Journal of Computer and Electrical Sciences (JCES)- Volume-1 Issue-4, April 2026
ISSN: 3049-2602

GESTURE CONTROLLED VIRTUAL MOUSE

T. Dheeraj[1], Mr. K. Kishore Kumar[2]
[1]Student, Department of Computer Science and Engineering
Andhra Loyola Institute of Engineering and Technology, Vijayawada, Andhra Pradesh, India
[2]Assistant Professor, Department of Computer Science and Engineering
Andhra Loyola Institute of Engineering and Technology, Vijayawada, Andhra Pradesh, India
Email: dheerajtalari16@gmail.com

Abstract: *The current paper outlines the design and development of a Gesture Controlling Virtual Mouse system that will allow users to function an entire computer without relying on the use of more traditional input hardware, completely using their hands. Live human hand movement is captured by a simple webcam and real time processed using a computer vision code that is created using the MediaPipe and the OpenCV library. The system takes advantage of an applicable gesture list, including cursor, left and right clicks, scrolling, drags, capturing of screenshots, controlling media playback, and volume control. Besides the gesture recognition, the project will also have smart modules of face recognition-based user authentication, eye-tracked scroll through gaze, auto lock to a screen when idle, voice command interpretation, and a free hand air-drawing board. The whole application is written in Python and can be run on any standard Windows 10/11 machine and regularly shows the frame rates required to have a smooth and responsive experience. The proposals were tested and it can be validated that the proposed system can be considered as a viable hands-free interface, which can be applied to accessibility, contactless computing and presentation settings.*

Keywords: *Gesture Recognition, Virtual Mouse, MediaPipe, OpenCV, Face Authentication, Eye Tracking, Human-Computer Interaction, Computer Vision, Contactless Interface.*

1. INTRODUCTION

Physical mouse and keyboard have been human computer interface of dominance over decades. Although useful in most places, they are inconvenient or unwelcome in places where touching is an issue - sterile medical centers, interactive kiosks in the public, user-accessible applications with motor impairment, and immersive presentation environments. A gesture-based virtual mouse solves these problems by making the actual human hand the pointing and clicking device.

The interfaces are now possible using consumer hardware due to recent advances in real-time computer vision and broad availability of open-source libraries. MediaPipe Hands is a product released by Google and provides 21 key points per hand in near-real time on a standard laptop processor. Combined with the use of the OpenCV on handling frames and the use of PyAutoGUI on injecting into the system, a fully operational gestural interface is all that is needed with a built-in webcam.

A much more advanced system is the one below, Virtual Mouse v9, which extends the cursor control by-pointing to a significant degree. It describes a gestures set of fifteen movements that include all common mouse functions and expands it to those of the media and slide navigation, and system shortcuts. Over that are face recognition, eye scroll, idle locking, voice recognition, and a drawing board where one can draw freely, making it one of the most complete feature gesture interfaces ever created using only commodity hardware.

The paper is structure as provided in the following way. Section 2 conducts a literature review. The system architecture is described in Section 3. Section 4 provides the description of gesture

https://doi.org/10.5281/zenodo.19605953

vocabulary and detection methodology. The AI-powered auxiliary features are in section 5. Section 6 uptakes implementation overview along with test outcomes. Section 7 contains recommendations on improvement in the future.

2. LITERATURE SURVEY

Studies on vision-based interaction of gestures are many decades old. The initial systems were based on the use of coloured gloves or fiducial in order to simplify the hand separation which limited the users as well as the environments. The transition to marker less skin-tone could be secured by the development and introduction of powerful feature descriptors and machine-learning classifiers that enabled practical execution of real-time skin-tone segmentation and contour analysis.

Mitra and Acharya (2007) created one of the powerful surveys that identified many types of motion patterns; essentially, a difference was drawn between those which were statical and those which were dynamic. Their paper emphasised the relevance of the temporal modelling with regards to the sequential gestures, which encouraged the use of Hidden Markov Models in the recognition pipelines that followed.

With the onset of deep learning, the area was advanced. Convolutional neural networks were found to be even better than hand-written pipelines, such that Molchanov et al. (2016) showed videos could be classified based on the gesture using 3D convolutional networks but due to the available resources at the time this was limited to powerful workstations.

