# Artificial Intelligence Gaming Assistant for Google Glass

Scott Bouloutian[1], Edward Kim[1,2]

[1]Department of Computer Science, The College of New Jersey, Ewing, NJ
[2]Department of Computing Sciences, Villanova University, Villanova, PA

**Abstract.** The interaction between humans and computers has traditionally been through the medium of desktop computing. However, in recent years, an alternative computing concept known as ubiquitous computing is ushering in various wearable computing technologies, such as Google Glass. These technologies enable even more immediate ways to share and access information. Our research seeks to explore novel methods in which these wearable technologies can be combined with more powerful computing techniques to compute useful context-specific information. The scope of this research is in utilizing Google Glass to act as an artificially intelligent game assistant. The approach of this work is to make use of Google's Mirror API to build a web-based service to interact with Glass. The Mirror API is used to share an image from Glass to a web-based service where the image is processed and key features are extracted. An appropriate algorithm is then used to compute a near-optimal game move depending on the game being played. The results are promising, and the Glassware that was implemented suggests appropriate moves while playing a game of Connect Four. Our results foreshadow what is possible when wearable technology is combined with artificially intelligent computation in the cloud.

## 1 Introduction

Ubiquitous computing is the idea that computing technology is everywhere, in any location, and in any device. New innovations have brought wearable technology into everyday society. By their very nature, these devices are designed to be small and lightweight, at the expense of longer battery life and faster processing capability. Also by their very nature, these devices may be expected to provide useful context sensitive information about one's environment in everyday life, which can be accomplished by different avenues of computer science, including computer vision and artificial intelligence. Depending on the application, these algorithms require greater computing power than can be handled reasonably on a wearable device. Our contributions are two fold. First, we describe the integration of Google Glass with a computing cloud infrastructure. Second, we demonstrate the increase in speed and usability that the cloud provides for heavy calculation processes (such as computer vision algorithms). These contributions have the power to deliver powerful context sensitive information to wearable devices such as Google Glass.

## 2 Background

The concept of presenting augmented computer feedback to a user of Google Glass is becoming more popular as computing becomes more pervasive. There have been other projects attempting to combine Google Glass with modern computer vision techniques. Google Glass, as well as some of the projects utilizing it will be outlined below and discussed with the target of a game assistant in mind.

### 2.1 Google Glass

Announced by Google X in 2012, Google Glass is a wearable computer with an optical head-mounted display in the shape of a pair of glasses. The project is currently in development by Google and the Glass used is part of the Glass Explorer Program.

### 2.2 Google Mirror API

The Google Mirror API [1] allows you to build web-based services that interact with Google Glass. It provides this functionality over a cloud-based API and does not require running code on Glass. A core component of Glass' software is called the time-line. The time-line is a chronological list of cards which you can swipe through. These cards contain information such as your emails, alerts, and photos. This API allows you to interact with this time-line by reading and inserting various cards.

### 2.3 Google Glass Playground

Google Glass Playground [2] is a project which combines Google Glass with the OpenCV computer vision library [3]. It functions as a test of the possibilities of combining these two technologies. Rather than using Google's Mirror API, the Google Glass Playground project uses the Glass Development Kit, or the GDK to implement its application. The consequence of this is that all of the code must be run locally on Glass, including the OpenCV library. A typical OpenCV demo such as face detection for example, while performing admirably well for an application running on Glass, ran at around 0.7 frames per second. Additionally, a noticeable heat output was emanating from the side of the device. This was something that was not desirable for a game assistant application, especially since the processing needed was greater than that of a face detection algorithm.

### 2.4 Cognitive Assistance

Glass has been used to assist with cognitive impairment [4]. Ha et al. [5] explored ways in which Google Glass can be used to perform real-time scene interpretation along with other assistive capabilities. They used a series of cloudlets and local computers used to offload processing from the Glass device itself. This application also utilizes the GDK but only to access glass specific features and to connect to the various cloudlets and assisting devices. Anam et al. [6] used Glass to help people with social isolation and depression using non-verbal social cues to facilitate social interaction. Way et al. [7] used glass to assist people with memory deficiency, detect new places and faces, and identify people, objects, and locations using visual cues and GPS coordinates. In contrast, our research will explore the possibilities of cognitive enhancement using computer vision and artificial intelligence algorithms.

