Keyboard Layout Analysis using Python

Siva Sundar, EE23B151

June 4, 2025

1 Overview

1.1 Abstract

In this assignment, we will develop a Python program that processes string input and computes the cumulative distance covered by a typist, defined as an individual adhering to a systematic typing methodology. Additionally, the program will generate a heatmap illustrating the frequency of keypresses for each key, where the highest frequency is represented in red and the lowest in faded blue, with the degree of fade corresponding inversely to the key's usage frequency.

1.2 Introduction

Since the invention of the typewriter, the arrangement of keys has evolved to **optimize speed**, **efficiency**, and **user comfort**. Today, the most widely used layout is the QWERTY layout, designed by Christopher Sholes in the 1870s. Initially created for *mechanical typewriters*, the QWERTY design was meant to reduce the likelihood of **mechanical jams** by spacing out frequently used letter combinations. While the original purpose of QWERTY is no longer relevant in the digital age, it remains the dominant layout due to widespread adoption and familiarity.

Figure 1: Piano-styled keyboard layout used in 1900s

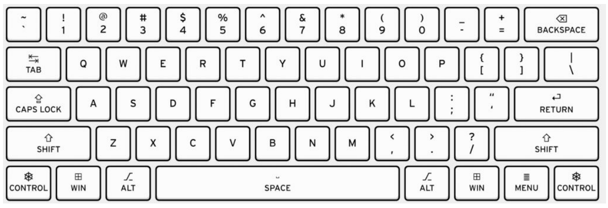


Figure 2: QWERTY layout

As typing became a fundamental skill with the rise of personal computers, alternative keyboard layouts, such as Dvorak and Colemak, were developed to improve typing efficiency by minimizing finger movement. Despite these innovations, QWERTY remains the standard due to the inertia of existing habits and systems.

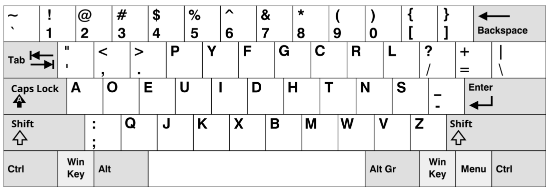


Figure 3: Dvorak layout

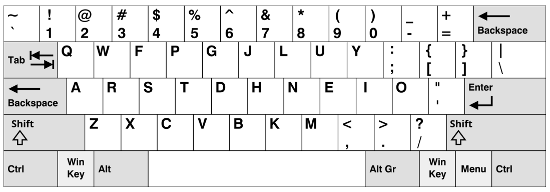


Figure 4: Colemak layout

In fields like programming and data entry, where speed and accuracy are crucial, keyboard layout efficiency remains important. Studying finger movements on a keyboard can improve ergonomics, reduce strain, and boost productivity. Our program simulates typing in QWERTY as well as other layouts to analyze patterns and visualize workload distribution via a heatmap.

1.3 Methodology

The approach for this assignment involves the following key steps:

1. **Data Extraction and Key-press identification:** Utilize Python to read data from command line and find the corresponding key presses used by the typist to type each of the letters in the input. To find the key presses, we use the given layout.

2. **Finding the key-press frequency for each key:** After knowing the sequence of key presses, we can find the frequency of key press for each key by iterating through the sequence.

3. **Visualizing the given keyboard layout:** `matplotlib` is used for visualization purposes. Using voronoi diagrams for the set of keys whose locations are given, we can easily draw a keyboard representation.

4. **Frequency heatmaps:** Using circles with different colors and transparency, I create a heatmap which can be used to visualise the key-press frequency for all the keys. Key with highest frequency gets marked by a red circle with least transparency and the key with least frequency gets marked with a blue circle of highest transparency.

5. **Tabular representation:** By identifying the keys used to type the input, their home keys, locations, and press frequencies, I can create a data table for verifying the distance calculations.

For **BONUS**, the same steps are done for **other keyboard layouts**, and by comparing the total distance, I find the best layout which minimises the travel distance. Also, **animations** are used for visualizing each keypress on the keyboard with a **modified heatmap** showing *cummulative* frequency distribution.

