Mouse cursor control using 3D head pose estimation

Blaž Kovačič
blaz.kovacic@student.um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

ABSTRACT

We developed a method of computer mouse cursor control based on 3D head pose estimation using a web camera image stream. This method can be especially applicable in the field of Assistive Technology interfaces for disabled persons. Our approach stands out by enabling direct conversion of head pose into an absolute position of the mouse cursor, the stability of which was achieved using a special approach of facial key-point filtering. In order to estimate head pose rotation angles we made use of the iterative version of the Perspective-n-Point algorithm together with a seven-point (symmetric) 3D head model, that adjusts itself to the head shape based on facial landmark positions in the starting head position. We demonstrate the effectiveness of our approach using practical comparative measurements.

KEYWORDS

Accessibility, 3D head pose estimation, Perspective-n-Point problem, dlib library, Human-computer interaction

1 INTRODUCTION

According to the World Health Organization, about 15 % of the world's population lives with some form of disability, out of which 2-4 % have significant difficulties in functioning [WHO 2011]. The field of Assistive Technology (AT) is concerned with the development of accessories that help persons with disabilities such as quadriplegia or cerebral palsy, in their everyday lives, work, communication, and have an impact on the general improvement of their quality of life.

Persons with special needs can make use of their devices through physical contact (for example using switches), with the use of facial movement (for example "Face Control" Android app [Obstino 2022]), with head movement (for example a gyroscopic mouse), with eye gaze (eye trackers), or by using their voice (for example Talon or Dragon Naturally Speaking [Nowogrodzki 2018]).

The choice of AT solution often depends on the type of disability and available mobility that a person has. For example, solutions that provide cursor control through head tracking, while requiring mobility of the head, may provide easier and more accurate cursor control for some disabled users compared to eye trackers [Zapała and Bałaj 2012], where the head is often required to be held still. Additional benefit of head trackers is that they can also be implemented in software, requiring only a web camera, presenting a cost effective AT solution.

Certain webcam head tracking solutions already exist on the application market, for example the CameraMouse [Betke et al. 2002] and eViaCam [Mauri 2019] PC software. The downside of the mentioned applications is that their underlying algorithm converts relative head motion into relative cursor movement, resulting in the effect that, with use, cursor will move to a position that is different from the current head gaze direction. For example, when we move the head from the starting position in a certain direction, and then back into the starting position, the mouse cursor is not at the same position anymore, but at a position that doesn't coincide with the head gaze direction, which might make it difficult for the end-user to interact with the computer.

To the best of our knowledge, an application solution that would remove the mentioned weakness doesn't yet exist on the application market, so we decided to research the feasibility of the approach of using the webcam image stream to convert 3D head pose into an *absolute* mouse cursor position, such that it always corresponds to the head gaze direction. Such an approach may potentially offer a favorable alternative to existing implementations.

2 RELATED WORK

[Nabati and Behrad 2010] developed an application solution for disabled persons, which captures an image stream using a monocular camera, and uses the head pose estimate (rotation, translation) using four marked points on the face to convert it into corresponding mouse movement. In their approach they divided the solution on a movement information retrieval module (3D head pose), and mouse movement module. When estimating the 3D head pose they assumed that 4 points on the face are on the same plane (a "N-point" planar face model where the front of the face is modeled as a flat surface) to obtain 3D rotation and translation between the web camera and head pose. Their approach of a 4-point head model is interesting and we expand upon the idea.

[Fu and Huang 2007] describe an approach of using the relative position of the head's bounding box in webcam image to set coarse cursor position on the screen, and they fine tune cursor position by means of estimating the head rotation. Although not specifically aimed at disabled population, the approach is interesting as it uses both head translation and rotation for fine cursor control.

[Tu et al. 2005] describe a 3D model based camera mouse control, implementing what they termed "direct mode" as a one-to-one mapping from obtained head rotations to cursor screen coordinates. Though exhibiting a degree of cursor localization error, with ease of implementation not being it's strong point, the approach does shows promise and prompts us to further research this method.

DOI: https://doi.org/10.26493/scores23.02

3 PROBLEM DESCRIPTION

Our goal was to develop a solution that uses the web camera image stream to estimate the 3D head pose and convert the obtained head rotation angles to absolute mouse cursor position such that it always corresponds to the direction of the head gaze.

