Modelling atmospheric turbulence and its influence on face recognition performance

H.G. Sikkema

***Abstract*—Atmospheric turbulence heavily affects image quality, and with it face recognition performance. As light travels through the complex air flows it is redirected, creating blurs and geometric distortions in the final image. A model proposed for atmospheric turbulence in free space optical communications applications is recreated for face recognition applications. It models atmospheric turbulence by recreating the complex airflows. They are recreated with bubbles with varying refractive indices. In two separate implementations, the edges of these bubbles are shown to cause artefacts, that distort the image differently from atmospheric turbulence. The effects of atmospheric turbulence on the performance of face recognition have also been tested, using a model based on Zernike coefficients. It showed that at mid to long ranges atmospheric turbulence can cause a significant decrease in facial recognition performance. Depending on the atmosphere, the face recognition can be either less accurate, or completely fail to detect any faces.**

I. Introduction

In the last decade, the accuracy of facial recognition methods has greatly improved. Some methods have even reached 100% accuracy [1], however that was achieved on a data set that varied only pose and illumination. The images in the data set were taken at close range to the subject, removing an important possible cause of distortions. When images are taken at larger distances, atmospheric turbulence will start to distort the image.

Atmospheric turbulence is used to describe the turbulent flow of air in the atmosphere. Similar to water, air can have laminar flow or turbulent flow. With laminar flow, all the air will flow in the same direction without mixing, similar to water flowing through a hose. With turbulent flow, the air will flow chaotically, similar to water flowing in a river. Instead of the air flowing in one direction, many smaller air flows will be flowing in many different directions with varying velocities. Each small airflow has its own humidity and temperature, which causes the light to change velocity slightly every time it transitions from one airflow to the next. The light will refract due to its change in velocity, as is described in Snell's law. An example of this transition can be seen in Figure 1. The strength of the refraction depends on the variations between air flows, in hotter more humid weather, these variations are much larger than in cold dry weather.

When an image is taken through atmospheric turbulence, the image will become distorted. These distortions can be severe enough that, facial recognition methods can no longer detect and recognise the subjects at all, as shown by Leonard *et al.* [3]. In the same paper, it is also shown that distorted images can be partially restored, which improves the performance of face recognition.

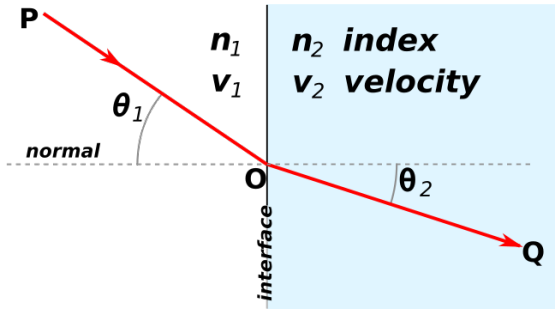

Fig. 1: The refraction of light at the transition from one medium to another, with $n_1 < n_2$ [2].

Due to the chaotic nature of atmospheric turbulence, neural networks are seen as a promising method to restore distorted images. In order to create these networks large training sets of atmospheric turbulence distorted images are required, but these are not readily available. Multiple methods to model atmospheric turbulence using undistorted data sets have already been proposed.

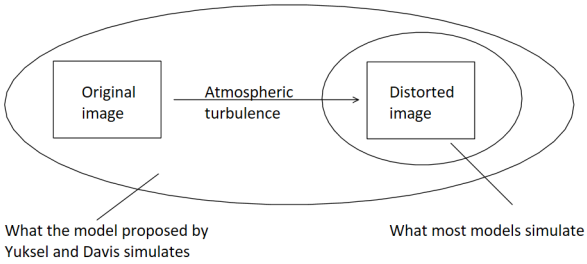

Fig. 2: A comparison between the approach of most models and the model proposed by Yuksel and Davis.

These methods use mathematical approaches to recreate the geometric distortions and blur created by atmospheric turbulence. In 2006 Yuksel and Davis [4] proposed a method for modelling atmospheric turbulence in Free-Space Optical (FSO) communication that is more closely related to the cause of atmospheric turbulence. The model tries to recreate the random air flows with varying refractive indices. It places bubbles with varying sizes and varying refractive indices in a 3D space between the emitter and the receiver, as can be seen in Figure 3 . The model is intended to model atmospheric turbulence for a single beam of light, but could also form the basis of a more accurate model for turbulence distorted images. This is a unique approach, since it recreates the cause of atmospheric turbulence distortion, instead of just recreating the distortion itself, as can be seen in Figure 2.

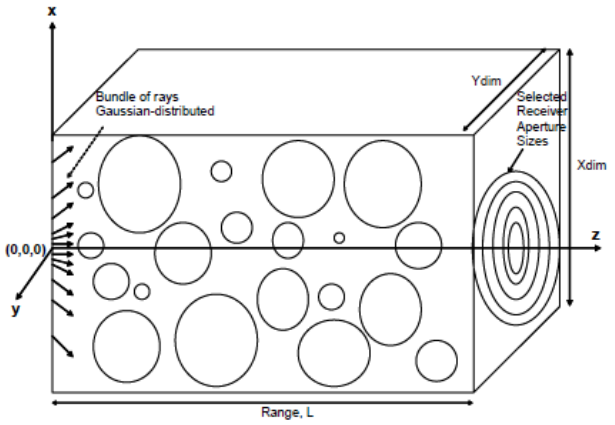

Fig. 3: Atmospheric turbulence model as proposed by Yuksel and Davis [4].