2 Data Extraction and Key-press Identification

Using `input()`, we take in the user input from terminal. this string is then passed to `find_key_press()` which returns a list of keys used and another list with the locations of these keys.

```python
def find_key_press(string, layout):
    key_press = []          # To store labels of the pressed keys
    key_press_loc = []      # To store location of these keys

    for letter in string:
        # if letter is space:
            # statements
        # elif letter uses shift key:
            # statements
        # else letter is lower key:
            # statements
    return key_press, key_press_loc
```

Code Snippet 1: `find_key_press()` definition

3 Finding the key-press frequency for each key

Now that we have the list `key_press` containing the sequence of key press, we can create a dictionary `key_freq` which stores each key label as dictionary's 'key' whose 'value' is the frequency. This is done using `key_count()` function.

```python
def key_count(key_press):
    key_freq = {}
    for key in key_press:
        if key not in key_freq:
            key_freq[key] = 1
        else:
            key_freq[key] += 1
    return key_freq
```

Code Snippet 2: `key_count()` definition

4 Visualizing the given keyboard layout

Given the keyboard layout, we have all the key locations (x, y) which can be stored in a list and this is used to create a `voronoi()` class which is defined in `scipy.spatial` library. Using the function `voronoi_plot_2d()` which takes in the voronoi class, we can plot the voronoi diagram which represents our keyboard layout.

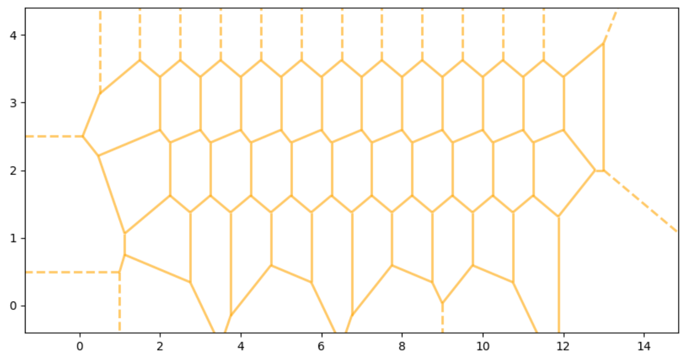


Figure 5: Voronoi diagram representing the physical layout (For QWERTY)

To add the key labels, we store the labels ([lower label, upper label]) for each key in a list which is then used to display the labels on the voronoi diagram. If the key only have lower label, upper label is set to `None`.

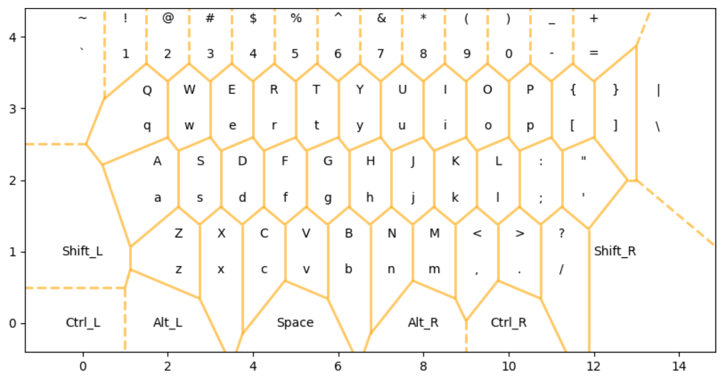


Figure 6: Voronoi diagram with physical and visual layout (For QWERTY)

Creating the lists for labels and key locations are done using `get_points_labels()` which takes in keyboard layout as input. The list of locations (`point`) is then used to make voronoi diagram and the list of labels (`labels`) is used to add the key labels. This part is done by `plotter()`.

5 Frequency Heatmaps

We can represent the frequency of each key using circles with different colors and transparency. The `coloring()` function takes a dictionary, `key_freq`, where each key is a label and the corresponding value is the frequency of that key. The function returns another dictionary, `key_color`, which maps each key to a list containing a color and an alpha (transparency) value.

