width and height it covered, sum  $\sum_{X \in R} X$ , and squared sum  $\sum_{X \in R} X^2$  of pixels in the region. By the property of segment tree, tree root start from 0, and each node  $X_i$  has four child  $X_{i \times 4 + 1}$ ,  $X_{i \times 4 + 2}$ ,  $X_{i \times 4 + 3}$  and  $X_{i \times 4 + 4}$ . Hence, we only need to allocate an large array and recursively process all nodes form root. Algorithm 1 shows how we generate the tree.

---

**Algorithm 1** Segment Tree Preprocess
 

---

```

1: Tree = Array()
2: function SETTREENODE(x, left, right, top, bottom)
3:   if left = right top = bottom then
4:     Tree[x].Sum = Image[left][top]
5:     Tree[x].SquareSum = Image[left][top]2
6:   else
7:     setTreeNode(4x + 1, left, (left + right)/2, top, (top + bottom)/2)
8:     setTreeNode(4x + 2, (left + right)/2, right, top, (top + bottom)/2)
9:     setTreeNode(4x + 3, left, (left + right)/2, (top + bottom)/2, bottom)
10:    setTreeNode(4x + 4, (left + right)/2, right, (top + bottom)/2, bottom)
11:    Tree[x].Sum =  $\sum_{i=4x+1}^{4x+4} \textit{Tree}[i].\textit{sum}$ 
12:    Tree[x].SquareSum =  $\sum_{i=4x+1}^{4x+4} \textit{Tree}[i].\textit{SquareSum}$ 
13:    Tree[x].SquaredError = Tree[x].SquareSum -  $\frac{\textit{Tree}[x].\textit{Sum}^2}{(\textit{right}-\textit{left}+1) \times (\textit{bottom}-\textit{top}+1)}$ 
14:    setTreeNode(0, 0, Image.Width, 0, Image.Height)

```

---

For region selection, we use a priority queue to retrieve the region of considerate regions with highest score. The priority queue start with only root of the segment tree. For each round the priority queue pop the item with highest score and push all its child in to the queue. Algorithm 2 shows how we select a region by the priority queue. After the selection finished, we will generate the data string to be sent. The regions in *seperatedRegions* will be  $\beta$  and others in *PriorityQueue* will be the average value, and then compress the string by Huffman Coding.

---

**Algorithm 2** Region Selection
 

---

```

1: seperatedRegions = Array()
2: PriorityQueue = Heap()
3: PriorityQueue.Push(Tree[0].SquaredError, 0)
4: for
5:   i = 0..SeperateRounds do
6:     value, x = PriorityQueue.Pop()
7:     seperatedRegions.push(x)
8:     PriorityQueue.Push(Tree[4x + 1].SquaredError, 4x + 1)
9:     PriorityQueue.Push(Tree[4x + 2].SquaredError, 4x + 2)
10:    PriorityQueue.Push(Tree[4x + 3].SquaredError, 4x + 3)
11:    PriorityQueue.Push(Tree[4x + 4].SquaredError, 4x + 4)

```

---



The complexity of our algorithm can be separated into 3 parts. First part is to initialize the segment tree. The size of segment is depends on the size of the image. If the number of pixels is  $N$ , the height of segment tree is  $O(N\log(N))$ , and the number of nodes will be  $O(N)$ . The time complexity of initialize is  $O(N)$ . Second part is loading the image. It will need to traverse whole tree from leaf to root. Since segment tree can be store in an array, it also takes  $O(N)$  time to load the image. Third part is to separate regions. For each round, we pop an element from heap and push four elements into heap. If we have separated image  $K$  times, the size of heap will be  $3K + 1$ . Time complexity of pop and push will be  $O(\log(K))$ , and do it  $5K$  times will be  $O(K\log(K))$ .

#### IV. PERFORMANCE EVALUATION

To evaluate the effectiveness of the proposed method, we do the different ratios of compressing on a thermal image by our method compare to JPEG image using different quality and png image, a lossless bit map image. We set the camera at the ceiling and view direction is perpendicular to the ground, and the image size is  $480 \times 640$  pixels. The JPEG image is generated by OpenCV 3.3.0, and image quality from 1 to 99.

