

Project Dataface:

Democratizing Facial Recognition with Google Glass

Jeremy Krach

[jakrach@umd.edu](mailto:jakrach@umd.edu)

University of Maryland, College Park

### Abstract

Lightweight and camera-equipped wearable devices such as Android-backed Google Glass—with their potential for wide-spread and mobile data capture—have piqued the imagination of technologists and privacy advocates alike. This paper describes an experimental system which confirms the feasibility of such devices for surveillance through live data collection and facial recognition. Furthermore, even though effective surveillance tasks are computationally demanding, this work illustrates that performance of such systems is scalable through careful architecting of communication between static servers and mobile collection devices. When the bulk of the complexity can be offloaded to the server, and with the availability of highly-available communication channels between collector and processor, we have the foundation upon which future surveillance systems might be constructed. Such systems awaken nightmares for those advocating privacy of the modern citizen, while inspiring innovators to push forward the bounds of what can be accomplished with today's technology. The present project enables advocates from both ends of the spectrum to debate privacy policy as it can be seen through the lens of systems that are possible today.

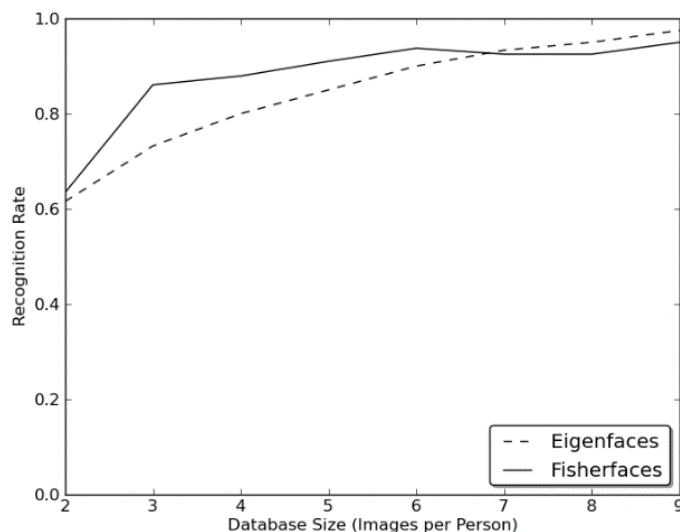
*Keywords:* Google Glass, mobile, scalable, privacy, surveillance, facial recognition

## Project Dataface:

## Democratizing Facial Recognition with Google Glass

Imagine yourself as a college student. As you walk from class to class, you see many faces that you are not familiar with on a day-to-day basis. Once in a while you recognize a face, but do not know the person. Although you have never met, you encounter this face frequently on your walk to class. Later that night, while browsing a social network, you see the familiar face again. Now you have access to personal information about that individual: perhaps their name, shared contacts, and interests. Even if you have never met the person, you certainly have seen their face many times without knowing any details of their life.

This project evaluated key features of mobile devices like Google Glass to do real-time capture of identity information on a daily or ongoing basis. Now return to the previous example,



**Figure 1:** Accuracy of two facial recognition algorithms based on database size. **Credit:** OpenCV.org

but add Google Glass to the mix. As you walk around your college campus or commute to work via the subway, you capture facial images of individuals you see using your wearable device. Every day you increase your dataset of facial images and every repeated individual strengthens their distinct facial profile. OpenCV, a popular open-

source image processing library, recommends approximately seven images of an individual in order to exceed a ninety-percent recognition rate (OpenCV Development Team, 2014). This is

represented graphically in figure 1. Assuming you collect data somewhere you traverse in day-to-day life, individuals may have a complete facial profile within a week (at best). Once you have collected enough pictures of a certain individual, your database server scans social networks for facial matches located in roughly the same area you collected the images (using GPS information embedded in the collected photos). Eventually, you may find a match and obtain a name or other personal information from sources like Facebook and Twitter.

