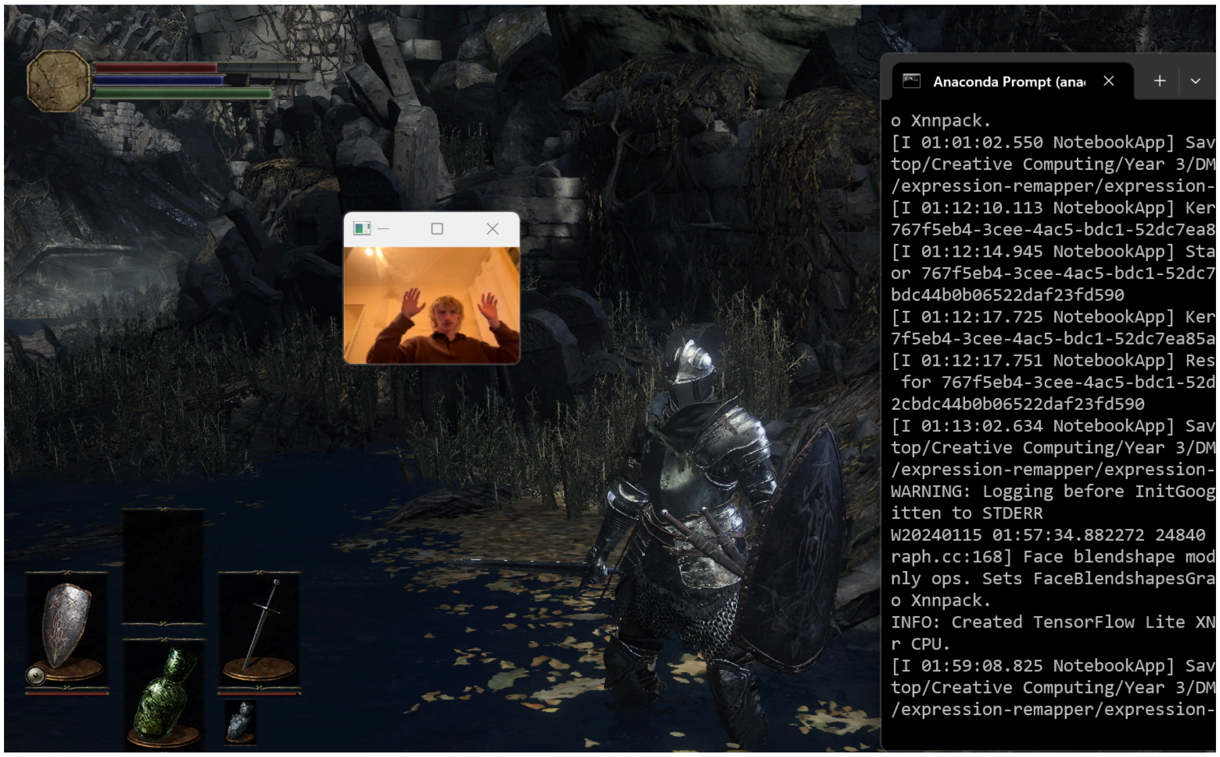


```
o Xnnpack.
[I 01:01:02.550 NotebookApp] Sav
top/Creative Computing/Year 3/DM
/expression-remapper/expression-
[I 01:12:10.113 NotebookApp] Ker
767f5eb4-3cee-4ac5-bdc1-52dc7ea8
[I 01:12:14.945 NotebookApp] Sta
or 767f5eb4-3cee-4ac5-bdc1-52dc7
bdc44b0b06522daf23fd590
[I 01:12:17.725 NotebookApp] Ker
7f5eb4-3cee-4ac5-bdc1-52dc7ea85a
[I 01:12:17.751 NotebookApp] Res
 for 767f5eb4-3cee-4ac5-bdc1-52d
2cbdc44b0b06522daf23fd590
[I 01:13:02.634 NotebookApp] Sav
top/Creative Computing/Year 3/DM
/expression-remapper/expression-
WARNING: Logging before InitGoog
itten to STDERR
W20240115 01:57:34.882272 24840
raph.cc:168] Face blendshape mod
nly ops. Sets FaceBlendshapesGra
o Xnnpack.
INFO: Created TensorFlow Lite XN
r CPU.
[I 01:59:08.825 NotebookApp] Sav
top/Creative Computing/Year 3/DM
/expression-remapper/expression-
```

# The worst way to play Dark Souls

By Johnny Henson

Using computer vision to map head movements and facial articulation to keyboard inputs, with the ultimate aim of defeating the first enemy of the game 'Dark Souls III' without using my hands.