MediaPipe Hands gained prominence due to the release of MediaPipe Hands which provided a two-stage pipeline palm-detecting then landmark regression, and this was implemented on the mobile computers in real-time. This made gesture-interface research democratised to a big extent. The majority of the published implementations that came after it are only cursor movement or small set of gestures and few of them integrate with biometric security or gaze interaction in a single unified application.

A study on gaze-direction estimation by using monocular cameras (Hansen and Ji, 2010) revealed that the scroll control is possible using geometrical information on facial landmarks without effectively complex machinery. When eye tracking is integrated it creates a complementary input channel which helps decrease the amount of fatigue in the hands when a long session is on. It is now possible to perform biometric authentication on a single frame of a camera with face recognition algorithms based on deep metrics learning, e.g. FaceNet, and employ embedded face-gating of interfaces that accept gestures, practically achieved by having constitutions comply with privacy regulations and fostering trust.

The current project integrates hand tracking, gaze estimating, face recognition, and speech processing into a single Python program that will not require any clouds to run on any consumer hardware.

3. SYSTEM ARCHITECTURE

The system is developed on the basis of a modular design, where each ability is described in a Python module. A central configuration file controls what modules are working on and what parameters are applied to them so that failure on any one component cannot impact the other parts of the application.

3.1 Architecture Overview

The end-to-end flow of the system is indicated in Figure 1. The webcam provides gesture input and hand tracking feeds into the Hand Gesture Recognition engine, which is used to drive a three stage gesture detecting pipeline including movement, click, as well as scroll. The identified gesture is then read and translated to tangible mouse behaviours by the Gesture

https://doi.org/10.5281/zenodo.19605953

Interpretations and Mapping module and finally, generates motion of cursor, mouse clicks and scrolling motion which act as command of the host personal computer.

.

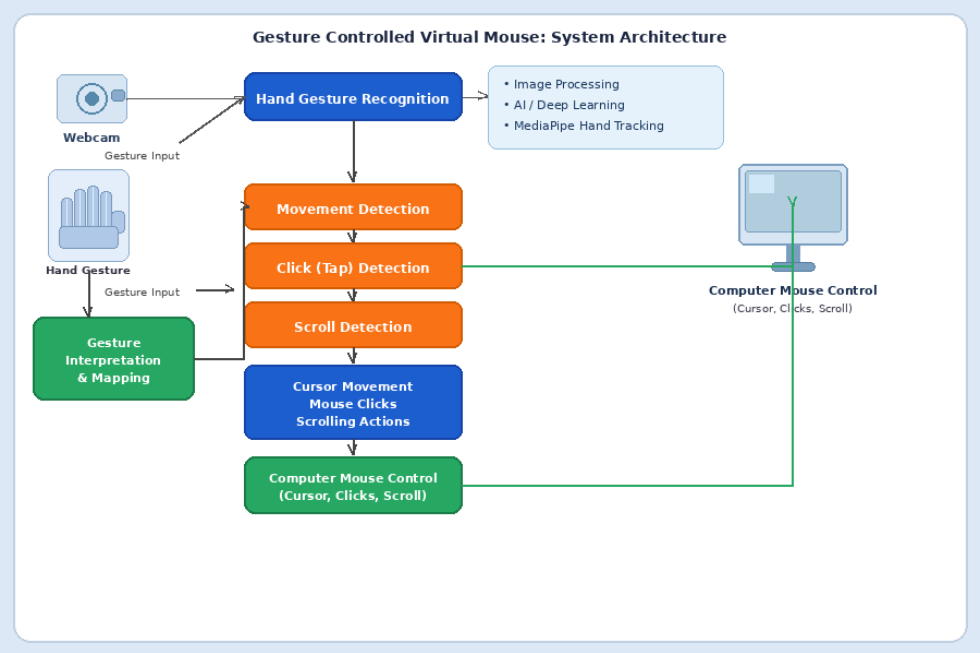

Fig. 1: System Architecture of the Gesture Controlled Virtual Mouse

3.2 Module Descriptions

The processing core comprises nine interconnected modules:

