AI-Based Eye Tracking for Human-Computer Interaction

David Sancho Crespo[1,2], Xinrui Yu[1] and Jafar Saniie[1]

[1] *Embedded Computing and Signal Processing (ECASP) Research Laboratory (http://ecasp.ece.iit.edu/) Department of Electrical and Computer Engineering Illinois Institute of Technology, Chicago IL, U.S.A*

[2] *E.T.S.I Industriales, Universidad Politécnica de Madrid, Madrid, Spain*

***Abstract*— Human-computer interaction (HCI) is a field that has grown enormously in importance in recent times, with the spread of the personal computer. However, HCI devices have had little evolution since their first designs. Prolonged use of the mouse can cause injury or permanent damage to the hands and wrists. So, this paper proposes an alternative to the conventional mouse that does not require the use of hands and works only with a webcam. A program has been developed that performs all the functions of the mouse based on the image of the user's face captured by the camera. The cursor moves to the position of the screen where the user is looking at in real time, while the rest of the functionalities such as clicks and scrolling are controlled with facial commands, such as winking or tilting the head. To implement these functions, several tools based on artificial intelligence have been used, such as a facial landmark detector, a gaze vector estimator, and a fully connected multilayer neural network.**

Keywords—human-computer interaction, eye tracker, gaze vector, artificial intelligence, MediaPipe, computer mouse

Introduction

Human-Computer Interaction (HCI) is the field of study that focuses on optimizing the way users interact with computers. To this end, interactive interfaces are designed for computers that meet the diverse needs of users. The most important types of HCI devices are the monitor, keyboard, touchpad, and mouse. The mouse is a handheld device that detects a two-dimensional movement. This movement results in the motion of a cursor or pointer on a screen. Generally, the mouse also includes three buttons, one of which is a scroll wheel.

In recent decades the use of the computer has spread hugely, with the arrival of the personal computer. In the United States, 74% of adults owned a computer in 2019 [1]. The mouse is one of the most important peripherals to control a computer, so they are almost inseparable.

This massive use of the computer mouse has brought to light some serious health problems that can be generated by its use over time. Repetitive and prolonged postures caused by the mouse can lead to musculoskeletal injuries to the arm. Mouse use for more than 20 hours a week can cause hand and wrist injuries while use extended to more than 30 hours a week can lead to carpal tunnel syndrome [2].

This paper analyzes and proposes an alternative that parts ways with conventional HCI methods, since it aims to use a system based on eye tracking technology that does not require the use of hands to perform mouse functions. The system follows the movement of the eye to identify where on the screen the user is looking at, and moves the pointer to that position. In addition, the system uses facial gestures, such as winking, opening the mouth, or tilting the head to perform the rest of the mouse functions such as scrolling or clicking.

This alternative has advantages for users beyond improving comfort or ergonomics for the arm. The ability to control traditional mouse functions hands-free improves accessibility for people with disabilities, injuries, or illnesses that affect their motor abilities in the arms in general. For example, people with arthritis.

Some examples of companies that market eye-trackers are Tobii or Gazepoint [3]. But these products have several disadvantages. The first problem is the cost of the software and the specialized hardware. Second, these devices only perform the function of eye tracking. To implement an HCI system, additional software is needed to act as an intermediary between the eye tracker and the computer, to perform all the functionalities of the conventional mouse.

This paper proposes a system that solves these problems. For this, a single computer program is developed that performs all the functions of a conventional mouse by means of facial instructions and an eye tracker. The user's face is captured only by a normal webcam. In this way, the system aims to be cheaper and more flexible than commercial applications using specialized hardware. Using a webcam also increases the accessibility of the program since most computers today include their own camera.

However, webcam-based eye trackers are the least accurate. Therefore, different image processing methods will be analyzed to evaluate which one provides greater accuracy. In this analysis, traditional methods and more innovative methods based on artificial intelligence are included.

I. Implementation

A. Analysis of Traditional Methods

To implement an eye tracker, it is necessary to detect the center of the user's eye. A common practice in object detection is to move from large objects to small ones. This saves a lot of computational power and makes the process much faster. In addition, it helps to avoid possible false detections. For this reason, the implementation of the eye tracker is divided into several steps: detecting the face in the full frame of the image, then detecting the eyes inside the sub-image of the face, and finally detecting the iris and center of the eye inside the sub-image of the eye region.

