Parameterized Data Reduction Framework for IoT Devices

Chi-Sheng Shih[1], Jyun-Jhe Chou[1], Wei-Dean Wang[2], and Kuo-Chin Huang[3]

[1]Graduate Institute of Networking and Multimedia, Department of Computer Science and Information Engineering, NTU IoX Research Center, National Taiwan University
Email: cshih@csie.ntu.edu.tw

[2]Department of Medical Education and Bioethics, NTU

[3]Department of Family Medicine, NTU Hospital

***Abstract*— In a IoT environment, there are many devices will periodically transmit data. However, most of the data are useless, but sensor itself may not have a good standard to decide transmit or not. Some static rule maybe useful on specific scenario, and become useless when we change the usage of the sensor. In this paper, we want to present a method to reduce the file size of thermal sensor which can sense the temperature of a surface and output 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

In a IoT environment, there are 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 useless. 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. Hence, a framework to select useful data is important.

On Raspberry Pi 3, while it is idling and turning off WiFi, it will consume 240mA and while uploading data at 24Mbit/s, it will consume 400mA. If we sent 640×480 pixels heat map images in png format (average 45KB) in 10Hz, it will consume about 264mA.

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.

Contribution The contribution of this work is to present a framework for user to choose either the bit-rate or the error rate of the video. 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 8×8 pixels image with $2.5 \circ C$ accuracy and $0.25 \circ C$ resolution at 10 frames per second. 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 1 shows an example of image from Grid-EYE sensor.

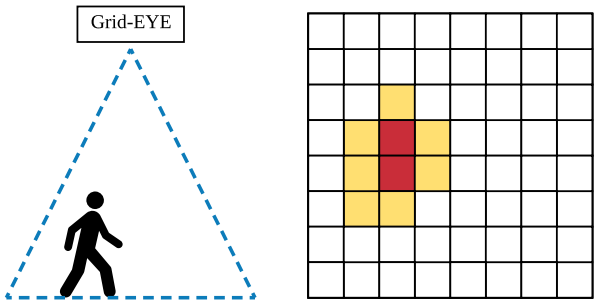

Fig. 1. Walking under a Grid-EYE sensor

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. 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 $5°C$ to $40°C$ and the resolution 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 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 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**], 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: In Figure 1, we can see that the sensor value will be 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. A Gaussian function $y = Ae^{-(x-B)^2/2C^2}$ has three parameter $A, BandC$. 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 $AandC$ to fit the image. Guo [**guo2011simple**] provide a fast way to get the fitting Gaussian function. In our testing, it will be about $0.5°C$ root-mean-square error, and only needs 5 bytes to store the position of peak and two parameters.

C. FLIR ONE PRO

FLIR ONE PRO can output a 480×640 pixels image with $3°C$ accuracy and $0.01°C$ 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 $°C$ 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.

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.

A. Heuristic Data Resolution Determination

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 means a region of image, and it can either use the average α of the pixels in the region to represent whole region or separate into four regions and left a remainder β. 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 heuristic value h 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 ω. One of the possible outcome is $\beta_1\beta_2\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6\beta_3\alpha_7\alpha_8\alpha_9\beta_4\alpha_{10}\alpha_{11}\alpha_{12}\alpha_{13}$. Figure 2 shows how the image is separated into several regions. By this method, we can continuously separate regions until the file size excess our requirement or the error rate less than a threshold.

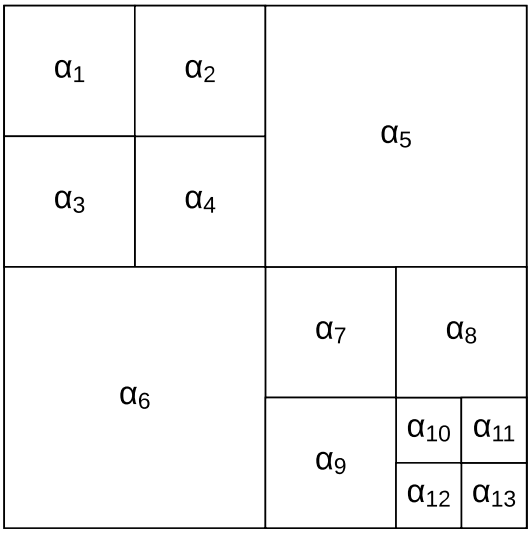

Fig. 2. Region separate by CFG

The heuristic function in the proposed method is the sum of squared error of the pixels in the region. We have also try to use the total squared error it can reduce as the heuristic function, but it will easily get stuck at a local minimum.

