Vision on Tap: An Online Computer Vision Toolkit

by

Kevin Chiu

M.S. Columbia University (2009)
B.S. University of Washington (2007)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Master in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2011

Author_____
Program in Media Arts and Sciences
June, 2011

Certified by_____
Prof. Ramesh Raskar
Associate Professor of Media Arts and Sciences
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by_____
Prof. Mitchel Resnick
LEGO Papert Professor of Learning Research
Academic Head
Program in Media Arts and Sciences

Vision on Tap: An Online Computer Vision Toolkit

by

Kevin Chiu

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on June, 2011, in partial fulfillment of the
requirements for the degree of
Master in Media Arts and Sciences

Abstract

In this thesis, we present an online toolkit, based on a combination of a Scratch-based programming environment and computer vision libraries, manifested as blocks within the environment, integrated with a community platform for diffusing advances in computer vision to a general populace. We show that by providing these tools, non-developers are able to create and publish computer vision applications. The visual development environment includes a collection of algorithms that, despite being well known in the computer vision community, provide capabilities to commodity cameras that are not yet common knowledge. In support of this visual development environment, we also present an online community that allows users to share applications made in the environment, assisting the dissemination of both the knowledge of camera capabilities and advanced camera capabilities to users who have not yet been exposed to their existence or comfortable with their use. Initial evaluations consist of user studies that quantify the abilities afforded to the novice computer vision users by the toolkit, baselined against experienced computer vision users.

Thesis Supervisor: Prof. Ramesh Raskar
Title: Associate Professor of Media Arts and Sciences, Program in Media Arts and Sciences

Vision on Tap: An Online Computer Vision Toolkit

by

Kevin Chiu

The following person served as a reader for this thesis:

Thesis Reader_____

Prof. Leah Buechley
Assistant Professor of Media Arts and Sciences
Program in Media Arts and Sciences

Vision on Tap: An Online Computer Vision Toolkit

by

Kevin Chiu

The following person served as a reader for this thesis:

Thesis Reader_____

Prof. Mitchel Resnick
LEGO Papert Professor of Learning Research
Academic Head
Program in Media Arts and Sciences

Acknowledgements

This thesis is dedicated to my parents, who instead of pressuring me to achieve arbitrary academic and extracurricular benchmarks, have always generously granted me unprecedented freedom to explore life and learn from my experiences.

I would like to thank my advisor, Ramesh Raskar, for allowing me to work on his vision of Visual Social Computing, my readers, Mitchel Resnick and Leah Buechley for their invaluable feedback on this thesis, Evelyn Eastmond and John Maloney for letting me use DesignBlocks as the basis for Vision Blocks and my former undergraduate assistants David Jia, Martin Martinez Rivera, and Phu Nguyen as well as visitors Kshitij Marwah and Abhijit Bendale for their work on implementing and debugging the Vision Blocks platform.

Finally, I would like to thank Jon Brandt and Adobe for their generous support and feedback regarding Vision on Tap and Vision Blocks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In 2009, Ramesh Raskar made several observations that make the need for this thesis apparent. First, cameras have become commonplace, with approximately 25 million sold each year in the United States alone [1]. Second, a relatively minuscule percentage of the population is literate in computer vision, despite having wants and needs that may well be met by vision systems. Third, researchers have started to look to online communities for both inspiration and implementation of new solutions [43, 42, 44].

Consumers are primed for education and researchers are ready to actively engage the wider online community. Visual social computing (ViSoCo), as described by Raskar, weaves a new social fabric based on visual computing of the people, by the people, and for the people. The technologies and platforms for visual exchange aim to overcome the traditional barriers of language and culture and consists of multi-disciplinary research in facilitating a positive social impact via the next billion personalized cameras. Within this movement is defined Vision on Tap, a platform upon which common end users create computer vision algorithms, share them through an online application store, and communicate directly with developers and researchers to provide the next generation of computer vision algorithms. The Vision on Tap project is a small aspect of the vision of ViSoCo.

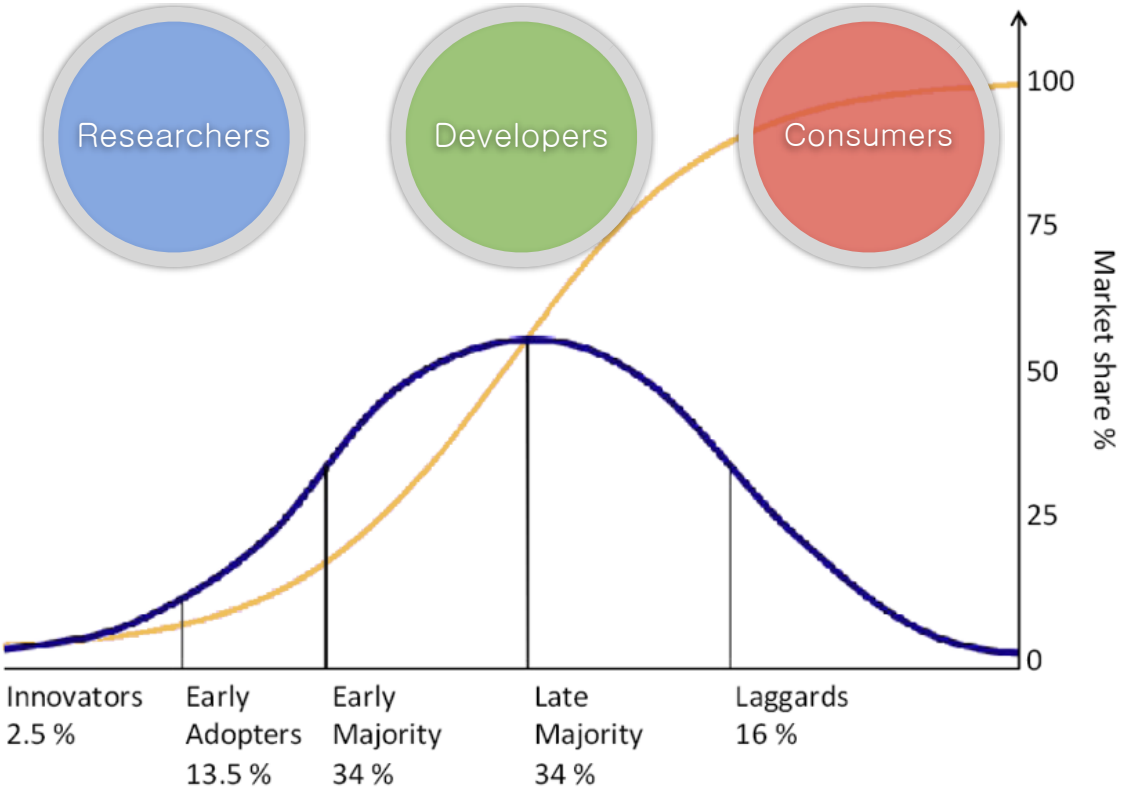

Figure 1-1: Diffusion as defined by Rogers. The Bell curve represents the social system in which the innovation is spreading. The population is organized into groups along the horizontal axis based on speed of innovation adoption. The yellow S-curve is proportional to the population over time that has adopted the innovation.

1.1 Background

1.1.1 Diffusion of Innovations and Toolkits

Diffusion, pictured in Figure 1-1 is the process by which an innovation is communicated through certain channels over time among the members of a social system. [33] In our context, the innovation is computer vision, the channels are through our online community and workshops, and the social system is people who have little or no experience programming computer vision algorithms although they could very well benefit from their use. Unfortunately, innovation uptake among our population has been slow. The computer vision community's contributions typically take years to reach consumers.

