# Computer Vision on Tap

Kevin Chiu Ramesh Raskar
MIT Media Lab, Cambridge, MA
{kgc,raskar}@media.mit.edu http://visionontap.com

## Abstract

*We demonstrate a concept of computer vision as a secure, live service on the Internet. We show a platform to distribute a real time vision algorithm using simple widely available web technologies, such as Adobe Flash. We allow a user to access this service without downloading an executable or sharing the image stream with anyone. We support developers to publish without distribution complexity. Finally the platform supports user-permitted aggregation of data for computer vision research or analysis. We describe results for a simple distributed motion detection algorithm. We discuss future scenarios for organically extending the horizon of computer vision research.*

## 1. Introduction

### 1.1. Background

In recent years, the Internet has become an increasingly important tool for computer vision research. Vast archives of visual data compiled by services such as Flickr, YouTube, and Google Images aid research in structure from motion [9], scene completion [4], and panorama detection [5], among many other areas. Commoditized human intelligence available through Amazon Mechanical Turk aids research in user interface analysis [12], image tagging [10], and scene understanding [8, 11]. Online user communities perform difficult computer vision tasks including optical character recognition [15] and image segmentation [14]. Researchers have attempted to enable long tail book publishing through online computer vision services [7]. The Internet now plays an important role both as a source of raw data and artifacts of human interaction.

### 1.2. Motivation

Bob is annoyed by cars speeding through his residential neighborhood and worried about the safety of children playing on the street. Alice wants to avoid eye-strain and wants her laptop to remind her if she is staring at the screen for too long. Jean is an artist and wants to estimate the percentage of people in the city sitting in front of a computer and wearing bright colored clothes or the frequency distribution of smiles per person in a day. Can Bob and Alice simply turn on a webcam and visit a website to get the services they require? Can Jean write a program and encourage others to go to a website so he can get real-time feedback?

We aim to provide a distributed computer vision system in which consumers casually interact with computer vision algorithms, producers create new algorithms without worrying about how to deploy and gather data from them, and all users are empowered to share their algorithms and content. With user and developer permission, the aggregated real-time streamed data can also be used for tagging, labeling, scene understanding, and prediction problems.

1. The system should be trivially easy to use for consumers; they should not have to download or install a new application or plug-in.

2. Computer Vision developers should not be concerned about managing the resources required to distribute their application.

3. Developers should be able to reuse algorithms easily.

4. Consumers should have fine grained control of their data.

There are many untapped resources relevant to computer vision on the Internet. One relatively untouched resource is users' webcams. We show how a distributed system consisting primarily of website visitors' browsers can be used to efficiently gather and process live video data.

The Internet is not only a host for large data sets, free processing power, and worker communities waiting for quick tasks. It can also be used as a platform for innovation through end users [6]. We discuss a future system in which users are empowered to experiment with and share their own computer vision algorithms. We propose using simple web technologies, such as Adobe Flash and JavaScript, along with commodity webcams to enable a lightweight distributed computer vision platform.

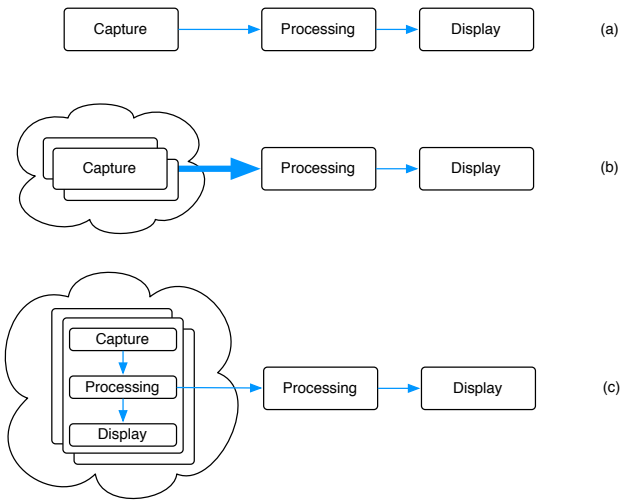



Figure 1. A standard computer vision pipeline (a) performs capture, processing, and display using a single serial workflow. A typical Internet vision pipeline (b) downloads large amounts of data from the Internet and then batch processes images to synthesize a result. Our proposed pipeline (c) distributes capture, processing, and display into the Internet.