Today these capabilities exist. With the ever increasing popularity of wearable devices, more and more exciting applications of technologies are being implemented in a wearable, and highly mobile, form. Since Google unveiled its Glass prototype in 2012, the gadget world has debated the threat to privacy such devices would offer (Newman, 2013). Facial recognition, in a similar sense, is becoming bigger and better as the years progress. In 2014, Facebook researchers published a paper unleashing a new state-of-the-art in facial recognition and reached levels of accuracy on-par with a human's facial recognition ability (Taigman, 2014, p. 1). Recognition databases such as CreepShield are beginning to crop up for public use (*CreepShield*, 2014), and many Americans fear the FBI, NSA, and local police implementing facial recognition as a tool to assist in law enforcement (Volz, 2014). Today, the platforms of wearable technology, facial recognition, and big data are beginning to blend, with Dubai already issuing Google Glass to its detectives (Maclean, 2014).

While the potential for surveillance via tools like Google Glass has always been known, there has also been a question of its scalability. Most substantive surveillance activities can be computationally demanding, yet the power of mobile platforms themselves remains relatively limited. The present project answers that question and shows how even though the mobile-side power is limited, substantive surveillance tasks are in fact practical in the presence of highly

available communication channels that allow the computation to be algorithmically split with a server. By reversing the design of a traditional approach to facial recognition, our design confirms that the capabilities of lightweight wearable devices are not only practical, but also scalable in a widespread surveillance context. Instead of collecting data online and attempting to bring it down to life (e.g. harvesting Facebook for images to use in a live facial recognition system), here we explore the opposite idea of collecting live data to then be applied to online realms at a later time. In addition, while a mobile device like Google Glass runs the collection, it immediately streams the images to a server over the internet in order to reduce its processing load. In this way, the facial recognition aspects become as scalable and unlimited as any existing methodologies that currently exist, since none of the heavy computational strain is put on the collection device. Instead, any processing of the images would occur (in this model) on a fully configurable backend of the user's choice.

The collection frontend for Google Glass described here utilizes readily available open source software. Without much extra assistance, an interested citizen could use the collection system described here to setup a custom surveillance apparatus using existing facial recognition libraries. The shallow learning curve required to apply this research inspires the idea of democratized surveillance: any citizen could surveil the public domain. This paper primarily explores how to confirm and maximize Google Glass' utility for this collection procedure, with some asides as to how it could eventually tie in to a larger process.

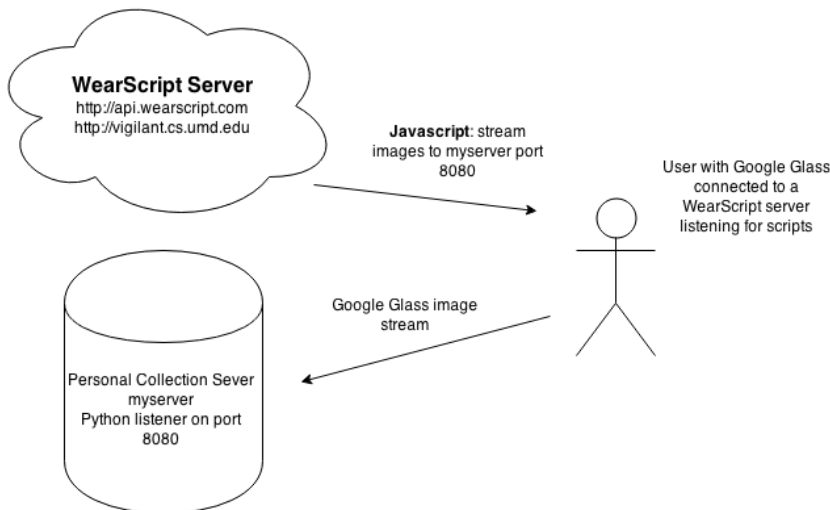
### **Design and Methodology**

Facial recognition depends heavily on data. While there are other projects pursuing facial recognition in real-time with Glass (*NameTag*, 2014), this project builds the large dataset on which existing facial recognition capabilities can be stacked. As mentioned before, this design

choice enables a scalable apparatus and a “plug-and-play” backend for processing the raw images collected. As a result, the primary goals of this project are as follows: 1) create a Glass application that easily interfaces with a storage backend, 2) ensure the application maximizes battery life to minimize data loss, 3) guarantee high-quality images on which both facial detection and recognition post-processing can be applied, and 4) the setup of the application and collection system must be straightforward enough that a typical citizen could configure their own collection system using Google Glass and the tactics explored in this paper.