Several techniques were tested for these tasks. Two methods have been studied for face detection: template matching [4] and Haar Cascade classifiers [5].

The previous methods can also be used for eye detection. However, in this paper, another method has been studied, which uses a detector of facial key points to distinguish the different elements of a face. This method is explained in [6]. The Dlib library of Python includes an algorithm that can detect 68 key points, including the 6 key points in the outline of the eyes. The Dlib library implements the method developed by Kazemi and Sullivan in 2014 in [7].

Finally, two image processing methods were tested to find the center of the iris or pupil, blob detection and contouring. With image processing tools found in OpenCV, the region of the eye can be turned into a black blob [8] on a white background. Then, the blob can be approximated to a circle to find the center. An example can be seen in Fig. 1.

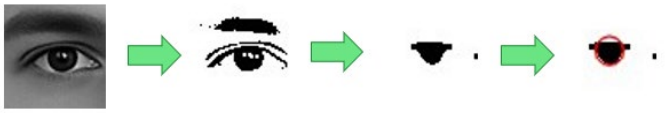

Fig. 1. Blob detection process to find the center of the eye

For contouring, first, the eye key points obtained with Dlib can be used to isolate the area between the eyelids. A binary threshold can then be applied to this area to differentiate the iris from the white part of the eye. Once the iris stands out from the rest of the eye the outline of the iris can be found with the OpenCV function findContours(). The center of the contour is then calculated using the moments() function. This process is explained in more detail in [9].

B. Advanced Tools Based on Artificial Intelligence

The previous methods and algorithms are based on image processing techniques or machine learning models that were developed years ago. After studying and testing some of them, it can be said that the facial and eye detection methods work correctly. But, iris detection methods present many problems. These methods require the user to stay still and be close to the camera. In addition, you have to keep your eyes wide open in an unnatural position. On the other hand, these methods are very dependent on lighting. Finally, the precision when positioning the center of the iris was insufficient.

Getting the pupil positioned accurately is essential to create an eye tracker so the previous methods were discarded. Then, two more advanced and recent methods that are based on more complex machine learning models were studied to track the eyes.

a) MediaPipe

MediaPipe is an open-source platform developed by Google that provides a wide range of tools and components for real-time video processing tasks using AI, that are designed to consume as little computational power as possible. Specifically, this paper uses one of MediaPipe's AI solutions, which is a facial landmark detector.

The detector uses machine learning models that can work with both individual images and videos. The detector generates 478 three-dimensional key points of the face to infer facial surfaces in real time [10]. To perform this task, MediaPipe combines the use of two models, a face detector and a facial mesh model.

The face detector is a variant of the BlazeFace model. BlazeFace is a lightweight and accurate detector optimized for GPU inference [11]. This model is so lightweight because it uses a mobile feature extraction network, similar to MobileNet [12]. BlazeFace is optimized for images of people from a smartphone camera or webcam, where the subject's face is at a short distance. So, this solution works especially well in the environment of this paper, for the application of HCI.

The facial mesh model analyzes the result of the BlazeFace model and performs a complete mapping of the face to estimate the 478 facial key points. The specifications of this model can be found in [13].

Therefore, the result of using this MediaPipe detector is a series of facial key points, that include points for both the iris perimeter and the pupil center. Then, this detector performs the three steps of an eye tracker that were discussed in the section on traditional methods: facial detection, eye detection, and pupil detection. In this way, the center of the eyes can be detected in a more precise way than with the traditional methods.

b) Eye tracker prototype

Using MediaPipe's facial point detector, a prototype eye tracker was made to move the computer pointer. This prototype relied solely on the movement of the user's pupils to move the cursor. The position of the pupils could be known thanks to MediaPipe's detector.

For this program to work, a previous calibration was carried out, which consisted of calculating the coordinates of the user's pupils when looking at 9 fixed points on the computer screen. These points are the center, the 4 corners, and the midpoints of each edge of the screen. Knowing the coordinates of the eyes looking at each of these points, the position of the pointer at the rest of the points on the screen was calculated based on a linear interpolation between the coordinates of the eyes looking at the reference points and the coordinates in real time.

c) L2CS-Net

The prototype had a serious lack of precision. The movement of the cursor was only based on the change of position of the pupils, but the distance that the pupil travels when looking from one side of the screen to the other is very small. So, it was hard to get the cursor to point to an exact place. Therefore, to improve

precision, We looked for more reliable options for representing eye movement than simple pupil position. The chosen method is the gaze vector.