One way to drastically decrease the complexity and increase the trial-ability of an innovation is to package it inside a product or service. One such product or service is a toolkit. Within the instructions, the user should learn to customize the technology, thus maximizing relative advantage. By transferring design capability to the user, the user should be able to create, from the toolkit, a custom product that is preferable to what could be attained as a ready-made product[45, 46, 30]. Vision on Tap is an online vision toolkit that will empower users to develop solutions for themselves that they would not be able to obtain off the shelf.

1.1.2 Computer Vision and the Internet

In recent work, the Internet has been exploited for its free and low cost resources, which include digital artifacts of human expression as well as menial mental labor. Vast archives of visual data compiled by services such as Flickr, YouTube, and Google Images aid research in structure from motion [36], scene completion [14], and panorama detection [18], among many other areas. Commoditized human intelligence available through Amazon Mechanical Turk aids research in user interface analysis [39], image tagging [37], and scene understanding [38]. Approaches such as [40, 37, 49, 34] focus on inserting the user into multi-class object classification systems whereas approaches such as [7, 47, 48, 27] use a human interface to train underlying computer vision algorithms. Online user communities perform difficult computer vision tasks including optical character recognition [44] and image segmentation [43, 39].

Rather than use the presence of the general online populace as a way to support computer vision algorithms, we aim to make computer vision algorithms useful to the population on the Internet [11].

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 her screen for an unacceptably long continuous period of time. 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 turn on a webcam, visit a website, and get the services they require? Can Jean write a program and post a link online to get real time feedback?

We aim to provide an online computer vision toolkit and supporting community in which consumers casually interact with computer vision algorithms, producers create new applications without worrying about how to deploy them, and all users are empowered to share their applications and content. With permission, the aggregated data can also be used for a variety of computer vision tasks, including but not limited to tagging, labeling, image recognition, and scene understanding.

1. The system should be trivially easy to use; users should not have to download or install a new application or plug-in.
2. Creators, users who choose to publish applications, should not be concerned about managing the resources required to distribute their applications.
3. Creators who choose to create vision applications should be able to reuse algorithms easily.
4. Consumers should have control of their data.
5. Researchers should be able to demonstrate their methods in real world conditions.

There are many untapped resources relevant to computer vision on the Internet. One relatively untouched resource is users' webcams. In an initial proof of concept, we show how a system consisting primarily of a website's visitors' browsers can be used to efficiently distribute interactive computer vision applications.

However, 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 [21]. We discuss a system in which users are empowered to experiment with and share their own computer vision applications. We propose using simple web technologies, such as Adobe Flash, along with commodity webcams and server architectures to enable a lightweight computer vision platform.

1.3 Contribution

This project makes several contributions, including a platform for creating and distributing lightweight computer vision applications, a visual programming interface for creating computer vision applications without prior knowledge of computer vision or programming, and a user study comparing the capabilities of the system in the hands of programming and computer vision novices vs. experienced in programming and computer vision.

1.4 Related Work

1.4.1 Computer Vision Toolkits

Many computer vision toolkits exist today. Maynes-Aminzade [22] has created an interactive system in which developers use visual examples to train a set of computer vision recognition algorithms. Users externally create their own programs to interpret the generated output, but cannot reprogram the included applications. SwisTrack [19], a generalized tracking system for life sciences proved to be an extremely valuable tool for life scientists. DAPRA's IUE program created number of "visual, plug and play" interfaces[24]. The IUE failed to gain wide usage because it presented an interface which built on top of advanced vision concepts, e.g. homographies, which limited its users to those who possessed both advanced vision background and software development skills. The former limited its use outside of vision, while the latter limited its use among vision researchers. Simpler but less extensive solutions such as OpenCV[6], gained wider acceptance but still required at least modest software development skills. There are a wide variety of languages available for building vision applications, such C, C++, Matlab, Python, Java along with libraries such as OpenCV and others. These have a high barrier of entry, which hinders their rapid diffusion amongst non-technical populations.

1.4.2 Computer Vision as a Web Service

Many projects have provided computer vision as a web service over the Internet. Skocaj et al. [35] discussed a model for delivering an image segmentation algorithm using a Java applet communicating with a Common Gateway Interface (CGI) server. Gazehawk[16] provides on-demand website usability studies, based on eye-tracking, as an on-demand service using cameras built into modern portable computers. Visym[9] provides a Matlab interface to cloud-hosted algorithms. IQEngines returns text labels for query images.

1.4.3 Social Coding

In our physical reality, in which we are constrained to meeting people in person, one of the core dependent variables of the diffusion of innovation, communication[33], is limited by mutual coincidence of time and space. By putting the community online, the channels of communication are made more flexible, easily supporting not only collaboration collocated in time and space, but also remote, asynchronous collaboration in the form of comments and code reuse[28]. Additionally, observability and trialability are greatly increased by bringing user projects online and making the source code available for easy replication[5, 23, 28]

The Scratch team at the MIT Media Lab have completed a number of works dedicated to bringing computation to learning environments, including building an online community surrounding the Scratch platform[23]. In Growing Up Programming[31], Resnick et al. highlight a panel of software projects[15, 20, 13, 26, 32] designed to foster adoption of programming among young, novice innovators in elementary and secondary school by providing a contextual literature to which the target audience can easily relate. Additional centralized social coding communities include Github[28], where code heritage is heavily emphasized, as well as OpenProcessing[5] and OpenCode[8], websites for sharing Processing[29] sketches. Other present-day examples include programming environments in the form of games, such as Minecraft and Little Big Planet, targeting casual players, and OpenFrameworks[17], targeting advanced users who have outgrown Processing.

1.4.4 Visual Programming

There are a handful of existing tools that provide computer vision algorithms for use in interactive art. They can be roughly split into two groups, a group of tools that appear to be descendant from real time music software, and another group stemming form early work in descriptive graphic languages in John Maeda's group at the Media Lab.

The first group consists of Max Jitter[2], EyesWeb[10], Isodora[41], Puredata Gem[50], and vvvv[25]. All of these are data flow based programming environments in which the

programs are interpreted at runtime. Max MSP is produced by Cycling 74, a music label. Jitter is an extension to Max that allows video processing and real time effects. Puredata is an open source version of Max created by the original author of Max. EyesWeb is Max-styled environment for computer vision; however, unlike modules available for Max in which each module in the data flow contains its own user interface for customizing parameters, EyesWeb modules tend to have very fine granularity. For example, in EyesWeb, a tutorial describing motion detection is results in linking ten modules together. Isodora goes in the opposite direction and provides characteristically monolithic modules. For example, the output module has 16 variables that can accept inputs. The environments in this group are similar in concept and design to Apple's Quartz Composer[4].

The second group consists primarily of Processing[29] and its descendant, OpenFrameworks[17]. Both provide a thin wrapper around a set of low level drawing procedures and also higher level packaged libraries. Both also require the end user to program using text entry. In Processing's case, the language is Java, and for OpenFrameworks, it is C++.

1.4.5 Inspired by Vision on Tap

Mayhem[12], a project by Paul Dietz in the Applied Sciences division at Microsoft, is an application that translates camera and other inputs into discrete, usable actions. It is designed as an offline platform for simple input processing in which a variety of triggers, such as motion or keyboard events, can be used to activate a variety of outputs, including Twitter posts or sound. By limiting the user to simple if/then logic, the learning curve is made extremely shallow.