In order to interface easily with a storage backend, this project utilizes Brandyn White’s WearScript application. White designed WearScript to allow rapid prototyping of Google Glass applications in Javascript. WearScript allows developers to “go from concept to demo in a fraction of the time,” since Javascript has a smaller learning curve than Android development—Glass’ traditional method for creating applications (*WearScript*, 2014). WearScript itself includes many projects: a general purpose server to send Javascript to Glass, various lightweight listener services (to subscribe to specific sensors for specific purposes), and the Glass application itself. To use the application, one must install the app’s APK using a short script found on the WearScript webpage. Once installed, the app will prompt the user to navigate to a WearScript server (<https://api.wearscript.com> or <http://vigilant.cs.umd.edu>) and link Glass to the server using a QR reader. Once linked, Javascript can be edited on the WearScript server and sent to Glass with a key shortcut.

WearScript has built in capabilities to stream a live image on a fixed interval. In order to make use of this feature, we tweaked an existing Python listener found on the WearScript



**Figure 2:** *The designed collection process.*

GitHub page. The Python program, in its original form, subscribed to the image stream from Glass and displayed meta-data about the images. To serve our purposes, we manipulated this listener service to instead save each image in its entirety.

Our listener opens a web socket, which listens for connections from Google Glass directly. Once connected, the socket subscribes to Glass' image stream and places each image in an arbitrary folder on the server. The listener needs the WearScript-Python library, but itself is a lightweight 20 line Python script. Figure 2 represents the designed process.

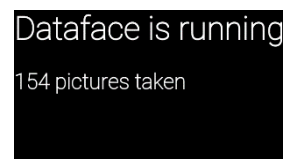
Once WearScript enabled the collection of images, battery usage and image quality became the next two concerns. In order to experiment with these variables, several tests were run using different Glass display interfaces and stream intervals. The maximum resolution that can be streamed from Glass' camera using WearScript is 1920 x 1080, which is more than high enough quality for any potential back-ends doing facial recognition or detection. All tests used this resolution in order to ensure that any post-processing of the images collected could be implemented.

Four experiments were designed to determine the application format to enable the longest battery life and the most images captured reliably.

Experiment 1 streamed images at one image per second. The Glass interface displayed each image captured in this experiment. Experiment 2 relaxed the stream speed to one image every five seconds, but maintained the same

interface as experiment 1. Experiment 3 maintained the five second speed of experiment 2, but displayed the interface seen in figure 3. Finally, experiment 4 streamed one image per four seconds. The breakdown of experiments (and their results) is summarized in figure 4.

Each experiment, with the exception of experiment 2, started with Glass unplugged but fully charged. Experiment 2 started at 50%, in order to see how Glass would perform starting without a full charge. The experiments were run until Glass either disconnected and could no longer reconnect, or completely drained its battery. The experiment runtime was measured as such. The majority of these experiments were conducted standing still while connected to a relatively strong Wi-Fi network. Experiments with tethering to a 4G-enabled smartphone were not conducted. While connectivity certainly would influence data retention, this preliminary research indicates performance under more ideal circumstances, which gives insight into which application format should yield best retention independent of connectivity and dependent on the controlled application variables: stream speed and Glass interface design.



**Figure 3:** The interface as seen on Glass for experiments 3 and 4.

## Results

### Experiment 1

Experiment 1 ran for a total of 38 minutes and 25 seconds. Collection ran for approximately nine minutes uninterrupted, at which point the Glass exited the application and returned to the home screen. Glass had heated up significantly. Upon reconnection, the



application ran for another eight minutes until it disconnected again. This process repeated itself twice more (respectively eight and ten minutes), until the battery was below 20% and Glass was unresponsive from overheating. Capturing 1 image per second, Glass was expected to capture 2305 images in the duration of the experiment, but instead only captured 451 images. The stream lost 80% of the images.