The gaze vector is a three-dimensional vector that indicates the orientation or line of sight of the person's eyes with respect to their head or the surrounding environment. Gaze tracking is a 3D problem, so calculating it is very difficult with a single monocular camera, as the camera only provides a 2D image. It can be done using a series of algebraic transformations as explained in [14]. But this method is extremely complex and it requires knowing a lot of data from the camera's environment or making estimates that decrease accuracy. Therefore, a second approach to calculating the gaze vector is machine learning.

Within the broad field of machine learning, convolutional neural networks have made progress in predicting gaze direction. In particular, for this paper, a recent model explained in an article from last year called L2CS-Net [15] has been studied. The researchers and authors of this article developed a state-of-the-art model to perform the estimation of the gaze vector and published it as open source to be used by other researchers.

L2CS-Net takes facial images as input and feeds them to a CNN with ResNet-50 base architecture to extract spatial features. It then uses two independent fully connected layers to predict the two exit angles of the gaze vector independently. These outputs are the yaw and pitch angles of the spherical coordinates.

Therefore, in this paper, the L2CS-Net model is used to process each frame captured by the camera and it estimates two angles, pitch and yaw, which indicate the direction of the user's gaze. These angles can be used to track where the user looks instead of the coordinates of the pupils.

d) Neural Network

The previous section explained the operation of the L2CS-Net model, which can be used to achieve the user's gaze vector for each frame captured by the webcam. This vector can be used to track the user's vision, but to implement an HCI program that performs the same functions as the mouse, it is necessary to translate the angles of the gaze vector to the position of the screen to which the user is looking at in each moment.

In the prototype, this task is achieved with a previous calibration step that finds the coordinates of the pupils when the user looks at 9 predetermined points on the screen. The rest of the screen positions are calculated by interpolating the known points. This causes the problem that the user has to remain stationary in the same position where the calibration was carried out.

However, for the final version of the program, this paper proposes to perform this task using a neural network, a multilayer perceptron. The network's inputs are numerical features obtained from each frame of the webcam such as pitch angle, yaw angle, coordinates of some facial points, and depth distance between the camera and user. While the output will be the x-coordinate and y-coordinate of the pointer on the screen on the computer.

This neural network performs a very specific task, so the database to train the model had to be created from scratch. Each entry of the database has 11 values:

- Yaw and pitch angle of the gaze vector estimated with L2CS-Net (2 values).
- X and y coordinates of the following facial points: nose, right ear, and left ear. Estimated with MediaPipe (6 values).
- Depth distance between camera and user, estimated with MediaPipe (1 value).
- X and y coordinates of the mouse pointer (2 values).

Each entry of the database was taken with a program that captures a frame of the webcam's footage when the user is looking directly at the pointer. This program then estimates the mentioned input features using the AI tools. These input features correspond to the position of the screen indicated by the pointer. In total, about 9000 samples have been taken during the writing of the paper to train the neural network.

This type of data is called labeled since, for each group of network input data or features, the corresponding outputs are also saved. The labeled data is used to train a type of machine learning algorithm that is called supervised. Supervised algorithms are trained by minimizing an error function that compares the output of the network in its present training state with the labels of the known data. To minimize error, different optimization methods based on the gradient of the error function can be used, for example, the stochastic gradient descent method. But to train the network of this paper we have used the Adam optimizer.

For the training process of the neural network, of the 9000 samples collected, 8000 have been destined for training data, of which 10% have been used as validation data. The remaining 1000 samples have been destined to test data.

To carry out the entire process of design and training of the neural network, Google Colab has been used, an online platform for Python programming. TensorFlow is the framework that has been chosen to develop the network model.

e) Depth distance measurement

It was mentioned that the depth distance was used as an input feature for the neural network. It is the distance between the camera and the head of the user. This is an important feature because it provides information about the position of the user's head in relation to the camera and the screen, but this distance is not easy to estimate with a single monocular camera, which is the only available resource of the paper. However, thanks to the good accuracy of MediaPipe's face key points detector, distance estimation can be made as explained in [16].

