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 to send 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 for IoT devices. In the algorithm, we iteratively separate a data region 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 an area has more dynamical diverse data, 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 0.5% of distortion is allowed, and up to 93% less when 2% of distortion is allowed.

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 sensors are used 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 8×8 pixels infrared array sensor, and FLIR ONE PRO, a 480×640 pixels thermal camera. Both are setting on ceiling and taking a video of a person walking under the camera.

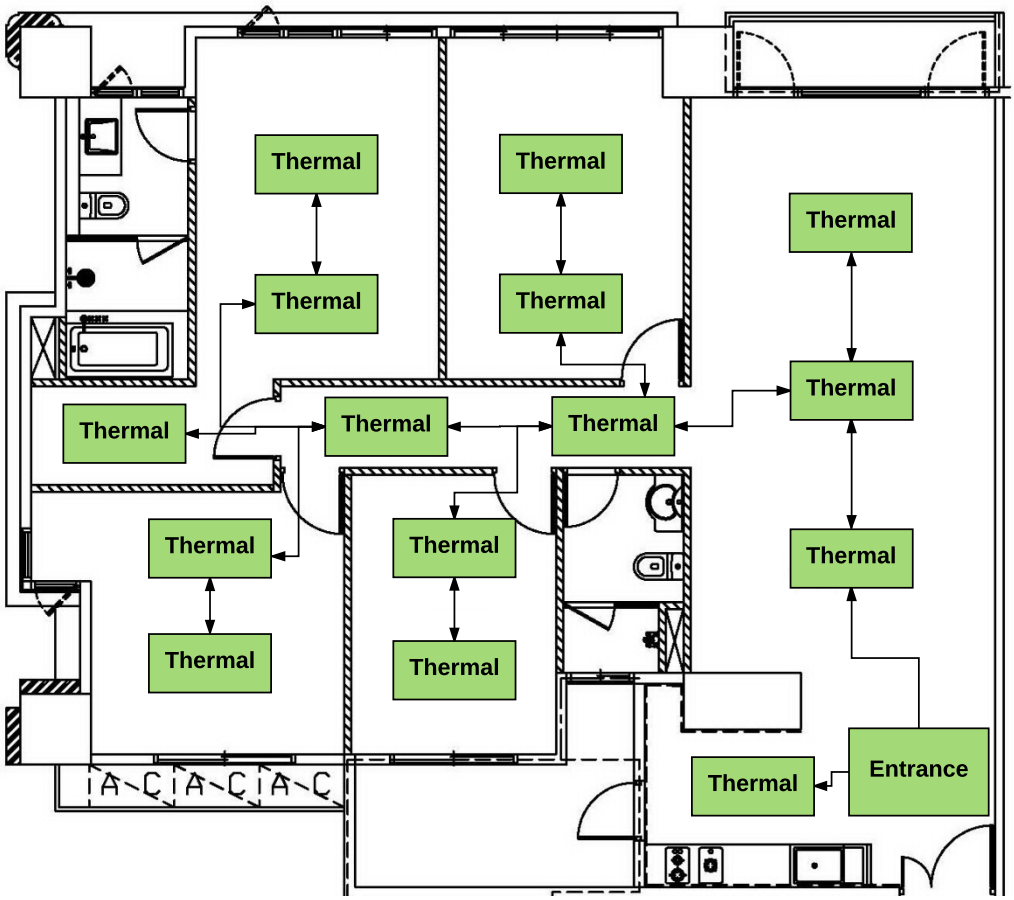

Fig. 1. Gait Velocity Measurement at Smart Homes

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 generates 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 data retrieved from FLIR ONE PRO to targeted data size and keep the quality of data. Nearby pixels in a thermal data mostly have similar value, so we can easily separate an data region into several regions and use its average value to represent it but will not cause too much error. By the method we proposed, the size of file can reduce more than 50% compare to using JPEG compression algorithm 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 8×8 pixels thermal data with $2.5°C$ accuracy and $0.25°C$ 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 thermal from Grid-EYE sensor. The sensor signal will form a signal cone. The pixel sampling our head temperature will have the highest reading, body is lower, and leg is the lowest except background because when the distance from camera to our body is longer, the area covered by the camera will be wider and the ratio of background temperature in the pixel will increase, also our head does not cover by cloth, so the surface temperature will higher than other place. While we are walking in an area, the temperature of air in the area will become warmer, and the shape of human will be more difficult to recognize.

