Calculating screen to face distance

Ivan Ludvig Tereshko [1]

[1]Moscow Institute of Physics and Technology

October 30, 2023

Abstract

A method of calculating screen to face distance is presented. The method relies on average distance between the user's eyes (pupillary distance) and does not require calibration. The algorithm is implemented as an Android application using face detection technologies provided by Android.

Calculating screen to face distance

Ivan Ludvig Tereshko

Tereshko.IV@phystech.edu

August 20, 2020

Abstract

In this paper, a method of calculating screen to face distance is presented. The method relies on average distance between the user's eyes (pupillary distance) and does not require calibration. The algorithm is implemented as an Android application using face detection technologies provided by Android.

I. Introduction

Screen to face distance has various applications in digital technology. The value can be used in order to implement a more responsive and user-friendly graphical interface. A more progressive application of screen to face distance may be in AR/VR technologies as it grants the ability to connect virtual and digital space.

Another application may lie in the medical field. For example, the measured value can be used to inform the user to keep the device at a more appropriate and comfortable distance. This is especially vital for people with poor eyesight, but the application can prevent any possible harm to any user's eyesight during continuous use of electronic devices.

II. State of the art

This problem was previously solved by KÃűnig, Beau and David (2014). The method presented uses the distance between a person's eyes to calculate screen to face distance. [3] The method presented in this paper uses the same principle, but has differences in the algorithm. Unlike the algorithm of KÃűnig et al. (2014), the following algorithm doesn't require calibration. Instead, it relies on the optical parameters of the camera.

III. Solution

i. Physics

The main equation used in this method can be derived from the lens equation. The following variables are used:
dist_mm - screen to face distance in mm,
F_mm - focal length in mm,
sensor_mm - sensor size in mm,
sensor_px - sensor size in px (image size),
obj_mm - real object size,
obj_on_sensor_mm - object size as it appears on the sensor in mm,
obj_px - object size in pixels.

Using basic lens optics (similar triangles on both sides of the lens), the following ratio is obtained [2]:

$$\frac{sensor\_mm}{F\_mm} = \frac{obj\_on\_sensor\_mm}{dist\_mm}$$

Considering the following relation (the ratio between millimeters and pixels) [2]

$$\frac{obj\_on\_sensor\_mm}{obj\_px} = \frac{sensor\_mm}{sensor\_px}$$

the final equaion is obtained

$$dist\_mm = F\_mm * \frac{obj\_mm * sensor\_px}{obj\_px * sensor\_mm} \quad (1)$$

In the current case pupillary distance represents the object. Therefore, the object's dimensions are calculated by subtracting coordinates

of one eye from the other. The object and sensor sizes are either their width or height. If the object's width is bigger than it's height, width represents size in (1) (both object and sensor width). Otherwise, height is used as size. The following approach let's us avoid borderline cases when, for example, width is much bigger than height.

ii. Algorithm

Android's face detection technology is used in the current method. The face is continuously detected using the front camera.

Overall, the method can be decomposed into the following steps:

1. Camera and face detector initialisation.
2. Obtaining camera parameters: sensor dimensions, focal distance.
3. Continuous update, which includes:
    - Face detection
    - Eyes position detection
    - Screen to face distance calculation, using equation (1)

In equation (1) the variable *sensor_px* represents image size, which is set during step 1. *F_mm*, *sensor_mm* are camera parameters, which are obtained at step 2. *obj_px* is pupillary distance in pixels (as it appears on the image), which is calculated at step 3.

Finally, *obj_mm* is a constant: it is the mean pupillary distance, which equals 63 mm. [1]

IV. Experiment

The algorithm has been implemented as an Android application, which displays the measured distance. The application has been tested on 3 Android devices:

- Huawei P10 Lite
- Huawei U8950
- Samsung J5

All devices have been tested at 4 distances: 100 mm, 150 mm, 200 mm, 400 mm. Each device was placed at a given distance from a person's eyes. Then, the value that was shown on screen was noted. The result's are presented on a graph (Figure 1).

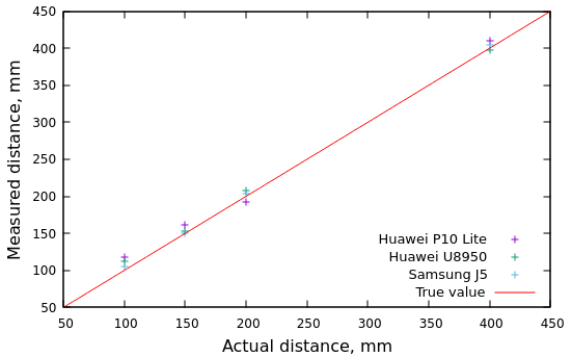

Figure 1: *Testing results*

Theoretically, all points should lie on the linear function (that is, at all points the measured value should equal the actual value). The method proved to be fairly accurate: the average error is 4,5 %.

V. Conclusion

A method of measuring screen to face distance was described in this paper. The algorithm doesn't require calibration and works continuously with updates every 40-70 ms (depending on device). The method proved to be fairly accurate. The algorithm has errors, as it uses the mean value of pupillary distance, which varieties in humans.

The following algorithm is implemented as an Android application. The full code is available on GitHub. [4]

Acknowledgements

I would like to thank Egor Rusakov for assisting me with testing the application.

References

[1] Neil Dodgson. "Variation and extrema of human interpupillary distance". In:

vol. 5291. Jan. 2004, pp. 36–46. ISBN: 0819451940. DOI: 10.1117/12.529999.

[2] Wayne Fulton. *Math of Field of View (FOV) for a Camera and Lens*. URL: https://www.scantips.com/lights/fieldofviewmath.html (visited on 08/14/2020).

[3] I. Konig, P. Beau, and K. David. "A new context: Screen to face distance". In: *2014 8th International Symposium on Medical Information and Communication Technology (ISMICT)*. 2014, pp. 1–5.

[4] Ivan Ludvig Tereshko. *Screen-to-face-distance*. July 2020. DOI: 10.6084/m9.figshare.12721595.v3. URL: https://figshare.com/articles/software/Screen-to-face-distance/12721595/3.