## 1.3. Contributions, Benefits and Limitations

We present a way to distribute an interactive real time computer vision algorithm, breaking away from the current trend in Internet computer vision of simply using the Internet as a large dataset of visual information or a pool of low skilled labor. We discuss a future computer vision platform for publishing and organically extending the field of computer vision research through intelligent user interaction.

The field of computer vision research benefits from our efforts by being exposed to a larger population of creative individuals. The next major contribution to our field may not come from our closed community of researchers, but instead, from the open community of the Internet.

However, the concept has limitations. Only certain tasks can be performed via a browser window. Computational tasks are easy but OS or peripheral interaction may be limited. We require (power and) Internet connectivity, possibly limiting use in developing countries. The platform needs an incentive model for general population to keep the webcams running even when the computers are idle. We have implemented a working prototype published at `http://visionontap.com`, but it is limited. It publishes content but does not provide a user friendly UI, developer tools, or any way to foster an active community of contributors and consumers. Although the implementation requires more development, we hope the experiments and the platform are sufficient to motivate further research and exploration.

## 1.4. Related Work

### 1.4.1 Online Programming Platforms

Our system follows in the footsteps of Scratch [6]. We take the basic concept of Scratch, an online programming and sharing platform designed to bring programming to under-served youth populations, and rethink the concept to fit the needs of the budding computer vision hobbyist. Scratch requires users to download and install a local application. However, our system allows content authoring in the browser. Scratch uses Java as its publishing language, making it difficult to deploy camera-based applications since standard client installations do not include camera access features. In contrast, our system uses Flash, which includes camera access capabilities in the standard client installation. In addition to adding camera access, our system also allows programs to interact with services other than our own, allowing third party integration.

### 1.4.2 Distributed Processing

Our system borrows ideas from Google MapReduce [2]. MapReduce is a programming model for processing and generating large data sets. In MapReduce, the general processing task is essentially split up into many map tasks, and a large reduce task. Map tasks compute on data in parallel, and reduce tasks take the output of the map tasks and combine them into a useful output. In our system, the map tasks can be viewed as the applications that that are run on users' machines, and the reduce task can be viewed as analyzing data returned from published applications. Our system is different in that it performs all of its calculations in real-time rather than as a parallel batch process.

### 1.4.3 Human Processing

Our system is inspired by systems such as [13, 14, 15], that use humans to perform tasks traditionally thought of as difficult for computer algorithms. However, we look beyond using users as a computer vision problem-solving commodity and aim to involve them in the innovation process.

# 2. Distributed Computer Vision

The system for distributed deployment of computer vision algorithms follows basic client-server architecture. The client consists of a user implemented as an Adobe Flash SWF object embedded into an HTML page, both which are served from Google AppEngine. The server consists of a heterogeneous system using both AppEngine servers and dedicated servers.

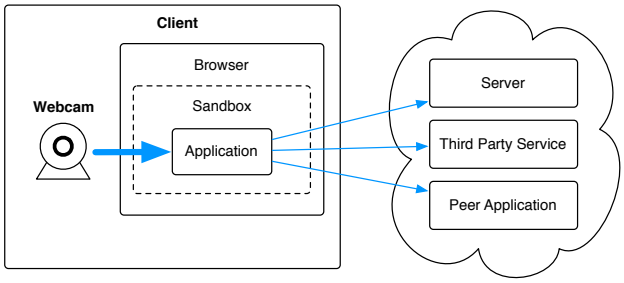



Figure 3. Data flow for user's client consumption. The width of the arrows is representative of the amount of data traveling between each component of the system. Most of the data is processed on the client, which publishes a summarized result set to one or multiple endpoints. For example, the camera may be capturing images at 30 fps. In most cases, no data is sent back to the server.

## 2.1. Clients

We refer to a user's computer and everything it hosts as the client. As shown in figure 3, the client includes the webcam, browser, and Flash application. When the Flash application is downloaded from our servers, a prompt appears asking for permission to access resources outside of the client's privacy and security sandbox. In our case, this is the camera. When the user accepts, the application's privileges are elevated to include camera access and the vision program runs. In future work, additional privacy concerns will be addressed.

The vision program that we implemented for initial testing was a simple motion detector. The motion detector accesses the camera and reports a motion value between 0 and 100 back to a standalone server. The reports are made once per second per client. All information is currently sent through URL parameters, but sending additional information through alternative communications methods, such as Jabber or through sockets is also possible.