Experiment	Stream Speed	Interface on Glass	Runtime	Images captured	Expected	Data loss
1	1 second	Image stream	38:25	451	2305	80.43%
2	5 seconds	Image stream	27:00*	274	324	15.43%
3	5 seconds	Text counter	50:44	591	608	2.79%
4	4 seconds	Text counter	53:23	754	800	5.75%

**Figure 4:** Displays the parameters of each experiment and their results.

*Stream Speed is defined as the time between each image.*

*\*Experiment 2 was run starting at 50% battery*

### Experiment 2

Experiment 2 started after recharging Glass up to 50% of its battery's capacity. In this case, Glass ran for 27:00 minutes before completely draining its battery. As in experiment 1, significant overheating remained a problem. Glass disconnected once at the 15 minute mark, and continued uninterrupted until the end from that point forward. Glass was expected to capture 324 images in this trial, but instead captured 274 images, a 15% rate of data loss. This rate is a significant improvement over the previous experiment.

### Experiment 3

Experiment 3 also used the conservative 5 seconds per image stream speed. However, this experiment displayed a text counter, rather than rendering the image stream. This effort was taken in order to reduce overheating and the drain on battery life. Glass ran this experiment uninterrupted for 50 minutes before shutting down due to a completely drained battery. Glass captured 591 images of 608, approximately a 3% data loss, the best rate found in this project.

## **Experiment 4**

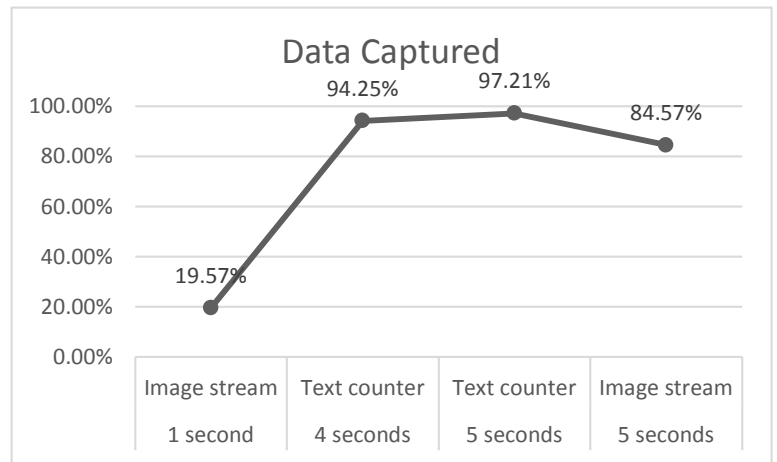
Experiment 4 shifted the 5 second stream speed down to 4 seconds per image. This experiment used the same interface developed for experiment 3 and had a similar (and slightly longer) battery life of 53 minutes. Glass only collected 754 pictures of the expected 800, in this case a 6% data loss. However, Glass did not overheat and ran until a complete battery drain, just as in experiment 3.

## **Facial Detection**

Once the above experiments completed, an initial facial detection test was run on a sample collected from the image stream. This test was designed in order to verify the third goal of this project: that the images received maintain a high enough quality to perform facial recognition. With a sample of six collected images, twenty matches for faces were found. Twelve (60%) of the detected faces were false positives and the other eight (40%) were true positives. Four faces were missed by the detection algorithm, or 33% of the total (12) faces in the sample. While better algorithms exist for detecting faces, this experiment met the limited capacity for this project and shows that the image quality is sufficient for most purposes. There may be a concern when exposing an entire collected set (50 minutes worth) to a facial detection algorithm. Many of the captured images will contain no faces, and the false positive rate will likely be extremely high. However, this issue somewhat dissipates when moving to facial recognition, since a match (or several) would be required before drawing any conclusions as to whether a given person had been seen multiple times or not.