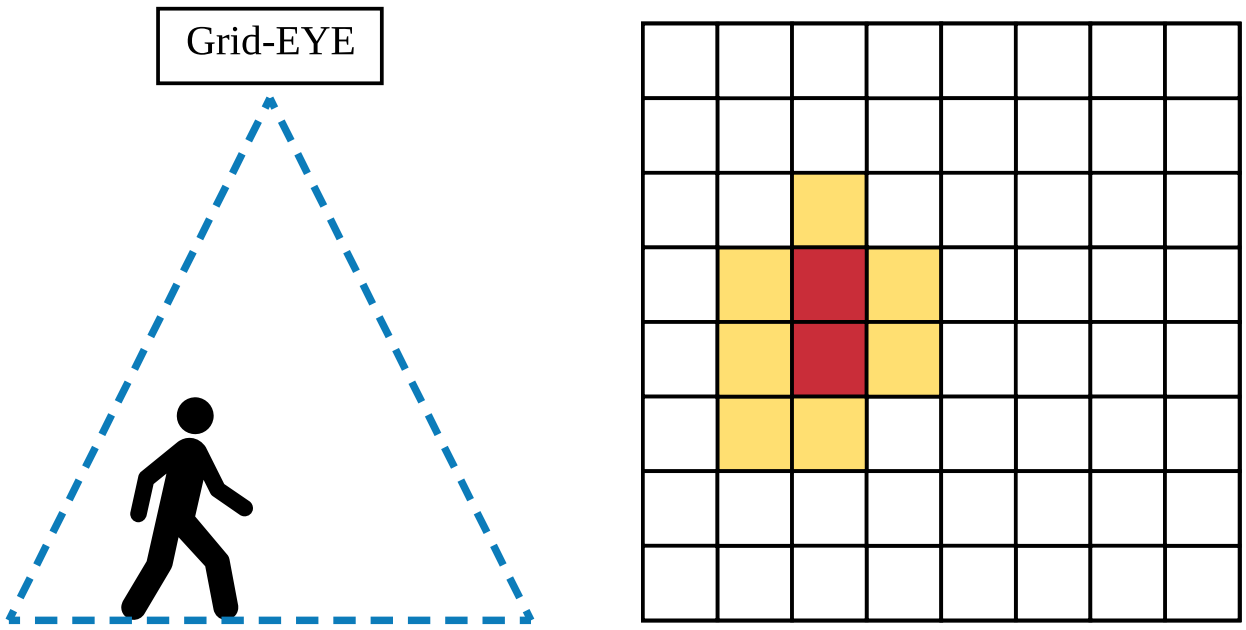

Fig. 2. Walking under a Grid-EYE sensor

The data we used is collected from a solitary elder's home. We deployed four Grid-EYE sensors at the corner of her living room, and recorded the thermal video for three weeks at 10 frames per second data rate, and the size of raw data is about 17.6GB.

B. FLIR ONE PRO

FLIR ONE PRO is a thermal camera that can output 480×640 pixels thermal data with $3°C$ accuracy and $0.01°C$ resolution, and capture speed is about 5 frames per second. 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 the data from picture taking mode as our test object. The data form FLIR ONE PRO has about 5000 times resolution compared to Grid-EYE. The shape of object is not just a cone. The temperature in a same object is similar, but an Obvious edge between different objects. Hence, we developed a method to compress the thermal data from FLIR PRO ONE. 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 idle 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 from Grid-EYE in a readable format, it will take about 380 bytes storage. However, the temperature range of indoor environment mostly from $5°C$ to $40°C$ and the resolution of Grid-EYE is $0.25°C$, so we can easily represent each temperature by one byte. Hence, we only need 64 bytes to store a frame. We can use different 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 send the pixels with higher temperature since thermal sensors are mostly used for detect heat source. Z-score is defined as $z = \frac{\chi - \mu}{\sigma}$ where χ is the value of the temperature, μ is the average of the temperature and σ 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 $10°C$ difference between day and night, and when people walk through the sensing area the Grid-EYE, the temperature reading will only increase $2°C$ to $3°C$. Hence, it is impossible to use a static threshold to detect human. In [**Shih17b**], the pixels with useful data only if the Z-score is higher than 2, so we can reduce the frame size by dropping all pixels with Z-score lower than 2. We can reduce the file 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 temperature readings of human body in a thermal data from Grid-EYE looks like a signal cone, we may use a Gaussian function to fit the thermal data. 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 use the pixel with highest temperature to be the peak of the cone, so we only need to adjust A and C to fit the thermal data. Guo [**guo2011simple**] provides a fast way to determine the fitting Gaussian function. In our testing, it will have $0.5°C$ root-mean-square error in average, 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 generate a data array that has less size compared to jpeg image when some of distortion is allowed. We use the thermal data from FLIR ONE PRO.