- Hand Tracker — wraps MediaPipe Hands and exposes both normalised and pixel-space landmark coordinates to downstream modules.
- Gesture Detector — applies rule-based geometry analysis and temporal smoothing to classify one of fifteen named gesture states per frame.
- Mouse Controller — translates gesture labels into system-level mouse and keyboard events via PyAutoGUI, covering move, click, drag, scroll, media keys, and system shortcuts.
- Face AUTH — handles face profile registration and authentication using a deep embedding model, serving both the startup access gate and runtime re-authentication after a lock event.
- Eye Tracker — estimates gaze direction from facial landmarks and maps it to vertical scroll events, providing an alternative input channel that does not require hand movement.
- Activity Monitor — timestamps every user activity and triggers an automatic screen lock once a configurable idle threshold is exceeded, releasing only on successful face authentication.
- Voice Controller — runs a speech-recognition listener on a background thread, injecting recognised commands into a queue the main loop processes each iteration.
- Air Drawing — maintains a transparent canvas overlay on the live frame, writing freehand strokes traced by the index fingertip when drawing mode is active.
- Presentation Control — detects rapid horizontal swipes and fires next-slide or previous-slide commands accordingly.

3.3 Data Flow

The channel followed by each frame of a webcam is: (1) horizontal flip (mirror-style interaction), (2) detecting landmarks on hands, (3) gesture recognition, (4) gesture-to-action, (5) HUD compositing and (6) window display. Face AUTH and Eye Tracker creates branch

https://doi.org/10.5281/zenodo.19605953

of main loop, which is triggered by configuration flags or keyboard shortcuts. Voice Controller works in a dedicated thread, and it adds its commands to a common command queue that the main loop fills up on a frame-by-frame basis***3.4 Configuration***

The tuneable parameters, which include camera index, frame resolution, MediaPipe confidence thresholds, face-matching threshold, idle timeout, eye-scroll speed, and enable flags of each module are controlled in a single config.py file. The only adjustment needed to adapt the system to various hardware or deployment cases is the adjustments to this file and not to application logic.

4. GESTURE VOCABULARY AND DETECTION

The system identifies fifteen different states of gestures that are organized into three functional groups: cursor control, system actions and media control.

4.1 Hand Landmark Representation

MediaPipe Hands yields 21 3-D landmarks per hand in normal form of hand binding box. Geometry-based calculations in finger-state are fed using the normalised coordinates, and pixel coordinates are used to telemap finger positions to screen positions. The extent of finger extension is determined by individually comparing the tip landmark with the the proximal knuckle in the primary axis of the hand and the wrist position is taken into consideration to accommodate the palm-up and palm-down orientations.

4.2 Cursor Control Gestures

The press of the index and middle fingers in combination - the peace sign - is used to move the cursor, with the location between the two finger tips being the scale of the screen image with a programmable smoothing factor. Left click is brought about by a single raised index finger. Right click is made by three fingers raised (index finger, middle finger, and ring finger). When the index and ring fingers are raised and the middle finger is left down, the scrolling is activated with the direction and speed movement being controlled by the vertical movement. A drag operation is initiated by holding a closed fist 0.6 seconds and terminated by opening up the hand.

4.3 System and Presentation Gestures

When all five fingers are spread, the screenshot gesture is triggered which puts the output as a timestamped PNG file on the disk. The Windows key is sent by four lifted fingers (index thumb to Pinky). Quick horizontal swipes of the index fingertips - followed by x-velocity between carbonated frames - issue next-slide commands or previous-slide commands..

4.4 Media Control Gestures

The volume can be controlled by the shaka sign (thumb and Pinky extended): the vertical movement of the hand can be designated to volume rise or fall. Middle finger up and down switches system mute. Clicking the ring finger alone will transmit a play/ pause keystroke to the currently running media player.

This step significantly decreases the jitter associated with digital video signal sounds.

.

4.5 Temporal Smoothing and Debouncing

Raw frame-level gesture labels are processed by a sliding-window majority which filters off temporary false-identifications due either to motion blur or partial occlusion. In addition, per-gesture cooldown timers protect click and system-action gestures. The drag gesture presupposes a fist that is in place outside 0.6 seconds, which is different than a short neutral fist position

https://doi.org/10.5281/zenodo.19605953

5. AI-POWERED AUXILIARY FEATURES

5.1 Face Authentication

When it is initiating, the application verifies the presence of face profiles. In case none of them are available, it offers the user a series of registration that captures the facial parameters entered by live webcam. In later launches an authentication gate is used to prevent entry until the registered face is found. The corresponding threshold can be configured in config.py, and the lock screen will show a live confidence score to assist the users with their positioning.