## 2.2. Server and supporting architecture

Conceptually, the publishing system consists of three parts including a pool of servers, job queues, and worker machines as shown in figure 4. The servers are AppEngine servers that act as an interface to the client. The job queues are used to provide a place to keep track of the status of jobs uploaded to the web server by clients. Workers strive to keep the job queues empty by compiling uploaded material and delivering binaries.

### 2.2.1 Server

The server handles two primary tasks. It serves applications to consumers and receives code from publishers. To accomplish these tasks, it relies on access to a distributed database known as Google Datastore and communication over HTTP to dedicated servers that perform native execution tasks disallowed on the AppEngine platform.

A request to publish begins with a request for the authoring interface. Currently the interface is a web form consisting of two text boxes, one for ActionScript, the primary programming language of Adobe Flash, and the other for Macromedia XML (MXML), which describes the interface in Adobe Flex. The user interface will be enhanced in future work. Submitting the form sends the code back to the server, where it is stored in the database.

When the uninterpreted code is stored in the database, a unique database key is generated that can be used to access the related database record directly. This unique key is passed on to the job queue so that workers will be able retrieve the code and return binaries and important messages back to the server using a remote API. This key is also used as a URL argument for communicating which application to retrieve.

A request to view an application begins when a client submits a Universal Resource Locator (URL) including a unique key describing the location of the program in our database. This key is used by the server to query the database for the associated entry. This entry may contain a text message detailing errors in compilation or a binary Flash file. If it contains a text message, the message is displayed with an apology. If it contains a binary, then the Flash application is loaded into the user's browser. If the associated entry for the key is not found, an error is returned.

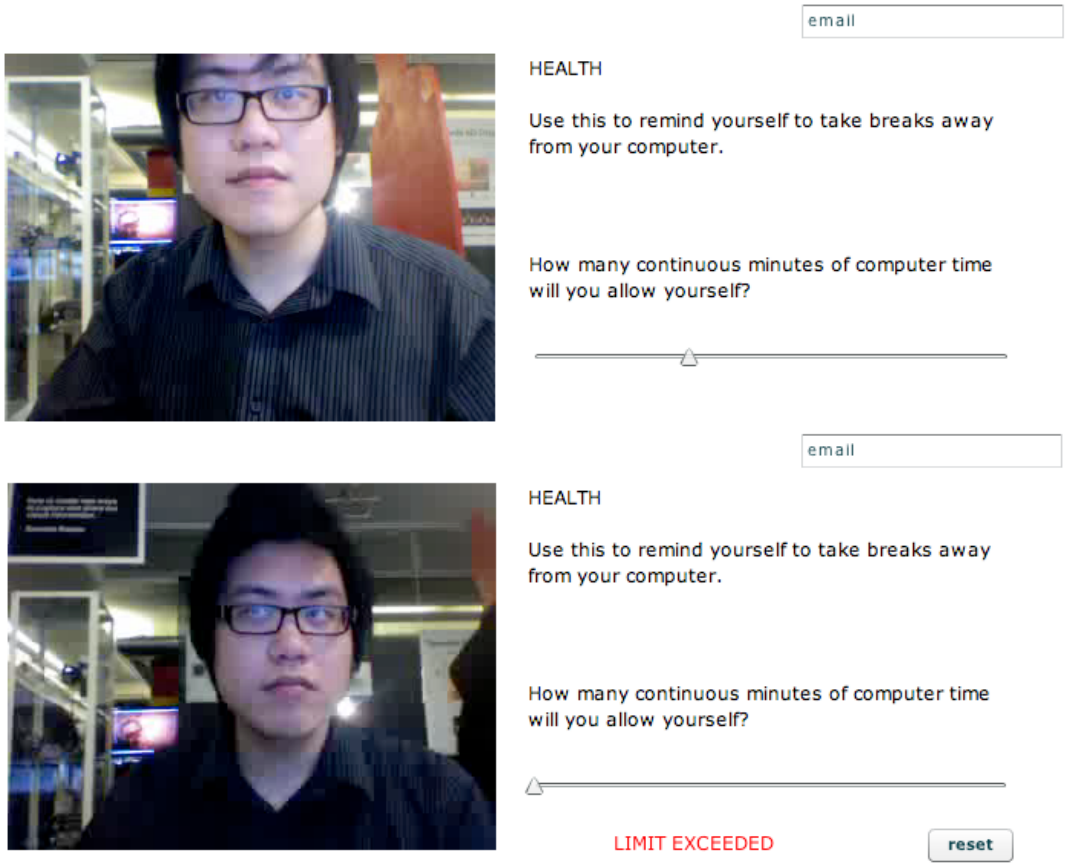
### 2.2.2 Worker

When building the application, we discovered that the AppEngine platform does not allow native execution of code. The goal of the worker machines is to perform all necessary native execution tasks.