Video demonstration (YouTube):
https://youtu.be/9dBxw7-ve9k

## Introduction, Concept, and Background Research

The *Dark Souls* (2016) series has developed a reputation for containing some of the most challenging and unforgiving video games ever created. This challenge has made the games something of a proving ground for people looking to test their skill and determination, but for some, just completing the game as it was intended is not enough of a challenge; and so a tradition of challenge runs was born (*Challenge run*, 2023).

Some challenge runs are straightforward, like completing the game without taking a single hit, but there is a more unusual sub-category of *Dark Souls* challenge runs where players use unorthodox, and often impractical, input devices to play and beat the game. Some notable controllers used to beat Dark Souls are a dance mat (Luality, 2018), bananas (SuperLouis64, 2020), a guitar, and voice commands (Kamen, 2015).

Most recently I discovered the work of Perri Karyal who designed a system that mapped her brain activity, monitored with an EEG, to keyboard inputs–and then used it to beat the newest Dark Souls game with her brain alone (Karyal, 2023). Karyal explains that she needed to train a software to recognize patterns in her neural activity while visualising different physical actions–for example, imagining pushing an object forward over and over again. Then she mapped the brain activity generated by these different visualisations to virtual keypresses in order to play the game. Karyal's work served as the primary inspiration for my project; I wanted to explore whether it was possible to achieve a similar hands-free style of play without the need for any additional hardware like the EEG or extensive training.

## Controls

| Face articulation | In-game action |
|---|---|
| Pitch head forward | Walk forward |
| Pitch head back | Walk back |
| Roll head left | Walk left |
| Roll head right | Walk right |
| Open mouth | Light attack |
| Pucker lips, Mouth left, Mouth right | Dodge roll |
| Raise left eyebrow | Toggle lock-on |

### Instructions

(1) Open the game you want to play, if you want. Otherwise a text editor will suffice to demonstrate.
    (a) Update the keybindings in the jupyter notebook to match the game you want to play.
(2) Run the cells of the jupyter notebook one by one ('Jupyter Notebook', no date).
(3) Tab into the window that pops up. It should show your webcam feed.
    (a) Look straight at your monitor/screen comfortably as you would when playing, then press the 'ENTER' key on your keyboard to calibrate.
    (b) When ready to play, press the 'o' key to enter play mode (enables programmatic keypresses).
(4) Tab back into the game window.
(5) Play according to the controls in the table above.
(6) Once done playing, tab back into the webcam feed window.
    (a) Press 'q' on your keyboard to quit the application.

## Technical Implementation

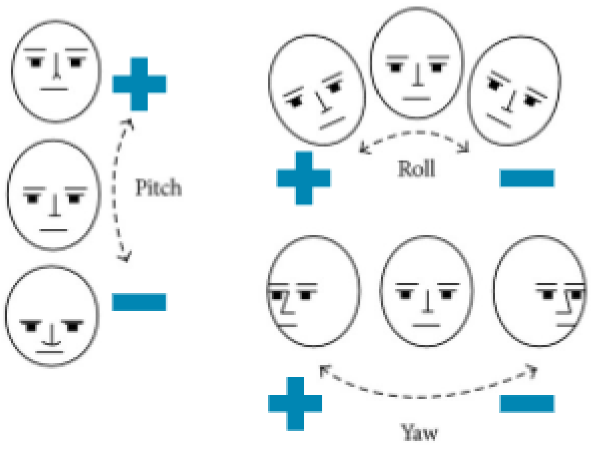
### Choice of tools and language

I built the computer vision aspect of this project using the Face Landmark Detection solution from MediaPipe ('Face Landmark Detection Solution', no date). Their API is usable with Python or JavaScript; I chose to use the Python version, despite being more proficient with JavaScript, because I anticipated better performance using it. Additionally, emulating virtual keypresses, which is a core requirement for this project, is very simple using Python and very challenging with JavaScript.

MediaPipe's solution stood out over others because, on top of the landmarks, it also outputs the approximated 3D orientation of the detected head, and blendshapes for different facial actions (e.g. *leftBlink*, *leftEyebrowRaised* etc.), which simplifies much of the process.

## Obtaining the orientation of the head

MediaPipe conveniently has an option to output the detected faces' transformation matrices. I originally used this data, and converted it to yaw, pitch, and roll, which I then used to trigger different actions when the values went beyond certain thresholds. For example, if the roll of the head is greater than 0.1, move left.



(Thierfelder, 2023)

During the testing phase the program was functioning with very high latency which made the game nearly unplayable. I discovered that disabling the output of the transformation matrices boosted the performance by up to 500% on my laptop, so I tried something else.