5.2 Eye Tracking Scroll Control

The mode of hand gestures is deactivated by the press of the eye-tracking mode (E key); it is replaced by gaze-controlled scroll. With the Eye Tracker module, it creates an estimation of the direction of gaze basing on the facial landmarks and converting upward gaze to upward scroll and downward gaze to downward scroll at a set speed. The initial run is a calibration step that the device requires to identify the neutral gaze. This mode minimizes fatigue of the hands when it comes to the long document reading.

5.3 Automatic Screen Lock

Activity Monitor module records the time of the last event that was detected. As the duration spent idleness nears the set time limit a colour-coded warning bar appears across the HUD becoming blue, and turning red towards the deadline. After a timer runs out, gestures cannot be entered, the camera display becomes grayed and only a notable match of a face can restart the work. There is also a manual lock that is accessed through the L key.

5.4 Voice Control

There is a background thread that is the speech-recognition listener that runs continuously. Known commands are compared to a command table and translated to mouse or system commands, and users who are tired of extended hand movements are given a spoken version. Its voice control is switched on and off via the V key without disrupting gesture recognition

5.5 Air Drawing Canvas

The mode of freehand drawing (pressing D) enables the index fingertip to draw over a transparent overlay, which is then overlapped onto the live camera image display. C can be used to clear the canvas and S to save as a PNG allowing on-screen annotation when making screencast and slides..

6. IMPLEMENTATION AND RESULTS

6.1 Development Environment

The system has been created in Python 3.10 on windows 11. It has core dependencies of OpenCV to receive and render frames, MediaPipe to detect hand and face landmarks, PyAutoGui to emulate a mouse and a keyboard, and the Speech Recognition library containing Google Web Speech API to process voice. The webcam was set to provide compressed MJPG at 30 fps and a resolution of 1920x1080.

6.2 Real-Time Performance

With a standard indoor light where only one hand is visible, the system maintained a display frame rate of 2530 fps on a mid-range laptop having an Intel core i5 processor and 8 GB of RAM. The average latency between frame capture and the output of gesture labels was less than 35 milliseconds which is well below what is considered to be perceived as real-time interaction. The FPS value was consistent over long durations of study as shown through the on-screen counter.

https://doi.org/10.5281/zenodo.19605953

6.3 Gesture Recognition Accuracy

Unofficial testing with five participants of each of the fifteen gesture classes was conducted on a non-randomized basis so that the stationary postures (cursor move, clicks, screenshot, win key) should be recognized at the first attempt rated at more than 90 percent in the indoor light conditions. Slide navigation using dynamic swipe gestures had an approximate first attempt accuracy of 85% with the difference being mainly due to individual variation in the speed of swipe. The temporal majority-vote smoothing decreased false-triggering significantly in comparison to the per-frame classification of the raw.

6.4 Face Authentication Performance

With less than ten seconds, registration with a new user was completed. The time to authenticate subsequent sessions was normally two to four seconds. The confidence rating on the screen came in handy to control users to change position where the baseline scores were moderate. False acceptances were not noticed in three different tests with different registered users.

.

6.5 User Experience

The HUD is positioned in small strips in both the upper and lower part of the window leaving the dominant section uninhibited with the tracking of the hands. A fold-out gesture guide (H key) contains the fifteen gestures which are displayed with coloured labels. The action in the form of clicks, setting volumes, and saving a screenshot confirmations are done through the notification banners without covering the tracking area. Every volunteer said that they were able to pick up the entire vocabulary of gestures in several minutes under the guide.

6.6 Test Cases and System Screenshots

The screenshots below were taken when the system was operating live to ensure the presence of gesture recognition, system startups tests, and tweezing the cursor. The tests in each case were done in the development machine under the normal indoor conditions.

