Ebola Prediction Model

Capstone project 2018

Prepared for:

Professor Bill Smart

Prepared By

Group 34

Brian Lee Huang

Claude Maimon

Bianca Beachamp

Introduction

The goal of this project is to be able to estimate a person's core body temperature by using just a person skin temperature. We tried doing this by using a thermal camera to get thermal data from a person's face and using a model to make some correlation between a person's skin temperature, and a person's core body temperature. The purpose of this project was to have it be used during an Ebola outbreak. The idea was to quickly quarantine potentially sick patients without doctor interaction. Removing doctor interaction removes the risk of the doctors getting sick and also reduces the possibility for the virus to spread to other patients.

Code

Our code can be found at https://github.com/maimonc/Ebola-Virus-Project/Code

Overview of the code

Our code has two main modes. One is a learning mode and one is a production mode. In the learning mode, the images processing code and the model code are used separately. In the production mode, the separate pieces of code are used together to output an estimated temperature.

In the learning mode, the image processing code processes a folder of CSV files that were taken in advance. This mode is to be used when trying to collect data to train the model. While using this mode, the user should collect the actual temperature of the people that are being photographed. After running the image processing code on the CSV folder, the thermometer temperatures need to be added to the file that the code produces. Then, that file (that holds both the actual and estimated temperature) needs to be sent into the model to create a new model.

In the production mode, the whole process happens at the same time. The code can only run while the camera is connected to the computer and can only run on Linux environment. When running the code, the camera will take a few pictures in a row, our code will process them and produce an estimated core body temperature.

Image Processing

The image processing code is used to process an image CSV file and produce a temperature mean value. That value can later be sent to the model code to create a model. The code finds the head of the person in the image, calculates the mean of the head pixels and returns that value. This code doesn't include the process of connecting to the camera and producing a CSV from the image. That code is available under the production mode.

Running the program

In order for the program to run, a folder with CSV files needs to be specified. That folder should hold the CSV files that were produced by the camera. There are two ways to include the folder. The first is via a command line argument. If a folder path is specified at runtime (python images2.py <folder path>) the program will use the files in that folder. However, if a folder was not specified, the program will use a built-in folder path that is specified in the MYPATH variable. It is recommended to use the built-in option when running this code in production more since the folder will not change. However, while training the model, it will be best to use the command line option.

Images.py line 124:

```
if (len(sys.argv) == 1):
   MYPATH = 'C:\Users\Tair\Documents\Pictures\Real\Resting'
elif (len(sys.argv)==2):
   MYPATH = str(sys.argv[1])
```

Correcting Values

In the problems and solution section of this report, we will discuss some problems that we experienced with the camera. Our main problem was calibrating the camera. Since we couldn't find a good way to calibrate the camera completely, we tried to use our code to "fix" the values a little.

It's important to state that this is not a good solution. We couldn't find a way to fix the problem so we tried this to help a little. We found that at some times the background values are higher than usual. It looked as if subtracting the background value from all values helps to get the head values into a better range. We couldn't rationalize why this

is happening and we don't think this approach is right. We only used it so we could start collecting data and proceed with the project.

The following function, correcting_values, demonstrate how we tried to help with the calibration problem. This function is simple, it finds the min value in the temperature Data list (that's that list that holds the CSV values), and then it substructs that value from all the values in the list. Since it's a "two-dimensional" list, the code finds the minimum value of each inner list and compares it to the absolute minimum variable min_value. Once the code finds the absolute minimum value, it iterates over the list and subtracts that value from every element of the list. Then, the code simply returns the modified list.

```
def correcting_values(temperatureData):
    #setting initial values so everything would be smaller
    min_value = 1000
    #goes through the list to find the absolute minimum
    for elements in temperatureData:
        temp_min_value = min(elements)
        if temp_min_value < min_value:
            min_value = temp_min_value
    # min_value = min(temperatureData)
    if DEBUG:
        print min_value
    #subtract the minimum value from all values
    for row_counter, elements in enumerate(temperatureData):
        for column_counter, element in enumerate(elements):
            element = element - min_value
            temperatureData[row_counter][column_counter] = element
            # [float(i)-min_value for i in elements]
            # print temperatureData[row][column]
    #returns the new list with changed values
    return temperatureData
```

Finding the Person's Head Starting Point

The following function, starting_point, receives the temperatureData list (that's that list that holds the CSV values) and returns a list. The list either holds two values, x and y, that represent the coordinates where the head begins, or it can be empty if the function failed. Y is the inner list number (the row in the CSV) and X is the element number inside that list (the column in the CSV). If the image is bad or if our code had an error, it is

possible that the code won't find a starting point. In this case, the returned list would be empty.