Figure 5 and Figure 6 show the different of JPEG and our method. JPEG image id generated by image quality level 3, and image of our method does 1390 rounds of separate and compressed by Huffman Coding. In this case, Huffman Coding can reduce 39% of our image size.

Figure 7 shows that the size of file can reduce more than 50% compare to JPEG image when both have 0.5%(0.18°C) of root-mean-square error. Our method has 82% less error rate when both size are 4KB image. The percentage of file size is compare to PNG image.

The computing time on Raspberry Pi 3 is:

- 1) *Date Structure Initialize*: XXX.XXX second.
- 2) *Image Loading*: XXX.XXX second.
- 3) *Region Separation*: XXX.XXX second. (a chart)

#### V. CONCLUSION

In this paper we present the design to reduce the data size of a two dimension thermal image. By using the property that thermal image is gray scale and nearby pixels are have similar value, we can use the average value to stand for whole region. By giving every regions different resolutions, we can reduce the file size to 50% less than JPEG when there is 0.5% of distortion, and up to 93% less when there is

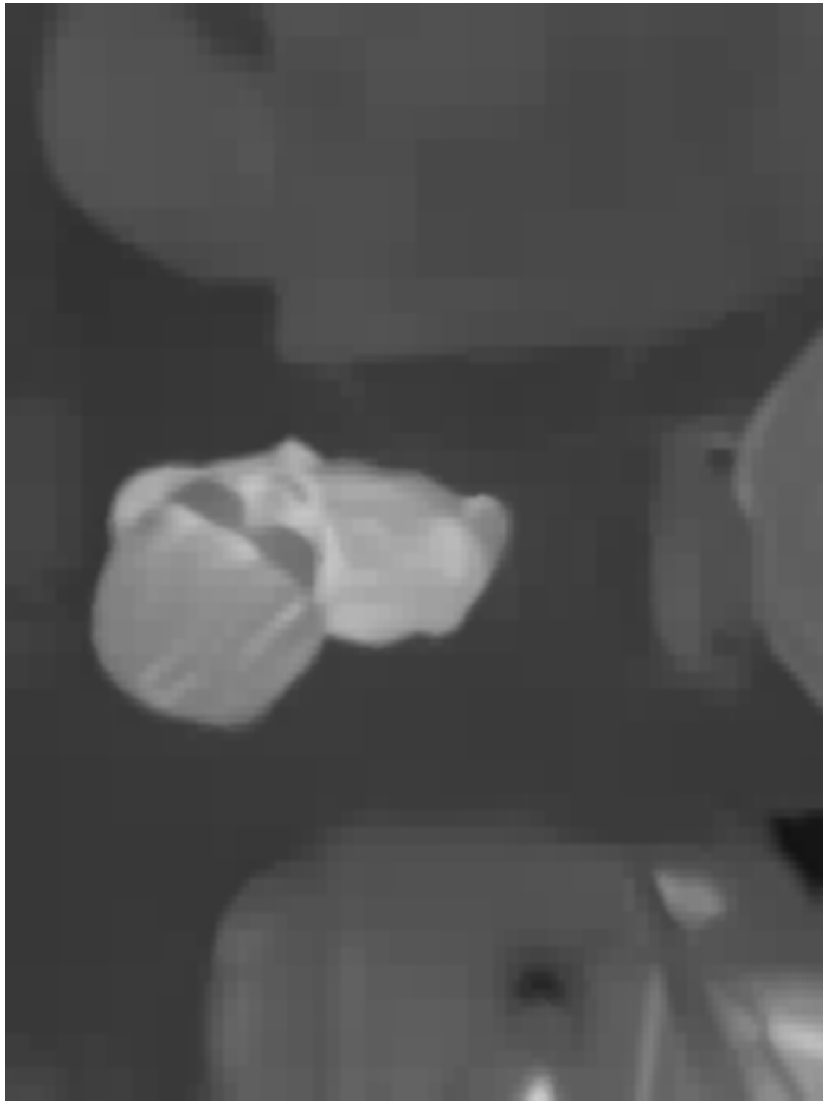


Fig. 5. 4KB Image by Proposed Method

2% of distortion. **Acknowledgements** This research was supported in part by the Ministry of Science and Technology of Taiwan (MOST 106-2633-E-002-001, MOST 106-2627-M-002-022-), National Taiwan University (NTU-106R104045), Intel Corporation, and Delta Electronics, and Advantech.