## 3 Methodology

In order to demonstrate this concept on Glass, the suitably complex game of Connect Four was chosen. Connect Four is a game in which two players take turns dropping colored tokens from the top of a 6 x 7 gridded game board. A Glassware application for Google Glass was created to provide intelligent feedback to a user playing a game of Connect Four. The first player to get four tokens in a row vertically, horizontally, or diagonally wins the game. This game was a good choice for use with Glass to demonstrate the capabilities of our system for several reasons. First, the game is constructed of simply colored components, not difficult to delineate from one another in a vision algorithm. Second, the game space is very large, i.e. there are $3^{42}$ possible board states. However, using an alpha-beta pruned minimax algorithm [8], our system was not so complex that it would be hard for a game algorithm to suggest a move. The minimax algorithm is a strategy for making predictions about future game states leveraging the fact that there are two players working towards opposite goals. The algorithm is a strategy for minimizing the possible loss of maximum loss scenarios. Alpha-beta pruning is an improvement over a naive minimax approach such that it eliminates branches of the tree of possible game states that will never be searched. In order to satisfactorily behave as a game assistant, some criteria were set out. The device should be able to provide a move suggestion without much effort from the user, and in a reasonable amount of time. In order to best meet these criteria, a web based service was built to interact with Glass. This solution was chosen over algorithms running directly on the Glass device to improve speed and battery life. With the limited computing power of a mobile device such as Glass, it seemed in the spirit of this technology to implement a cloud based service.

The game assistant consists of two separate interacting modules. More on their implementations, functionality, and purpose will be discussed in the following sections. The first module on Google App Engine is built on top of the Java Quick Start application that Google provides. This Quick Start application provides a starting point for building a Glassware Application. A Java class was added to the Quick Start code to communicate with a second module. When an image is received by the first module, it is uploaded via an HTTP POST request to a PHP script on an Apache server running on Amazon EC2 [9]. Once the

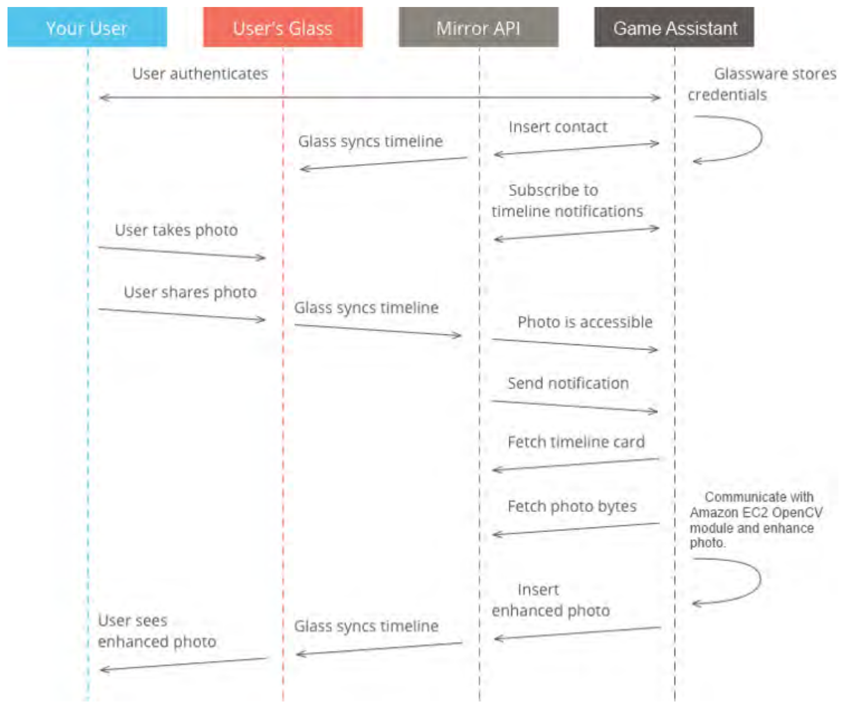


**Fig. 1.** The API flow for the glassware service.

image is uploaded it waits for a response from the server. The PHP script saves the image and runs a Java jar file with the image path as a parameter. This is how the two modules are stitched together. The OpenCV module is responsible for the analysis of the image and minimax computation. Many vision algorithms are used throughout the OpenCV computation, although their implementations were left up to OpenCV.

