Surveillance using Thermal-Imaging Sensors

Leonidas Deligiannidis
Department of Computer Science and Networking
Wentworth Institute of Technology
Boston MA, USA
deligiannidisl@wit.edu

Abstract
In this paper we present a solution for remote area surveillance utilizing a thermal camera. This is an extension to our earlier paper where the details of the algorithms and their applications were presented. Our earlier work was based on a color video camera where this solution is based on a thermal camera. Thermal-imaging enables our surveillance algorithms to work and be useful even in total darkness as they detect radiation in the long-infrared range of the electromagnetic spectrum. We also show how to calibrate the sensor to map accurately sensor values into absolute temperatures.

Keywords: Surveillance, Thermal Imaging

1. Introduction
Thermo-Imaging cameras are sensors that can detect radiation in the long-infrared range of the electromagnetic spectrum. This radiation is emitted by all objects whose temperature is above the absolute zero. Thermo-Imaging cameras have been in existence in the market for many years, but their high cost prevented their usage by the broad population. They not only cost a lot, they even come pre-packaged as sensor units and this makes it difficult for programmers to access the raw data in real time. In the last couple of years however, low resolution (80x60 pixels), inexpensive thermo-camera sensors appeared in the market. The manufacturer, FLIR Systems, Inc., designed a low cost sensor – the cost of the sensor is around a couple of hundred dollars. GroupGets.com, a crowd-funding site, sells a breakout board which enables the sensor to be connected to a Raspberry Pi single board computer. The manufacturer provides an API, and now utilizing the Serial Peripheral Interface (SPI) protocol we can receive the raw data stream from the camera in real time. Controlling the sensor requires the usage of the Inter-Integrated Circuit (I2C) protocol. Both protocols are supported by the Raspberry Pi.

There are many applications where such thermo-cameras can and are being used today. They can detect energy waste and heat loss [1], moisture and mold [2], construction, electrical [2][3], and many other problems related to heat energy [1][4] and air conditioning [5]. IR technology can also help us improve concrete structures [6] by pinpointing areas of high stress [7], and even design better wall material for buildings and houses [8] and could also be used to prevent terrorist attacks [9].

Because the sensor is sensitive to long-infrared range of the electromagnetic spectrum, it can operate in total darkness, from a distance. This makes it an attractive solution where hazardous locations need to be inspected [10]. Thermo-cameras measure the amount of radiation emitted by a surface. These values can then be mapped to color pixel values to form a 2-dimensional picture depicting the temperature distributions [11]. They help us convert the invisible long-infrared waves to visible waves for analysis and feature extraction.

2. The Sensor
The thermo-imaging sensor streams its data frames over the Serial Peripheral Interface bus (SPI), and it can be controlled over its Inter-Integrated Circuit (I2C) bus. The Raspberry Pi single board computer supports both interfaces. We mounted the thermal camera and the Raspberry Pi on the back of a touch screen LCD display as shown in figure 1.

Figure 1. The Thermo-imaging sensor attached at the back of an LCD display.

This enables us to move around with the sensor while looking at what is in front of the sensor. We run a server on the raspberry pi where many client subscribers and request the feed of the thermo-camera. The software of course can also
run locally on the Raspberry Pi that can be battery powered—which makes the entire built portable as a stand-alone sensor. We implemented the algorithms presented in [12] which include detection of movement in a user defined area or above/below a specified line, and line crossing (tripwire). Figure 2 shows a screen shot of what the sensor sees. The feed is transmitted to a remote laptop. Upon motion detection in the specified area, an alarm could be triggered.

Figure 2. Screenshot of what the sensor sees. Motion in the specified area is shown in highlighted color (red) on the second view at the right. Upon an alarm event, we could send an email, activate someone’s pager, sound a siren, etc.