The depth estimate is based on the fact that the horizontal diameter of the iris of the vast majority of the world's population has a constant measurement of 11.7±0.5 mm. This is coupled with the fact that MediaPipe's facial point detector includes dots that mark the ends of the horizontal diameter of the iris. Then, using the pinhole camera model approximation as can be seen in Fig. 2, the distance d between the subject and the camera can be calculated if three values are known: the actual diameter of the iris (11.7 mm), the diameter in pixels of the iris captured by the camera (calculated with MediaPipe) and the focal length of the

camera f. The distance can be calculated with (1), following the triangle similarity theorem.

$$d = \frac{real\ iris\ width}{pixel\ iris\ width} \times f \quad (1)$$

The f was calculated experimentally, clearing the focal length from the previous formula and calculating the distance d with a measuring tape in several positions. This method calculates the distance between the camera and the user with a relative error of 10% according to [16].

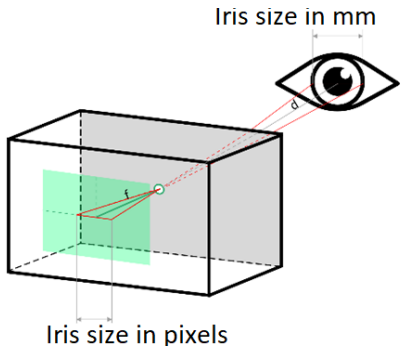

Fig. 2. Pinhole camera model

C. *Final HCI Program to Replace Mouse*

The code of the final HCI program of the paper was written using all the models described in B. This program is developed in Python and performs all the functions of a conventional mouse, which are:

- Move the cursor.
- Make left and right button clicks.
- Double-click.
- Scroll down and up.
- Press and hold the left click for selection.

To execute these functionalities, the program imports the PyAutoGUI library, which executes functions that control mouse functionalities. In addition, the program imports the Pytorch library, a face detector, and the L2CS model to calculate the gaze vector. MediaPipe is also imported to calculate the coordinates of facial points and depth distance. The keras.models library of TensorFlow is imported to be able to load the neural network that was trained in Google Colab.

The following two sections explain in more detail how human-computer interaction works.

a) Pointer Movement

As explained, L2CS-Net and MediaPipe are used to extract features from the frame. These features then enter two neural networks that calculate the xy position that the pointer should take on the screen. This method follows an eye tracker approach, in which the pointer is placed where the user is looking. However, eye trackers never have perfect accuracy and it is rarely better than 2 centimeters. In addition, those that achieve greater accuracy are those that use specific hardware such as IR sensors. On the other hand, webcam-based trackers, like the one in this paper, are the less accurate ones. Finally, the tracker proposed in this paper achieves about 3-4 cm of accuracy.

This level of precision is not enough to move the computer cursor, so a method has been devised that performs the movement of the pointer in two phases. First, the described eye tracking system is used, this phase is designed to quickly move the cursor to an approximate point of the screen, close to the target. Once the pointer approaches the target, the second phase is activated. In this phase, the eye tracker is deactivated and the computer pointer mimics the movement of the facial point of the user's nose.

To choose which method controls the mouse pointer, a facial command has been chosen, which is to open the mouth. That is, when the user's mouth is closed, the pointer moves to the position where the user is looking. When the user opens the mouth, the eye tracker is deactivated and the pointer moves with the nose. The program detects whether the mouth is open or not using the coordinates of two facial points that are calculated with the MediaPipe's detector.

b) Other functionalities

This section explains the other functionalities of the HCI program. These are, left mouse click, right click, double click, scroll down or up, and hold left click to select. These actions are executed by different facial commands that the program can detect due to the relative positions of various facial points that can be calculated with MediaPipe.

Mouse clicks are achieved by making a quick wink. When the right eye is winked, a right click is made and if the left eye is winked, a left click is made. If both eyes are closed at the same time, the program does nothing because it considers it to be a normal blink. To detect if one eye is closed or not, MediaPipe's facial point detector has dots on the edges of each eyelid. By comparing the y-coordinate of those points you can discern if the eye is closed or not.

