PHYS 319: Hot Object Detector

Jasmin Goh

April 7, 2019

1 Abstract

This is a report on designing and building a hot object detector using a microprocessor, infrared camera and servo motor. The goal of this project is to create a system that detects the location of a hot object within its proximity, and point the camera in that direction. Data from the infrared camera is received by the host computer, and is analyzed using a program written in C.

2 Introduction

The hot object detector is made up of 3 main hardwares: MSP430 microprocessor, MLX90614 infrared camera and SO3N servo motor. The infrared camera is mounted on the servo motor, allowing the camera to have a 180 degree view. The infrared camera and the servo motor are both connected to the microprocessor. Data from the camera is transferred to the host computer via the microprocessor, and is analyzed in a program written in C. The degree of rotation is decided by algorithms in the C program.

The goal of this proejct is to create a system that can detect the location of a hot object. This system can potetially be embeded onto a robot platform which utilizes the direction that the camera is pointing to to move away from any thermal hazards (i.e. hot objects) in its proximity.

3 Setup

There are 3 parts in setting up the hardwares for the hot object detector: (1) between the microprocessor and the green prototype board, (2) between the infrared camera and the microprocessor, and (3) between the microprocessor and the servo motor.

3.1 The connection between the microprocessor and the green prototype board

The green prototype board is used to power up the servo motor and the infrared camera. This means that the gound pin of the microprocessor must be connected to the ground pin of the green prototype board.

3.2 The connection between infrared camera and microprocessor

The infrared camera consists of 4 pins as shown in the diagram.

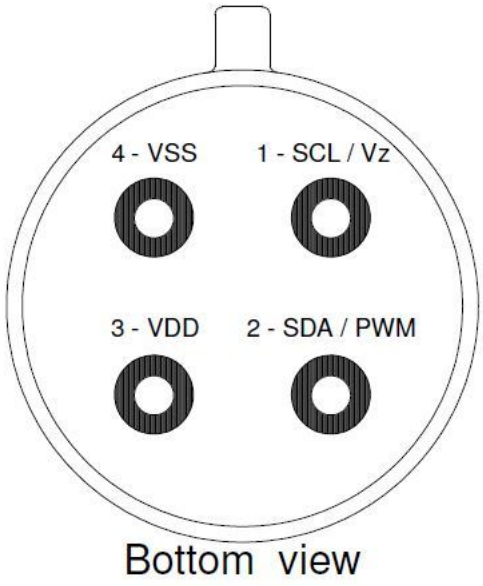

Figure 1: Pins of the infrared camera

Ensure that VSS is connected to a ground pin, and VDD is connected to a

+5V pin on the green prototype board. The SDA/PWM pin is responsible for sending thermal data to the microprocessor. (Note: because I was not able to establish the communication between the camera and the host computer, I left this pin connected to the ground)

Note: The infrared camera is a sensitive piece of equipment, so the unused pins should be connected to the ground to avoid it from picking up electrical charges.

3.3 The connection between microprocessor and servo motor

The servo motor consists of 3 wires of different colors (black, red and white). Ensure that the black wire is connected to the ground pin, and the red wire is connected to a +5V pin on the green prototype board. The white wire is connected to P1.2 of the microprocessor.

4 C program

Because of the limited time for this project and the lack of resource, I was not able establish the connection between the infrared camera and the host computer. Hence, I wrote code that mocks the data produced by the infrared camera. The next part of the C program sets up the microprocessor, analyzes the (mock) data from the infrared camera and finally, computes the degree of rotation of the camera.

4.1 Mocking data from the infrared camera

The infrared camera used for this project produces an image of 32 x 24 pixels, where each pixel consists of 1 byte (i.e. 8 bits) of data. The mock data is an array of size 32 x 24, and each element of the array is an int that ranges from 0 to 255 (i.e. $2^8 - 1$, where 8 is the number of bits that a pixel can represent). Figure 2 and 3 contain the code that creates the array for the mock data.