The primary challenge that we address in this work is obtaining very stable estimates of head rotation angles through facial key-point filtering, and implementing a mapping function to translate head rotations to cursor screen coordinates.

The problem can be divided into the following parts: 1) image acquisition and face region detection, 2) detection of facial landmarks within the face region, 3) head modeling (for example an N-point 3D model), 4) 3D head pose estimation, 5) stabilization of the obtained head rotation angles, and 5) conversion of the head rotation angles into stable mouse movements.

3D head pose estimation problem

3D head pose estimation is a subset of the more general problem called pose estimation, where the 3D pose (rotation and translation) of an arbitrary object is being estimated. A 3D pose can be estimated by minimizing the so-called reprojection error. A reprojection error is the smallest at that rotation and translation of object points, such that the difference between the positions of those points projected from the 3D object coordinates to the 2D image plane, and between corresponding (observed landmark) points in the captured image is as small as possible (see Fig. 1).

The problem of reprojection error minimization is solved by the so-called Perspective-n-Point (PnP) algorithm [OpenCV.org 2023b], which, additionally, takes into account the camera parameters described by a so-called camera intrinsic matrix (containing information about, for example, focal length and optical center) and distortion coefficients of the matrix (describes how much straight lines are distorted when the image is captured).

The 3D pose estimation is, therefore, obtained when we find the appropriate rotation and translation matrix that minimizes the error when projecting object points into the image plane (point (u, v)). This projection can be written mathematically as [OpenCV.org 2023b] the matrix product given in Eq. (1). The equation describes the projection of world coordinate points $(X_w, Y_w, Z_w, 1)$ onto the image plane points $(u, v, 1)$, where **r** and **t** are parameters of the rotation and translation matrix, and **f** and **c** are focal lengths and optical centers respectively.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{1}$$

4 SOLUTION IMPLEMENTATION

The solution was implemented in the C++ programming language, where the popular OpenCV library was used for computer vision algorithms. To obtain 68 key-points of the facial region, also known as facial landmarks, we used the dlib C++ library, a toolkit containing machine learning and computer vision algorithms and tools [Dlib 2017].

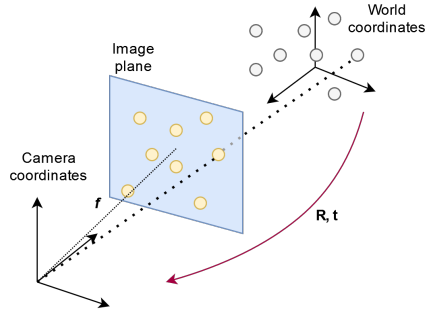

Figure 1: Visualization of the Perspective-n-Point problem (according to [OpenCV.org 2023b]).

Table 1: Computation of 7-point 3D head model given object points in starting head position.

Point	x	y	z
Between the eyebrows	0	$27.y - 33.y$	$0.56 \cdot (45.x - 36.x)$
Left eye corner	$-(27.x - 36.x)$	$36.y$ - $33.y$	$0.56 \cdot (45.x - 36.x)$
Right eye corner	$(27.x - 36.x)$	$36.y - 33.y$	$0.56 \cdot (45.x - 36.x)$
Tip of the nose	0	0	$1.36 \cdot (45.x - 36.x)$
Left mouth corner	$-(54.x - 48.x)/2$	$48.y - 33.y$	$0.70 \cdot (45.x - 36.x)$
Right mouth corner	$(54.x - 48.x)/2$	$48.y - 33.y$	$0.70 \cdot (45.x - 36.x)$
Middle of the chin	0	$8.y - 33.y$	$0.56 \cdot (45.x - 36.x)$

We captured the image stream with 3-channel RGB images of resolution 640x480. First, we detected the facial region using the widely used Viola-Jones algorithm [Viola and Jones 2001] upon which we decided because of its fast OpenCV implementation, using the default FrontalFace pre-trained model from the OpenCV library. Then, we tracked the obtained facial region using the MOSSE (Minimum Output Sum of Squared Error) tracker [Bolme et al. 2010] from the OpenCV contrib repository, which helps reduce computation time, and therefore increases the resulting frame rate, which would otherwise have been affected negatively if we were to detect the facial region repeatedly using the Viola-Jones algorithm. What follows is the detection of facial landmarks using the dlib library, which, when given an input image and location of the facial region, returned 68 landmark points (see image 2) in the starting head position.