Once initialized, the worker queries the job queue for available jobs. If a job is available, then it is time-stamped and the status is changed to being in progress. The worker then reads the unique database key from the job description. The key is used to retrieve the code to be compiled from the database.

When the code is retrieved, it is extracted into a folder named after the unique key to avoid conflicts with workers working on the same machine. The contents of the folder are compiled using MXMLC. Then a remote connection to AppEngine is made to upload the resulting SWF to the database and to set a `compiled` flag to `true`. If errors occurred during compilation, then the messages are sent back to the database and the flag is set to `false`.

Once results are sent back to the database, the worker reports back to the job queue and sets the status of the job to completed. Then it requests a new job and the cycle continues.



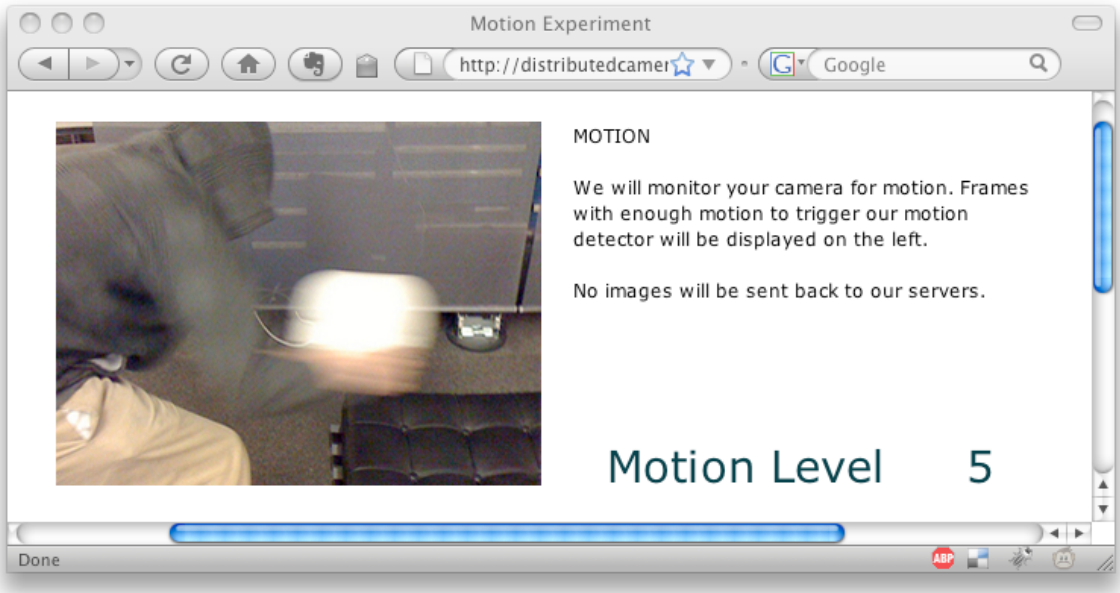




Figure 2. The interfaces of two sample applications. The first two images are of an application that reminds the user to take breaks away from their computer. Two timers are used in conjunction to provide the desired service. One thirty second timer is reset when motion above a certain threshold is detected. A second timer, whose countdown time is set by the user, is reset if the thirty second timer reaches zero. The third image is of a simple motion detection application.