To test whether such a model is possible, two implementations will be tested. One is created in Java and it works similar to a ray tracer. Except that it traces the beams of light from the camera to its origin. This allows it to only look at those beams that can be seen by the camera. The second implementation is created in three.js, a JavaScript library that can create 3D computer graphics. In a 3D scene, the bubbles will be placed in between the original image and the camera.

The first aim of this paper is to study the feasibility of the model proposed by Yuksel and Davis, by recreating it with two different implementations and compare the results to the results of existing models. The second aim of this paper is to analyse the effect of variations of atmospheric turbulence on the performance of face recognition.

II. Related Work

1) Modelling atmospheric turbulence: Various approaches to modelling atmospheric turbulence distortions in images have been already been proposed, as shown in Figure 2, these focus on modelling the resulting distortion of the images. Deledalle *et al.* [5] used the Fried kernel and added white Gaussian noise on top of that, in order to model atmospheric turbulence. Chak *et al.* [6] recreates atmospheric turbulence by geometrically distorting a set area around a randomly selected pixel, this process is repeated for numerous pixels after which the entire image is blurred. Yasarla *et al.* [7] uses Gaussian blur kernels. Chimitt and Chan [8] model atmospheric turbulence Zernike coefficients, this model is explained in more detail in Section III-B.

2) The effect of atmospheric turbulence: Evaluating the effects of atmospheric turbulence on the performance of facial recognition has also been done before. Yasarla *et al.* [7] and Deledalle *et al.* [5] both use the distorted images in their comparisons, were they discuss the results of their turbulence mitigation methods. Espinola *et al.* [9] use real atmospheric turbulence distorted images, taken at a distance of $300m$ to discuss its effects on face recognition performance. Leonard *et al.* [3] show the effects of atmospheric turbulence at ranges up to $400m$, they also show the improvement in face recognition performance by partially restoring the distorted images.

III. Methods

A. A geometrical optics approach

The method proposed by Yuksel and Davis was created for FSO communication applications, where signals are sent using EM waves from an emitter to a receiver. The main focus of this study was to estimate the distance from where the signal arrives at the centre of the receiver. Using these estimates an appropriate size for the receiving aperture can be chosen in order to receive the required amount of power. To model atmospheric turbulence distortion in an image using a similar approach some changes need to be made to the model.

The model places bubbles with varying refractive indices in a 3D space. The image is placed on one side and the camera on the other. As the light passes through the bubbles, it is distorted. Similar to how light is distorted after passing through the air flows in atmospheric turbulence. In order to recreate the chaotic nature of atmospheric turbulence, the model uses multiple random distributions. The index of refraction is determined using a normal distribution with a mean of 1.0001 and a variance of 10^{-5} and the radius of the bubbles is determined using a uniform distribution ranging from $1mm$ to $1m$.

In the original model [4], 14000 bubbles were placed in the 3D space, modelling so many bubbles is very demanding on a computer. In the original model, one run would be enough to generate the required data for certain parameters. To generate accurate data on image distortion caused by certain parameters, hundreds of images would need to be distorted. So the implementations of the bubbles need to be created with efficiency in mind. Since the only interesting bubbles are those that influence the light coming from the images, these are the only bubbles that will be placed.

Two different implementations of the model were created. The first one functions similar to a raytracer, but it only calculates the rays of light that eventually reach the camera. The second implementation creates the bubbles as 3D computer graphics, while this allows for more complex light refraction and reflection, it is also more demanding on the hardware of a computer. So this second implementation can render far fewer bubbles than the first.

1) Java implementation: The first implementation was created in Java, the pipeline for this implementation was created by Zeinstra. As mentioned before it functions similar to a ray tracer. It starts off by determining the size of the distorted image it wants to create, for each pixel it traces the path of light back. If the light passed through a bubble it will determine how it passed through the bubble. Depending on the angle between the light beam and the surface of the bubble, it either refracts or reflects. If the light can be traced back to a pixel of the original image, the new pixel will have the same RGB value. If the light cannot be traced back to the original image, the pixel will be white.

In this model 'the camera' is as large as the image that it wants to create, this is set to be as large as the original image that is distorted. So without bubbles, the light would travel in a straight line from the original image to the camera. With the bubbles, the light can be redirected slightly to the outside and still be directed back and reach the camera.

The radius of the bubble is chosen from a uniform distribution ranging from $1mm$ to $1m$, so the average bubble has a radius of $0.5m$. The area where the centre of the bubbles can be placed extends two times the average radius in each direction from the size of the images themselves. In Figure 4 a schematic overview can be seen of the placement of the bubbles. This implementation takes approximately 30 seconds to simulate distortion with 100 bubbles in one image.

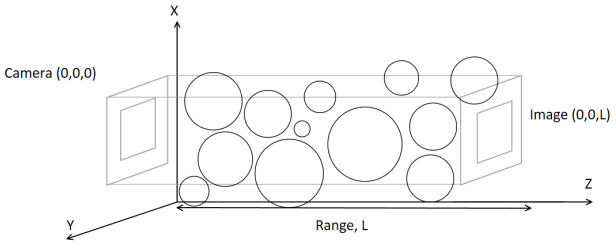

Fig. 4: A schematic overview of the placement of the bubbles in the Java implementation.