Chapter 2

Vision on Tap: Computer vision as a real time web service

Vision on Tap is a proof of concept in which we demonstrate an implementation of computer vision as a secure, live service on the Internet. We provide a basic publishing platform to distribute a real time vision application using simple widely available web technologies, including 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.

Vision on Tap aims to provide computer vision as a web service in a variety of commonplace situations, meaning any situation in which visual information retrieved via a user's camera can be used to create useful feedback. Several examples, such as parking a car and keeping a dog off of a couch, are shown in Figure 2-1.

Our system is inspired by systems such as [42, 43, 44], that use humans to perform tasks traditionally thought of as difficult for computer algorithms. However, we look beyond using

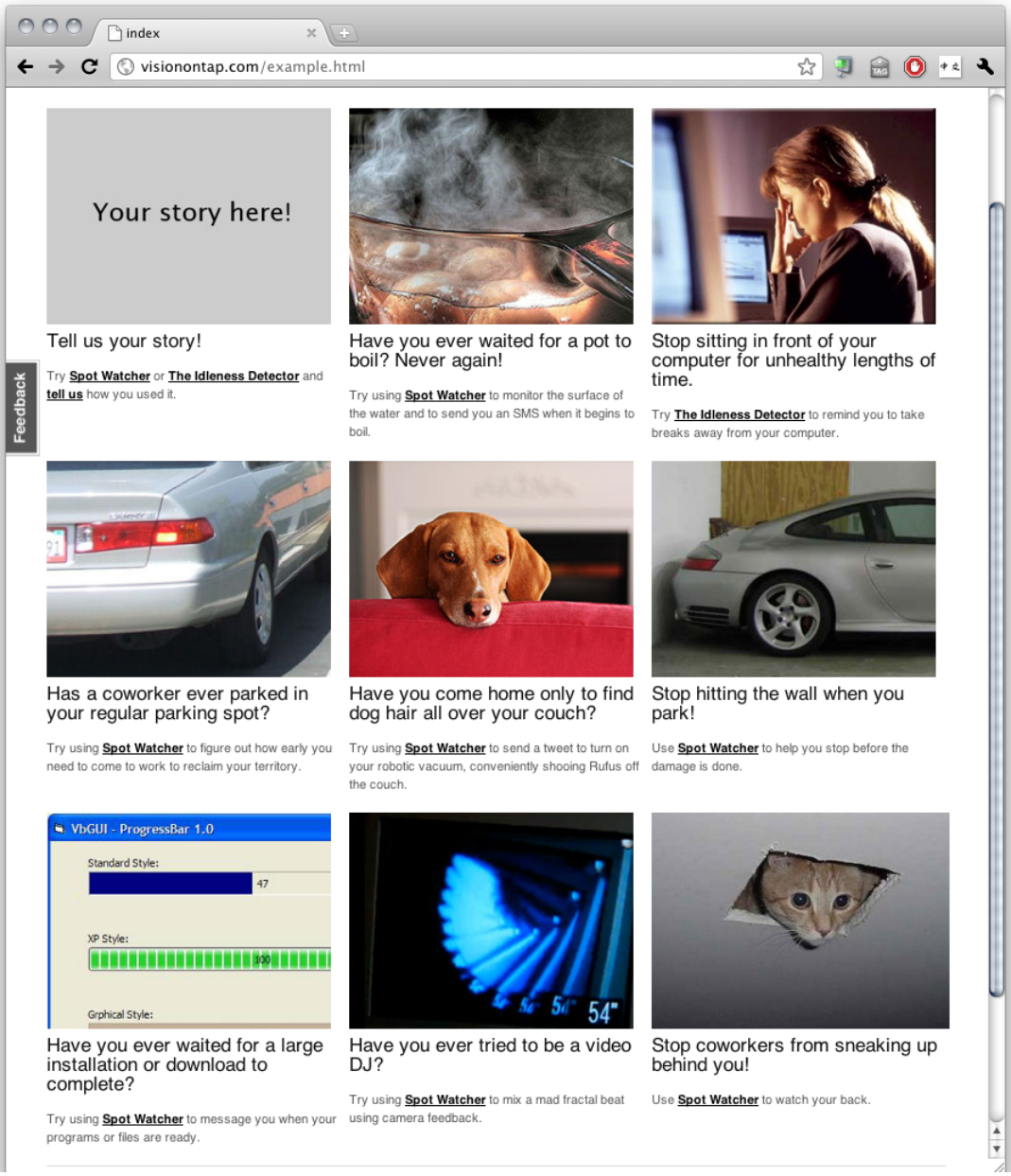

Figure 2-1: A number of applications are possible using the Vision on Tap platform, including but not limited to alerting the user when a pot begins to boil, reminding users to take breaks when doing sedentary activities, observing parking spot activity, triggering a robotic vacuum via Twitter to scare a dog off a couch, assisting parking in confined spaces, download progress monitoring, video fractal creation, and maintaining awareness of a user's physical environment by alerting the user of approaching individuals outside the user's immediate line of sight.

users as a problem-solving commodity resource and aim to involve them in the innovation process.

2.1 Design

The design for Vision on Tap provides a basic pipeline for distributing computer vision applications without requiring anything except a browser for both the publisher and the consumer. A publisher submits an application to the system via a simple web form and receives a URL in return. The URL can then be shared with users. The platform provides resources that are not typically available to offline platforms, including a mobile text messaging gateway, Twitter, and email.

Vision on Tap provides a minimalistic publishing interface, as shown in Figure 2-2 that is not intended for public consumption, but rather for proof of concept. The user is presented with a web form. The form consists of two text areas and a file upload field. One text area is the input for the code for the view, and the other is the input for code representing the back end logic. Alternatively, rather than coding the program into the presented text areas, the user might upload a compressed file containing the required files. Once the form is submitted, the user is provided with a URL where the system will publish the application. To share the application, users can simply share the URL.

2.2 Implementation

The system for distributed deployment of our computer vision applications follows a basic client-server architecture. In order to provide access to the client side camera as well as to avoid requiring the user to manually download a separate program, the core of the client is implemented as an Adobe Flash program embedded into a Hyper Text Markup Language (HTML) page. The server is written in Python and based on CherryPy, an object-oriented server library and framework.

Figure 2-2: The initial creation interface provided by the Vision on Tap system. The author may either edit the code directly in the web form for simple applications or upload the project source in zip format by specifying a file via the file choosing widget. After the form is submitted, the user is provided with the URL for their application.

2.2.1 Clients

We refer to a user's computer and everything it hosts as the client. As shown in Figure 2-4, 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 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.

2.2.2 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 2-5. 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.