When modeling the head, we used seven of those landmark points, namely, the points corresponding to the eye corners (points 36 and 45), the point between the eyebrows (27), nosetip point (33), corners of the mouth (48 and 54), and the point in the middle of the chin (8). We centered the model in a way shown in Table 1. The Table shows the x, y, and z coordinates of the model points, where, for example, $27.y$ means the y position of the 27th point (the point between the eyebrows) that we obtained using the facial landmark detector at the starting head position.

4.1 Obtaining head rotation angles using OpenCV

In our implementation we measured the camera intrinsic matrix and distortion coefficients using the example calibration tool that comes with the OpenCV library package. The tool measures these

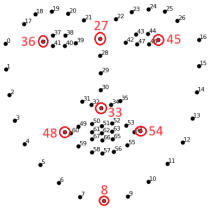

Figure 2: Distribution of 68 facial landmark points [1].

parameters by being shown an image of the chessboard pattern from different angles through the webcam image stream.

Inside the OpenCV environment we made use of the *solvePnP* function [OpenCV.org 2023a], in order to obtain the rotation and translation vector, followed by the use of the *Rodrigues* function [OpenCV.org 2023a], in order to transform the rotation vector into a rotation matrix, and, finally, we used the function *RQDecomp3x3* [OpenCV.org 2023a], in order to transform the rotation matrix into Euler rotation angles. Out of those angles, we made use of the yaw (horizontal head rotation) and pitch (vertical head rotation) angles, and converted them into the absolute position of the mouse cursor.

4.2 Filtering algorithm

With the intent of stabilizing the obtained head rotation angles, we made use of the approach of filtering the x and y positions of each of seven points (a total of 14 filters) that the dlib facial landmark detector returns. For this purpose we implemented an exponential smoothing (IIR) filter with difference equation given by Eq. (2).

$$y[n \quad 1] = (1 - \alpha)x[n] \quad \alpha y[n] \tag{2}$$

We defined $\alpha = 1 - \frac{2\pi f_0}{f_s}$ as the memory factor, basing the definition on properties of the RC low-pass filter from whose Laplace transfer function the Eq. (2) can be derived by taking the inverse Laplace transform and substituting derivatives with finite differences. Here f_0 is the filter cutoff frequency, and $f_s = 30\,\text{Hz}$ is our frame rate.

The final solution implementation used Eq. (2) to filter the seven landmark coordinate points $(u_i[n], v_i[n])$ for $1 \leq i \leq 7$, by applying the filter equation to u and v coordinate components. After we used these points in the PnP algorithm, we obtained estimations of the head rotation angles. We then transformed these rotations into the final cursor position point, and, finally applied the same filter on this point as well, which resulted in an even more stable cursor position.

We determined the appropriate cutoff frequency for filters applied at each point as $f_0 = 0.19\,\text{Hz}$ empirically, which corresponds to $\alpha = 0.96$.

The undesired effect of filtering is the time delay introduced into the movement of cursor during application use. From the filter's transfer function, the filter step response can be shown to be equal to $y(t) = 1 - e^{-2\pi f_0 t}$. The time at which this function reaches 80 % of it's maximum value can be shown to be $t_{delay} = \frac{\ln(0.2)}{2\pi f_0}$. Using this, we could estimate the time delay of our filter at a specified cutoff frequency as $t_{delay}(f_0 = 0.19\,\text{Hz}) \approx 1.35\,\text{s}$, which in a practical setting, manifests as a time delay, that is needed in order for the cursor to move from one side of the screen to the other, when the head is moved quickly between the two positions.

4.3 Conversion of rotation angles into an absolute cursor position

After having obtained the Euler rotation angles of the head, we transformed them into an absolute cursor position according to the equations (3) and (4) given by:

$$x_{pix} = \frac{W}{2} \times \left(1 - \frac{\tan(\theta_x)}{\tan(\theta_{xmax})}\right) \tag{3}$$

$$y_{pix} = \frac{H}{2} \times \left(1 - \frac{\tan(\theta_y)}{\tan(\theta_{ymax})}\right) \tag{4}$$

Here, W and H are the screen width and height in pixels, θ_x and θ_y Euler angles for yaw (horizontal head rotation) and pitch (vertical head rotation) respectively (obtained through the PnP algorithm), and θ_{xmax} and θ_{ymax} are the estimated head yaw and pitch angles at the extreme head positions (when looking at leftmost/rightmost and topmost/bottom side of the screen). In our case we set $\theta_{xmax} = 6.5°$ and $\theta_{ymax} = 8.0°$, which enabled us to use smaller head movements to obtain extreme positions of the cursor.