### 3.1 Glassware on Google App Engine

The module hosted on Google App Engine consists of the Glassware application which handles all interaction with the Glass device itself. This module is needed to allow a photo taken by Glass to be uploaded, modified, and returned to the user's Glass. This basic functionality is included in Google's Mirror API Quickstart for Java, and is what this module was built upon.

The API flow for the glassware service can be broken down into ten steps as illustrated in Figure 1 and described below.

1. The user authorizes with OAuth 2.0, and the service stores their credentials.
2. A contact for the service is added to the user's Glass.
3. The service subscribes to updates in the user's time-line by inserting a subscription for the time-line collection.

4. User takes a picture of the game setup.
5. User shares the photo with the added contact, making it accessible to the service.
6. Due to the fact that the service is subscribed to time-line updates, a notification is sent to the service including the id of the time-line card containing the shared photo.
7. The service uses this ID to fetch the correct time-line card.
8. The service uses the card's attachment ID to fetch the binary data of the photo.
9. A call to the AI module is made, and a near optimal game move is returned.
10. Finally, a new time-line card is constructed with information returned from the AI module and inserted into the user's time-line.

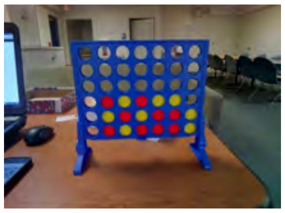
### 3.2 OpenCV on Amazon EC2

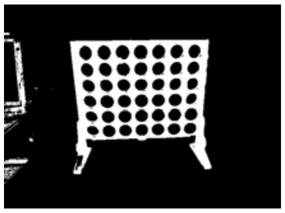
The second module hosted on Amazon EC2 and combines powerful computer vision and artificial intelligence algorithms to deliver a near optimal move suggestion. Requests can be made to the EC2 instance through an Apache web server. The image of the game board that needs to be analyzed is uploaded to this server via a POST request to an Apache server. It is at this point that the heavy computation begins. The process taken by OpenCV to interpret the Connect Four board image can be broken down into the following steps illustrated in Figure 2 and described below.

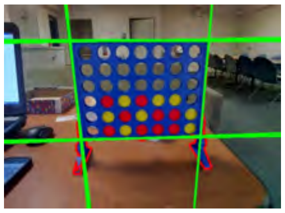
1. The original game state image is passed as a command line argument.
2. Threshold is taken by selecting for the color of the game board in an HSV range.
3. Contours of the threshold are found from a border-following algorithm and border lines are found and averaged from a Hough transform.
4. The possibly askew game image is rectified using a perspective matrix.
5. A threshold, contour, and minimum circle algorithm is applied to find the relative positions of the colored game tokens to the board.
6. Finally, a minimax algorithm is applied to produce an intelligent move suggestion.

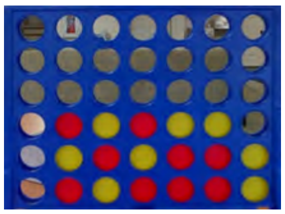
### 3.3 Minimax Algorithm

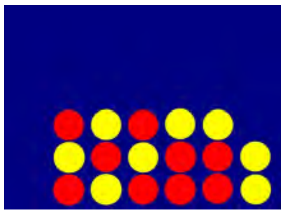
As mentioned previously alpha-beta minimax is used to calculate an approximate best move. For the purposes of this research, a standard implementation of this algorithm was used. In order to arrive at a move suggestion in a reasonable amount of time, a depth limit was applied to the minimax tree of game states. The depth limit is easily adjustable to suit the needs of the application. It follows from the imposed depth limit that a leaf node of the tree of game states may not necessarily be a terminal game state (i.e. one in which the game has ended). As the minimax algorithm requires the utility of each leaf node to operate, there must be a way to calculate the utility of a non-terminal game state. In order


(a) Original image taken in as command line argument.