B. Data Structure and Region Selection Algorithm

We use the sum of squared error of pixels in the region when we use the average of them to replace them as the heuristic value. In order to reduce the heuristic value's calculating time, we design a four dimension segment tree to preprocess all possible regions. For each node, it store the range on both width and height it covered, mean $E[X]$, and squared mean $E[X^2]$ of pixels in the region. By the property of segment tree, tree root start from 0, and each node X_i has four child $X_{i\times4+1}$, $X_{i\times4+2}$, $X_{i\times4+3}$ and $X_{i\times4+4}$. Hence, we only need to allocate an large array and recursively process

all nodes form root. Algorithm 1 shows how we generate the tree.

For region selection, we use a priority queue to retrieve the region of considerate regions with highest value. The priority queue start with only root of the segment tree. For each round the priority queue pop the item with highest value 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 β and others in $PriorityQueue$ will be the average value, and then compress the string by Huffman Coding.

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 480×640 pixels. Figure 3 shows an example of image which was took by FLIR ONE PRO. The JPEG image is generated by OpenCV 3.3.0, and image quality from 1 to 99.

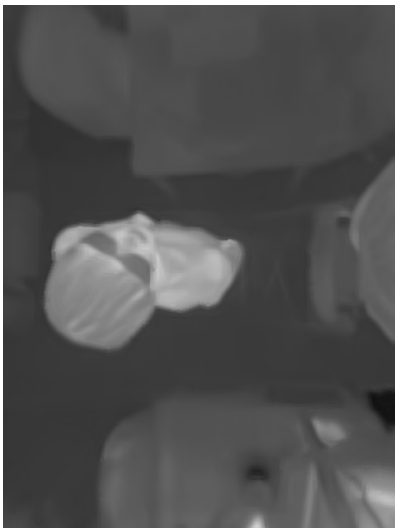

Fig. 3. PNG image, size = 46KB

Figure 4 and Figure 5 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 **??** 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.

V. Conclusion

In this paper we present the design to reduce the data size of a two dimension thermal image. By using the property

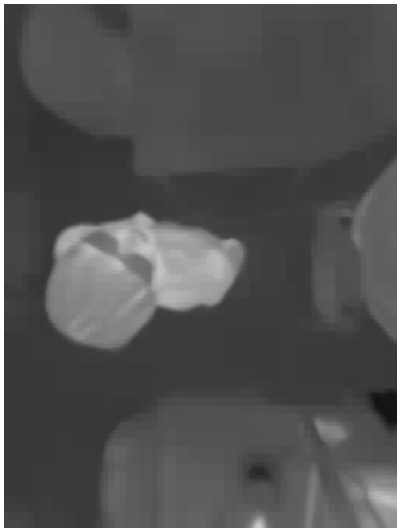

Fig. 4. 4KB Image by Proposed Method

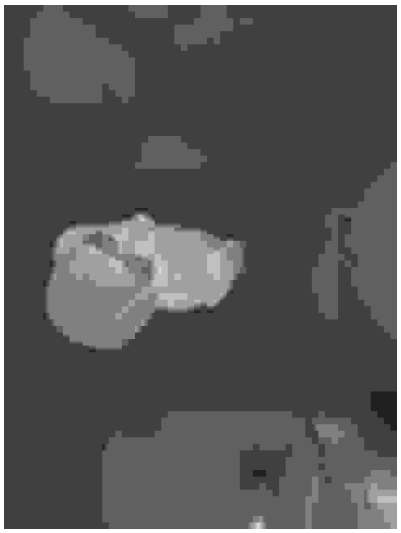

Fig. 5. 4KB Image by JPEG

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 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.

Algorithm 1 Segment Tree Preprocess

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

Algorithm 2 Region Selection

$seperatedRegions = Array()$
$PriorityQueue = Heap()$
$PriorityQueue.Push(Tree[0].SquaredError, 0)$
for
    $i = 0..SeperateRounds$ **do**
    $value, x = PriorityQueue.Pop()$
    $seperatedRegions.push(x)$
    $PriorityQueue.Push(Tree[4x+1].SquaredError, 4x+1)$
    $PriorityQueue.Push(Tree[4x+2].SquaredError, 4x+2)$
    $PriorityQueue.Push(Tree[4x+3].SquaredError, 4x+3)$
    $PriorityQueue.Push(Tree[4x+4].SquaredError, 4x+4)$