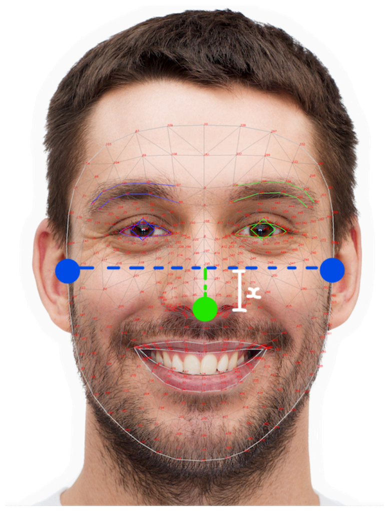
*Matrix output ENABLED*

```
[RESULT] ~11.27 FPS
```

*Matrix output DISABLED*

```
[RESULT] ~50.59 FPS
```

As an alternative, I used the landmarks of the face to loosely approximate the pitch and roll of the head. For the pitch (up/down tilt) I averaged the y position of the vertices by each ear to act as an approximation of the center of the head, and then compared that position to the y position of the tip of the nose. The y distance between these points would be around zero when facing forward, positive when looking up, and negative when looking down. I then used this information trigger different actions when the distance (approximated pitch) grew or shrank beyond a certain threshold. I followed a similar process for roll but with different points.



*A diagram of the method for approximating pitch. The blue dots are the ears, the blue dotted line is the average y position of the two ears, the green dot is the nose position, and x is the distance between the ears and nose on the y axis (the approximated pitch). Diagram drawn on top of image from MediaPipe docs. Available at: https://mediapipe-studio.webapps.google.com/studio/demo/face_landmarker*

Using the above method I was able to maintain the same ~50fps speeds.

## Calibration

The method described above works well enough, but the head orientation it calculates is relative to the *camera*, not the person. This means that if the webcam isn't perfectly centered the readings will be wrong; to a webcam at a high angle, you will appear to be looking down even if actually looking forward. To solve this I implemented calibration. The user looks comfortably at the screen as they would when playing, and then presses the calibrate hotkey 'ENTER', which uses their current orientation as a base point, and measures all future orientation *relative* to this.

## Choosing appropriate landmarks

The API I'm using returns 52 facial blendshape scores, but only a few are suitable for this task. I chose which landmarks to use as controls by going through each and considering the following:

**(1) Ease of articulation**
The frequency and urgency of an in-game action should correlate with the ease of articulation. The most common in-game actions like attacking and dodging should be mapped to comfortable and easy to articulate parts of the face. This is why I've bound attacking to the opening of the mouth (as silly as it looks). Some less common actions like locking-on to an enemy or healing could reasonably be bound to less comfortable articulations without causing fatigue, because of their less frequent use.

**(2) Reliability of tracking**
The reason I chose face tracking over full body pose-estimation for this project is because of reliability. I found that, in my own tests, face tracking models were able to more reliably estimate landmarks than pose models, and with a lower inference time too. But even within face tracking, some blendshapes are more reliably tracked than others; the blink blendshapes in particular were not reliable enough to be used for any crucial actions. Similar to above, I chose the mouth for attacking because it had reliable tracking.

## Selecting score thresholds

The model returns 52 blendshapes representing unique areas of articulation, along with a score to indicate the extent of articulation; a closed mouth would have a score of 0 and a widely opened mouth would have a score closer to 1, with a range between. I had to decide appropriate thresholds at which the in-game actions would be triggered.

Selecting these thresholds was a balance between responsiveness and accuracy; higher thresholds meant I could be more certain that the action had been *intentionally* triggered, but would require more extreme (and therefore more fatiguing, and slower to execute) articulation. Conversely, an action with a lower threshold could be triggered much quicker and with less effort, but with an increased risk that the expression could be misread, and the action could trigger unintentionally.

With this in mind I selected higher and lower thresholds for different actions based on their nature. For example, for the dodge roll action, I chose a very low threshold because:

(1) The effectiveness of the move relies heavily on timing, so responsiveness is crucial.
(2) In most cases and unintended dodge roll will not be of significant harm to the player.

On the other hand, I would have chosen a high score threshold for the healing action because:

(1) Healing leaves the character vulnerable for a few seconds.
(2) Healing consumes limited healing resources, which should be used with purpose.

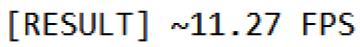
## Using delay

Some gameplay actions can only be triggered with specific timing. For example, dodge rolling can only be performed by tapping the dodge button–in other words, pressing, then releasing less than a couple of hundred ms after; pressing and releasing the button on the same frame does not register, nor does holding the button down. To solve this problem I used threading. I create a new thread temporarily, on which I would press the key, wait, and then release the key. By delegating this to a separate thread I can have it execute asynchronously without blocking image processing on the main thread.