The pixels that the hot object occupy is entred in Line 1 in Figure 3 (in the hot_object variable). In the sample code, the hot object only occupies 4 pixels. The code works by reading which pixels the hot object occupies, and then produces a 1-D array.

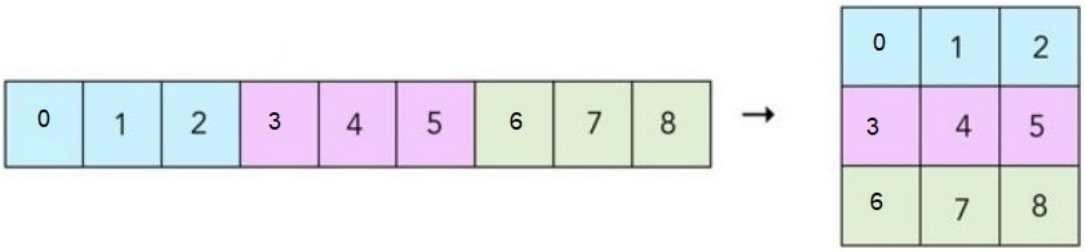

Figure 2: Conversion from a 1-D array to a 3 by 3 matrix

In that 1-D array, the index of the array corresponds to the position of the pixel, and the value stored corresponds to the temperature at that position.

The index of the 1-D array can be converted to the position of the pixel of the image from the infrared camera using the conversion as shown in Figure 2 (just scale the matrix up to a 32 by 24 matrix). Here's a simple formula to convert the index of the 1-D array to an x-y coordinate. (Note: the top left cell is (0,0)):

x-coordinate = index mod 32

y-coordinate = round down(index / 32)

Also, the air around the hot object gets warmed up by convection. Hence, through the infrared camera, the pixels around the hot object should have a high value too. Those are handled of the "else if" cases as shown in Figure 3. The values assigned in the "else if" cases (i.e. Line 10, 16, 21, 24, 27 of Figure 3) will be explained in Section 5.

The code in Firgure 3 also assumes that if there's not hot object, the temperature at that region will be 0. This is defintely not the case in real life; it is designed this way for simplicity.

```
int hot_object[4] = {340, 341, 372, 373};
    int acc[768];

    for (int a = 0; a < 768; a++ ) {
        if (contains(hot_object, a)) {
            acc[a] = 250;
        }
        else if (contains(hot_object, a+1) || contains(hot_object, a-1)
                || (a > 32 && contains(hot_object, a-32)) || (a < 32*23 && contains(hot_object, a+32)))
        acc[a] = 220;
        }
        else if (contains(hot_object, a+2) || contains(hot_object, a-2)
                || (a > 32 && contains(hot_object, a-31)) || (a < 32*23 && contains(hot_object, a+31))
                || (a > 32 && contains(hot_object, a-33)) || (a < 32*23 && contains(hot_object, a+33))
                || (a > 64 && contains(hot_object, a-64)) || (a < 32*22 && contains(hot_object, a+64)))
        acc[a] = 190;
        }
        else if (contains(hot_object, a+3) || contains(hot_object, a-3)
                || (a > 32 && contains(hot_object, a-30)) || (a < 32*23 && contains(hot_object, a+30))
                || (a > 32 && contains(hot_object, a-34)) || (a < 32*23 && contains(hot_object, a+34)))
        acc[a] = 140;
        }
        else if (contains(hot_object, a+4) || contains(hot_object, a-4)) {
        acc[a] = 120;
        }
        else if (contains(hot_object, a+5) || contains(hot_object, a-5)) {
        acc[a] = 50;
        }
        else {
        acc[a] = 0;
        }
    }

```

Figure 3: Code that mocks data from the infrared camera

4.2 Setting up the microprocessor

The code for seting up the microprocessor is shown in Figure 4. P1.0 (red LED) is used for debugging purposes, and P1.2 is used for controlling the servo motor.

```
1       WDTCTL = WDTPW + WDTHOLD;          // Stop WDT
2       P1DIR |= BIT1 ;                    // Set P1.0
3       P1DIR |= BIT2;                     // P1.2 to output
4       P1SEL |= BIT2;                     // P1.2 to TA0.1
5       CCTL1 = OUTMOD_7;                  // CCR1 reset/set
6
```

Figure 4: Code that sets up the microprocessor

4.3 Analyzing data from the infrared camera

The image produced by the infrared camera has a width of 32 pixels, i.e. 32 columns. For analyzing, I splitted the image into 3 sections, where the 0th to 10th column is the left section, 11th to 20th column is the middle section and 21st to 31st column is the right section (shown in Figure 6).