The nearby pixels usually have similar values, except at the edge of objects. Hence, we can divide 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 divide an image into some polygon region needs a lot of computation power and difficult to describe the edge of each region. Also, determining the number of region is a challenge. Hence, to effectively describe regions we design that every region most be a rectangle, and every region can divide 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 divide a region into 4 pieces.

Our method is shown in Figure 3. Data structure initialization only needs to do once if the size of a frame doesn't change. A thermal data will be loaded into our data structure and divide into several regions. Finally, the compressed data will be encoded by Huffman coding, and transmitted to database. When users want to use the thermal data, they can restore the data from the encoded data in database.

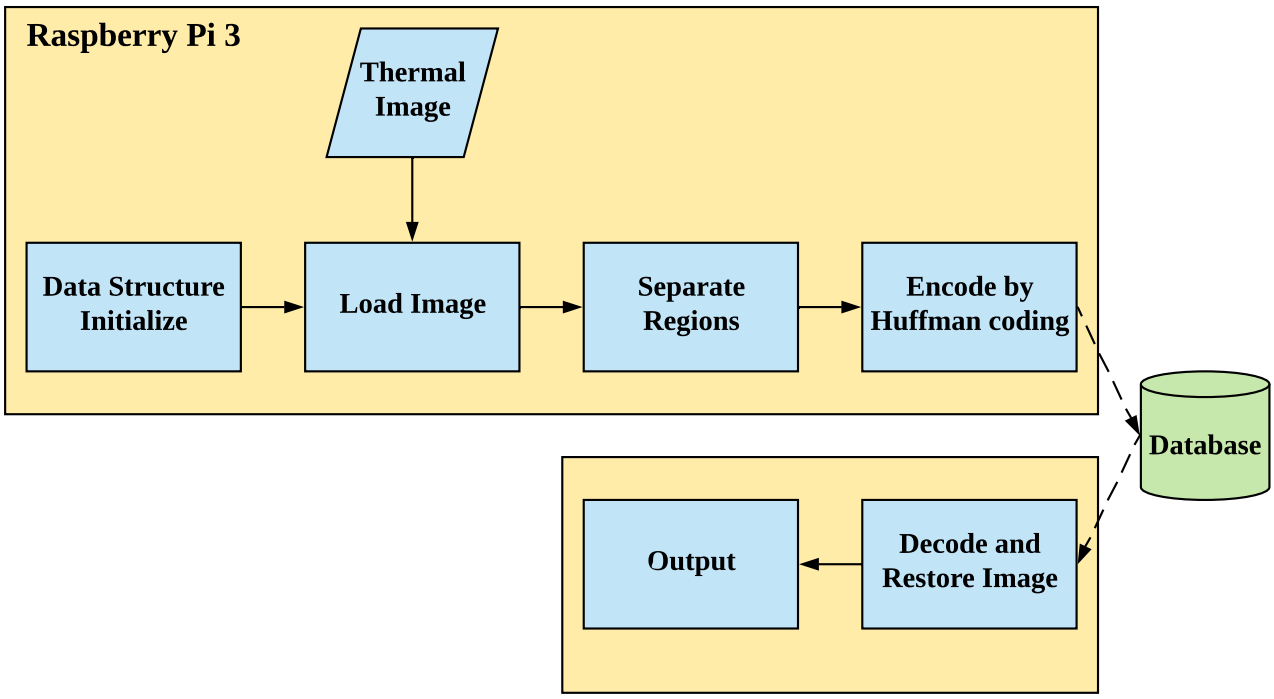

Fig. 3. System Architecture

A. *Region Represent Grammar*

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

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