Test Case 1: Hand Landmark Detection — Open Palm

An open-palm gesture will be identified by the system as depicted in figure 2. The 21 MediaPipe landmarks are real-time drawn on the hand and landmarks of the fingers are indicated with a white circle and the knuckle landmarks with a green dot. By overlaying the skeleton on the hand tracking pipeline it is established that the hand tracking pipeline accurately identifies all the fingers joints which is the foundation upon which all the subsequent gesture classification relies upon.

https://doi.org/10.5281/zenodo.19605953

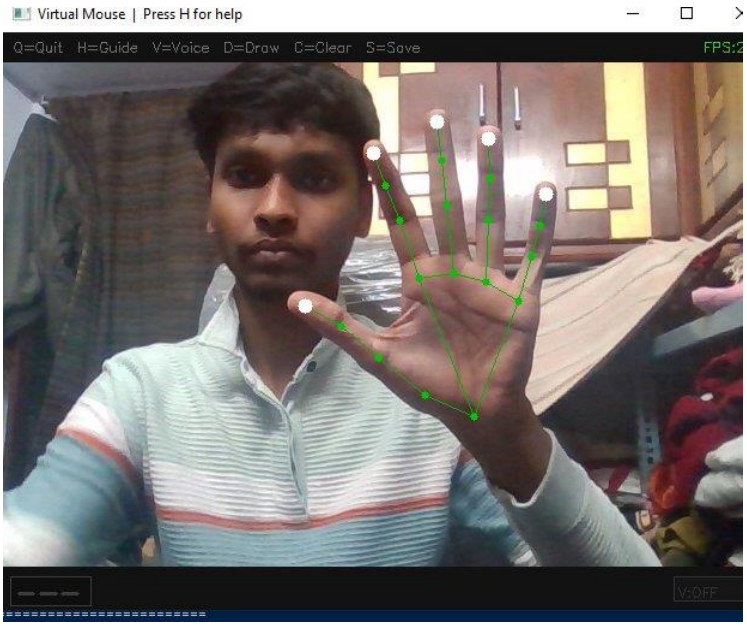

Fig. 2: Live hand landmark overlay on an open-palm gesture (all 21 Media Pipe key points visible)

Test Case 2: System Startup and Library Validation

As seen in figure 3, the project startup script (run-project.py) gives the terminal output. The script is a sequential check of all the necessary and optional Python packages, namely, cv2, media pipe, pyautogui, numpy, PIL, speech recognition, and pyaudio, and the presence of a web-camera. Checks are all positive and therefore indicate the environment is appropriately configured before the main application loop commences.

The gesture tested in the case 3 is the Peace Sign where the fingers move like a cross to form a peace sign.

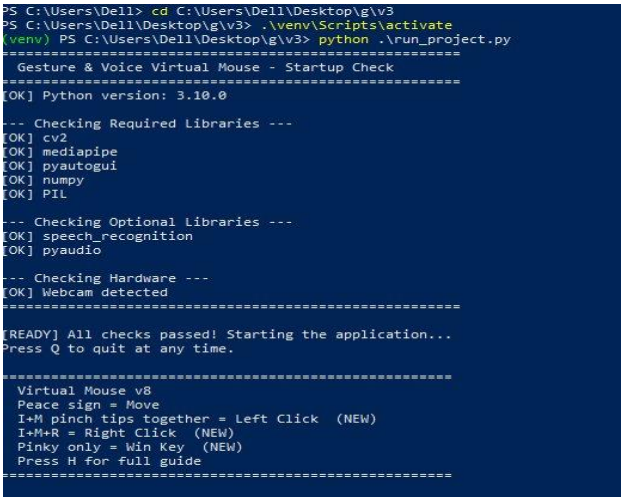

Fig. 3: Startup validation output confirming all required libraries and webcam are detected

Test Case 3: Cursor Movement Gesture — Peace Sign

Just as the Figure 4 shows, the peace-sign (index and middle fingers up) gesture is used to activate cursor movement mode. And the HUD component, the MOVE, appears clearly as it is written near the bottom-left of the window in green, which it distinctly reveals that the Gesture Detector has properly identified the gesture state. The system positions the centre of the two elevated fingertips to screen coordinates (in real time), and the FPS display (to the

https://doi.org/10.5281/zenodo.19605953

right of the top) displays 23 fps, which is significantly lower than the range of values needed to display any image smoothly.