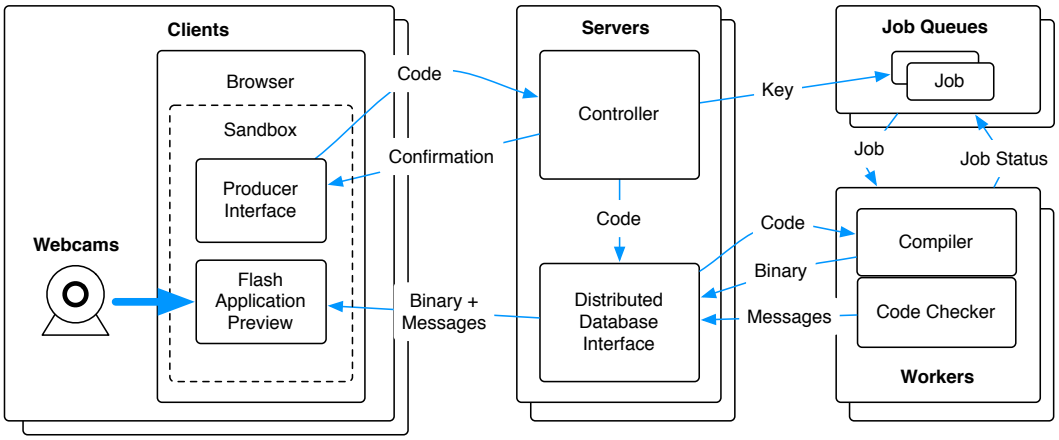



Figure 4. Data flow for publishing an application. A producer uploads code through an online interface into one of many AppEngine servers. From here, the code is stored in a distributed database and a compilation job is queued. As they become available, dedicated worker machines monitoring the job queue process compilation jobs, generating binaries, or in the case of errors or warnings, messages that will be presented to the client.

The worker cycle is maintained using `cron` and a series of folders representing stages of code retrieval, compilation, and uploading. `Cron` is used to make sure the Python scripts used to complete each stage are running.

#### 2.2.3 Job Queue

A job queue is used to reduce the risk of over saturating the dedicated servers with too many simultaneous tasks. If a job queue were not present, a sudden surge in publication requests could overextend the fixed resources dedicated to native execution. With a job queue, a sudden surge in publication requests merely lengthens the waiting time for completing new jobs and leaves currently running jobs unencumbered.

After submitted code is stored on the server, the job queue is asked to create a compilation job that includes a unique database key provided by the AppEngine server. This job is placed at the end of the queue with an available status.

#### 2.2.4 Servers, Third Party Services, and Peer Applications

Applications are currently allowed to communicate arbitrarily with any other processes running on the Internet. This allows interesting interactions with not only AppEngine, but also third party services such as Twitter and Flickr, remote dedicated servers, and even other client-side Flash applications.

We implemented a simple third party server using CherryPy. Our server maintains an access log that records each incoming request. Each record includes IP address, date, HTTP command, URL, return code, browser, operating system, and preferred language.

## 3. Limitations

Incentivizing user participation is challenging. We found the job acceptance threshold on Amazon Mechanical Turk with regards to compensation to be exceptionally high compared to the results achieved by previous research [10]. Through feedback from one of the workers, we discovered that they were apprehensive about taking the job because they believed that photos were being taken of them. What we learned from this is that, although our software stated clearly that no images were being sent back to the server, there was still an issue of trust between the consumer and the producer. In future work, we aim to assure a mutual agreement of privacy expectations before accessing a personal camera.

Currently, our system requires advanced knowledge to use fully. Our publishing interface, a simple web form, is minimal interface usable by advanced users or users with access to an external development environment. Additionally, we have not tested our system under heavy publishing load. Currently we have published one application in a testing environment, located at `http://visionontap.com` and have not encountered any systemic problems during our testing.

Users were recruited both organically and through Amazon Mechanical Turk. Initially the compensation offered for keeping the program running for twenty-four hours was 0.05 USD, the default price suggested by the Mechanical Turk system. At this price, we commissioned 100 Human Intelligence Tasks (HITs). However, after one hour of waiting no HITs had been accepted by workers. The price was

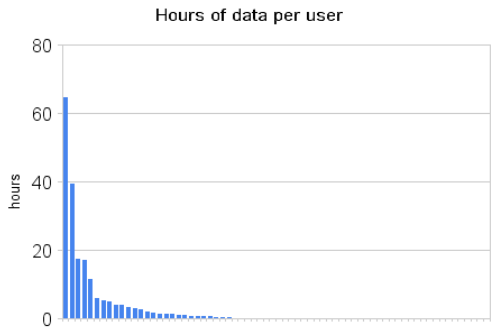



Figure 5. The distribution of hours of user participation follows a long tail distribution. The most dedicated user provided over sixty hours of video. The least dedicated user provided less than an hour of video.