R refers to a region of frame, and it can either use the average α of the pixels in the region to represent the whole region or divide into four regions and left a remainder β. Dependence on size of compressed data, we can choose the amount of dividing regions. The context-free grammar starts from a region containing whole frame. For each R we calculate a score based on the data the quality of data we can improve by dividing it into smaller regions. Figure 4 shows an example of thermal data which was took by FLIR ONE PRO. One of the possible outcome is Shown in Figure 5 if we divide the frame 6 times and it will have 19 regions. By this method, we can iteratively compress the thermal data until the number of regions reach our file size requirement or the error rate is less than the requirement.

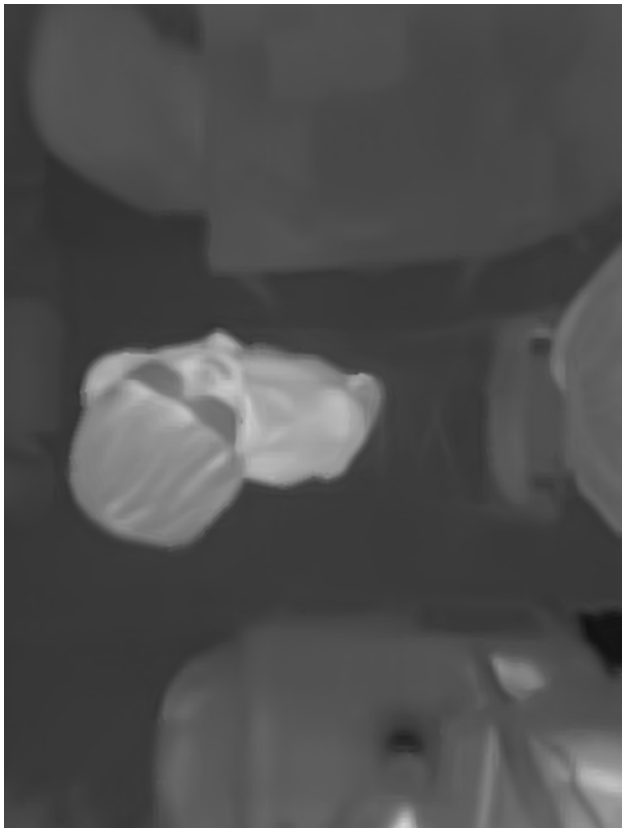

Fig. 4. PNG image, size = 46KB

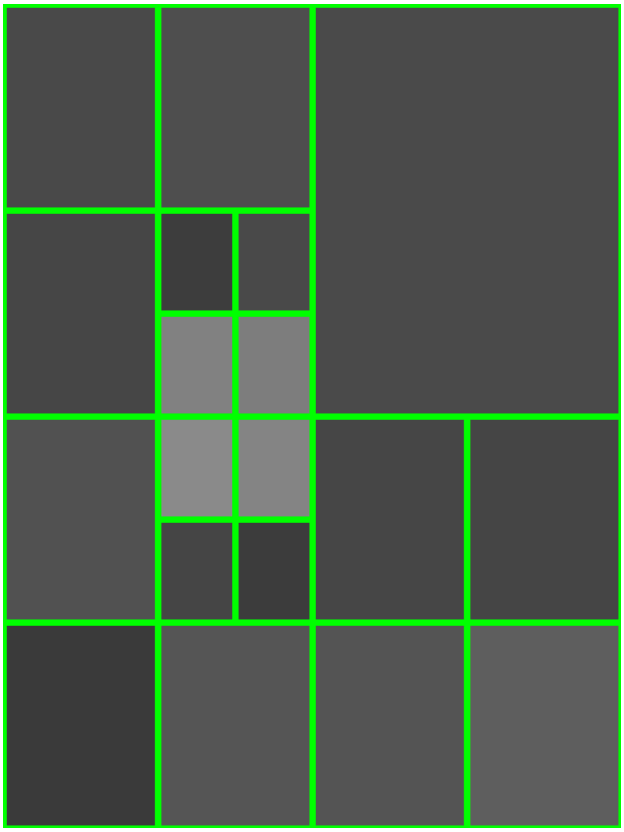

Fig. 5. Region divided by our method

B. Data Structure and Region Selection Algorithm

To help us choose which region to be divided, we give every region a score, and put them into a heap. For each round, we pick the region with the highest score, and divide 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.