Fig. 6. 4KB Image by JPEG

Proposed method and JPEG comparing

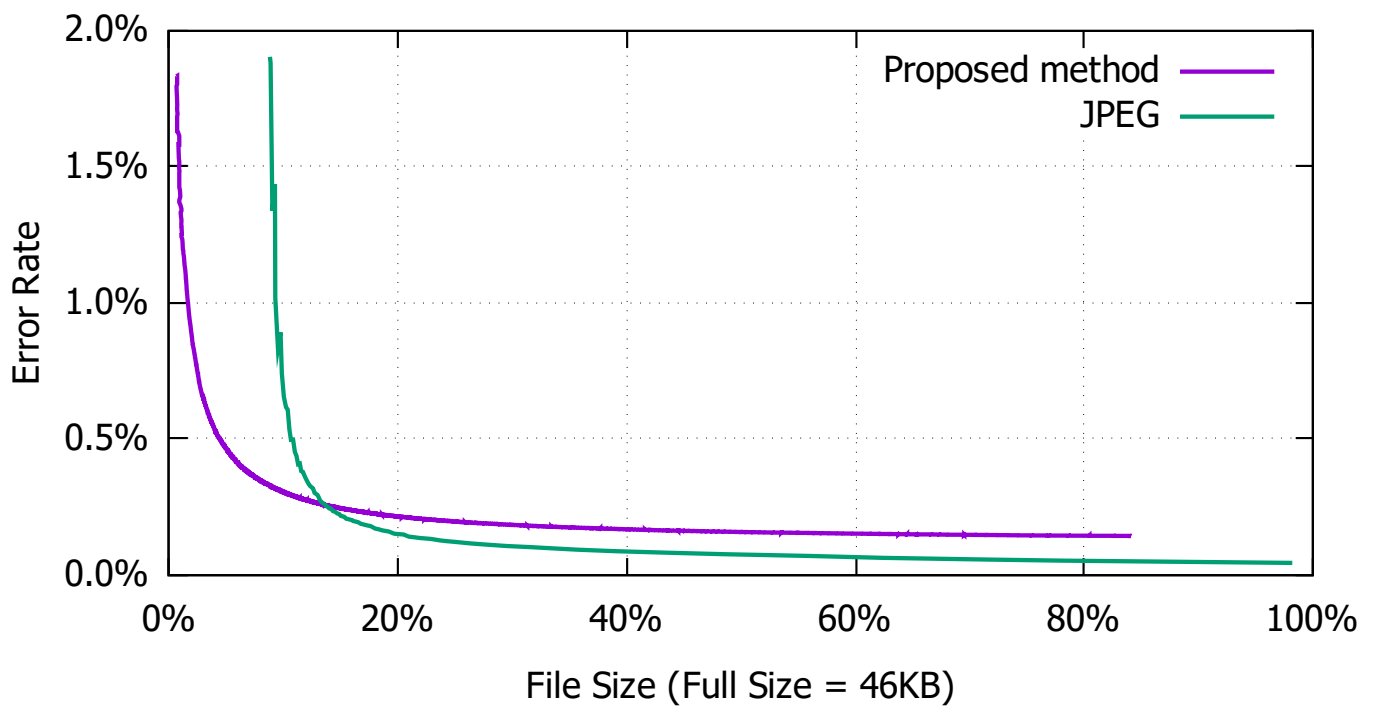


Fig. 7. Proposed method and JPEG comparing