2) Three.js implementation: The second implementation uses three.js, three.js is a JavaScript library for creating 3D computer graphics [10]. It renders these 3D computer graphics with WebGL using the GPU. In order to render the bubbles with a certain index of refraction, the material property refractionRatio in three.js is used. The refractionRatio is equal to $\frac{n_1}{n_2}$ from Snell's law [11]. Parts of the implementation are based upon three example implementations of spheres with refraction [12] [13] [14].

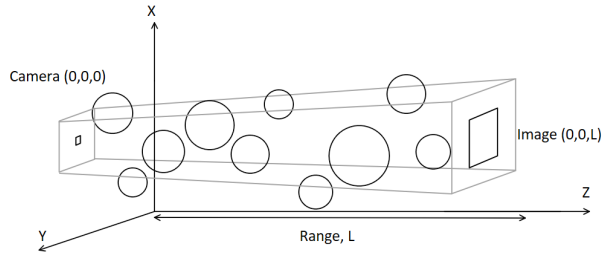

Fig. 5: A schematic overview of the area where the bubbles are placed.

The area where the bubbles are placed is slightly different from the Java implementation. In three.js the camera has no real size and can be seen as a single point in space. The area where the bubbles are placed can be seen in Figure 5. The area extends two times the average radius in each direction, similar to the Java implementation.

While three.js is fast in generating graphics, it is slow at generating random numbers. Java was used to generates all the random numbers to speed up the process. It was also used to automatically open the three.js file and capture the resulting distorted image. The three.js implementation generates an image distorted by 20 bubbles within 5 seconds, but an image distorted by 50 bubbles already takes more than one minute.

The scene in three.js where the bubbles, the camera and the image are placed, also has a background. In this implementation the background shows a simple landscape, this allows the face to 'blend' with the background, due to the distortions. To even better recreate this effect, the background should match the background in the image with the face, but this was not yet done in the three.js implementation. The bubbles in three.js have a predetermined resolution, described by the size in pixels. This size was set to 1024 pixels, this helped giving images a blur effect.

B. Zernike coefficient approach

Chimitt and Chan [8] propose a method to model atmospheric turbulence using Zernike coefficients. Most methods use convolutions to distort images. The images are transformed to the Fourier domain and multiplied by the Point Spread Function (PSF), a PSF is an impulse response of an imaging system. These convolutions are computationally expensive, the authors speed up the process by using Zernike coefficients. Zernike coefficients come from Zernike polynomials, which are used as mathematical descriptions of distortions in wavefronts [15]. Each order of Zernike coefficients is orthogonal to the others, which allows the distortion to be separated into different types of distortions. The authors calculate the second- and third-order which represent tilts, separate from the higher-order coefficients, which represent blurs. By splitting the Zernike coefficients up into tilts and blurs, the complexity of the calculations is reduced.

One of the main aims of the author's research was to create an open-source simulator for atmospheric turbulence. This simulator was used to determine the effects of variations of atmospheric turbulence on the performance of face recognition. In this paper no equations are shown from this model, as the equations are both complex and numerous, the paper published by Chimitt and Chan [8] explains the model in more detail.

The simulator allows for many variations on atmospheric turbulence, it has 9 different variables which all influence the simulator in their own way. 3 of these influence the way the simulator upsamples the images, these will not be varied. The focal length of the receiver will also not be changed. In Table I the variables, that are varied during the experiments, can be seen. The model gives their own default values for every variable, the distortion caused by these values are used as a baseline for the experiments. Some variations on these values have been tested, the ranges were in these values have been varied can be seen in Table I as well.

C. Face recognition

To test the effects of the atmospheric turbulence distortions on face recognition, a face recognition implementation was required. The purpose of this implementation is to be used as a quantification tool, the decrease in performance of the face recognition is used to quantify the distortion caused by atmospheric turbulence. The face_recognition library for Python [16] was used, as it is a simple yet effective implementation. The library will be used to compare every distorted image with all non-distorted images. The library first tries to detect a face in each image, each detected face gets its own face encoding of 128 dimensions [17]. To compare two face encodings u_1

TABLE I: The variables that will be varied, their default values and the ranges wherein the values will be varied.

Symbol	description	default value	Lower Limit	Upper Limit
D	The diameter of the receiving aperture	$0.2034m$	$0.1017m$	$0.4068m$
λ	The wavelength of the light	$525nm$	$400nm$	$750nm$
L	The distance between the image and the receiver	$7000m$	$1m$	$9000m$
C_n^2	The index of refraction structure parameter	$5 \cdot 10^{-16} m^{-2/3}$	$5 \cdot 10^{-17} m^{-2/3}$	$5 \cdot 10^{-15} m^{-2/3}$
K	The number of PSFs per row	16	8	32

and u_2, the following formula is used. $||u_1 - u_2|| = d$, this results in the face distance d, which is a measure of how much the two faces differ. If it is 0, the faces are the same, if it is 1 they are completely different [18].

All of these values will be used to create a Receiver Operating Characteristic (ROC) curve [19]. This is a curve that shows how well the distorted images can be matched with the originals. The Area Under the Curve (AUC) is used to describe how well the two sets of images can be matched. When the AUC is equal to one, it is a perfect match. If the AUC is 0.5 the matches are made at random.