2.2.3 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

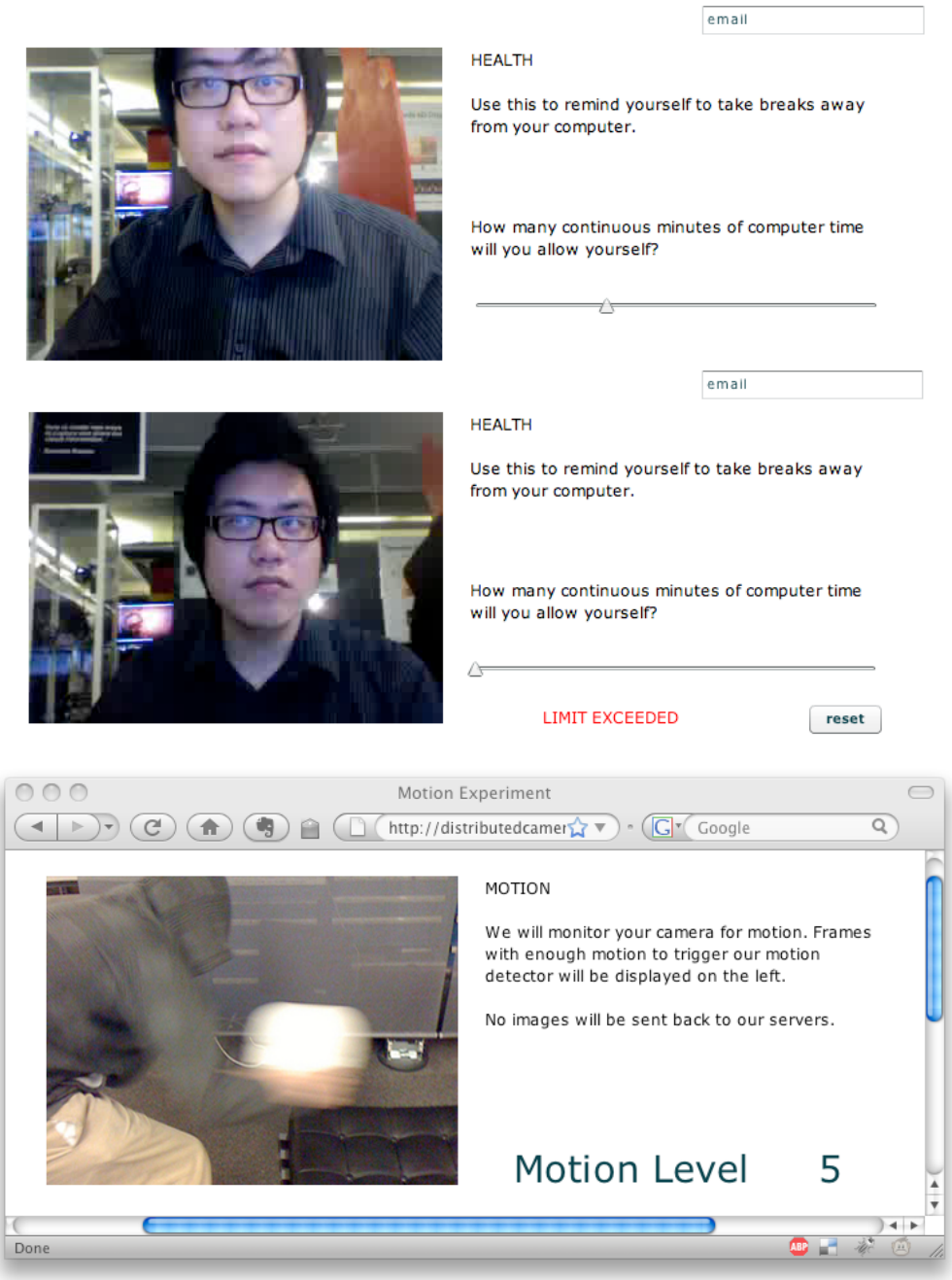

Figure 2-3: The interfaces of two sample applications. The first two images are of an application that reminds the user to take breaks away from their computer. Two timers are used in conjunction to provide the desired service. One thirty second timer is reset when motion above a certain threshold is 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.

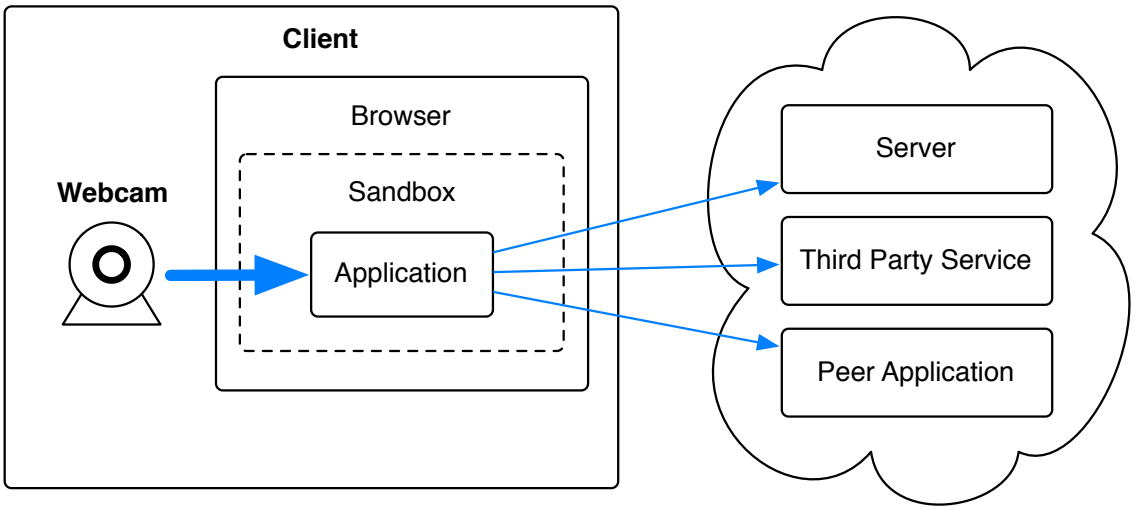

Figure 2-4: Data flow for user's client consumption. The width of the arrows is representative of the amount of data traveling between each component of the system. Most of the data is processed on the client, which publishes a summarized result set to one or multiple endpoints. For example, the camera may be capturing images at 30 fps. In most cases, no data is sent back to the server.

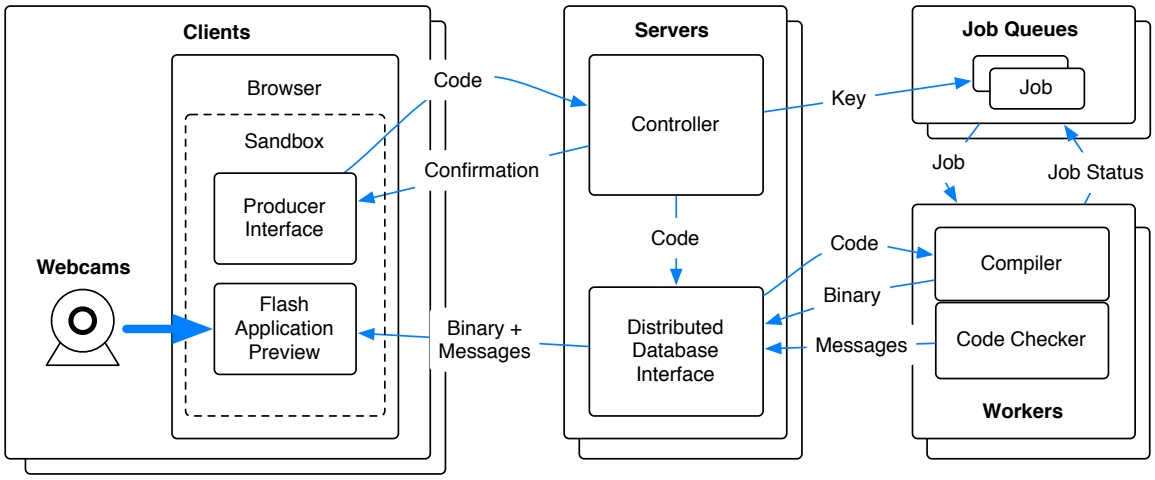

Figure 2-5: Data flow for publishing an application. A producer uploads code through an online interface into one of many AppEngine servers. From here, the code is stored in a distributed database and a compilation job is queued. As they become available, dedicated worker machines monitoring the job queue process compilation jobs, generating binaries, or in the case of errors or warnings, messages that will be presented to the client.