When an eye is reopened after making a wink, if it has been a quick wink, the program executes the click function of the PyAutoGUI library. In addition, if the user has winked the left eye, when the eye is opened a timer is activated. If the user makes another quick wink of the left eye before a second has passed, the program executes the double-click instruction instead of performing a normal left-click. The one-second time can be adjusted in the program according to the user's preferences.

Before, there was talk about a quick wink. However, if the user keeps the left eye closed for longer, instead of making a click, the program performs the function of holding the left mouse button down to select. The moment the left eye is reopened, the button is no longer pressed.

However, to select, you also have to move the cursor while holding down the click. To achieve good precision during the selection, a special way of moving the cursor has been implemented that is activated exclusively when the left eye is making a long wink. When the program begins to select, the position of the tip of the nose at that moment is saved as a reference. The program calculates four different sectors around the reference, as can be seen to the left of Fig. 3. Then, while the user keeps his eye winked, when the point of the tip of the nose enters one of the sectors, the cursor begins to move in that direction with a constant speed. Therefore, while the cursor is

being selected it can only move in four directions. The speed can be adjusted from the program code.

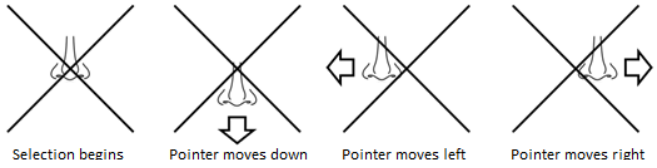

Fig. 3. Example of the pointer movement system while the selection process is taking place

The last functionality that remains to be explained is the scroll. To perform this action, the facial commands that the user has to do are to tilt the head up to scroll up or down to scroll in this direction. To know if the user is tilting his head, the program uses the vertical coordinates of 3 points found with the MediaPipe's detector. These points are the tip of the nose, upper lip, and forehead. Fig. 4 shows the flowchart of the full HCI program that has been discussed in this section.

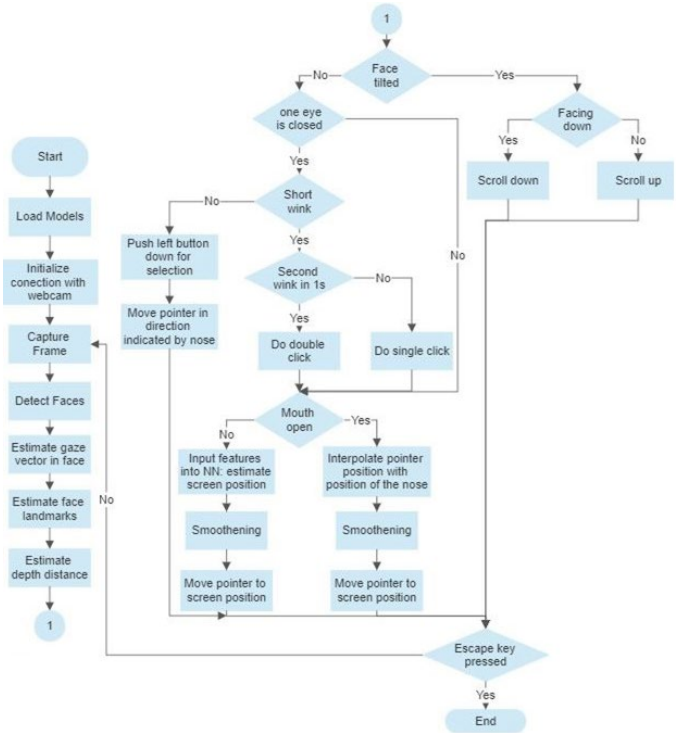

Fig. 4. Flowchart of the HCI program to replace the mouse

II. Results and Discussion

The paper has been developed and tested on an HP Pavilion Gaming Laptop 16-a0xxx, with an Intel(R) Core (TM) i5-10300H processor @2.50 GHz and 32 GB RAM.

A. Results of the Eye Tracker

To design the neural networks, the loss in the validation data was analyzed to choose the values of several hyperparameters. Some of the most important hyperparameters are the number of layers of the network and the number of neurons in each layer. For the neural network that calculates the x-coordinate of the pointer, a network of 6 fully connected layers has been used. The first two layers have 64 neurons, the next two have 128, and the last two have 256 neurons. Apart from these two layers, there is the input layer that depends on the number of inputs and the output layer that only has one neuron, because the network only calculates one value. All layers use the RELU activation function. In addition, it is very important to add a normalization layer at the beginning of the network, because with the unnormalized data, the networks do not work.