A normal AUC does not look at undetected faces, it will simply compare all the detected faces to the original images. So it does not take failure to detect faces into account, as this is the worst outcome, it is important to take it into account in comparisons between sets of images. When a face is not detected it gets a face distance of 1, with the original face that was distorted in the image. The face distances with all other faces are set to 0. This is the worst possible result, which gives a better representation of the performance of face recognition. AUC2 will be used to refer to the AUC that takes failure to detect into account.

IV. Results

This section will be split into two parts, the first part will show the results from the two geometrical optic approaches for modelling atmospheric turbulence. The resemblance between distortion caused by atmospheric turbulence and distortion created by the model of Yuksel and Davis is evaluated two times. One for each implementation of their model. The second part will show the results from the approach using Zernike coefficients and the effects of variations of the variables on the performance of face recognition.

A. A geometrical optics approach

1) Java implementation: To evaluate the performance of the Java implementation of the model, an image will be distorted, the result is evaluated on how good it represents atmospheric turbulence. The first test was done with values that closely represent the original model as proposed by Yuksel and Davis [4]. The refractive indices come from a normal distribution with the mean of 1.0001 and a variance of 10^{-5}. The radii come from a uniform distribution ranging from $1mm$ to $1m$. The bubbles are placed in the area as described in Section III-A1. For the test, 100 bubbles were placed in between the image and the camera.

For the test image, a face was generated using the site thispersondoesnotexist.com [20], the site uses styleGAN to generate realistic faces [21]. The face is used with a black background to create the test image. The image can be seen in Figure 6. To compare the distortion with an atmospheric turbulence distortion, an image distorted by the model of Chimitt and Chan [8] is used, as this model was shown to be a good representation of atmospheric turbulence by the authors. The image distorted by this model can be seen in Figure 7.

Fig. 6: The test image with a person that does not exist.

Fig. 7: A distorted version of Figure 6, using the model created by Chimitt and Chan [8]. It has a $C_n^2 = 1 \cdot 10^{-15} m^{-2/3}$, all the other values are the default values.

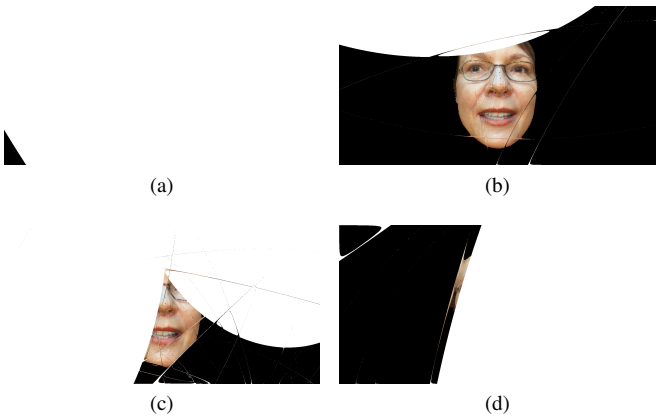

Fig. 8: Images distorted using the Java implementation.

In Figure 8 some of the resulting images can be seen. The Java implementation shows a white pixel if the light that reached that pixel did not originate from the original image, this occurred much more frequent than expected. Some images, like Figure 8a, were almost completely white. Others, like Figure 8d, were not completely white but still had the entire face missing from the image. Approximately 40% of all images were distorted in this way. The remaining 60% had two different problems. The first one can be seen in Figure 8b, the image is still largely undistorted. With 100 bubbles

the resulting image still looks very similar to the original. Assuming that with more bubbles, the image will become distorted, a second problem will still need to be solved. In Figure 8c this problem can be seen most clearly, but it is also visible in Figures 8b and 8d. The lines created by the edges of the bubbles can be clearly seen in the distorted images. In Figure 7, which is assumed to be distorted by real atmospheric turbulence, no such lines are visible.

2) Three.js implementation: To evaluate the performance of the three.js implementation a similar approach is used as with Java implementation. The values for the indices of refraction and the radii were again chosen using the same distributions as the original model proposed by Yuksel and Davis [4]. The bubbles were placed in the area shown in Figure 5 in Section III-A2. The image that is distorted can be seen in Figure 6 and the results are compared to Figure 7.

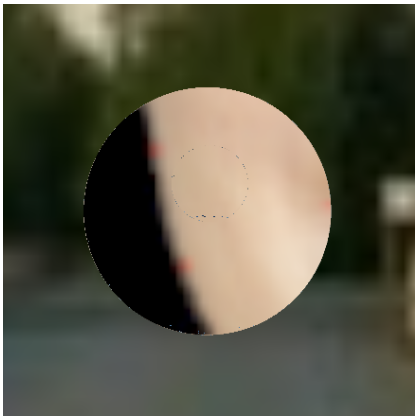

Fig. 9: An image distorted using the three.js implementation, with 20 bubbles with radii ranging from $1mm$ to $1m$.

In Figure 9 one example of the images distorted by the three.js implementation can be seen, this image was distorted by 20 bubbles. The refraction of the bubbles is stronger than that of the Java implementation. And the distortion does not look like atmospheric turbulence. To get a better understanding of what is happening with the distortion, multiple variations to the values used by Yuksel and Davis were tested.

In figure 10 some example images are shown from eight different sets of distorted images. The first four Figures 10a, 10b, 10c and 10d have varying radius ranges. A smaller radius range, resulted in more subtle distortions. Figures 10e, 10f, 10g and 10h, show images with radii in the range of $1mm$ to $50mm$ and varying numbers of bubbles. To more clearly show the distortions caused by the bubbles, the size in pixels was doubled, to 2048 pixels. Figure 10e is distorted by 50 bubbles, the most notable distortions are caused again caused by the edges of the bubbles.