## Optimization

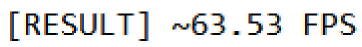
I used threading to boost the FPS of the program. I created a separate thread just to poll the camera each frame while the main thread dealt with processing the current frame. The framerate went from ~20 FPS to ~60 FPS from threading alone. I also experimented with minor optimisations like resizing the webcam frames before processing (using nearest interpolation for speed) and converting opencv matrices to np.arrays before passing them to functions.

*Pre-optimisation*



```
[RESULT] ~11.27 FPS
```

*Post-optimisation*



```
[RESULT] ~63.53 FPS
```

## Reflection and Future Development

While the concept is novel, the result was frustrating, unresponsive, and unreliable. This came largely down to a bottleneck in the processing of the images on my CPU. Moving forward I'd like to explore some other landmark detection and pose estimation APIs and, crucially, I'd like to get them to make use of GPU acceleration. While working on this project I discovered that MediaPipe's Python API, specifically, does support GPU inference, while the JavaScript and mobile APIs do. Unfortunately this discovery came late and I did not have enough time left to port my code.

Moving forward, mobile in particular could be an interesting platform to explore because it would allow me to offload the processing to another device, and then send relevant data (landmarks, blendshapes) to a separate app on my PC that would then execute the keypresses. Offloading the heavy computer vision tasks to a separate device would free up my computer to be able to run more resource demanding games, and mobile phones would be an ideal device to use because of their accessibility.

I would like also like to invest in a USB webcam–I don't have one so I've been developing this project using my phone's camera, streamed over WiFi to my PC. I expect a USB webcam to have lower latency.

I also found that *just* using my head and face was not practical–there aren't enough distinct and easy to articulate expressions. MediaPipe has a 'holistic' landmarker available that combines face, pose, and hand landmarkers, which could open up more possibilities for control, but the benchmarks show that it comes at a performance cost. I'd like to experiment with it regardless–I can imagine some fun interactions, like holding your fists up to enter combat (lock-on).

The binary inputs of a keyboard mapped to the granular motion of a face felt unintuitive, even after practice. It would have been nice to map the head orientation to smooth character motion so that turning your head to different degrees would result in a range of walking speeds. Dark Souls III already supports this functionality, but the binary nature of a keyboard does not permit these granular inputs. I would need to look into Gamepad emulation to achieve this.

## References

BoppreH (2024) ‘keyboard’. Available at: https://github.com/boppreh/keyboard (Accessed: 15 January 2024).

*Challenge run* (2023) *Pikipedia*. Available at: https://www.pikminwiki.com/Challenge_run (Accessed: 13 January 2024).

*Dark Souls 3 - Banana Controller Full Run* (2020). Available at: https://www.youtube.com/watch?v=-jayN_X9p6Y (Accessed: 13 January 2024).

‘Dark Souls III’ (2016). FromSoftware Inc.

‘Face Landmark Detection Solution’ (no date). MediaPipe Studio. Available at: https://mediapipe-studio.webapps.google.com/studio/demo/face_landmarker (Accessed: 13 January 2024).

*How I Play Elden Ring With My Mind (EEG)* (2023). Available at: https://www.youtube.com/watch?v=rIbfNUA5pWk (Accessed: 13 January 2024).

‘imutils’ (2024). PyImageSearch. Available at: https://github.com/PyImageSearch/imutils (Accessed: 15 January 2024).

‘Jupyter Notebook’ (no date). Available at: https://jupyter.org (Accessed: 15 January 2024).

Kamen, M. (2015) ‘“Dark Souls” superfan beats game using only voice’, *Wired UK*, 8 December. Available at: https://www.wired.co.uk/article/dark-souls-voice-control-fan-hack (Accessed: 13 January 2024).

Karyal, P. (no date) *This Gamer Turned EEG Tech Into a Game Controller - IEEE Spectrum*. Available at: https://spectrum.ieee.org/elden-ring-hands-free-controller (Accessed: 13 January 2024).

‘MediaPipe’ (no date). Google. Available at: https://developers.google.com/mediapipe (Accessed: 15 January 2024).

*Nameless King on dance pad by Luality* (2018). Available at: https://www.youtube.com/watch?v=UxKsRf6R-vc (Accessed: 13 January 2024).

‘opencv’ (2024). OpenCV. Available at: https://github.com/opencv/opencv (Accessed: 15 January 2024).

Rosebrock, A. (2015) 'Increasing webcam FPS with Python and OpenCV', *PyImageSearch*, 21 December. Available at: https://pyimagesearch.com/2015/12/21/increasing-webcam-fps-with-python-and-opencv/ (Accessed: 15 January 2024).

Thierfelder, S. (2023) 'Head Pose Estimation with MediaPipe and OpenCV in Javascript', *Medium*, 21 February. Available at: https://medium.com/@susanne.thierfelder/head-pose-estimation-with-mediapipe-and-opencv-in-javascript-c87980df3acb (Accessed: 15 January 2024).