CursorVision:

System Overview,
High Level Software Architecture,
& Ethical Considerations/
Mitigation Strategies

01/16/2026
Software Engineering

By: Kody Graham
For: Md Jobair Hossain Faruk

System Overview

CursorVision is a conceptual accessibility focused software system that is designed to allow for hands free control of one's computer, using real time eye movement and blink detection. The system will allow users to control the cursor movement and left/right mouse clicks using only a standard web cam, after initial setup.

The motivation for CursorVision comes from the widespread accessibility challenge that many individuals face due to either temporary or permanent loss in their ability to use their arms. I have personally had three different family members who, due to extreme injuries, have not been able to use either of their arms in two cases for four+ months and in one case permanently. Whatever the reason, there are no doubt millions of people worldwide who may be unable to interact with traditional input devices like a mouse or keyboard. CursorVision offers a **non-invasive**, software-only solution that will run locally on users' hardware, which will make it more accessible and affordable than many existing technologies. Emphasis on non-invasive because the only other options that come to mind achieving this same goal are external hardware based and must be installed on the user; Neuralink or MouthPad (a retainer that allows you to use your tongue on the roof of your mouth to control cursor).

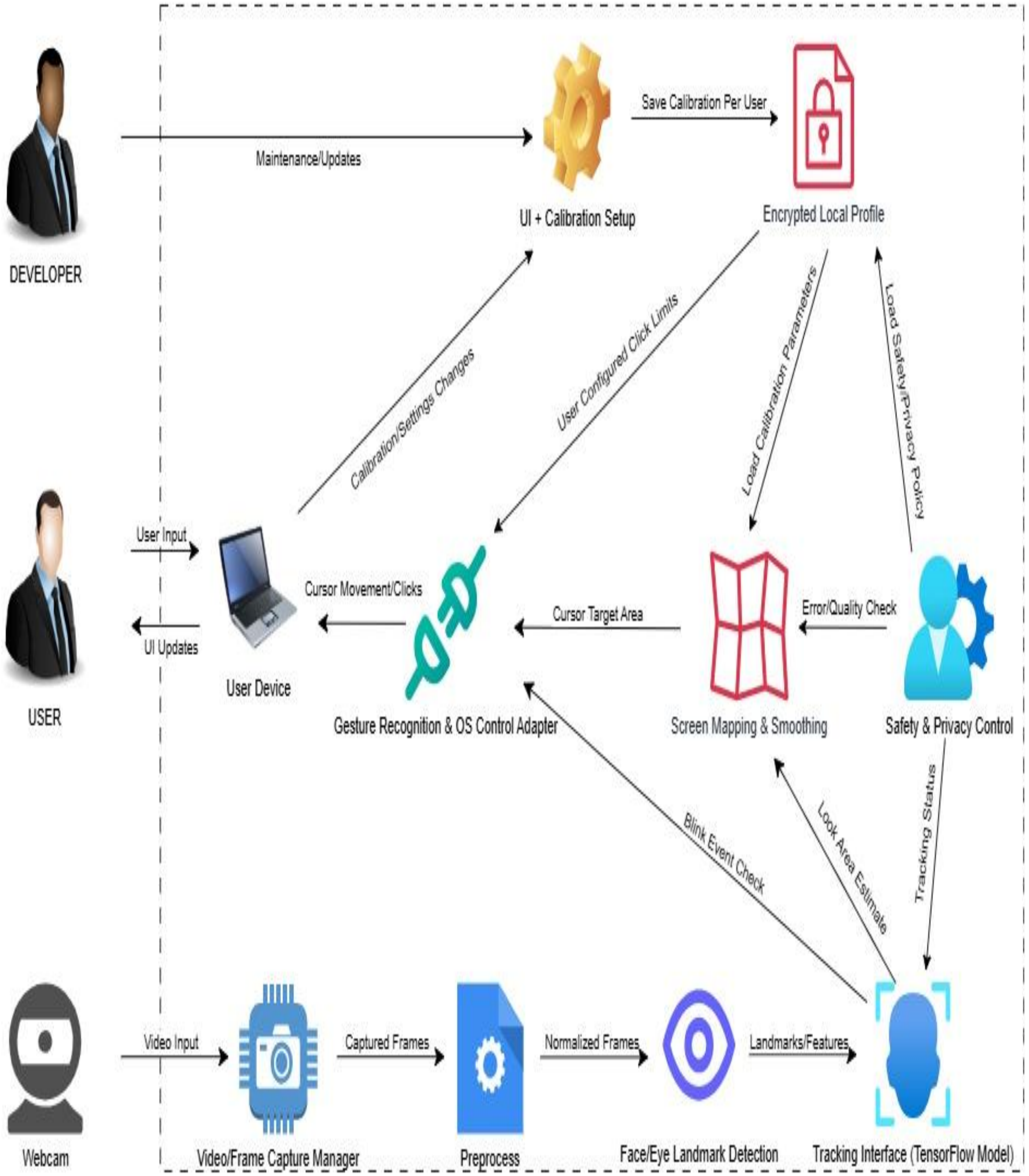
From a software engineering standpoint, CursorVision presents several real-world challenges. These challenges include real time performance, camera variability (mainly different lighting/environments and the quality of the webcam), the accuracy and stability tradeoffs, and even some ethical concerns about privacy (which I address by keeping everything local). The system is designed to be class oriented so that each individual aspect of the system can be independently improved or rolled back while protecting the system as a whole.