To better understand what is happening at these edges, some different shapes have been tried as well. In Figures 11a and 11b these can be seen. While the lines are not visible in these images, the abrupt transitions are still there. This is what the model by Yuksel and Davis does differently from atmospheric turbulence. The focus of the model is recreating the air flows that form atmospheric turbulence, but these air flows do not abruptly transition from one airflow to the next. The two airflows differ in velocity, temperature and concentrations of various gasses and water. Two objects with different temperatures will form a temperature gradient between them, this also

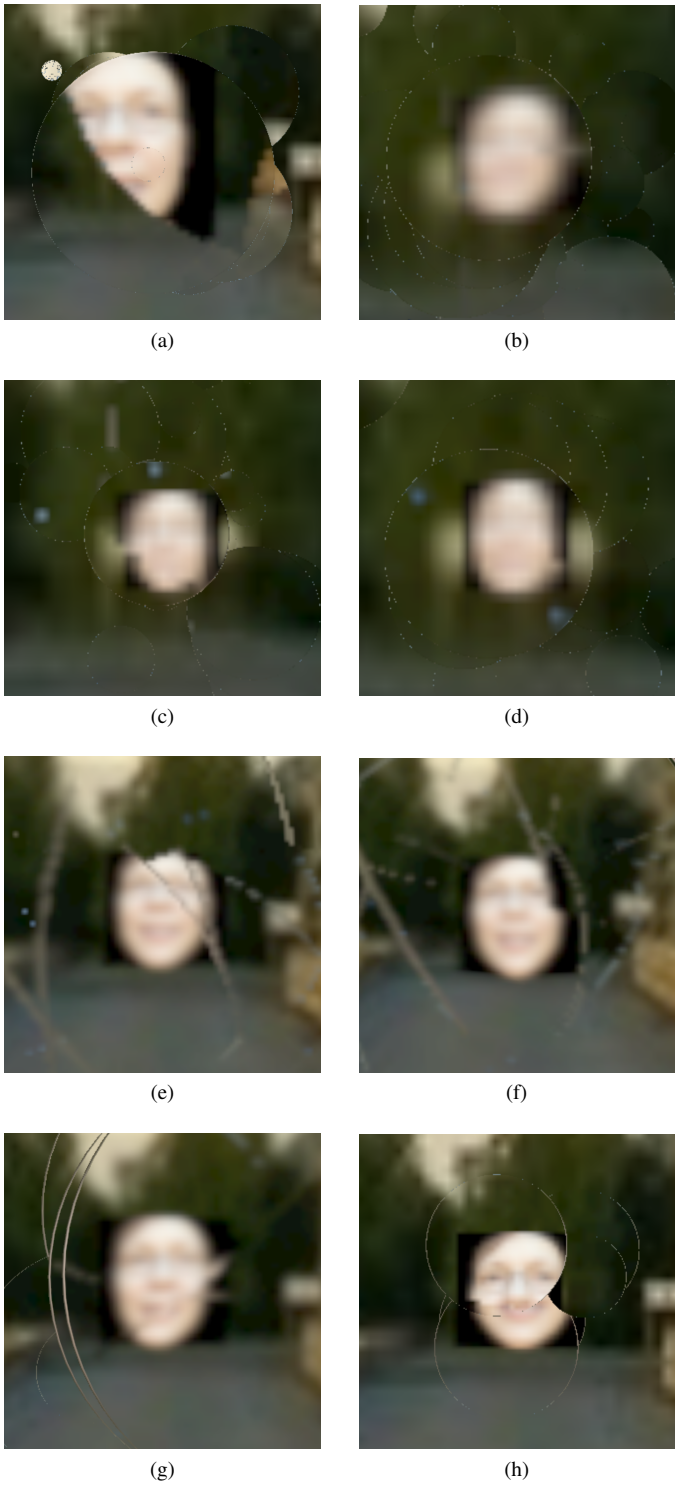

Fig. 10: Distorted versions of the figure 6. a, b, c and d are distorted by 20 bubbles with radii ranging from $1mm$ to $0.5m$, $0.1m$, $50mm$ and $25mm$ respectively. e,f,g and h are distorted by bubbles with radii ranging from $1mm$ to $50mm$, but with 50, 40, 30 and 10 bubbles respectively.

(a) (b)

Fig. 11: Images distorted by 20 cubes (a) or by 20 rectangles (b).

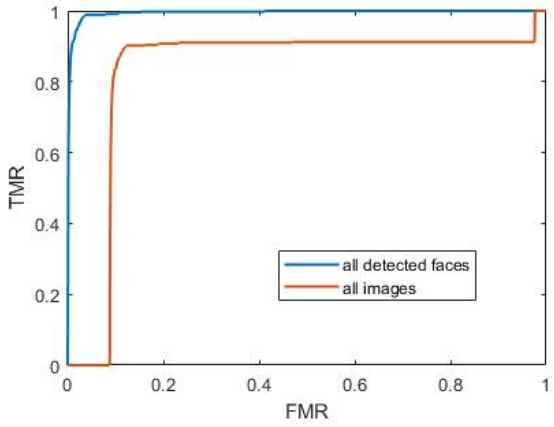

Fig. 12: The two ROC curves, one showing the performance over all detected faces and one showing the performance over all faces.

happens with velocities and concentrations. These gradients will cause a smooth transition from one airflow to the next, unlike the model.