The code shown in Figure 5 works by accumulating the value of pixels for each section. The higher the value for a section, the hotter the section is.

```
1   int num_row = sizeof(acc)/sizeof(acc[0])/32; // Each row has 32 ele
2   int row_counter = 0; // For tracking row in iteration
3
4   int right = 0;
5   int left = 0;
6   int mid = 0;
7
8   for (int k = 0; k < num_row; k ++) {
9      int start_left = 0 + 32*row_counter;
10     int start_mid = 11 + 32*row_counter;
11     int start_right = 21 + 32*row_counter;
12     int end_right = 32 + 32*row_counter;
13
14     left += sum_array(acc, start_left, 11);
15     mid += sum_array(acc, start_mid, 10);
16     right += sum_array(acc, start_right, 11);
17
18     row_counter++;
19  }
20
```

Figure 5: Code that analyzes the data from the infrared camera

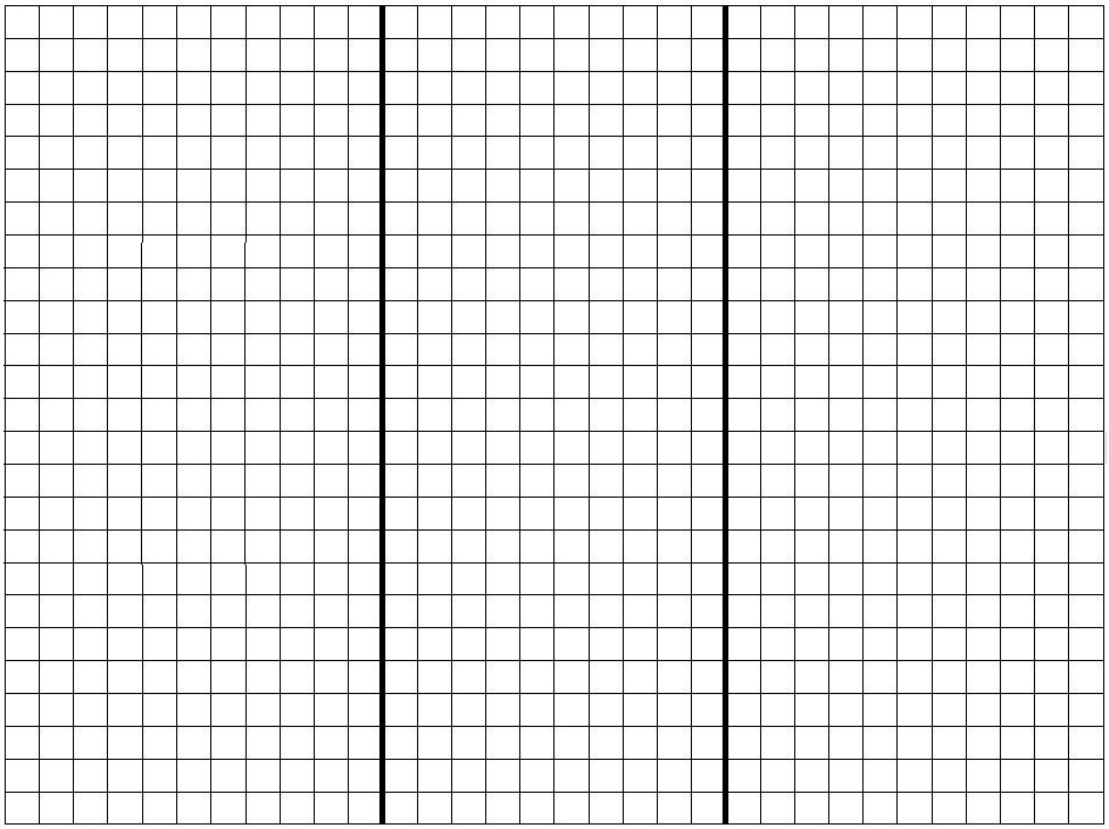

Figure 6: Left, middle and right section of the image from the infrared camera

4.4 Computing the degree of rotation

The code that computes the rotation of the servo motor is shown in Figure 7. The start and end variables control how much to rotate the servo motor, which in turns determine where the camera will be pointing to.