The network to calculate the y-coordinate is simpler since no significant difference was appreciated when increasing the size of the network. This network only has 3 fully connected layers of 64 neurons each. It is also necessary to normalize the data for this model.

Three other hyperparameters that are also very important are the error function, the optimizer, and the learning rate. After testing several options provided by Tensorflow, the best error function was the mean square error. The optimizer chosen was Adam, after trying others such as SGD and Adadelta that provided worse results. The learning rate that performed best in the tests was 0.0005.

The performance of the neural network has been measured with the mean absolute error across the test data, comparing the difference between the expected x and y coordinates with the real value of the screen position in each test sample. The mean absolute error for all test data is 103 pixels for the x-coordinate and 115 pixels for the y-coordinate. The screen used for the paper has 1920x1080 pixels and measures 35.5x20 cm. Therefore, the mean absolute error is approximately 1.9 cm for the x-coordinate and 2.1 cm for the y-coordinate.

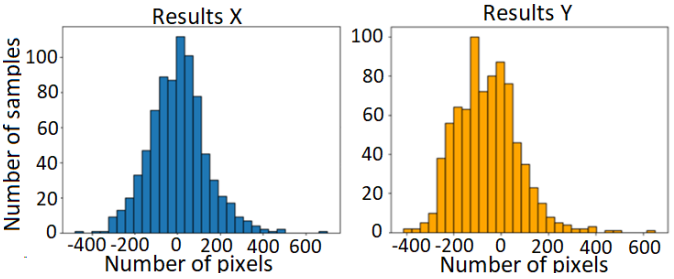

Fig. 5. Representation of the position error between the network's predictions and the real labels for the test data

Fig. 5 shows the histograms of the errors for individual samples. For the x-coordinate, it seems that the histogram is quite centered on zero so there is no pronounced error in one direction than in another. While in the y-coordinate there is a greater number of samples with negative error. The coordinate system used by the program places the origin at the top of the screen and the coordinates grow downwards. So, this error means that the network calculates a position for the pointer lower than it should be, since the value predicted by the network is greater than the actual value of the sample.

B. Results of the Final Program

The final program brings together the tools described: MediaPipe's detector, L2CS-Net, and the neural network. In addition, the program periodically queries facial commands to detect when it has to execute a mouse function. Putting together the execution of so many models and functionalities makes the operation of the program lower its processing speed.

When the program is using neural networks to move the cursor it has a stable speed of 3.5 FPS. When the user opens the mouth, neural networks are not used, as the cursor moves with the motion of the nose. In this case, the program only processes the frame with MediaPipe and L2CS-Net, so the program works at 6 FPS. Clicks are triggered with a quick wink. This is the most difficult facial command to detect by the program since eyelid movement is very fast. However, clicks are usually made when the user has their mouth open because they have more precise control over the pointer. At that moment the program processes 6 frames per second so it is fast enough to detect most winks.

Sometimes the program fails to detect a wink or detects a wink erroneously when the user blinks, but based on system tests these glitches are unusual. The rest of the facial commands to select or scroll are continuous in time so the program detects them without errors.

The accuracy of the movement of the pointer when moving with the nose is good since the MediaPipe facial landmark detector is very accurate and locates the tip of the nose well. When the movement of the cursor is made with the gaze using neural networks, the precision corresponds to that described in the previous section, which was an error of approximately 2 cm in both directions. However, this error is not homogeneous across the screen. Table 1 breaks down the mean absolute error by screen quadrants in the test samples.

TABLE 1. ABSOLUTE ERROR BY SCREEN QUADRANT

Screen Quarter	Error X pixels	Error Y pixels
Top right	91	90
Top left	108	88
Bottom right	117	135
Bottom left	99	168

A larger error can be observed in the lower half of the screen. The webcam is located just above the screen, so this is possibly because the bottom is farther away from the camera. However, getting a real measure of pointer accuracy is very difficult as accuracy varies, not only depending on the quadrant of the screen being looked at, but also depending on the position in which the user is in relation to the computer and camera.