CursorVision is intended to operate entirely on the local machine of the user and will not require any cloud connection. All the model training, processing, unique user facial calibration data, and mouse control happen locally to reduce the privacy risk to near zero. A bonus is this will also improve responsiveness while CursorVision is running by not introducing any additional latency. In short, keeping everything local should strengthen privacy and increase responsiveness.

Architecture Diagram

Taking your advice from class, I decided to design the system in a very compartmentalized way so that each aspect of the system can be individually updated or rolled back. This kind of layered approach will allow me to separate concerns between input capture, computer vision processing/mapping, machine learning/calibration, and system controls. The data follows a simple path from sensor input to interacting with the operating system. This simple path should make debugging easier and ensure very predictable behavior when implementing the system.

CursorVision
(Local Desktop App)



Note: All video processing and calibration storage occur locally on the user's device. This is to prevent raw frames or personal data from being sent off of the device.

Key Components

Action	Reaction
Eye Movement (Up, Down, Left, Right)	Cursor (Up, Down, Left, Right)
Left Eye Wink (1x)	Left Mouse Click (1x)
Right Eye Wink (1x)	Right Mouse Click (1x)
Both Eyes Blink (1x), Blink Fast (3x)	1x=Ignore, 3x=Pause Cursor/Click Actions

* I know this probably seems a little odd, all the left/right wink and blink fast stuff, but I have seen firsthand how much someone's life is turned upside down when they lose the ability to use their arms and while this can seem like a small or insignificant thing, anything that allows someone in that situation to have self-controlled interactions can mean a lot to them.

The way I have visualized it, the system will operate like this:

1. The user is in front of a computer with a standard webcam (for the best mapping I would suggest a built-in webcam or one that is close to centered on the screen for the most uniform mapping).
2. The webcam will continuously capture live frames in the video/frame capture module.
3. The computer vision (preprocess and Face/Eye Landmark Detection) module will process each frame to detect eye direction and blink states.
4. That processed data will then be passed to the Tracking Interface module where my TensorFlow model will estimate the users' look direction and whether they are winking/blinking.
5. The UI Calibration module will provide training data for look and blink states and send this data to the Encrypted local Profile module which will encrypt/decrypt and store the specific user calibration data.
6. The safety and privacy module will provide validation for the accuracy of predictions made by the TensorFlow model before being sent to the last module.
7. Finally, the Gesture Recognition and OS Control module will convert the predicted user's intent into operating system level cursor controls, undoubtingly using OS APIs. Likely pyautogui as it will allow me, depending on how accurate I can get the eye tracking, to be able snap to locations on the screen when users look there, ultra responsiveness if the user so chooses, like Neuralink allows. This will not likely be a feature in the initial version of CursorVision but with the developer being able to introduce optional updates through the UI and Calibration Module, we can add that cursor jump functionality as a later optional feature.