B. Zernike coefficient approach

To measure the effects of various variables of atmospheric turbulence on the performance of face recognition, the AUC and AUC2 are used, as previously explained in Section III-B. The data set that is distorted, consists of 500 images generated using thispersondoesnotexist.com [20]. Using face recognition software each distorted face is compared to all undistorted faces and the results are shown in a ROC curve. Using the AUC and AUC2 of that ROC curve gives an indication of how strong the atmospheric turbulence distortion was. In the appendix examples of every set mentioned are shown, all examples are distorted versions of the same image to make comparisons easier. In Table III in the appendix, a complete overview of all the various distorted sets, the used variable values and the resulting AUC and AUC2 can be seen. In Figure 12 the two ROC curves can be seen, both are from the dataset created with the default distortion. The blue one shows the ROC curve for all detected faces, the orange one shows the ROC curve for all images. The AUC for the blue one is 0.995, which makes it appear as a reasonable result, but the face recognition software detected only 456 out of the 500 images. The orange ROC curve has an AUC2 of 0.829, which better reflects the performance of face recognition.

Using this AUC2 the effect of variations for each of the five variables can be quantified. One variable will be different from its default value per set of images. Each variable is changed twice, once to a higher value and once to a lower value. In Table II the variations that were tested and the resulting AUCs can be seen. Examples of the distorted images and a complete overview of all distorted sets can be found in the appendix.

1) Aperture size: From the results, it can be seen that a smaller aperture size improves the performance of face recognition. In the appendix, it can be seen that the images still experience a lot of tilt distortion but are far less blurred, when the aperture size is decreased.

2) Wavelength: The wavelength affects the performance just as Cauchy's equation predicts, according to his equation longer wavelengths experience a slightly lower refractive index in the same material as shorter wavelengths do. So the data set with a wavelength of $400nm$ experiences more refraction, which results in more distortion.

3) Distance: The distance between the image and the camera behaves as expected, a larger distance creates a larger distortion.

4) C_n^2: The index of refraction structure parameter affects the distortion significantly, with $C_n^2 = 5 \cdot 10 - ^{-15}\ m^{-2/3}$ in only 137 of the 500 images a face was detected. C_n^2 represents temperature and humidity, but also more subtle variations like the composition of gasses in air. With $C_n^2 = \cdot 10 - ^{-17}\ m^{-2/3}$ almost no distortion was present in the images, this shows that in the right environment atmospheric turbulence is barely noticeable.

5) Number of PSFs: The number of PSFs barely changed the AUC and AUC2 at all and no noticeable difference can be seen from the images. The number of PSFs did have a large impact on the computational complexity of the model, as it took more than four times as long to distort the images.

TABLE II: The AUCs of sets of images with each one changed value, compared to the default values.

Variable	Value	AUC	AUC2
all	$default$	0.995	0.82945
D	$0.1017m$	0.998	0.955
D	$0.4068m$	0.975	0.519
λ	$400nm$	0.989	0.713
λ	$750nm$	0.999	0.933
L	$5000m$	0.999	0.872
L	$9000m$	0.987	0.759
C_n^2	$5 \cdot 10 - ^{-17}\ m^{-2/3}$	1.000	0.96842
C_n^2	$5 \cdot 10 - ^{-15}\ m^{-2/3}$	0.772	0.05662
K	8	0.995	0.815
K	32	0.995	0.822

In face recognition applications distances of $7km$ are not very realistic, cameras used for these applications do not have enough resolution for these distances, also the earth's curvature will hide most of the person at those distances, assuming there is an unobstructed sightline of $7km$ to start with. Distances

around $500m$ are more realistic for mid- to long-range face recognition applications. In Figure 13 the resulting AUC2 by varying the distance can be seen, at distances below $1km$ there seems to be almost no change in AUC2.

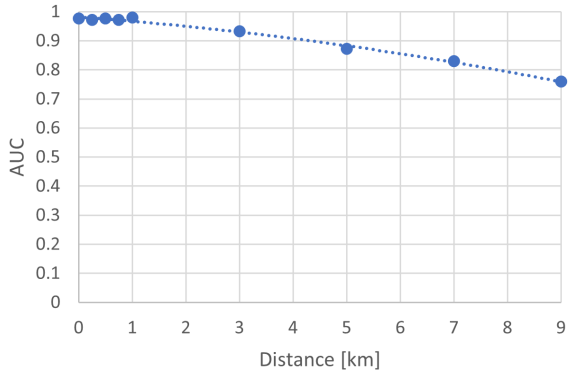

Fig. 13: The AUC for distances ranging from $1m$ to $9km$.