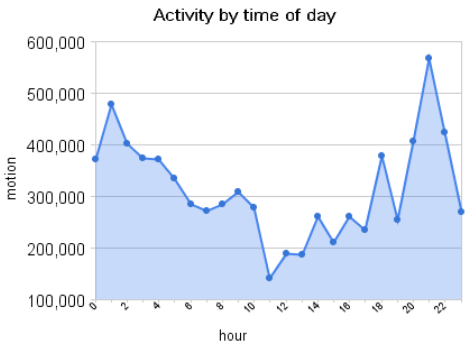



Figure 6. The distribution of activity of hours in the day shows more activity at nighttime than daytime. The units on the vertical axis are sums of motion levels, each in the range 0 through 100 inclusive.

then raised 0.50 USD, leading our first Mechanical Turk worker to accept the job. Although we found it difficult to recruit anonymous workers online, we found it trivially easy to recruit graduate student volunteers with the promise of 0.50 USD worth of free food from a nearby vending machine.

## 4. Results

Here is an example of a single record we collected using our stand alone server monitoring a distributed motion detection algorithm:

18.85.24.87 - - [22/Mar/2009:06:32:37] ”GET /report/email/8 HTTP/1.1” 200 51 ”” ”Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.0.7) Gecko/2009021906 Firefox/3.0.7”

We collected over 700,000 records over the course of approximately five days. Our first record is dated `17/Mar/2009:20:22:22`, and our last record is dated `22/Mar/2009:06:32:37`. In raw form, the log file is 151.3 MB. Compressed, it is 3.2 MB, which we estimate to be significantly smaller than five days worth of video from connected clients.

Although we can’t be certain of the exact number of users due to dynamic IP addresses and lack of an authentication system, we estimate, based on the number of unique IP addresses in the log, that 68 individuals from the eight countries shown in figure 1 provided a combined 197 hours of video data using our application.

From the data gathered in this experiment, we were able to determine that more movement occurred in front of our participating clients’ cameras at night, suggesting that users were more active at night than during the day.

| Country | Unique IPs |
|---|---|
| United States | 58 |
| India | 3 |
| Germany | 2 |
| Taiwan | 1 |
| France | 1 |
| European Union | 1 |
| Serbia and Montenegro | 1 |
| Canada | 1 |

Table 1. Number of unique IP addresses. Participants were originally recruited locally by word of mouth. However, due to low participation rates, additional participants were recruited through advertisement on Twitter, through email lists, and through Amazon Mechanical Turk.

Figure 7. The next big computer vision killer app may be invented by this guy. Computer vision research benefits from being exposed to a larger population of creative individuals. The next major contribution to vision in learning may not come from a closed community of researchers, but instead, from the open community of the Internet. (Photo courtesy Wired Blog.)

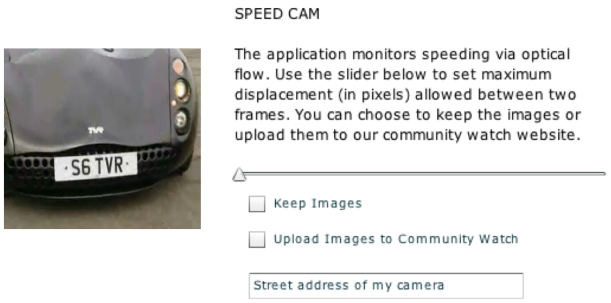



Figure 8. Potential Application: Citizen Speed Watch. Bob, concerned about speeding vehicles, can aim his camera towards the street and visit a website for computer vision service. One limitation, however, is that license plates may be blurred.

# 5. Potential Applications

## 5.1. Real-Time Social Mood Mapping

Determining the mood of an individual is relatively easy for humans. Facial feature recognition and expression understanding are hardwired into the human brain [3]. However, determining the distributed mood of a group of people, for example, in an entire office building, can be a nontrivial exercise.

With our system, John can write a smile detector paired with a mapping application and distribute it over the Internet to colleagues in his office. His coworkers can participate in creating a mood map of their office by allowing the in-office map application to use their location and mood information to populate local office regions on their floor. Running the application for each floor of a building can help produce a three dimensional volume of mood measurements. It would be interesting to investigate the correlation between office floor number and daily mood fluctuations.

## 5.2. Neighborhood Watch

Speeding in residential areas is a life-threatening nuisance that pervades neighborhoods across the country. Despite presenting an exceptionally high risk to school-aged children, this public nuisance is low on police priority lists due to the frequency of higher priority and higher profile crimes such as drug trafficking and violent crime. Police forces are too preoccupied with more immediate threats to dedicate forces to stop neighborhood speeding.