To achieve the steps mentioned above for a seamless user to system interaction, each component will have to be designed carefully but should also be somewhat flexible. While that seems contradictory, this is the start of the software development process and as I have seen on many projects I have worked on, sometimes the way we have it in our head at the start of a project does not translate to the best/most efficient way to accomplish our overall goal once we actually start writing code. With that in mind, here is an outline of how I think structuring my key components to start makes the most sense:

1. Video/Frame Capture Manager Layer
   - This layer is where the continuous live frames from the user's webcam are captured.

2. Preprocessing Layer
   - This layer normalizes frames to reduce variability that could be caused by several factors, camera quality, or lighting conditions for instance. Doing this early will help with consistency before we train our agent on what to look for.

3. Face/Eye Landmark Detection Extraction Layer
    - This is where the user's face will be detected and key eye and facial landmarks from each frame will be saved. These will provide a structure for my model to base its decisions on.

4. Tracking Interface Layer
    - Here the TensorFlow based regression model will estimate the look intent from the extracted features. These estimates will be used to predict the direction the user wants to move the cursor. Not necessarily the exact spot on the screen in the initial build but the direction up/down/left/right.

5. Calibration & User Configurations Layer
    - This layer will manage the per user calibrations. It will collect samples in a similar fashion to how you calibrate a touch screen. Just look at different highlighted spots on the screen so the system can build a prediction map for what it looks like when you are looking at distinct locations of the screen.

6. Screen Map & Smoothing of Cursor Control
    - This is where the actual conversion from look intent into cursor movement on screen will happen. There will also have to be some filtering done in this layer to smooth the cursor movement. I am not sure exactly how yet, but it will happen in this layer for sure.

7. Gesture Recognition and OS Control Adapter Layer
    - Here the winks or blinks will be translated into OS events, click left/right. The cursor control will ultimately go through this layer as well so it may make the most sense to implement it here too; I have not had to work with cursor movement as an output before so we shall see.

8. Safety and Privacy Control Layer
    - Finally in this layer, we will enforce system wide safeguards. For instance, check prediction confidence and more importantly ensure that sensitive calibration or user data is encrypted and kept local. We will check for errors in all layers, so the checks happen and can be addressed before leaving a module. This means there is not a true error checking layer, but we should have an error log stored in this Safety Layer.

Ethical Considerations & Mitigation Strategies

CursorVision raises many ethical considerations because of its reliance on continuous input from the web cam. Facial imaging and eye tracking data is inherently sensitive so improper handling could compromise the user's privacy. To mitigate privacy risks, CursorVision is designed to operate entirely on the user's local machine. No video frames or facial images are sent or stored off the user's system. All calibration data and trained models are also stored locally and should, for an added layer of security, be encrypted.

Safety concerns about inaccurate intent interpretation causing clicks on unintended areas that have consequences, on Amazons one tap purchase button for instance, should also be addressed. The simplest solution I feel is to just have a threshold for acceptable prediction confidence to approve an action like click. In addition, if camera input is interrupted the system should enter a safe mode and prompt for recalibration to prevent any potential unexpected cursor actions.

Bias and fairness concerns can also come into play on a system that relies on AI and camera vision if trained on disproportionate group data. For instance, if the model was pretrained on a dataset of images featuring only men, it is likely that the facial patterns/landmarks it learned will translate poorly for users that are women. These biases are mitigated in my design of CursorVision by per user calibration and model training. Rather than relying on a generalized model that may perform poorly across different demographics of users, CursorVision will train a dedicated model for each user.

Finally, the ethical consideration of the user remaining in control of the system is especially important. AI is becoming more mainstream but there are case studies where AI has overstepped or tried to find a work around to achieve a goal other than the users/developers specified one. This may give some users a valid sense of worry when AI is involved in something they use. However, you can rest assured that with CursorVision the AI is strictly limited to cursor movement and click intent translations. This AI will be far from complex enough to try to overstep any boundaries. The user has full control over configuration, calibration, and system activation.

All these ethical safeguards are embedded directly into the system architecture of CursorVision rather than trying to figure out what should be considered ethically after the software is built.