I computed transparency (alpha) based on the ratio of the key's frequency to the maximum frequency in the dataset: (factor of 1/2 is just to make it more transparent)

$$\alpha_{\text{key}} = \frac{\text{frequency of key}}{2 \times \text{maximum frequency}}$$

The color for each key is assigned using an `if-elif-else` structure as seen in the code snippet below:

```python
def coloring(key_freq):
    # Showing a part of this function
    for key in key_freq:
        alpha = key_freq[key]/max_freq
        if alpha >= 1:
            key_color[key]=[color[0],alpha/2]
        elif alpha < 1 and alpha >= 0.6:
            key_color[key]= [color[1],alpha/2]
        elif alpha < 0.6 and alpha >= 0.2:
            key_color[key]=[color[2],alpha/2]
        elif alpha < 0.2:
            key_color[key]=[color[3],alpha/2]
```

Code Snippet 3: `coloring()` definition

After creating the `key_color` dictionary, using the `plotter()`, we can plot circles over the keys with the appropriate color and transparency.

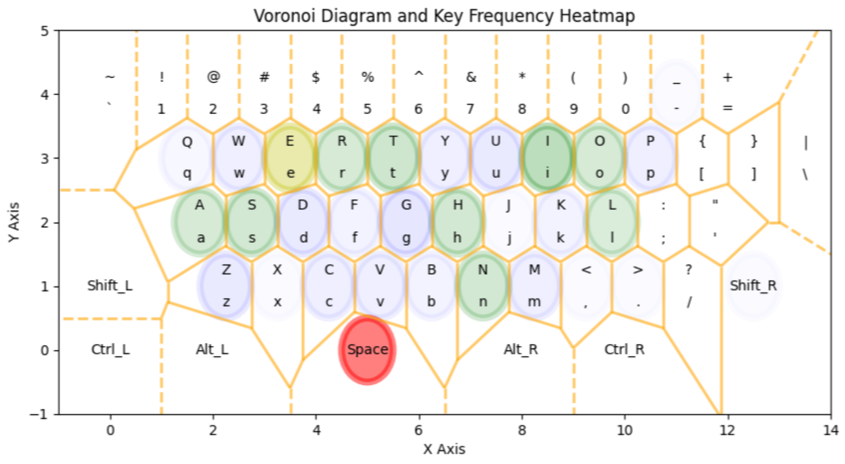
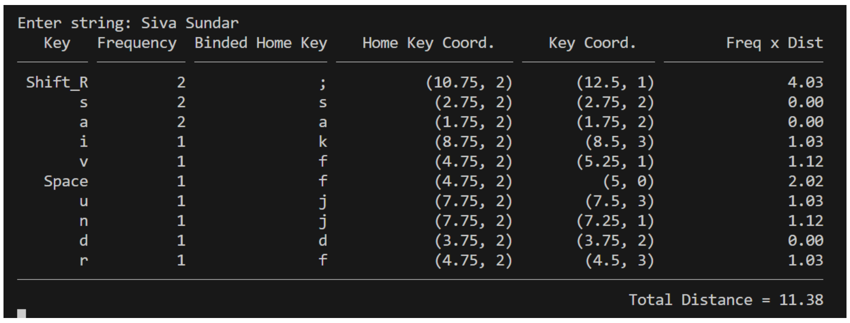


Figure 7: Heatmap example

6 Tabular Representation

Using the `key_freq` dictionary and the given layout, we can identify the home key associated with each key in the frequency dictionary and their respective locations. By calculating the Euclidean distance using `math.dist()` and multiplying it by the key's frequency, we determine each key's contribution to the total distance. Summing these contributions yields the total distance.



```
Enter string: Siva Sundar
   Key    Frequency  Binded Home Key     Home Key Coord.      Key Coord.         Freq x Dist

 Shift_R          2                ;         (10.75, 2)       (12.5, 1)                 4.03
       s          2                s          (2.75, 2)       (2.75, 2)                 0.00
       a          2                a          (1.75, 2)       (1.75, 2)                 0.00
       i          1                k          (8.75, 2)        (8.5, 3)                 1.03
       v          1                f          (4.75, 2)       (5.25, 1)                 1.12
   Space          1                f          (4.75, 2)          (5, 0)                 2.02
       u          1                j          (7.75, 2)        (7.5, 3)                 1.03
       n          1                j          (7.75, 2)       (7.25, 1)                 1.12
       d          1                d          (3.75, 2)       (3.75, 2)                 0.00
       r          1                f          (4.75, 2)        (4.5, 3)                 1.03

                                                                     Total Distance = 11.38
```

Figure 8: For the input string "Siva Sundar"

The `print_table` function organizes these steps, formatting the output into a visually appealing table. The dictionary is sorted so that the first row corresponds to the key with the highest frequency, decreasing thereafter.