One solution to help neighborhoods defend themselves from irresponsible motorists is to use our system. Concerned citizens can publish speed camera software, shown in figure 8, to be deployed by homeowners in their windowsills using commodity hardware, including just webcams, personal computers, and the Internet. Additionally setup would be extremely simple since there is no installation procedure. Simply visiting a website links the camera back to a local neighborhood watch group or the local police station.

# 6. Conclusion and Future Work

We have demonstrated a concept of computer vision as a live service on the Internet. We showed a platform to distributed a real time vision algorithm using simple widely available web technologies. There are several avenues for future work.

## 6.1. Privacy

In [1], Avidan and Butman describe a system in which the algorithm producer guarantees privacy for the algorithm consumer. In our system, we expect to be able to allow the consumer to specify, in a fine-grained manner, the level of privacy expected, so that provided applications can be loaded with a social contract respecting end user privacy concerns.

## 6.2. User Interface

Our current user interface is lacking and does not meet user expectations. In future iterations, we hope to create a more inviting experience for end users. This can be accomplished through creating interfaces to existing highly developed publishing services, such as Scratch and Eclipse, or creating our own environment.

## 6.3. System Robustness

Our initial prototype client, located at `http://visionontap.com`, only reports back to a single endpoint. In future iterations, we hope to add interfaces for multiple servers, third party services, and other clients.

# References

[1] S. Avidan and M. Butman. Blind vision. In *in Proceedings of the 9th European Conference on Computer Vision*, pages 1–13. Springer, 2006.

[2] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. pages 137–150, December 2004.

[3] E. Halgren, T. Raij, K. Marinkovic, V. Jousmaki, and R. Hari. Cognitive response profile of the human fusiform face area as determined by meg. *Cereb. Cortex*, 10(1):69–81, January 2000.

[4] J. Hays and A. A. Efros. Scene completion using millions of photographs. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 4, New York, NY, USA, 2007. ACM.

[5] F. Liu, Y.-h. Hu, and M. L. Gleicher. Discovering panoramas in web videos. In *MM '08: Proceeding of the 16th ACM international conference on Multimedia*, pages 329–338, New York, NY, USA, 2008. ACM.

[6] J. Maloney, L. Burd, Y. Kafai, N. Rusk, B. Silverman, and M. Resnick. Scratch: A sneak preview. In *C5 '04: Proceedings of the Second International Conference on Creating, Connecting and Collaborating through Computing*, pages 104–109, Washington, DC, USA, 2004. IEEE Computer Society.

[7] P. Reddy, J. Fan, J. Rowson, S. Rosenberg, and A. Bolwell. A web service for long tail book publishing. In *BooksOnline '08: Proceeding of the 2008 ACM workshop on Research advances in large digital book repositories*, pages 45–48, New York, NY, USA, 2008. ACM.

[8] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: A database and web-based tool for image annotation. Technical report, Tech. Rep. MIT-CSAIL-TR-2005-056, Massachusetts Institute of Technology, 2005.

[9] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: Exploring photo collections in 3d. In *SIGGRAPH Conference Proceedings*, pages 835–846, New York, NY, USA, 2006. ACM Press.

[10] A. Sorokin and D. Forsyth. Utility data annotation with amazon mechanical turk. pages 1–8, June 2008.

[11] M. Spain and P. Perona. Some objects are more equal than others: Measuring and predicting importance. In *ECCV '08: Proceedings of the 10th European Conference on Computer Vision*, pages 523–536, Berlin, Heidelberg, 2008. Springer-Verlag.

[12] S. Steinbach, V. Rabaud, and S. Belongie. Soylent grid: it's made of people. In *ICCV*, pages 1–7. IEEE, 2007.

[13] L. von Ahn and L. Dabbish. Labeling images with a computer game. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 319–326, New York, NY, USA, 2004. ACM.

[14] L. von Ahn, R. Liu, and M. Blum. Peekaboom: a game for locating objects in images. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 55–64, New York, NY, USA, 2006. ACM.

[15] L. von Ahn, B. Maurer, C. Mcmillen, D. Abraham, and M. Blum. recaptcha: Human-based character recognition via web security measures. *Science*, pages 1160379+, August 2008.