Eq. (3) was derived by observing $\tan(\theta_x) = \frac{\hat{x}}{D}$, where $\hat{x}$ is the desired cursor x position relative to the center of the screen, θ_x is the yaw angle of the head rotation, and D is the distance of the user's head to the computer monitor. For example, $\hat{x} = 0$ would, in this case, correspond to the center of the computer screen. Then, we observed that, in practice, there exists a maximum yaw angle θ_{xmax}, where the user is looking at the extreme side of the computer screen (for example rightmost), and that there also a similar relationship holds, namely, $\tan(\theta_{xmax}) = \frac{\hat{x}_{max}}{D} = \frac{W/2}{D}$. Dividing the two resulting equations eliminates D, giving $\frac{\tan(\theta)}{\tan(\theta_{xmax})} = \frac{\hat{x}}{W/2}$, which we can now solve easily for $\hat{x}$. Additionally, accounting for the pixel offset of $\frac{W}{2}$ pixels at $\hat{x} = 0$ resulted in $x_{pix} = \frac{W}{2} - \hat{x}$, which, after insertion of $\hat{x}$, yielded Eq. (3). Without loss of generality, the same approach can be used to derive Eq. (4).

5 MEASUREMENTS AND RESULTS

We evaluate our solution by measuring cursor stability during use with three cursor movement tasks, and compare the results to those when using noisy baseline head rotation estimates.

The measurements were performed by an able bodied test subject at a distance of about 60cm from the screen, using a 144 PPI (Pixels Per Inch) screen with a screen resolution of 1920x1080 pixels, using the default parameter values described in Section 4.3 and the filter cutoff frequency $f_0 = 0.19\,\text{Hz}$.

In the first part of the measurements we compared graphs of cursor x and y displacement from a reference center point at the task where the user sequentially moved the head in two directions: 1) left-right, 2) up-down. We measured the stability of the cursor position relative to horizontal axis of the computer screen (x displacement) and vertical axis of the computer screen (y displacement). The results can be seen in Fig. 3, where we can see that the non-filtered cursor positions exhibited a large degree of noise.

[1] Based on: https://www.researchgate.net/figure/The-68-landmarks-detected-by-dlib-library-This-image-was-created-by-Brandon-Amos-of-CMU_fig2_329392737

Table 2: Results of a t-test for cursor position when observing a fixed point.

	No filter	Filter
Mean [pixels]	46.25	8.39
Variance	787.73	16.10
Number of samples	132	201
t Stat	15.39	
P(T<=t)	< 0.00001	
t critical two-sided	1.97	

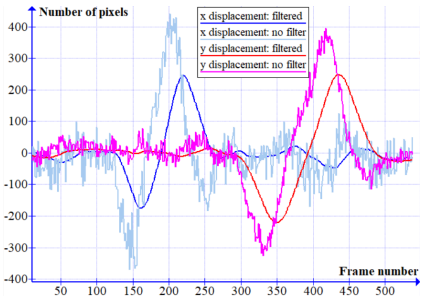

Figure 3: Visualization of filtered and non-filtered cursor displacement along the x and y axes when looking left-right and up-down sequentially.

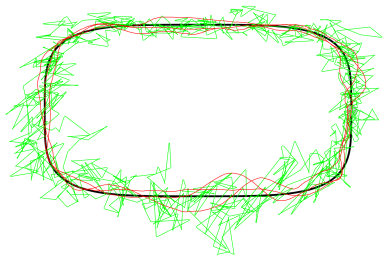

Figure 4: Display of filtered (red) and non-filtered (green) cursor path during a detour around the reference curve (black).

In the second part of the measurements we recorded the cursor location when observing a fixed point - first without the filtering algorithm, and then with the filter. Measurement samples are absolute values of cursor deviation from the point. Then, we performed the t-test, where we obtained the results shown in Table 2. We can see that filtering introduces a statistically significant difference in the results.

In the final part of the measurements we measured cursor deviation magnitude from an ellipse-like curve (with half the width and height the screen dimensions, i.e. 25% side-margins). The subject had the task of moving the cursor using the head mouse three times around the curve, starting and ending in mid-right side of the curve.