The method to find the starting point is the following: the code goes through every inner list (every row) until it reaches an element that is in the right range (> 30 and <45). Then, the code checks if the next four elements in the list are also in the right range. If there are five values that are in the right range in a row, our code assumes that where the head begins.

```
def starting_point(temperatureData):
   for row_counter, elements in enumerate(temperatureData):
       for column_counter, element in enumerate(elements):
           if element < 45 and element > 30:
               good = []
               for iterator in range(1, 5):
                   # print len(elements)
                   if (column_counter + iterator) < len(elements):
                       if elements[column_counter + iterator] < 45 and
elements[column_counter + iterator] > 30:
                           # print "added to good"
                           good.append(iterator)
               # print len(good)
               if len(good) == 4:
                   # print "in"
                   coordinates = []
                   coordinates.append(row_counter)
                   coordinates.append(column_counter)
                   return coordinates
   return []
```

Getting the Mean of The Head Pixels

In order to get the mean temperature value for the head, we had to come up with a method to decide where the head is. Bill allowed us to assume that the head will usually be about the same size. After finding the coordinate where the head "begins", we assume the head is going to be around 40 columns wide and 30 rows long. These numbers are base on an image of a person who is standing 2 meters away from the camera.

The code uses those values (40, 30) in the following way: It uses the starting point (from the last function) as the top of the head. However, since the head of a person is wider in

the center of the face (as shown by the temperatures in the CSV file), we assumed that the top of the head is about the center of the head horizontally. This is why the code goes from y to y+Height vertical but from (x - width/2) to (x + width/2).

There might be better ways to extract the head from the picture. We used this method for simplicity and because we're only using pixels that are in a specific range. That is, if we go too far from the head it doesn't matter because those values will probably be too cold and won't be counted towards the mean.

```
def get_mean(x, y, WIDTH, HEIGHT, temperatureData):
    if ((y + HEIGHT) > 120 or ((x + (WIDTH / 2)) > 160)):
        print "PERSON OUT OF RANGE"
        return 0
    for row in range(y, (y + HEIGHT)):
        # y is the row it begins, h is the width of the length of the head
        for column in range(x - (WIDTH / 2), x + (WIDTH / 2)):
            if (temperatureData[row][column] > 43 or temperatureData[row][column] <
35):
                continue
            else:
                good_range.append(temperatureData[row][column])
    if (len(good_range) == 0):
        return 0
    range_mean = sum(good_range) / float(len(good_range))
    return range_mean
```

The output of the program

In the training state (not in the production mode) the program outputs a CSV file. In the file, there are two entries in every row, one for the calculated mean and one for the name of the processed file. Outputting the name of the files helps with matching the calculated mean value with the thermometer temperature. After processing the CSV folder, the user should add the temperatures taken with a thermometer to this file. After matching the calculated value with the thermometer value, the file name column should be deleted. At that point, the file would be ready to be sent into the model code.

Model

Our model

Our current model is created by using linear regression, specifically, Least Squares Linear Regression. This method works well for the most part and can be made more robust with larger datasets. However, it does not account for other variables that could alter a person's skin temperature or their core body temperature. This could potentially prove to be troublesome in the long run, so different models should be tested to try and solve those problems.

Other possibilities and their benefits

One possibility for another model to test is a multivariable regression line, where the slope is dependant on multiple different factors. This way factors like weather, ambient temperature, humidity, if the picture is taken indoor or outdoor, windchill, and activity before the picture was taken can all be taken into account. However, this method requires a little bit more thought behind it and may require more user input as they need to input those variables everytime the program is run.

Production Mode

The production mode is the culmination of all the parts of this project. It combines the image processing and the model into one program. The code has two parts, one to create the model and one to take the picture and process it. Creating the model is very simple, it creates a Least Squares Regression Line with the data provided. It is run with the same program, but with a different command. We currently only have a linear regression, but more can be added in at a later time.