In Figure 14 the variation of the AUC2 with distance can be seen for three different values of C_n^2. With a higher C_n^2 the AUC2 drops significantly faster over distance. With $Cn^2 = 5 \cdot 10 -^{-16} m^{-2/3}$ there is almost no change in the AUC2. Face recognition fails to recognise approximately 1% of all faces on each of the distances. When the C_n^2 is increased to $1 \cdot 10 -^{-15} m^{-2/3}$ the AUC2 starts to decrease faster with distance. With $C_n^2 = 5 \cdot 10 -^{-15} m^{-2/3}$ the AUC2 decreases even faster. At $500m$ the face recognition fails to detect 31 faces, at $1km$ it already fails to detect 65 faces. The performance of recognising the detected faces also decreases. Depending on the environment where the camera will be placed, atmospheric turbulence can become a real issue for face recognition. Measurements done in Hawaii [22] showed that C_n^2 can reach up to $10^{-15}m^{-2/3}$. Whereas measurements done in The Netherlands [23] showed values of C_n^2 up to $10^{-13}m^{-2/3}$. With such high values of C_n^2, most faces will not be detected, and of those that are detected, few will be recognised. A study by Espinola *et al.* [9] on real atmospheric turbulence distorted faces at $300m$ distance, shows a similar decrease in performance of face recognition. The study shows that from $C_n^2 = 7.5 \cdot 10 -^{-14} m^{-2/3}$ the performance starts to decrease fast at $300m$, similar to how the performance decreases fast with $C_n^2 = 5 \cdot 10 -^{-15} m^{-2/3}$ at $500m$. It seems that at a set distance the performance fast from a certain value of C_n^2. With enough data, it should be possible to create a graph showing at what distance and what C_n^2 face recognition becomes unreliable.

V. Conclusion

The geometrical optics approach to modelling atmospheric turbulence in free-space optical communication applications has been recreated for face recognition applications. In two separate implementations, it has been shown that a bubble is too simple an approximation of the air flows. The abrupt transitions cause artefacts that distort the images differently

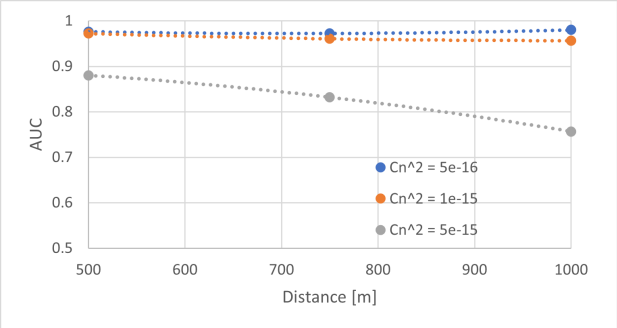

Fig. 14: The variation of AUC2 over distance at three different values of C_n^2

from atmospheric turbulence. The effects of variations of atmospheric turbulence on face recognition performance have also been studied. Atmospheric turbulence can heavily decrease the performance of face recognition on mid to long ranges. The software fails to detect a significant number of faces and fails to recognise a significant number of the detected faces.

References

[1] J. K. Appati, H. Abu, E. Owusu, and K. Darkwah, "Analysis and implementation of optimization techniques for facial recognition," *Applied Computational Intelligence and Soft Computing*, vol. 2021, 2021.

[2] "File:Snells law.svg - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/File:Snells_law.svg

[3] K. Leonard, J. Howe, and D. Oxford, "Simulation of atmospheric turbulence effects and mitigation algorithms on stand-off automatic facial recognition," vol. 8546, 10 2012, p. 85460O.

[4] H. Yuksel and C. C. Davis, "A geometrical optics approach for modeling aperture averaging in free space optical communication applications," in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, ser. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, S. M. Hammel and A. Kohnle, Eds., vol. 6303, Sep. 2006, p. 630302.

[5] C.-A. Deledalle and J. Gilles, "BATUD: Blind Atmospheric TUrbulence Deconvolution," Oct. 2019, working paper or preprint. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02343041

[6] W. H. Chak, C. P. Lau, and L. M. Lui, "Subsampled turbulence removal network," *CoRR*, vol. abs/1807.04418, 2018. [Online]. Available: http://arxiv.org/abs/1807.04418

[7] R. Yasarla and V. M. Patel, "Learning to restore a single face image degraded by atmospheric turbulence using cnns," 2020.

[8] N. Chimitt and S. H. Chan, "Simulating anisoplanatic turbulence by sampling intermodal and spatially correlated zernike coefficients," *Optical Engineering*, vol. 59, p. 1, 8 2020. [Online]. Available: https://doi.org/10.1117/1.OE.59.8.083101

[9] R. Espinola, K. Leonard, K. Byrd, and G. Potvin, "Atmospheric turbulence and sensor system effects on biometric algorithm performance," vol. 9452, 04 2015.

[10] "GitHub - mrdoob/three.js: JavaScript 3D Library." [Online]. Available: https://github.com/mrdoob/three.js

[11] "Meshstandard material refractionratio – three.js docs." [Online]. Available: https://threejs.org/docs/api/en/materials/MeshStandardMaterial.refractionRatio

[12] "Refraction (Three.js)." [Online]. Available: https://stemkoski.github.io/Three.js/Refraction.html

[13] "three.js webgl - materials - cube refraction [Lucy]." [Online]. Available: https://threejs.org/examples/webgl_materials_cubemap_refraction.html

[14] "three.js webgl - materials - environment maps." [Online]. Available: https://deml.io/three/examples/webgl_materials_envmaps.html

[15] V. Lakshminarayanan and A. Fleck, "Zernike polynomials: A guide," pp. 545–561, apr 2011. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/09500340.2011.554896

[16] "face-recognition · PyPI." [Online]. Available: https://pypi.org/project/face-recognition/

[17] "face_recognition package - Face Recognition 1.4.0 documentation." [Online]. Available: https://face-recognition.readthedocs.io/en/latest/face_recognition.html

[18] "face_recognition.api - Face Recognition 1.4.0 documentation." [Online]. Available: https://face-recognition.readthedocs.io/en/latest/_modules/face_recognition/api.htmlface_