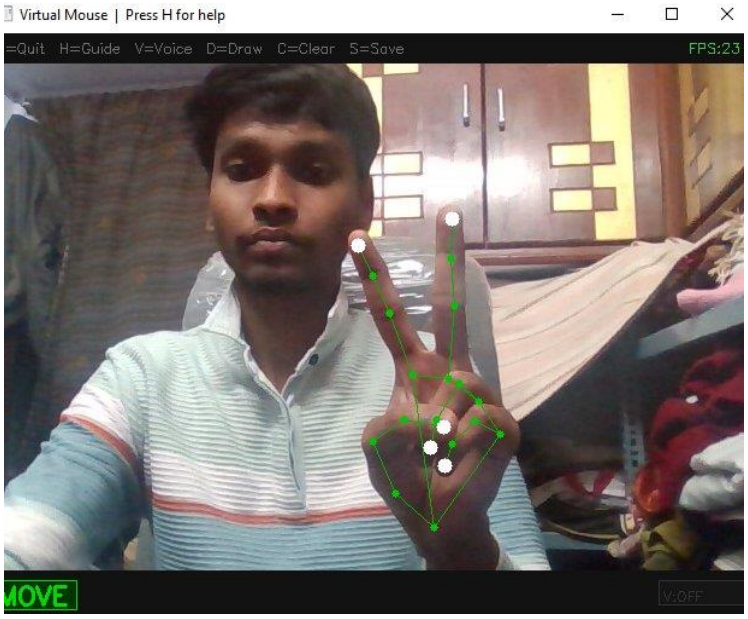

Fig. 4: Peace-sign gesture activating MOVE mode — HUD label and FPS counter confirm real-time operation

Collectively, these three test cases prove the basic pipeline: initialisation of a library and a hardware perform its duties properly with no errors, hand landmarks are correctly detected with all 21 key points, and gesture classifier recognises the posture of the cursor movement and shows it in the HUD in the real time.

7. CONCLUSION AND FUTURE WORK

This article has introduced Virtual Mouse v9 an interface that is a gesture-based human-computer interaction which is driven solely on the laptop with only a webcam on it. The system is based on the fifteen-gesture mouse and media control vocabulary plus intelligent auxiliary modules to face authentication, gaze scroll, auto screen lock, voice commands and air drawing designed in Python without specialised hardware.

Real time testing was confirmed at over 90, and 85 percent accuracy with static and dynamic gesture and swipe, respectively. Initial tests of face authentication did not provide any false acceptances and worked perfectly well. The modular implementation enables one to upgrade separately the individual components as the underlying libraries are changed. Live test cases also established proper landmark detection, clean startup verification as well as proper real-time gesture classification.

Future research will also exchange the rule-based gesture engine with a lightweight neural classifier in order to achieve greater robustness in different hand orientations and under different conditions of light. The expansion of the vocabulary to include two-hand gestures would allow pinch-to-zoom and rotation interactions. It would be possible to extend the range of deployment by porting the application to Linux and macOS and optimising the face-authentication pipeline on embedded platforms, including the Raspberry Pi.

https://doi.org/10.5281/zenodo.19605953

REFERENCES

1. S. Mitra and T. Acharya, "Gesture Recognition: A Survey," IEEE Transactions on Systems, Man, and Cybernetics, Part C, vol. 37, no. 3, pp. 311–324, 2007.

2. P. Molchanov, S. Gupta, K. Kim, and J. Kautz, "Hand Gesture Recognition with 3D Convolutional Neural Networks," in Proc. IEEE CVPR Workshops, 2016.

3. F. Zhang et al., "MediaPipe Hands: On-device Real-time Hand Tracking," arXiv:2006.10214, 2020.

4. D. W. Hansen and Q. Ji, "In the Eye of the Beholder: A Survey of Models for Eyes and Gaze," IEEE Trans. Pattern Anal. Mach. Intell., vol. 32, no. 3, pp. 478–500, 2010.

5. F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," in Proc. IEEE CVPR, pp. 815–823, 2015.

6. G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, 2000.

7. GoogleLLC,MediaPipeSolutionsDocumentation, https://developers.google.com/mediapipe, 2023.

8. A. Rosebrock, OpenCV with Python for Image and Video Analysis, PyImageSearch, 2021.

9. R. Szeliski, Computer Vision: Algorithms and Applications, 2nd ed., Springer, 2022.

10. C. Kanan and G. W. Cottrell, "Color-to-Grayscale: Does the Method Matter in Image Recognition?"

    PLOS ONE, vol. 7, no. 1, e29740, 2012.

https://doi.org/10.5281/zenodo.19605953