III. CONCLUSION

The paper consists of the design and implementation of a computer program for human-computer interaction or HCI, which performs all the functions of a traditional mouse without the need to use hands.

Moving the pointer is the most complex since it is difficult to create a system fast and comfortable to use and with enough precision to be able to move the cursor of the computer without hands. Therefore, a system has been implemented that uses the eye tracker to quickly move the cursor around the screen without much effort, while in short distances a secondary system can be used that moves the pointer mimicking the movement of the nose. The secondary system is only used to move the cursor from 2 to 3 cm, so the system achieves pinpoint accuracy without completely discarding the comfort and speed of the eye tracker.

In this way, the paper proves that new and improving AI technologies plus better and more widely available webcam hardware can be an alternative to the mouse, for numerous users who suffer from injuries or diseases that affect their motor abilities in arms or hands. In addition, it can also prevent the damage derived by the continued use of a computer mouse, for any computer user.

REFERENCES

[1] T. Alsop, "Desktop/laptop ownership among U.S. adults 2008-2019," Statista, https://www.statista.com/statistics/756054/united-states-adults-desktop-laptop-ownership/#:~:text=As%20of%20February%202019%2C%2074,a%20desktop%20or%20laptop%20computer. (accessed Jul. 11, 2023).

[2] N. Dehghan, "Designing a new computer mouse and evaluating some of its functional parameters," researchgate, https://www.researchgate.net/publication/261609866_Designing_a_new_computer_mouse_and_evaluating_some_of_its_functional_parameters (accessed Jul. 11, 2023).

[3] "Eye tracking market research," Explorer Research, https://explorerresearch.com/eye-tracking-market-research/ (accessed Jul. 11, 2023).

[4] K. Hashimoto, "Template matching using DSP slices on the FPGA," researchgate, https://www.researchgate.net/publication/262424493_Template_matching_using_DSP_slices_on_the_FPGA (accessed Jul. 11, 2023).

[5] "Face detection using Haar Cascades," OpenCV, https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html (accessed Jul. 11, 2023).

[6] A. Rosebrock, "Facial landmarks with dlib, opencv, and python," PyImageSearch, https://pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/ (accessed Jul. 11, 2023).

[7] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," semanticscholar, https://ieeexplore.ieee.org/document/6909637 (accessed Jul. 11, 2023).

[8] "Blob detection using opencv," LearnOpenCV, https://learnopencv.com/blob-detection-using-opencv-python-c/ (accessed Jul. 11, 2023).

[9] V. Agarwal, "Real-time eye tracking using opencv and dlib," Medium, https://towardsdatascience.com/real-time-eye-tracking-using-opencv-and-dlib-b504ca724ac6 (accessed Jul. 11, 2023).

[10] "Face landmark detection guide | mediapipe," Google, https://developers.google.com/mediapipe/solutions/vision/face_landmarker (accessed Jul. 11, 2023).

[11] "Face detection guide | mediapipe," Google, https://developers.google.com/mediapipe/solutions/vision/face_detector#blazeface_short-range (accessed Jul. 11, 2023).

[12] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for Mobile Vision Applications," arXiv.org, https://arxiv.org/abs/1704.04861 (accessed Jul. 11, 2023).

[13] G. Yan and I. Grishchenko, "MediaPipe Face Mesh," Google developers, https://storage.googleapis.com/mediapipe-assets/Model%20Card%20MediaPipe%20Face%20Mesh%20V2.pdf (accessed Jul. 11, 2023).

[14] A. Aflalo, "Eye gaze estimation using a webcam," Medium, https://medium.com/mlearning-ai/eye-gaze-estimation-using-a-webcam-in-100-lines-of-code-570d4683fe23 (accessed Jul. 11, 2023).

[15] A. A. Abdelrahman, T. Hempel, A. Khalifa, and A. Al-Hamadi, "L2CS-net: Fine-grained gaze estimation in unconstrained environments," arXiv.org, https://arxiv.org/abs/2203.03339 (accessed Jul. 11, 2023).

[16] A. Vakunov and D. Lagun, "MediaPipe Iris: Real-time Iris Tracking & depth estimation," Google Research Blog, https://ai.googleblog.com/2020/08/mediapipe-iris-real-time-iris-tracking.html (accessed Jul. 11, 2023).