The rotation of the servo motor can be controlled using the PWM, but it seems like the degree of rotation and the value of pulse width is not linear. Hence, trial and error was used to determine the start and end value for each case.

```
if (left > mid && left > right) {
    start = 2290;
    end = 2250;
} else if (left == mid && left > right) {
    start = 1915;
    end = 1875;
} else if (mid > left && mid > right) {
    start = 1540;
    end = 1500;
} else if (right == mid && right > left) {
    start = 1165;
    end = 1125;
} else if (right > mid && right > left) {
    start = 790;
    end = 750;
} else {
    start = 6000;
    end = 0;
}

while (1) {
    for (int i = start; i >= end; i-= 20) {
        CCR1 = i;
        for (int j = 0; j < 10000; j++);
    }
}

```

Figure 7: Code that computes the rotation of the servo motor

As shown in the code, the condition that determines where to rotate the camera to is very simple, and it only supports the case where only one hot object is in its proximity.

The first case corresponds to when the left section contains the hot object (hence, having a value higher than the mid and right). In that case, the camera is suppose to be pointing at 45 degree to the left. An example of the placement of the hot object is shown in Figure 8.

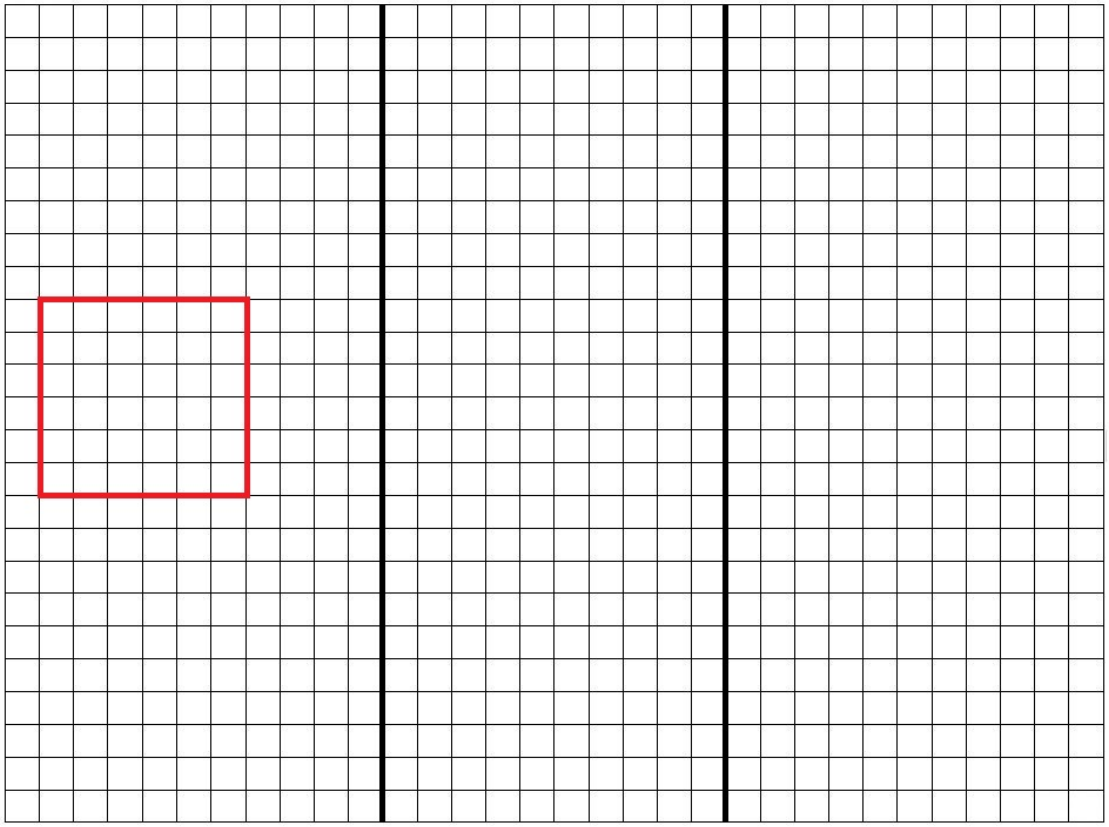

Figure 8: Hot object on the left section of the image from the infrared camera

The second case corresponds to when the left and mid section contain equal amount of the hot object (hence, having a same value for left and mid). In that case, the camera is suppose to be pointing at about 23 degree to the left. An example of the placement of the hot object is shown in Figure 9.

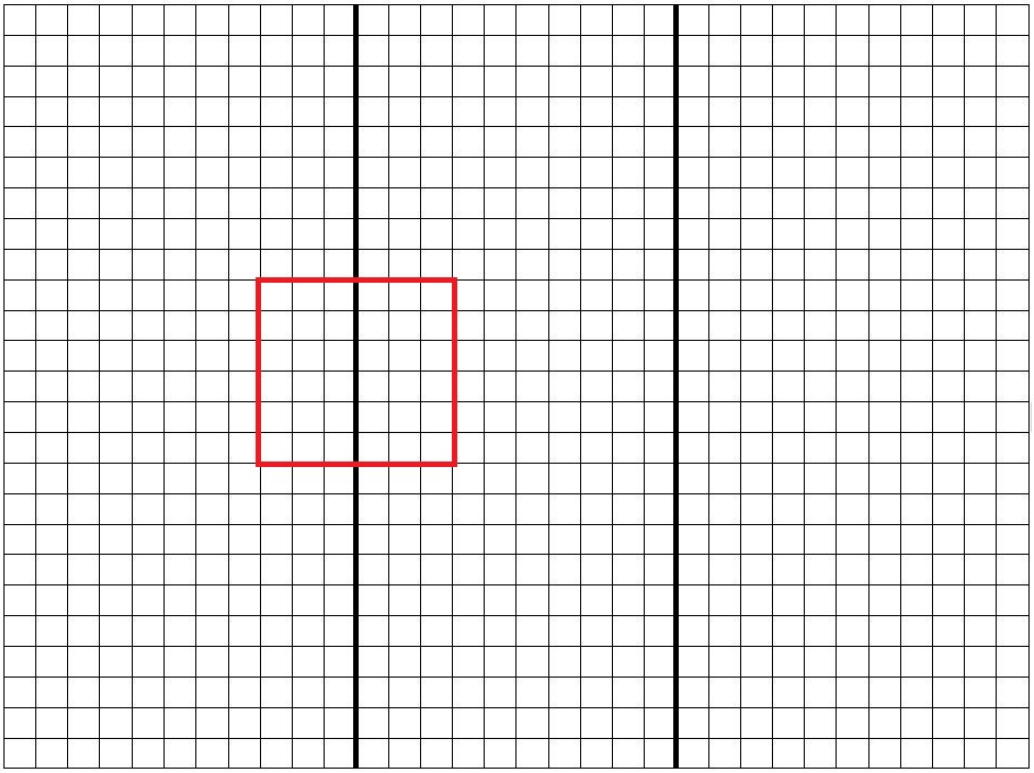

Figure 9: Hot object slightly on the left section of the image from the infrared camera

The explanation is very similar, just that the objects are placed in the middle, slightly to the right, and on the right. In these cases, the camera will be pointing to the respective directions.

The "else" case corresponds to when there is no hot object in the proximity of the detector. Hence it is doing a full rotation (i.e. 180 degree due to the limitation of the servo motor), scanning for potential hot object in its surrounding.