**Analysis**

The above results verify some important hypotheses about Glass’ use for image collection in a large-scale surveillance system. First, the current hardware on Glass appears to be incapable of supporting a 1 image per second stream of images. The device overheats too rapidly, causing disconnection and sporadic shutdowns, even when Glass has substantial battery remaining. A more conservative stream speed of 4 or 5 seconds appears to prevent overheating, with minor data loss (see figure 5). The data loss here could possibly be mitigated by implementing a parallel listener in order to be saving images on their own threads. However, without considering such an alternative, the best option would be to use a minimalistic interface and stream at a rate of 5 seconds per image. This collection rate



**Figure 5:** Shows the data retention rates for each of the four experiments run.

yields a large amount of images when captured over the course of the expected runtime. Returning to the first goal of this project (implementing a Glass application that interfaces with a storage backend), these results show that the application developed within this project builds a sufficient dataset on a storage backend (rather than Glass itself).

Second, the battery life does not appear to extend beyond 1 hour. This is consistent with most online reports of Google Glass’ battery life. While many users found with moderate usage it could last 3-5 hours, many testers found with heavy usage such as constant image collecting, application use, or video capture, the battery would frequently last between 30 and 60 minutes. Given Glass is still a prototype, the hope is that as more consumer-friendly products hit the

market, their battery life could extend even longer. However, the fact that this application can last so near the maximum expected length of a heavy usage estimate shows that the restriction is on Glass' hardware, not the application itself. This meets the second goal of maximizing battery life.

Collecting images in this way can be extremely noisy. In a very restricted sample of images, a strong majority of facial detections were false positives. With facial recognition's stricter requirements, the false positive rate would be expected to decrease. However, improperly discovering faces at such a high rate yields a large amount of noise, especially if facial detection builds the dataset for facial recognition. Potential solutions include making the Glass application smarter by taking a picture every 5 seconds if a face is present. However, the increased complexity of the application is bound to drain battery life at a faster rate and increase potential for overheating. However, the fact that 1920 x 1080 resolution images stream reliably meets the third goal of this project: high quality images sufficient for a facial detection or recognition backend.

Finally, the implementation of this system meets our fourth goal of ease-of-use. The server that listens for Glass' image stream requires little configuration: a single Python script with around twenty lines of code and a small library to interface with WearScript are the only files that require any user-interaction. The process of setup could easily be scripted, but is simple enough most people familiar with the installation of a web server would have no trouble. Once the server is running, the user only needs install WearScript on Glass and connect the device to a WearScript server (<https://api.wearscript.com> or <http://vigilant.cs.umd.edu>). Using the Javascript application at <https://gist.github.com/jakrach/38583e9e17d73ac49f4c>, the user

could then stream images to their server by changing one line to tell Glass where the server exists.

While it may not necessarily be news that Google Glass can support image collection with a battery life of roughly 50 minutes, this project solidifies some important objectives. First, the design of this collection scheme shows a scalable method for collecting large databases of facial data on an ongoing or daily basis. Further, the design shows the capabilities to customize the backend platform to work for an individual's needs. Next, this experiment formally records performance and life-expectancy of Google Glass in a real collection scenario, establishing a gauge on what can be expected out of wearable technologies used for similar purposes in the coming years. Finally, this project shows that large-scale facial recognition or surveillance systems are feasible with today's technology. This final observation opens the floor for debate between policy makers and privacy advocates as to the potential benefits and costs of adopting such a technology.

### **Future Work**

This paper explored the practicality of using Google Glass as a collection tool for a democratized surveillance system. In these experiments, Glass operated as a dummy collector without receiving any information back from the server. There are many extensions of this project and much work yet to be done. Areas to continue work on include everything from further proving and exhibiting Google Glass' usefulness for the collection of large facial datasets to building a backend that is compatible with the collector built for this project. More specific extensions are explored below.

First, experiments must be run using a 4G connection tethered to a smartphone, rather than a stable Wi-Fi connection, as the primary purpose of using Glass in this case is its mobility.