(b) Threshold by selecting pixels in an HSV range.


(c) Contours from border-following and lines from Hough transform.


(d) Rectified image using perspective matrix.


(e) Threshold, contours, and min circles.

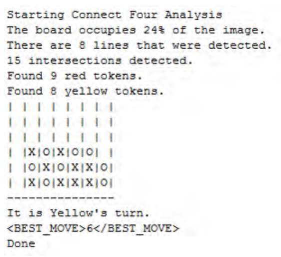

```
Starting Connect Four Analysis
The board occupies 24% of the image.
There are 8 lines that were detected.
15 intersections detected.
Found 9 red tokens.
Found 8 yellow tokens.
| | | | | | | |
| | | | | | | |
| | | | | | | |
| |X|O|X|O|O| |
| |O|X|O|X|X|O|
| |X|O|X|X|X|O|
---------------
It is Yellow's turn.
<BEST_MOVE>6</BEST_MOVE>
Done
```
(f) Console output and minimax.

**Fig. 2.** Visualization of the intermediate steps performed by the computer vision and artificial intelligence algorithms in our system.

to calculate this value every possible four-in-a-row on the board is considered paying attention to how close the current player is to achieving a four-in-a-row, assigning to it a weighted score. Negative weighted scores are used in the case that an opponent is close to achieving a four-in-a-row. The scores for each possible four-in-a-row on the board are summed resulting in a suitable heuristic value.

## 4 Experiments and Results

In order to use the game assistant, the user first takes a picture of the game board with Glass either through the hardware button or the voice activated menu. Next, the user taps the image and shares it with the Game Assistant Application. Glass uses its wireless connection and shares the image with the server. When a suggested move is found, the image on Glass' time-line is updated to show text with the suggested move. In our tests, this process takes 10-15 seconds to return a result to the user's glass; however, we note that this entire process is very reliant on the user's Internet connection.

The computation time of the OpenCV module on Amazon EC2 can be measured more accurately, due to the fact there is to connection latency involved. Running ten trials with example images on a machine with minimax looking ten moves ahead yielded an average move calculation time of 391 milliseconds.

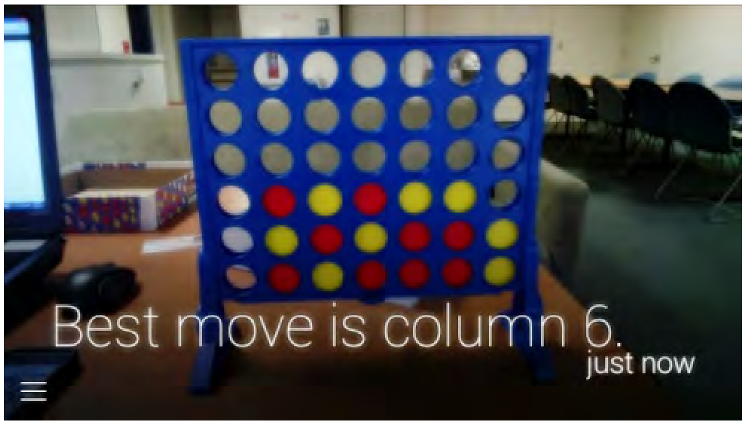


**Fig. 3.** A screenshot of the end result of our system looking through Google Glass. The best move is overlaid on the image processed by the cloud.

The computation time for this module is not very significant, and the main bottleneck seems to be network connection speeds. Another point in the process where things may be slowing down is the way in which the Mirror API handles subscriptions in its notification service. Our service on Google App Engine has to wait for a notification from the Glass device in order to start the process.