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

By the equation shown above, we just need to know the sum of square and the sum of all pixels in the region to calculate the score of the region. We use a 4-dimension segment tree as a container to store all possible regions and its scores. Since segment tree is a complete tree, the size of tree is less than 2 times the number if pixels. For each node of segment tree, it records the range on both width and height it covered, sum $\sum_{X \in R} X$, and sum of square $\sum_{X \in R} X^2$ of pixels in the region. The root of segment tree starts is node number 0, and each node i has four child from node number from $i \times 4 + 1$ to $i \times 4 + 4$. Hence,

we only need to allocate a large array and recursively process all nodes form root. Algorithm 1 shows how we generate the tree and calculate the sum of square and the sum of all nodes.

Algorithm 1 Segment Tree Preprocess

$Tree = Array()$
function SETTREENODE($x, left, right, top, bottom$)
  if $left = right\ top = bottom$ **then**
    $Tree[x].Sum = Frame[left][top]$
    $Tree[x].SquareSum = Frame[left][top]^2$
  else
    $setTreeNode(4x+1, left, (left+right)/2, top, (top+bottom)/2)$
    $setTreeNode(4x+2, (left+right)/2, right, top, (top+bottom)/2)$
    $setTreeNode(4x+3, left, (left+right)/2, (top+bottom)/2, bottom)$
    $setTreeNode(4x+4, (left+right)/2, right, (top+bottom)/2, bottom)$
    $Tree[x].Sum = \sum_{i=4x+1}^{4x+4} Tree[i].sum$
    $Tree[x].SquareSum = \sum_{i=4x+1}^{4x+4} Tree[i].SquareSum$
  $Tree[x].SquaredError = Tree[x].SquareSum - \frac{Tree[x].Sum^2}{(right-left+1)\times(bottom-top+1)}$
$setTreeNode(0, 0, Frame.Width, 0, Frame.Height)$

For region selection, we use a priority queue to retrieve the region of considerate regions with highest score. The priority queue is made by heap, and 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.

The compressed data size is depended on how many iterations of dividing the regions. The compressed data size will be about the number of iterations times four bytes. Algorithm 2 shows how we divide regions until specified data size.

Algorithm 2 Dividing regions depends on compressed data size

$seperatedRegions = Array()$
$PriorityQueue = Heap()$
$PriorityQueue.Push(Tree[0].SquaredError, 0)$
for $i = 0..NumberOfIterations$ **do**
  $value, x = PriorityQueue.Pop()$
  $seperatedRegions.push(x)$
  for $j = 1..4$ **do**
    $PriorityQueue.Push(Tree[4x+j].SquaredError, 4x+j)$

The error rate of the compressed data is the sum of the squared error of regions in priority queue. Algorithm 3 shows how we divide regions until specified RMSE.

After the region dividing finished, we will generate the data string to be sent. The regions in $seperatedRegions$ will be replaced by a reminder for dividing and others in $PriorityQueue$ will be the average sensor reading

Algorithm 3 Dividing regions depends on compressed data RMSE

$seperatedRegions = Array()$
$PriorityQueue = Heap()$
$PriorityQueue.Push(Tree[0].SquaredError, 0)$
$SquaredError = Tree[0].SquaredError$
while $\sqrt{(SquaredError/FrameSize)} > SpecifiedRMSE$ **do**
    $value, x = PriorityQueue.Pop()$
    $seperatedRegions.push(x)$
    $SquaredError$ -= $value$
    for $j = 1..4$ **do**
        $PriorityQueue.Push(Tree[4x + j].SquaredError, 4x + j)$
        $SquaredError$ += $Tree[4x + j].SquaredError$

in the region, and then compress the string by Huffman Coding.