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.

2.2.4 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.

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.

2.2.5 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.

2.2.6 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 other instances of our service, but also third party services such as Twitter and Flickr, remote dedicated servers, and even other client applications.

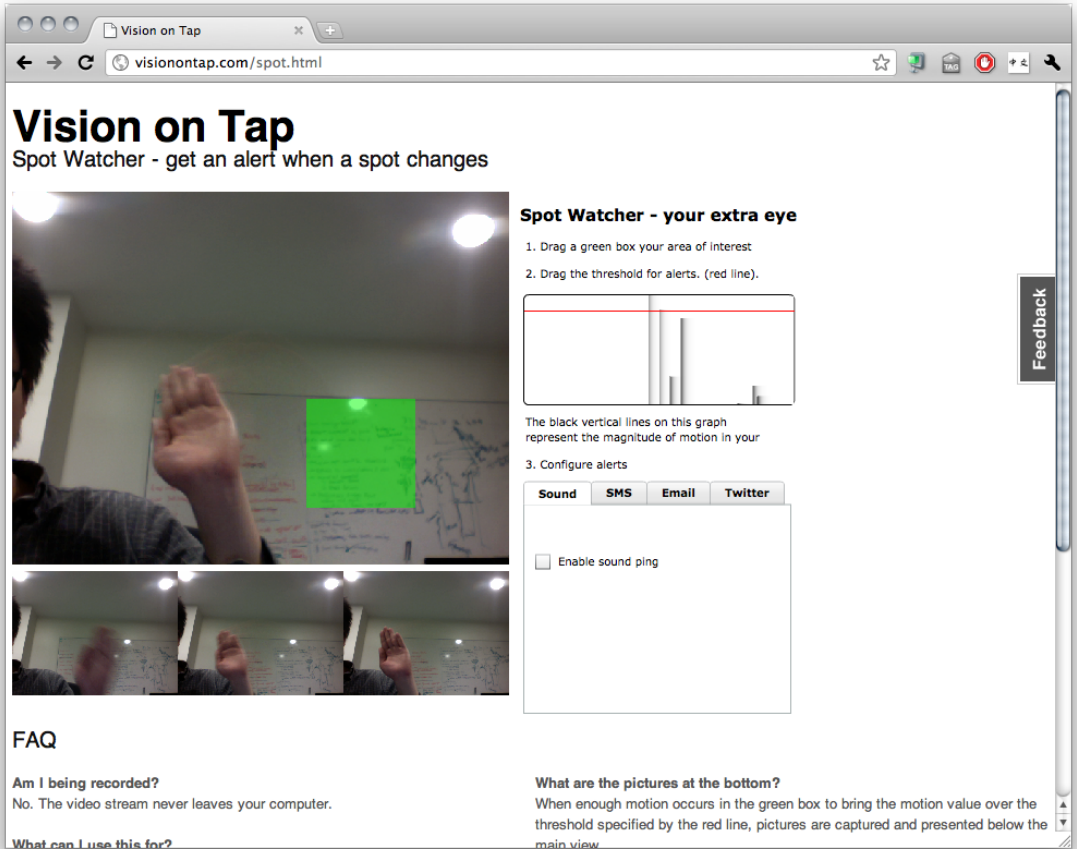

Figure 2-6: A screenshot of the Spot Watcher demo in action. There is a large, real time, camera view on left. The green box specifies a region of interest. The graph on the right displays the current motion level within the region of interest. When the motion level exceeds the level indicated by the horizontal line on the graph, the customizable trigger, located on the bottom right, is activated. Below the camera view are three small snapshots that appear when motion within the region of interest exceeds the threshold. The images represent the current camera view before, during and after the motion threshold is exceeded.

2.3 Applications

For demonstration purposes, we created a single implementation demonstrating an instance in the class of applications we made possible, an application named Spot Watcher, shown in Figure 2-6. Spot Watcher takes as input the live stream from a connected camera as well as a region selected using the mouse. The region is represented as a green box that the user can position over an area of interest. When movement was detected in the area of interest, it is translated into an integer between 0 and 100 and graphed. The graph features a threshold line that can be dragged vertically. When the graph exceeds the threshold, a customizable alert is triggered that outputs a combination of sound, twitter, email, alert, and text message. The SMS and email alerts may optionally include an image.

This single instantiation covers several applications as shown in Figure 2-1, including, but not limited to alerting the user when a pot begins to boil, observing parking spot activity, triggering a robotic vacuum via Twitter to scare a dog off a couch, parking assistance in small garages, download progress monitoring, and maintaining awareness of a user's physical environment by alerting the user concerning approaching individuals outside the user's immediate line of sight.

2.4 Proof of Concept Deployment and Collected Data

We implemented a simple 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. The URL encodes the outputs from each client. We asked users to simply navigate to the designated website and leave the browser program active for the extent of the study.

Sixty-eight individuals from the eight countries provided a combined 197 hours of video data using our application. We collected over 700,000 activity level 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`. The cleaned and anonymized data, both in

original form and summarized into timestamps, activity levels, and obfuscated IP addresses is available in the supplementary materials.

Chapter 3

Vision Blocks: A Visual Programming Environment for Vision on Tap

The original version of Vision on Tap was essentially a blank box within which users could write and submit code in a similar fashion to OpenCode[8]. The vast majority of users failed to create anything. In an attempt to remedy this, we created a more user-friendly interface, Vision Blocks.

Vision Blocks is the visual programming interface, or integrated development environment (IDE), for the Vision on Tap system. It is a descendent of the Scratch programming environment from the Media Lab's Lifelong Kindergarten group (LLK). Scratch is programming environment that appeals to children and allows them to create programs using an interlocking block metaphor that somewhat resembles puzzle pieces. Evelyn Eastmond, a former member of LLK, and John Maloney, a current member of LLK, created a Flash port of Scratch named DesignBlocks. An early version of DesignBlocks is the basis for Vision Blocks. Vision Blocks adds a real time vision pipeline and a set of compatible computer vision algorithms.

3.1 Visual Programming with Blocks

Vision Blocks relies on Scratch's block-based programming metaphor. In this approach to visual programming, the user selects blocks representing code snippets from a palette and arranges them on a canvas. As they are positioned on the canvas, the blocks snap together, providing an affordance to communicate block compatibility.

There are three types of blocks. Hats, stack blocks, and reporters. Hats form the beginning of a code section. They initialize the stack they are attached to. Stack blocks can be attached to hats as well as each other. Stack blocks may also have inputs in the form of text, integers, or booleans. In some cases, such as with the `if` block, stack blocks may wrap other stack blocks. Reporters generate boolean or integer values and can be used as inputs to other blocks. Some reporters have inputs within which other reporters can be nested. The relationship between the various types of blocks and the similarity between Scratch and Vision Blocks is illustrated in Figure 3-1.