For this sample of measurements we obtained the following average values with their Standard Deviations (units are the number of pixels of cursor deviation magnitude from the reference curve): $\mu_{\text{no filter}} = 83.5$ ($\sigma_{\text{no filter}} = 101.1$); $\mu_{\text{filter}} = 17.9$ ($\sigma_{\text{filter}} = 13.8$).

6 CONCLUSION

In this paper we presented a promising method for computer mouse cursor control using webcam 3D head pose estimation, based on filtering of facial key-points. Such an approach is relevant and applicable in the field of Assistive Technology Human-Computer Interfaces.

Our approach stands out by it's ability to map the head rotation vector, $(yaw, pitch)$, directly into corresponding (x, y) positions of the cursor. This differs from available application solutions on the market, which instead translate relative head motion into relative mouse movement, where the head gaze direction need not coincide with cursor position.

In our experimental work we demonstrated the effectiveness of our key-point filtering approach on cursor stability during use. During the task of observing a fixed point, the mean absolute difference between the observed point and actual cursor position was 8.39 pixels with $\sigma^2 = 16.10$, as opposed to the mean of 46.25 pixels with $\sigma^2 = 787.73$ without the filter. During the task of user moving the cursor around the reference curve, we obtained a mean absolute cursor position difference (relative to the curve) of 17.9 pixels ($\sigma = 13.8$ pixels), as opposed to the mean of 83.5 pixels ($\sigma = 101.1$ pixels). Additionally, we demonstrated the cursor denoising ability visually during the task of sequential head movements.

In the future work, we would like to improve the cursor stability by increasing the filter memory factor when the head is held still, which would allow fine-tuned cursor movements and potentially improve the ease of use. Additionally, we would like to include the implementation of facial gesture detection for mouse click simulation and automatic fixation of cursor on UI elements (e.g. buttons).

REFERENCES

Margrit Betke, James Gips, and Peter Fleming. 2002. The camera mouse: visual tracking of body features to provide computer access for people with severe disabilities. *IEEE Transactions on neural systems and Rehabilitation Engineering* 10, 1 (2002), 1–10.

David S Bolme, J Ross Beveridge, Bruce A Draper, and Yui Man Lui. 2010. Visual object tracking using adaptive correlation filters. In *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE, 2544–2550.

C++ Dlib. 2017. Dlib C++ Library. *Dlib. net* (2017).

Yun Fu and Thomas S. Huang. 2007. hMouse: Head Tracking Driven Virtual Computer Mouse. In *2007 IEEE Workshop on Applications of Computer Vision (WACV '07)*. 30–30. https://doi.org/10.1109/WACV.2007.29

Cesar Mauri. 2019. *Enable Viacam Website.* http://eviacam.crea-si.com/index.php Accessed on 2023-06-30.

Masoomeh Nabati and Alireza Behrad. 2010. Camera mouse implementation using 3D head pose estimation by monocular video camera and 2D to 3D point and line correspondences. (2010), 825–830. https://doi.org/10.1109/ISTEL.2010.5734136

ANNA Nowogrodzki. 2018. Writing code out loud. *Nature* 559, 7712 (2018), 141–142.

Obstino Association - Društvo Obstino. 2022. *Face Control Android application.* https://play.google.com/store/apps/details?id=com.obstino.facecontrol

OpenCV.org. 2023a. *OpenCV documentation Camera Calibration and 3D Reconstruction.* https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html Accessed on 2023-3-3.

OpenCV.org. 2023b. *OpenCV documentation Perspective-n-Point (PnP) pose computation.* https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html Accessed on 2023-3-3.

Jilin Tu, T. Huang, and Hai Tao. 2005. Face as mouse through visual face tracking. In *The 2nd Canadian Conference on Computer and Robot Vision (CRV'05)*. 339–346. https://doi.org/10.1109/CRV.2005.39

P. Viola and M. Jones. 2001. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Vol. 1. I–I. https://doi.org/10.1109/CVPR.2001.990517

WHO. 2011. *World Report on Disability 2011.* https://www.who.int/teams/noncommunicable-diseases/sensory-functions-disability-and-rehabilitation/world-report-on-disability Accessed on 2023-05-17.

Dariusz Zapała and Bibianna Bałaj. 2012. Eye Tracking and Head Tracking–The two approaches in assistive technologies. (2012).