The complexity of our algorithm can be divided into 3 parts. First part is to initialize the segment tree. The size of segment is depends on the size of the frame. If the number of pixels is N, the height of segment tree is $O(Nlog(N))$, and the number of nodes will be $O(N)$. The time complexity of initialize is $O(N)$. Second part is loading the thermal data. It will need to traverse whole tree from leaf to root. Since segment tree can be stored in an array, it also takes $O(N)$ time to load the thermal data. Third part is to divide regions. For each round, we pop an element from heap and push four elements into heap. If there is K iterations, 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 $O(Klog(K))$.

IV. Performance Evaluation

To evaluate the effectiveness of the proposed method, we did the different ratios of compressing on a thermal data by our method compared 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 thermal data size is 480×640 pixels. The JPEG image is generated by OpenCV $3.3.0$ which is using libjpeg version 9 13-Jan-2013, and image quality from 1 to 99.

Figure 6 and Figure 7 show the different of JPEG and our method. JPEG image id generated by image quality level 3, and thermal data of our method does 1390 rounds of separate and compressed by Huffman Coding. In this case, Huffman Coding can reduce 39% of compressed data size.

Figure 8 shows that the size of file can reduce more than 50% compared to JPEG image when both have $0.5\%(0.18°C)$ of root-mean-square error. Our method has 82% less error rate when the compressed data size is $4KB$. The percentage of file size is compared to PNG image.

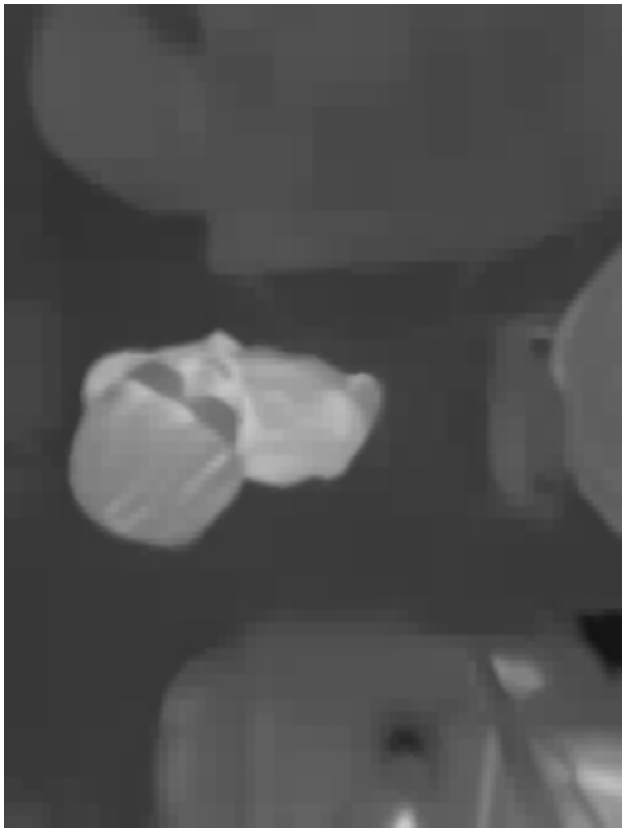

Fig. 6. Data compressed by Proposed Method (4KB)