3.2 Design

3.2.1 Creation Interface

The Vision Blocks IDE, as shown in Figure 3-2 is segmented into four main elements. From left to right, the palette menu allows the user to select a functionally themed palette, the palette pane presents a group of blocks to the user, the workspace provides an area for manipulating the blocks, and the canvas presents an output from the user's program. Above the canvas are controls where the user can specify a title and publish the application. The IDE loads in the browser and provides tools organized into program control (Start, Logic), inputs (Video, mouse in Logic), output (Draw, Alert, Move).

There are a number of departures from the design of the DesignBlocks interface. The palette menu is labelled using text rather than images. We found that although text required knowledge of the English language, it was more readily distinguishable when compared with

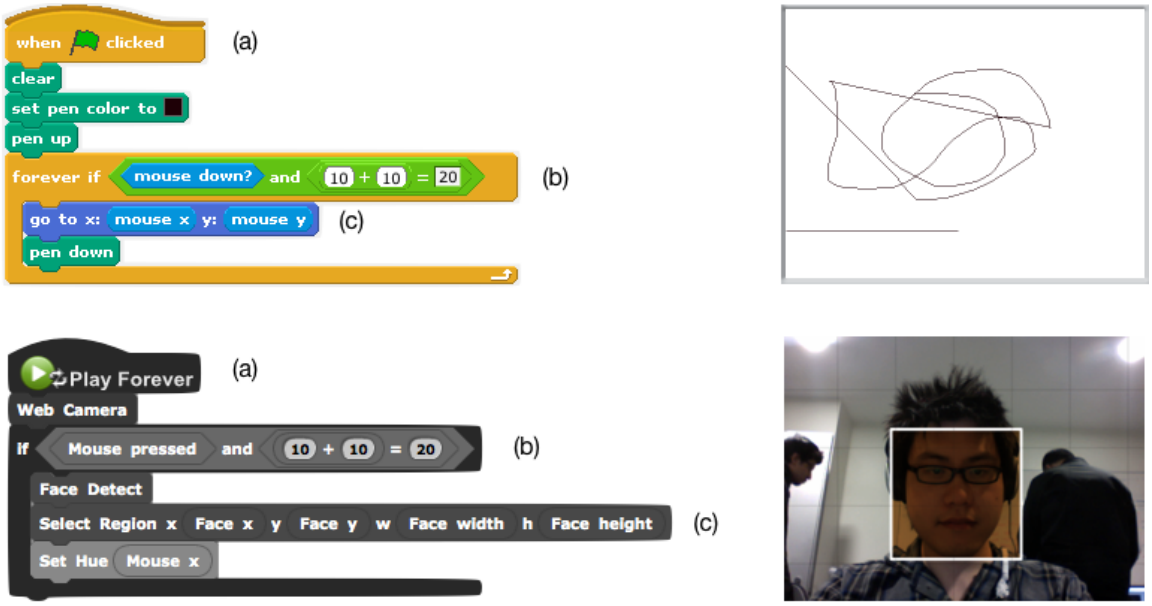

Figure 3-1: Scratch Blocks code that implements the ability to draw a line (top). Vision Blocks code that implements the ability to track a face in a camera image and surround it with a translucent rectangle the color of which is based on the current position of the mouse (bottom). This figure shows the similarity between Scratch Blocks and Vision Blocks and illustrates the underlying blocks metaphor. (a) A hat block acts as an initializer for the visual code stack. (b) A stack block takes a boolean reporter as an argument. The input for this stack block illustrates concept of nested reporters. (c) Some blocks accept multiple reporters.

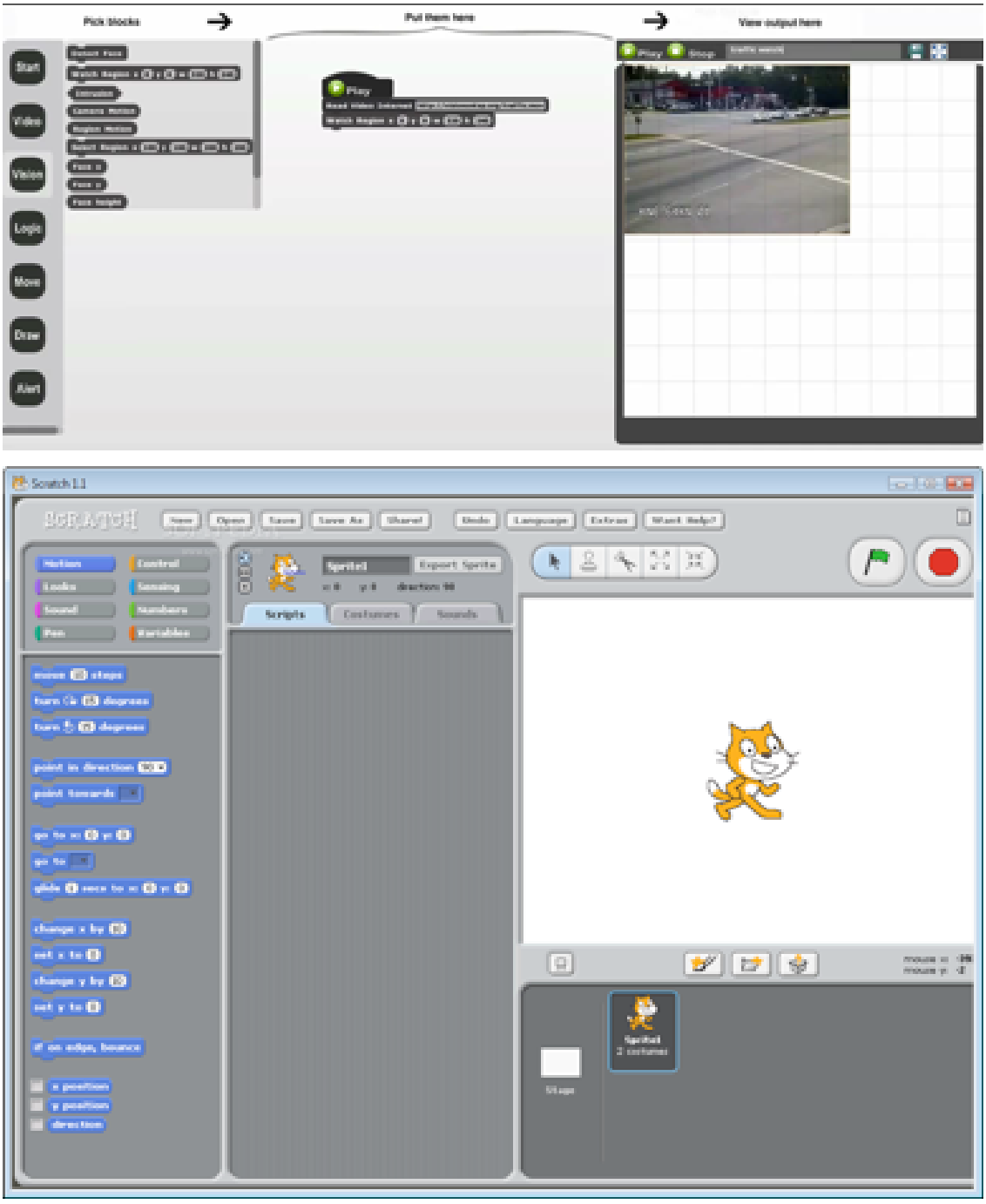

Figure 3-2: A screenshot of the Vision Blocks interface showing the loading of video over the web (top). The Scratch interface showing a cartoon cat sprite. (bottom).