The actual production mode where a picture is taken and an estimated core body temperature is produced is ran with a different command. It is run with two different functions within the program. The first function creates a folder called 'Images' in the current directory which will be where all the images are created, stored and analyzed as to not clutter up the current directory. Then it takes 40 frames worth of images, but only the last 30 are kept. We do this to prevent any frames where the camera is warming up where it produces garbage frames of data. However, there is a chance that it can produce 40 frames of garbage, so it is up to the user to watch to make sure that doesn't

happen. However there is error checking in the image processing so an error will be thrown.

```
def takeThermalImage(configFile):
    path = "./Images"
    name = "image"
    with IrCamera(configFile) as ir_cam:

        for i in range(40):
            data_t, data_p = ir_cam.get_frame()
            cv2.imshow('visual data', data_p)
            print '----------------'
            print data_t
            print '----------------'
            cv2.waitKey(5)

            n_rows, n_cols = data_t.shape
            # print data_t[0]

            if (i > 10):
                filename = str(name + str(i) + ".csv")
                with open(os.path.join(path, filename), 'w') as csvfile:
                    thermalwriter = csv.writer(csvfile)
                    for row in range(n_rows):
                        thermalwriter.writerow(data_t[row])
```

After the first 10 frames, the code will then start to save the data as csv files in the ‘Images’ folder. The second function that the code is the `processImage()` function in the image processing module.

```
config = sys.argv[1]
modelData = sys.argv[2]

# Taking the image using the first config file.
takeThermalImage(config)

# Getting the means of all the files.
imageMeans = images2.processImages()

if (0 in imageMeans):
    print "An error has occurred. Try again."
else:
    # Getting the mean of all the means
    totalMean = mean(imageMeans)

    # Running the total mean through the model.
    print "Raw Processed Temperature: ", totalMean
    print "Predicted Core Body Temperature: ", modelPredict(totalMean, modelData)
```

```
    print "DONE"
```

The program depends on two command line arguments, the first is the configuration file for the camera and the second is the model. The function that runs the camera requires the configuration file, while the prediction function requires the model information. The piece of code above is the main portion of the program. It runs the function that takes the pictures, then runs the image analysis code we wrote on all the csv files that were produced. The analysis code analyzes the images and produces a mean temperature of the persons head. The variable `imageMeans` is a list of means created by the code.

Within the image analysis code there is error checking to check for if the person is out of frame, garbage frames or irregular temperature spikes produced by the camera. If any of these are found it produces a 0 and immediately returns. The production mode code then check if there is a 0 within the list of means, and if there is an error is thrown. Otherwise it takes the average of the list of averages and inputs it into the least squares model. The output of the model is then output onto the screen as the predicted core body temperature.

```
def modelPredict(rawTemp, modelDataName):
  modelData = reader.commaReader(modelDataName)
  predictedTemp = (rawTemp * float(modelData[0])) + float(modelData[1])
```

Since we are using a linear regression line the model is a simple linear line. The model data contains a slope and intercept

Command Line

The production mode has two modes. The first being model creation and the second being taking the pictures and analyzing it. These two commands are ran based on the command line arguments passed to the program. Creating the model is ran by simply adding the '-c' tag and the training data. Where training data is a list of number pairs where the first is the skin temperature produced by the camera and the second is the real core body temperature. These two values should be separated by a space. The command should look like this:

```
Python productionMode.py -c trainingData.txt
```

To take the picture there is a slightly longer command as there are more arguments that need to be passed. The command should look like this:

```
Python productionMode.py config.xml model.txt
```

The first command argument is the configuration file for the camera and the second is the model information for predicting core body temperatures. For our current model we used a least squares regression line, so the model file is simply slope and intercept. More model types can be added in later if needed but we only have a linear regression right now.