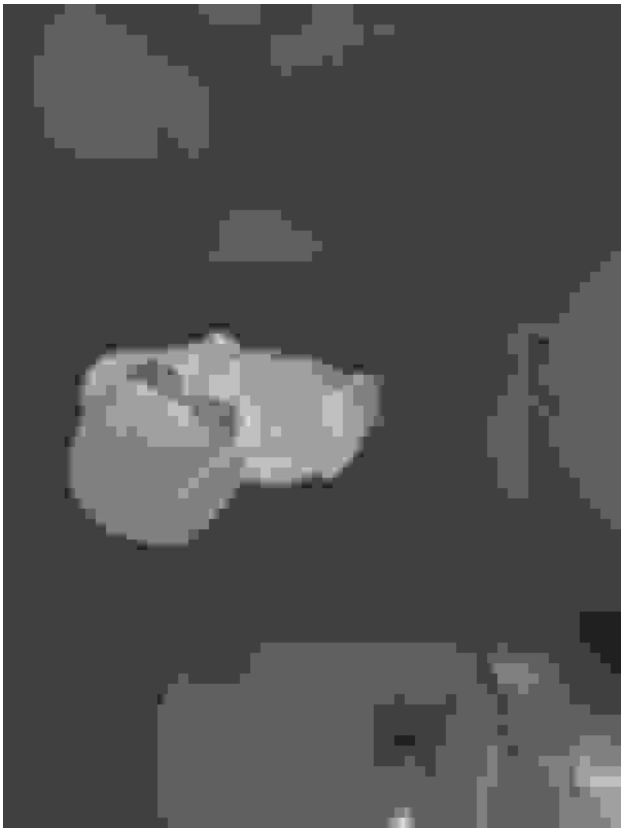

Fig. 7. Data compressed by JPEG (4KB)

The computing time of a 480×640 thermal data on Raspberry Pi 3 is:

1) Date Structure Initialize: 0.233997 second.

2) Thermal Data Loading: 1.268126 second.

3) Regions dividing: About 4.6 microsecond per separation. Figure 9 shows the computation time of Region dividing.

Total time is about 1.5 second.

V. Conclusion

In this paper we present the design to reduce the data size of a two dimension thermal data. Nearby pixels in a thermal data mostly have similar value, so we can easily separate an data region into several regions and use its average value to represent it but will not cause too much error. The method we proposed can either choose the data size or error rate of compressed data. 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 2% of distortion.