[19] C. G. Zeinstra, "Forensic Face Recognition : From characteristic descriptors to strength of evidence," Ph.D. dissertation, University of Twente, Enschede, The Netherlands, nov 2017. [Online]. Available: http://purl.org/utwente/doi/10.3990/1.9789036543750

[20] "This Person Does Not Exist." [Online]. Available: https://thispersondoesnotexist.com/

[21] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2019-June. IEEE Computer Society, jun 2019, pp. 4396–4405. [Online]. Available: https://github.com/NVlabs/stylegan

[22] J. P. McHugh, G. Y. Jumper, and M. Chun, "Balloon Thermosonde Measurements over Mauna Kea and Comparison with Seeing Measurements," *Publications of the Astronomical Society of the Pacific*, vol. 120, no. 874, pp. 1318–1324, dec 2008.

[23] A. Van De Boer, A. F. Moene, A. Graf, C. Simmer, and A. A. M. Holtslag, "Estimation of the refractive index structure parameter from single-level daytime routine weather data," 2014. [Online]. Available: http://dx.doi.org/10.1364/AO.53.005944

APPENDIX

(a) (b)

Fig. 15: a shows the gray-scale version of the original image, b shows the image distorted with the default values.

(a) (b)

Fig. 16: a shows the distorted image with $D = 0.1017m$, b shows the distorted image with $D = 0.4068m$.

(a) (b)

Fig. 17: a shows the distorted image with $\lambda = 400nm$, b shows the distorted image with $\lambda = 750nm$.

(a) (b)

(c)

Fig. 18: a distorted at a distance of $1m$, b distorted at $250m$ and c distorted at $3000m$.

(a)

(b)

(c)

(d)

(e)

(f)

(g)

(h)

(i)

Fig. 19: Examples from the sets shown in Figure 14. a,b and c show at $500m$ with $C_n^2 = 5 \cdot 10^{-16} m^{-2/3}$, $C_n^2 = 1 \cdot 10^{-15} m^{-2/3}$ and $C_n^2 = 5 \cdot 10^{-15} m^{-2/3}$ respectively. d,e and f show the same at $750m$ and g,h and i show it at $1000m$.

(a)

(b)

Fig. 20: a shows the distorted image with $L = 5000m$, b shows the distorted image with $L = 9000m$.

(a)

(b)

Fig. 21: a shows the distorted image with $Cn^2 = 5e - 17m^{-2/3}$, b shows the distorted image with $Cn^2 = 5e - 15m^{-2/3}$.

(a)

(b)

Fig. 22: a shows the distorted image with $K = 8$, b shows the distorted image with $K = 32$.

$D[m]$	$\lambda[nm]$	$L[m]$	${C_n}^2[m^{-2/3}]$	K	AUC	$AUC2$	# Figure
0.2034	525	7000	$5 \cdot 10^{-16}$	16	0.995	0.829	15b
0.1017	525	7000	$5 \cdot 10^{-16}$	16	0.998	0.955	16a
0.4068	525	7000	$5 \cdot 10^{-16}$	16	0.975	0.519	16b
0.2034	**400**	7000	$5 \cdot 10^{-16}$	16	0.989	0.713	17a
0.2034	**750**	7000	$5 \cdot 10^{-16}$	16	0.999	0.933	17b
0.2034	525	**1**	$5 \cdot 10^{-16}$	16	1.000	0.976	18a
0.2034	525	**250**	$5 \cdot 10^{-16}$	16	1.000	0.972	18b
0.2034	525	**500**	$5 \cdot 10^{-16}$	16	1.000	0.976	19a
0.2034	525	**500**	$\mathbf{1 \cdot 10^{-15}}$	16	1.000	0.972	19b
0.2034	525	**500**	$\mathbf{5 \cdot 10^{-15}}$	16	0.999	0.881	19c
0.2034	525	**750**	$5 \cdot 10^{-16}$	16	1.000	0.972	19d
0.2034	525	**750**	$\mathbf{1 \cdot 10^{-15}}$	16	1.000	0.961	19e
0.2034	525	**750**	$\mathbf{5 \cdot 10^{-15}}$	16	0.998	0.832	19f
0.2034	525	**1000**	$5 \cdot 10^{-16}$	16	1.000	0.980	19g
0.2034	525	**1000**	$\mathbf{1 \cdot 10^{-15}}$	16	1.000	0.957	19h
0.2034	525	**1000**	$\mathbf{5 \cdot 10^{-15}}$	16	0.997	0.757	19i
0.2034	525	**3000**	$5 \cdot 10^{-16}$	16	1.000	0.934	18c
0.2034	525	**5000**	$5 \cdot 10^{-16}$	16	0.999	0.872	20a
0.2034	525	**9000**	$5 \cdot 10^{-16}$	16	0.987	0.759	20b
0.2034	525	7000	$\mathbf{5 \cdot 10^{-17}}$	16	1.000	0.968	21a
0.2034	525	7000	$\mathbf{5 \cdot 10^{-15}}$	16	0.772	0.057	16a
0.2034	525	7000	$5 \cdot 10^{-16}$	**8**	0.995	0.815	22a
0.2034	525	7000	$5 \cdot 10^{-16}$	**32**	0.995	0.822	22b

TABLE III: Complete overview of all sets of distorted images, the variables used to distort them and the resulting AUC and AUC2. The last column references the figure in the appendix that displays an example image from the respective set.