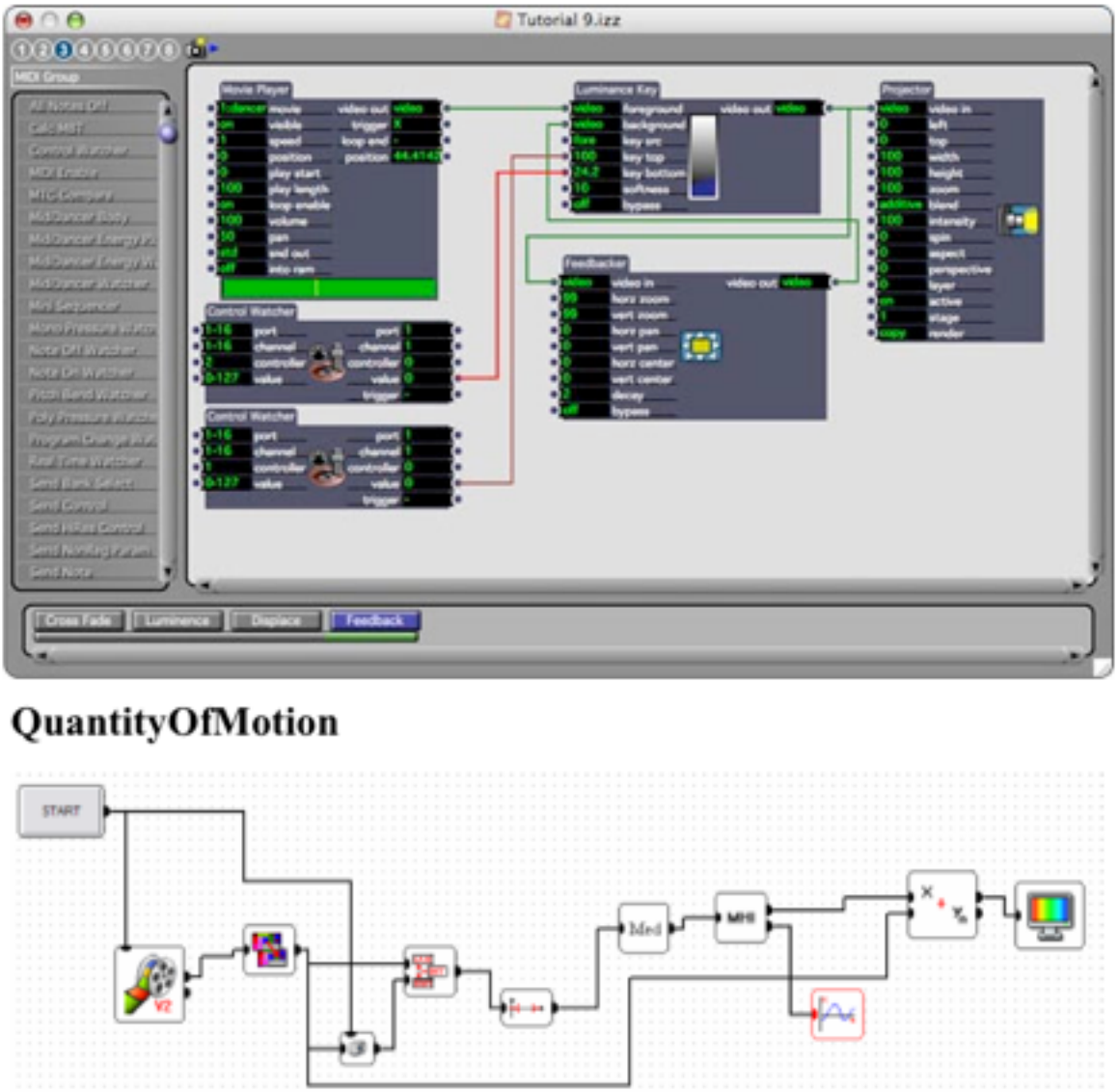

Figure 3-3: Isadora (top). Eyesweb (bottom).

images in initial prototypes. The interface also features a large label at the top indicating the intended workflow, in which users are expected to "pick" blocks, "put" them into the workspace area, and finally "view" the output on the canvas. Additionally, the metadata area, in which users can view cursor coordinates and other debugging information, was removed.

3.2.2 Vision Pipeline

Typically, in a computer vision system, the input is provided via a natural image in the form of a photograph or video. The source of this is usually a camera. From this initial input, the goal of a computer vision system is to understand the scene, extracting semantic information that can be output via a low dimensional expression, typically a coordinate, a label, or a map of some type. Between the high dimensional input and the low dimensional output, a number of subsystems might be put in place to gradually reduce the complexity of the input. For example, a raw image input might be first placed through a blurring filter to remove noise, then through a skin detection algorithm to remove portions of the image that do not concern a skin-colored target. This map might then be used to segment out a portion of the scene that might be used for further processing in which the image is translated into just a few numerical outputs, face recognition, for example. In each step of a vision pipeline, each stage typically reduces the complexity of the data being passed along.

When composing computer vision or image processing elements, the order in which they are applied is important. For example, applying smoothing filters and then sub-sampling an image produces a different effect than first sub-sampling and then smoothing. Similarly, detecting an event at one stage of a pipeline than at another stage can produce differing results.

A Necessary Departure from Scratch Blocks

Given the nature of the vision pipeline, it is necessary for us to depart from a few of the metaphors provided by the Scratch Blocks framework. Consider the mouse input provided by Scratch. The x and y coordinates of the mouse are made available upon access to the respective "mouse x" and "mouse y" blocks. The face detection blocks, which are analogous to the mouse input blocks in that, as part of a larger set of outputs, provide a "face x" and "face y" output. However, unlike the mouse outputs, the face outputs require the face detection block to be initialized before they can be accessed. This behavior is a result of the face detection being designed to be triggered at a stage in the vision pipeline that isn't known in advance. The detection could be triggered immediately on raw camera input data, or it could be used after a skin detection filter. The order in which it is triggered is important and shouldn't be forced to be where the data is accessed. By separating the triggering of the face detection block and the use of its outputs, the user can place the face detection block in any place within the vision pipeline without having to worry about triggering the block upon access to its outputs.

Thus, we introduce to the Scratch Blocks metaphor the existing concept of parent and child blocks. When the parent block is triggered, the state of the program at that point in time is used as the input to the parent block, and the child blocks are populated with the output from the parent block. The child blocks can then be repositioned without re-triggering the parent block and overwriting the desired output data.

3.2.3 Component Choices

We would like to recognize Ramesh Raskar for his clear determination of target applications and specifications for the components necessary for their realization. These critical aspects of the project would be lacking without his experience and leadership.

The overarching theme of the interaction between the components added to the Scratch system is a vision pipeline of three stages, input, processing, and output. Input blocks

bring in information to populate a common staging area. Processing blocks observe the information in the staging area and provide reduced dimensionality outputs, such as coordinates or scalar values. Finally, output blocks, in conjunction with the remainder of the Scratch system, provide a means for the process to communicate with users who are not actively observing the direct feedback from the processing blocks.

The motivating applications include allowing a grandmother to keep watch over her grandchild without having to rely on partitions designed to confine toddlers into specific regions of the residence as well as monitoring neighborhood traffic for vehicles that might be exceeding legal speed limits. These motivating applications made the components apparent. There would need to be remote and local video input, pixel-based image analysis, skin filters, remote notifications, and the ability to record events.