Testing

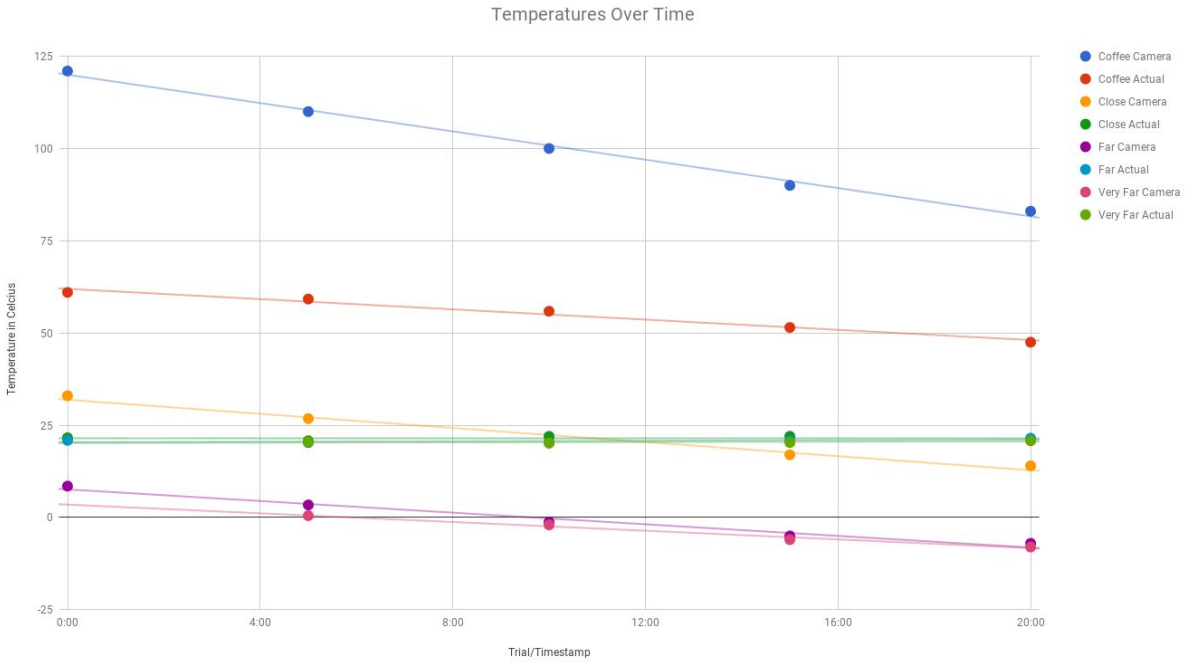
While running the code we ran into some edge cases where errors will occur. These included things like the person not being in frame, the person being too tall or too short, or camera errors such as garbage frames or strange temperature spikes. We tried to account for all of this in the image analysis portion by adding in error handling where we would output an error to the user. There may be even more edge cases we haven't run into yet, but for best results try and have the person's head in the center of the frame.

Problems and Solutions:

Camera

Our projects rely on data from the camera. In order to create a good model and even a good image processing process, we needed the camera to be reliable. We ran into many problems with the camera, mainly due to bad calibration. In the beginning, we used the camera software to test the camera. While using the software it was easy to see that the camera returned bad values. In the beginning, it returned about 60 degrees Celsius as the temperature of a person. It also returned about 100 degrees Celsius for a cup of coffee even after it set for a while. We tried a few methods to test the camera, one of them was to test the change of temperature of a cup of coffee on a table. We put a hot coffee cup on a table and took a picture of the table every few minutes. The data that we got is included below. As times passed and the coffee's temperature went down so did the table temperature.

Trial/Time stamp	Coffee Camera	Coffee Actual	Close Camera	Close Actual	Far Camera	Far Actual	Very Far Camera	Very Far Actual
0:00	121	61	33	21.6	8.5	20.9		
5:00	110	59.2	26.8	20.8	3.4	20.3	0.5	20.5
10:00	100	55.9	21	22	-1.2	20.2	-2	20.2
15:00	90	51.5	17	22	-5	20.8	-6	20.3
20:00	83	47.5	14	21	-7	21.5	-8	20.8



We also tried to change a lot of the variables in the software such as ambient temperature and adding a reference temperature. None of these helped. There is a folder of a few images with temperature values in the camera folder. It's called Samples. We used that folder as a calibration folder for the camera and it made the camera work better, but it was still a little off. In the production mode, where we don't use the camera's software to take the pictures, the code just uses the default calibration file.