7 BONUS

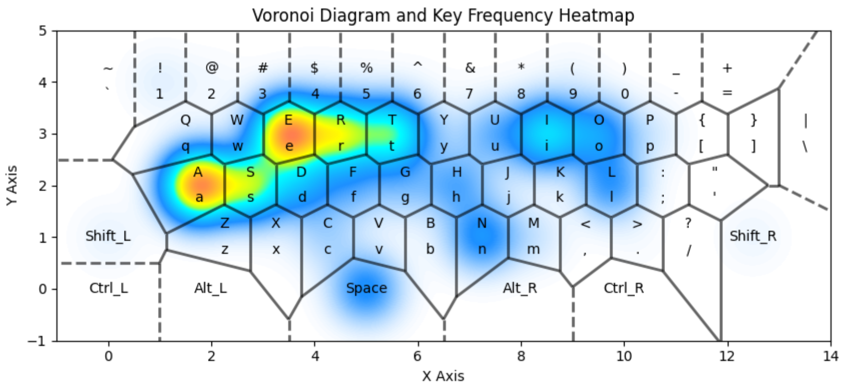
7.1 Using Different Layouts

I created two additional files: `dvorak_layout.py` and `colemak_layout.py`, each containing 'keys' and 'characters' dictionaries similar to those in `qwerty_layout.py`. These files are assumed to be in the same directory as the notebook. I performed the same steps as for the QWERTY layout, generating tables and plots accordingly.

7.2 Animation and Modified Heatmap

To create a smooth heatmap, I used the reference [2]. The example utilized the `imshow()` function, which I adapted to use key locations and frequency values. For the animation, I based my implementation on a reference code provided by the professor, which uses `FuncAnimation` to create animated plots.

You can view the animated heatmap for a larger input string (using the Dvorak layout) by clicking this link: **CLICK HERE**.



8 Results

See **jupyter notebook** (which also contains **BONUS** part) for the python code. How to run the code, is explained in the notebook. Below is a table of results for some string **inputs** I gave for different layouts:

Layout	**str_1**	**str_2**	**str_3**	**str_4**
QWERTY	509.96	1123.93	93.24	1590.03
Dvorak	367.64	822.31	87.96	1594.57
Colemak	332.94	792.00	80.87	1602.44

Table 1: Total distance taken in each string by a layout

9 Conclusion

We can see that, Colemak is the **best** for str_1 (small paragraph), str_2 (big paragraph) and str_3 (small code), while for str_4 (big code), QWERTY layout is slightly better than the two.
Hence, I conclude that Colemak is best for writing eassays, novels and similar literature, while QWERTY is best for writing codes (especially, large codes).

10 How to run the Notebook?

1. Download the notebook and layout files.
2. Ensure all of these files are in the same directory.
3. Press `Run All` in the notebook. It will prompt for string input four times:
   - First, for running the code snippet showing analysis and frequency heatmap for the QWERTY layout.
   - Second, for the code snippet that analyzes all three layouts.
   - Third, for the code snippet for animation, which will ask for string input for both the text and the layout.
4. Comments are provided in the notebook to guide you through the code.

11 References

[1] How to Properly Draw Circles in Python and Matplotlib?

[2] Matplotlib documentation: Blend transparency with color in 2D images

[3] Matplotlib documentation: List of named colors

[4] Matplotlib documentation: matplotlib.axes.Axes.imshow

[5] Matplotlib documentation: Animations using Matplotlib

[6] SciPy documentation: `voronoi_plot_2d`

[7] Printing formatted tables in Python

[8] Wikipedia: Keyboard Layout