The 10-15 seconds it takes to return a result to the user's glass is significant given the short move calculation time of the OpenCV service. Further analysis of the time spent in both services revealed delays in network communication. From the time the Glassware service received the image to the time it was ready to update the timeline card with the result an average of 4.233 seconds had passed. The remaining 5-10 seconds in the process are due to the initial upload of the original image to the glassware service, and the delay between when a timeline card is updated by the service and when it appears on the user's device, see Figure 3.

### 4.1 Discussion and Future Work

The advantages of using a perspective matrix transform when computing the state of the game board is that it allows for moderately skewed boards to be detected. In Figure 4 for example, the algorithm is able to estimate the bounds of the playing area accurately enough to end up achieving a correct result. In the case of a more extreme skew such as in Figure 5, a correct result was not achieved. Extraneous border lines were detected near the base of the game board, interfering with the calculation of the bounds of the playing area.

There are a few mechanisms which could be used to generally improve this entire process. One way in which speed up the process of returning a result to the user's glass would be to use the GDK as a frontend for the OpenCV service. In such a case OpenCV would still be running on a remote machine

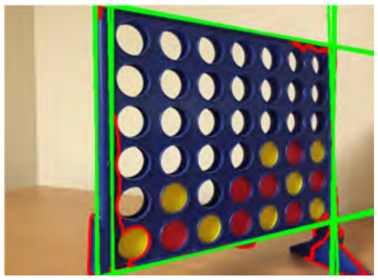


**Fig. 4.** Skewed board 1, our algorithm is able to correctly rectify this image.

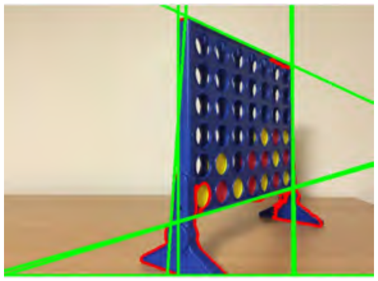


**Fig. 5.** Skewed board 2, our algorithm fails to correctly process this image.

and not on the Glass device itself. Making use of the GDK would eliminate the Glassware service and it's tight integration with the Glass timeline. However, the elimination of a service would also mean the elimination of its associated network latencies. In addition to improvements in the chain of network requests between services, the algorithms employed in the recognition of the game board could also be enhanced. The addition of mechanisms to enhance the image or to better handle it under extreme skew could be employed.

## 5 Conclusion

The results of this research are promising, showing that complex computation, while not in real time, is possible to achieve with wearable technology such as Google Glass. The end result was a functioning Glassware application which given an image of a Connect Four board can intelligently suggest a move. Due to the fact that the implementation was separated into two distinct modules, additional functionality can be added to the application by simply changing out the code hosted on Amazon EC2. This allows for the continuation of this research in terms of adding functionality for different games (physical or otherwise). The future of computing truly lies in this paradigm of lightweight technology having access to intelligently driven powerful computation.

## References

1. Google: The mirror api. `https://developers.google.com/glass/develop/mirror/index` (2014)
2. Google: Google glass playground. `https://github.com/space150/google-glass-playground` (2014)
3. Bradski, G.: Open cv library. Dr. Dobb's Journal of Software Tools (2000)
4. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for vm-based cloudlets in mobile computing. IEEE Pervasive Computing **8** (2009) 14–23

5. Ha, K., Chen, Z., Hu, W., Richter, W., Pillai, P., Satyanarayanan, M.: Towards wearable cognitive assistance. MobiSys (2013) 68–81
6. Anam, A., Alam, S., Yeasin, M.: Expression: a google glass based assistive solution for social signal processing. In: Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility, ACM (2014) 295–296
7. Way, T., Reimers, C., Bemiller, A., Wagner, B.: Memory enhancement using machine learning: The glassessibility project. Google Glass Accessibility Workshop (2014)
8. Knuth, D.E., Moore, R.W.: An analysis of alpha-beta pruning. Artificial intelligence **6** (1976) 293–326
9. Amazon Web Services, I.: Aws amazon elastic compute cloud (ec2) - scalable cloud hosting. (2014)