Second, attempting facial recognition on a dataset obtained from Glass' stream would give insight into the capabilities of a democratized surveillance system using large datasets from various citizens. Third, creating a smarter application for Glass that only captures images if certain conditions are met, such as favorable lighting or the presence of faces, would enable a more interactive surveillance system. Finally, it would be useful to attempt to have the server notify Glass upon recognizing a familiar face and state how many times that person has been seen before. Once a face had been seen a certain amount of times and a profile has been built of that given person with fairly high confidence, it would be useful to see if a backend could tie into social media and crawl publicly available information for a match. This leads even deeper into the idea of democratizing surveillance, and would allow for a recognition system of standard citizens in the capacity seen with the "NameTag" app—which seems to use the reversed logic of building the dataset from online sources (*NameTag*, 2014).

On the policy side, there is room for work on the definition of privacy. By stepping outside your home, do you consent to have your image collected by any stranger with a camera? Is there a difference between being caught in the background of a tourist's photograph and having your face captured and catalogued for recognition purposes by a fellow citizen? In the modern era, society must define where the line exists so that technologies that live on the fringe, like Google Glass, can be readily accepted. Early in Google's campaign there were reports of violent insults and even physical attacks on users of Glass. Around this new technology, many people felt their privacy threatened. With this paper as evidence to the systems and capabilities we can create, there must be discussions and research into how these new technologies will be accepted, if at all.

### **Conclusion**

Google Glass, due to its discreet profile and mobile nature, is an ideal candidate for creating a democratized facial recognition system. In this study, an attempt was made to determine the practicality of such a device in building a large-scale surveillance system. By flipping a traditional model for such systems, this research showed a scalable design that would utilize mobile platforms like Google Glass to collect live data and send it to a much more configurable server backend. Although issues such as battery life, network connectivity, and noisy image data present potential issues for such a system, Glass certainly has the capabilities to create a democratized surveillance system. By using the open source WearScript project, one can easily customize a listener to collect images at any given interval and completely customize the interface on Google Glass during collection. The system built for this project is extremely lightweight, and would only require a small Python listener to run on a public facing server. This process could easily be packaged more formally, or scripted due to its simplicity. This would enable any citizen, no matter their experience with technology, to pick up Glass and assist in the movement of democratized surveillance.

## References

- CreepShield.com - Facial Recognition for Online Dating. (2014). *CreepShield*. Retrieved February 25, 2015, from <http://www.creepshield.com/>.
- Maclean, W. (2014, October 2). Dubai detectives to get Google Glass to fight crime. *Reuters*. Retrieved February 25, 2015, from <http://www.reuters.com/article/2014/10/02/us-emirates-dubai-google-police-idUSKCN0HR0W320141002>.
- Newman, J. (2013, May 2). The Real Privacy Implications of Google Glass. Retrieved February 25, 2015, from <http://techland.time.com/2013/05/02/the-real-privacy-implications-of-google-glass/>.
- OpenCV Development Team. Face Recognition with OpenCV. (2014, April 21). *OpenCV*. Retrieved February 25, 2015, from [http://docs.opencv.org/modules/contrib/doc/facerec/facerec\\_tutorial.html](http://docs.opencv.org/modules/contrib/doc/facerec/facerec_tutorial.html).
- Taijman, Y., Yang, M., Ranzato, M., & Wolf, L. (2014). DeepFace: Closing the Gap to Human-Level Performance in Face Verification. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Volz, D. (2014, September 15). FBI's Facial-Recognition Technology Has Achieved 'Full Operational Capability.' *National Journal*. Retrieved February 25, 2015, from <http://www.nationaljournal.com/tech/fbi-s-facial-recognition-technology-has-achieved-full-operational-capability-20140915>.
- WearScript Source. (2014). Retrieved February 25, 2015, from <https://github.com/wearscript>.
- White, Brandyn. WearScript: JS with Batteries Included for Glass. (2014). *WearScript*. Retrieved February 25, 2015, from <http://www.wearscript.com/en/latest/index.html#>.



With NameTag, Your Photo Shares You. (2014). *NameTag*. Retrieved February 25, 2015, from <http://www.nametag.ws/>.