3. Temperature Values

Even though we re-implemented the algorithms presented in [12] using a thermo-camera, instead of a color-camera, that works in total darkness, we found useful to also detect drastic temperature changes. While we can configure the sensor to detect motion inside or outside of a predefined area and detect predefined line crossings, we can also focus, in parallel, for any drastic temperature changes in the field of view of the sensor. Thermo-imaging cameras do not provide color pixel values. Instead, they provide a relative number per pixel that represents temperature. The hotter the pixel, the greater the number. However, there is no direct mapping of this number to an absolute temperature. High-end sensors are factory-calibrated and they can display absolute temperatures for each pixel. To be more effective with temperatures, we need to be accurate of the absolute temperature of different objects in the field of view of the sensor. However, the most inexpensive sensors—such as the one we were using, are not factory calibrated.

It is not very difficult, however, to calibrate such sensor as long as we know some absolute temperatures in the field of view of the sensor. For this we used the Melexis MLX90614 10° field of view, non-contact, thermometer which can be controlled using the I²C bus. One of the problems is that the Raspberry Pi exposes to the user only one of the I²C busses—one used by the thermo-camera and we need one more to read the absolute temperatures from the MLX thermometer. The problem was solved by using a $5 Raspberry Pi zero. The thermometer was attached on the Raspberry Pi zero. The thermo-imaging camera was attached to a Raspberry Pi 3. We connected the two Raspberry Pi computers via their UART chips. This made the thermometer and the computer it was attached on, an extension to the computer that the thermo-imaging camera was connected to. Figure 4 shows a diagram on how all the sensors and computer components are connected together.

Figure 4. The thermo-imaging sensor is connected to the primary computer. The MLX thermometer is connected to a secondary computer. The two computer are connected together utilizing the Universal Asynchronous Receiver/Transmitter (UART) interface via the General Purpose Input Output (GPIO) header.

4. Calibration

Each thermo-imaging sensor is different by construction. In other words, different sensors of the same make measure different values for the same input. Thus, they need to be calibrated before mapping sensor values to temperatures. We use the Least Square Fit (LSF) method to calibrate the sensor. We make about 100 unique measurements with the MLX thermometer and at the same time we keep track of the values.
the thermo-imaging sensor generates for the same measurements. We then fit the data using the Least Square Fit method. The $x$ represents the value from the thermo-imaging sensor and $y$ represents the temperature of the same input, which is generated by the MLX non-contact thermometer. We first find the average of the $x$ and $y$ as shown below.

$$
\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}
$$

$$
\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}
$$

We then compute the $S_x$, $S_y$, and $S_{xy}$ values as shown below:

$$
S_x = \sum_{i=1}^{n} (x_i - \bar{x})^2
$$

$$
S_y = \sum_{i=1}^{n} (y_i - \bar{y})^2
$$

$$
S_{xy} = \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})
$$

The regression coefficients $a$ and $b$ are given by:

$$
a = \frac{S_{xy}}{S_x}
$$

and

$$
b = \bar{y} - a\bar{x}
$$

Based on these $a$ and $b$ value, we can approximate / estimate future mapping from sensor values ($\text{thermoValue}$), to temperatures by computing:

$$
\text{temperature} = a \times \text{thermoValue} + b
$$

The overall quality of the fit is then parameterized in terms of a quality known as the correlation coefficient:

$$
r = \frac{S_{xy}}{\sqrt{S_x S_y}}
$$

The closer to the value of 1 it is, the better the quality of the fit. We use a value of 0.9 and above as a good fit. Otherwise, we inform the user that a recalibration is needed. A typical calibration result is shown in figure 5.

---

**Figure 5.** A sample calibration graph. Temperature values are measured at a distance of approximately 6 feet from the sensor, while the user moves around the sensor. MEASURED are the actual measurements – on the X axis are the values from the thermo-imaging sensor and on the Y axis are the measurement from the MLX thermometer. After calibration, we compute the $a$ (0.00986295256853018) and $b$ (-9.929496147198051) values as well as the quality of the fit, $r$. If the value of $r$ is over 0.9 we accept the calibration result. In this graph we have $y = 0.00986295256853018 \times x - 9.929496147198051$ and $r= 0.92686859703352$. Based on these values, we can get the EXPECTED temperatures for the same measurements. We can also map new thermo-imaging sensor values to temperatures.
References