Data

This kind of project relies heavily on data. We ran into many problems with collecting data. First, as mentioned before, the camera returned bad values. We still collected some data but it's not reliable. It was hard to predict how the camera will behave when taking pictures. Another problem, even bigger than the camera problem is the nature of the data. In order to create a model that predicts high fever we needed to have pictures of people with a fever or high body temperatures. This kind of data is hard to collect. As part of this class we weren't allowed to use random people to collect data from. We ended up using only the three of us for collecting data. Bill's graduate student helped us by providing some extra data but it was still of people with normal temperatures.

Results

We were able to create a program which takes a picture at and produces a estimated core body temperature. However our current version may not be accurate as the model may be weak due to the lack of data. As a proof of concept were were successful in making something that could work, if provided with a large data set, and a fully functioning calibrated camera. The current methods for image processing and model creation are somewhat simple, and could be improved upon at a later time. The model is a least squares regression line so it does not account for many other variables that could affect a person's skin temperature and core body temperature. The image processing code is a simple search through a csv to return a batch of values roughly in the area where the person's head is. This works for the most part, but does not account for things like long hair or beards.

Overall we create a simple proof of concept to estimate core body temperature through a thermal image. The overall accuracy of this project is dependant on the data set used, and the complexity of the model. With a larger data set the program could be fairly accurate.

Recommendations for Future

Model

We are currently using a least squares regression line for our model creation. This creates a simple linear line which produces a slope and intercept to correlate a person's skin temperature to their estimated core body temperature. This simple method could be the best method, but we did not much data to create a more robust model.

Multiple linear regression is the next logical step to test for the model. It can account for more variables that could possibly change someone's core body temperature. Some of these variables could be the weather, the ambient temperature, and the humidity.

Camera

The camera must be calibrated correctly in order for this project to work. Currently, the code uses the default calibration. We were stuck on this problem through the whole project. We couldn't find a way to get the camera to work correctly. It also wasn't in our requirements to calibrate it. We think that a possible fix to the problem might be to take more pictures and videos with the camera, add to them their temperatures, and use those as calibration files. We weren't able to do this, but it might help increasing the accuracy of the camera.

Data Collection

While presenting our project at the undergraduate engineering Expo, we've noticed many factors that could change the camera readings. It might be helpful to pay attention to these factors while collecting data to train the model.

- The distance from the camera: the closest to the camera, the better the reading. People that were standing farther than 2.5 meters away seemed colder in the image.
- The camera's range: When standing 2 meters away from the camera, a lot of people were either too short or too high to fit in the frame. We had to change the camera's angle many times to get the people in the frame.
- Glasses: glasses are really cold on the camera. It might be best to make people take their glasses off for the picture. It might make the mean smaller.

- Hair: hair is colder than the skin. Therefore it might be problematic to run the code on people with a big beard or mustache. It might be helpful to train the model on people with such characteristics.
- Some people are just way hotter than others. It was clear that even though people were in the same conditions, some were really hot even though they didn't have a fever and didn't work out. When collecting data, it might be helpful to collect data on people that are usually colder and people that are usually hotter.

Code Modifications

We are currently using somewhat rudimentary methods for both the model and finding the person's head in the image. Our model is a simple least squares regression line so it simply creates a best fit line based on a person skin temperature and core body temperature. This can be thrown off my many different factors, like if it is cold outside skin temperature will be lower while their core stays the same, or if the person has just exercised their core body temperature will be higher than normal. If a different model is used it should be more robust, and take into account different factors such as the weather, and ambient temperature.

Our image processing code simply scans the csv file, looking for temperatures within skin temperature range then outputs an average of those temperatures. It does some error checking in the production mode and it also tries to isolate the head of the person. However, it does not account for things like long hair, beards, glasses or piercings. These things are colder than human skin and could potentially produce errors with our code. Our code does a decent job, but it could be improved upon by using something smarter like some kind of facial recognition where hair, beards and glasses won't throw off the data gathering.

Conclusions

This project relies heavily on data. As mentioned in this report, we weren't able to collect good data. Both the camera problems and the nature of the needed data made it hard to achieve that goal. We believe that once the camera is fixed and enough data is collected, our code can be used to estimate a core body temperature of a person using a thermal image. There are some code modifications that need to be done. Also, new prediction models need to be tested. But more importantly, a lot of data needs to be collected for this project to succeed.