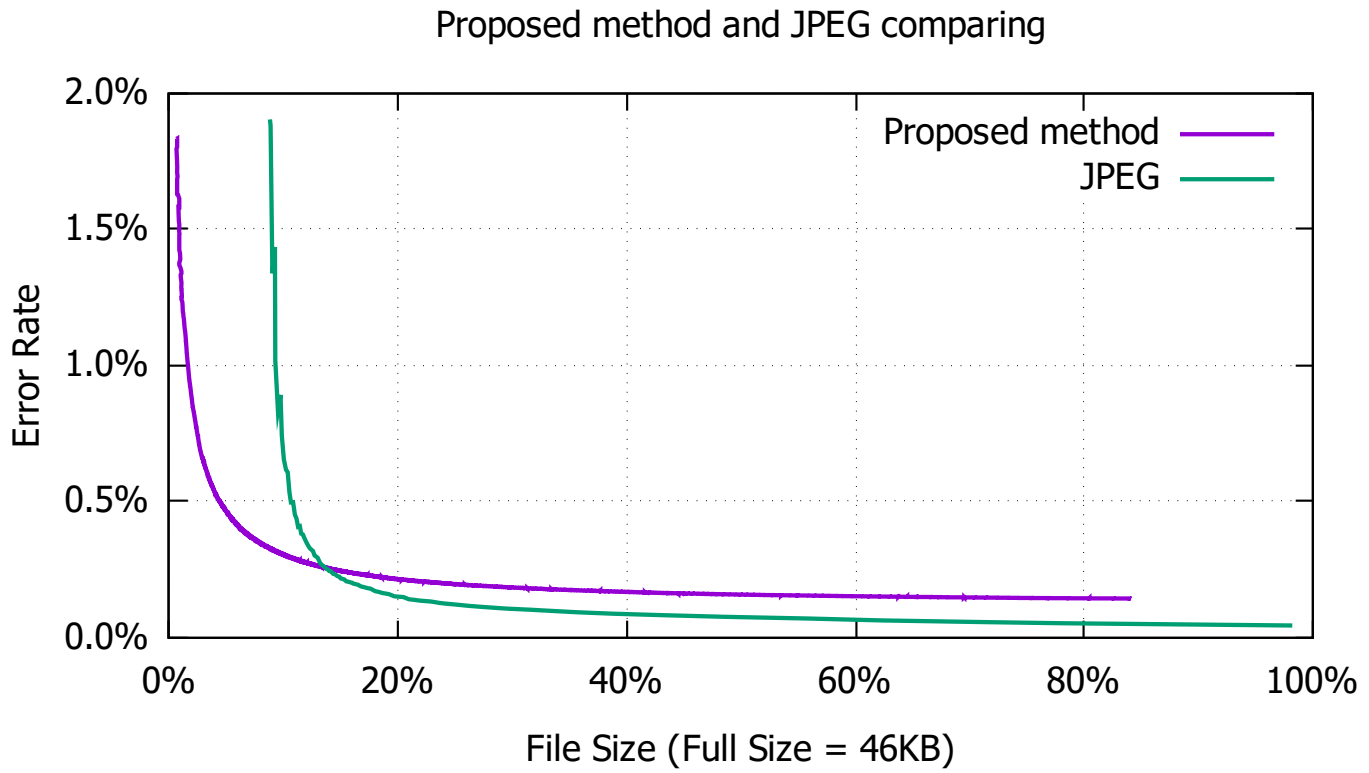

Fig. 8. Proposed method and JPEG comparing

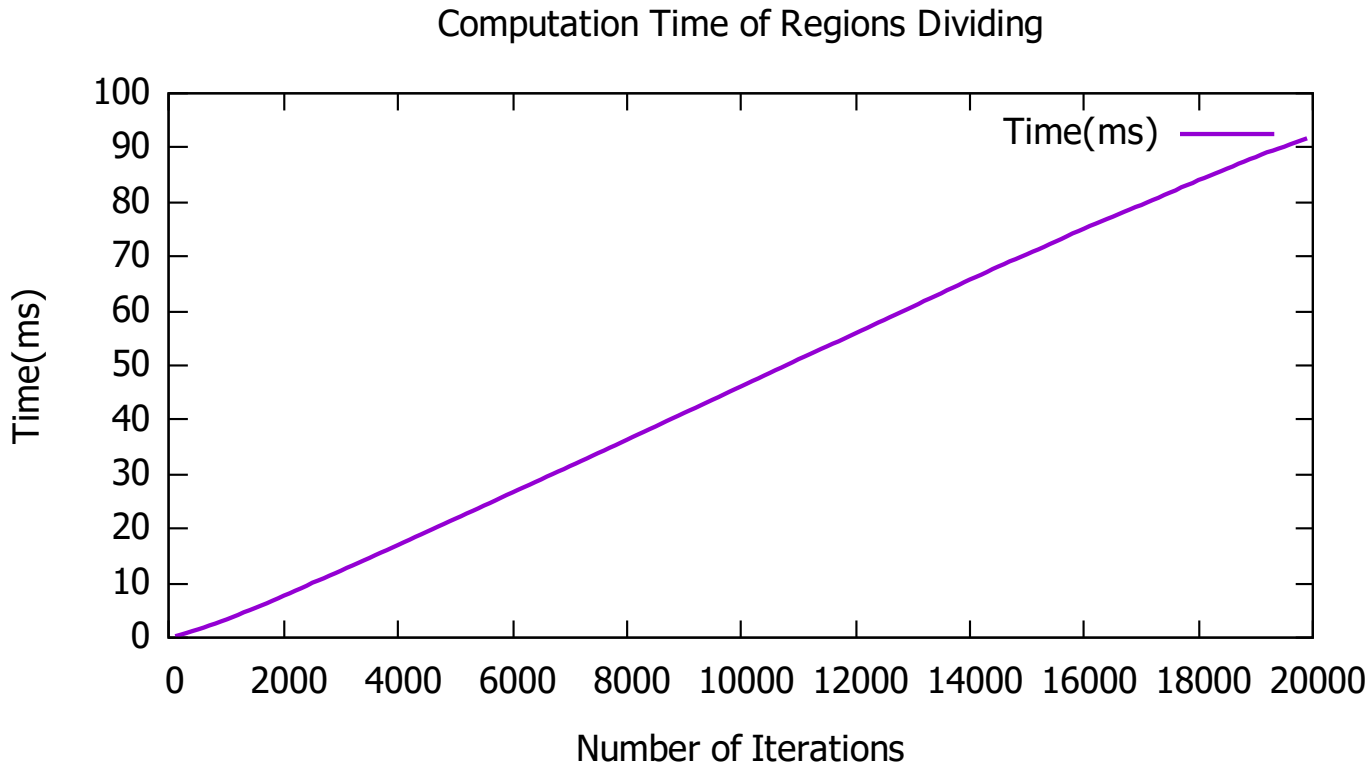

Fig. 9. Computation Time of Regions Dividing