The start and end variables are used inside of the while loop in Line 21 of Figure 7. Within the loop, the CCR1 will continuously be taking the value of i, which is restricted by the start and end variables, hence restricting the rotation of the servo motor. The nested for loop in Line 24 of Figure 7 is just to slow down the rotation to make it smoother.

5 Python program for visualization

I also wrote a python program that visualize what the image from the camera would look like. It helps to check if the code in Figure 3 is produing an expected array.

The python code is shown in Figure 10, 11 and the output of the program is shown in Figure 12.

As mentioned in section 4.1, the seemingly arbitrary values assigned in the "else if" cases are just to make sure that the image produced by the Python program looks smoother.

```
def main(lon: List[int]) -> Image:
    """
    Takes in a list of 32 * 24 ints, converts it into an Image
    """
    size = 10 # CHANGE SCALE OF IMAGE HERE
    row = 1
    acc = square(size, "solid", "white")
    one_row = square(size, "solid", "white")
    for i in range(0, len(lon)):
        if (i < 32 * row):
            one_row = beside(one_row, convert_num_to_col(size, lon[i]))
        if (i == (32 * row)-1): # prep or next row
            acc = above(acc, one_row)
            row = row + 1
            one_row = square(size, "solid", "white") # reset row accumultor

    return acc
```

Figure 10: Code that converts an array (produced by the code in Figure 3) into an image that represents to the output of an infrared camera

```python
from cs103 import *
from typing import *
import numpy as np

def convert_num_to_col(size: int, n: int) -> Image:
    """
    Converts a number into a square of the appropriate color
    Variation of colors: white, red, orange, yellow, green, cyan, blue, darkblue, purple, black
    Note: n should be in the range of (0, 255)
    """


    step = 25
    if (n < step):
        return square(size, "solid", "black")
    elif (n < step * 2):
        return square(size, "solid", "purple")
    elif (n < step * 3):
        return square(size, "solid", "darkblue")
    elif (n < step * 4):
        return square(size, "solid", "blue")
    elif (n < step * 5):
        return square(size, "solid", "cyan")
    elif (n < step * 6):
        return square(size, "solid", "lightgreen")
    elif (n < step * 7):
        return square(size, "solid", "yellow")
    elif (n < step * 8):
        return square(size, "solid", "orange")
    elif (n < step * 9):
        return square(size, "solid", "red")
    else:
        return square(size, "solid", "white")

```

Figure 11: Helper function that converts an int into a pixel of a color based on the value of the int

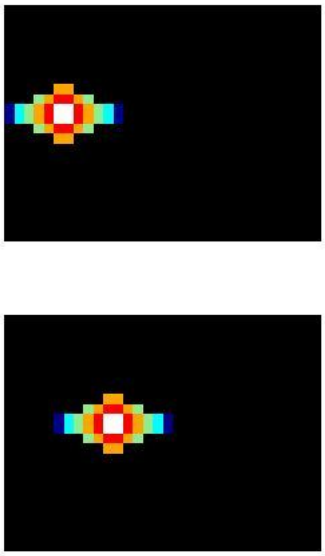

Figure 12: Mocked infrared camera images produced by the Python program (the first image corresponds to Firgure 7, where the hot object is on the left side of the image, and second iamge corresponds to Figure 8, where the hot object is slightly on the left side of the image)

6 Limitations/ Improvements

As mentioned briefly in section 4.4, the servo motor can only turn 180 degree. This means that this system will not be able to detect hot object on the other side of the camera. One way to solve this problem is to use 2 infrared cameras, so that the system can cover a 360 degree view of its proximity.

Also, the condition that determines where the camera should rotate, and how much to rotate is very simple. An improvement can be made by dividing the image up into more sections, i.e. 8 sections instead of just 3. This will improve the precision and accuracy of the location of hot object detected by the system. Also, with a more complex condition, the system can detect more hot objects, hence, able to make better decision on where to turn the camera to. This will allow the potential robot platform (mentioned in section 2) to have higher chances of survival in an environment with thermal hazards.

[1]Note: All codes written for this project is on my GitHub repo.