3.2.4 Orthogonal Visual Feedback

An early design decision was to separate the visual feedback from the internal representation of the image. This way, a user can have real time visual feedback that a particular block is processing the inputs in the expected manner without affecting the visual inputs for the next block in the pipeline.

3.2.5 Website

A website was created that allows users to create and share their own applications. There are three primary sections.

The front page, as shown in Figure 3-4, includes three demonstrations to help users get started with the system. Each demonstration includes an annotated video tutorial as well as a ready-to-run example program that allows users to experiment with a previously prepared, working system.

The "browse" page acts as a directory for users' saved applications. As shown in Figure 3-5, each application is represented by a screenshot of the canvas taken when the user saves the

program as well as the title that the user specifies. When a user clicks on an entry in the browse page, the IDE loads with the code saved from the authoring user's session.

The "create" page houses the Vision Blocks IDE. Here, users can create, title, and share their creations. Additionally, this view loads with pre-populated blocks when users select a project to load from the "browse" page.

When an application is saved, a new page is created to host it. At the bottom of the page, the public is allowed to leave comments. Users who leave comments on a project are notified when additional comments are left. By following this notification scheme, the comment system attempts to engage past commenters in the present conversation. From the application page, users are also allowed to edit and re-save the application, created a new instance and a new application page.

Additional pages of the website include an "about" page that describes the goals of the project, a frequently asked questions page (FAQ) that features a troubleshooting guide and addresses privacy concerns, a "documentation" page that describes the functionality of each block in the Vision Blocks interface, a "contact" page that lists contact emails, and an "acknowledgements" page.

3.3 Implementation

The components provide an array of inputs, processing, and output options for the casual user. The components are primarily implemented using a Flash front end tied to a back end implemented in Ruby on Rails. The development environment is standardized on Adobe Flash Builder and git to coordinate development efforts.

We would like to recognize Abhijit Bendale for creating several of the components meeting the array of capabilities specified by the motivating applications, as well as going beyond the motivating applications to create components that were applicable to a wider range of situations, and Kshitij Marwah for thoroughly debugging the components and reimplementing them when necessary. The following describes components that were completed as

of December, 2010. Since this date, several more components have been started, including color segmentation, marker-less tracking, and recording.

3.3.1 Components

Input

Local Camera: If there is a local camera attached, the user may use it to provide either still frame or video input at 320 x 240 pixel resolution. This block, when triggered, draws a single frame from the camera to the canvas and also populates an internal camera buffer that may be used for further processing.

Remote Camera: Given a URL in the form of an RTMP stream, the user may use a remote camera for frame or video input. The internal behavior is similar to the local camera block, except that the resolution is determined by the stream.

Online Videos: Given a URL, online video sources provided in MP4 or Quicktime container format encoded with H.264 as well as Flash Video container format encoded with either the ON2 VP6 or Sorenson Spark video codecs may be used for video input. The internal behavior is similar to the remote camera block.

Online Images: Given a URL, online images in JPEG, PNG, GIF, or SWF format may be used as image input. The image is displayed at its native resolution on the canvas.

Processing

Face Tracking: The face tracking is provided via the Viola-Jones algorithm. The implementation is provided via an OpenCV-based ActionScript 3 library named Marilena.

Motion Sensing: Efficient frame-wise motion sensing is provided by subtracting previous and current video frames or regions of interest within a video frame using a difference filter.

A preliminininary implementation relied on accessing the individual camera pixels, which caused a noticeably negative impact on performance.

Motion Tracking: Like the motion sensing block, The motion tracking block keeps a reference to the previous frame and computes a difference between the two. In the case of the motion tracker, the image resulting from the difference filter is processed further. The contrast is increased to remove regions with less movement, a blur filter is applied to mitigate noise, and the image is thresholded to create a movement map in which light pixels represent movement that surpassed the threshold and locations in which movement did not pass the threshold are dark. A bounding box is calculated and placed around the light pixels in the map, and the center is provided as x and y coordinates. The bounding box is indicated to the user via the canvas.

Edge Detection: This block applies a Sobel filter to the current frame. First the frame is converted to greyscale, then vertical and horizontal Sobel operators are applied. The result is an image whose edges are dark and whose smooth regions are light.

Skin Detection: Human skin color tends to fall into a narrow region in YCbCr space. By selecting a segment in the color space according to this YCbCr region using a simple filter implemented using Adobe's Pixel Bender framework, skin detection is completed in real time.

Output

Mobile text messaging: The text messaging block takes two inputs: a phone number and a message. When triggered, a text message is sent to the specified number. This is implemented via a server-side function call to a third party mobile text messaging gateway.

Email: The email block is similar to the text messaging block, but takes as input an email address instead of a phone number. This is implemented via a server-side function call to an outbound email provider.

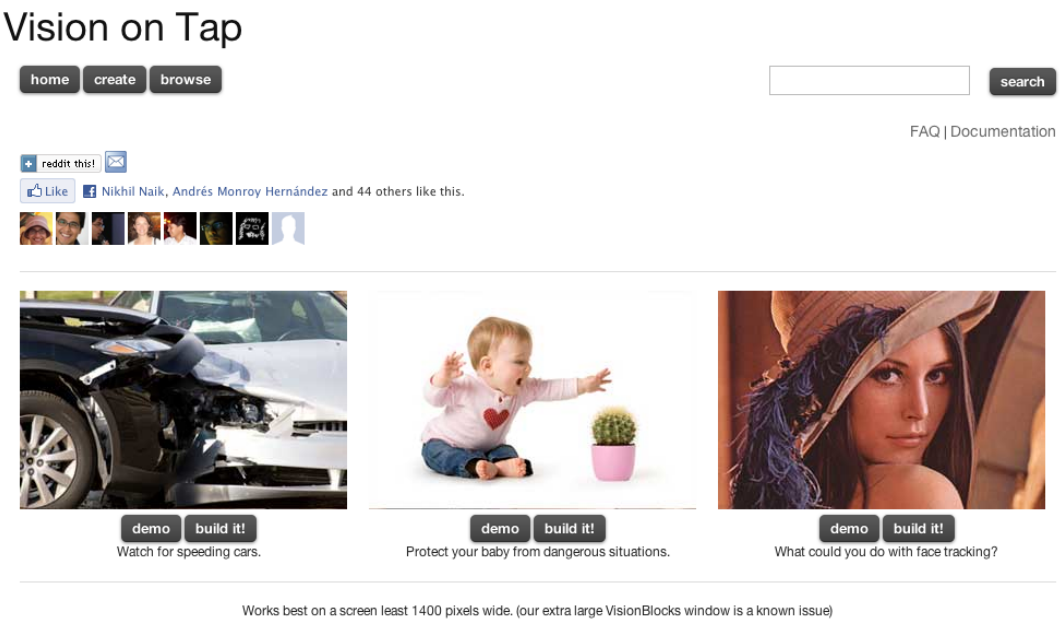

Figure 3-4: A screenshot of the website's front page. The user is initially presented with three optional tutorials for familiarization with the platform.

Audio: When this block is activated, a preset sound plays. This is implemented by reading in a URL, currently hard coded, for the sound, downloading the target file, and playing it using the ActionScript sound API.

3.4 Applications

The applications afforded by the Vision Blocks system are very similar to those presented in the original Vision on Tap system without the Vision Blocks interface. However, when compared to Vision on Tap's text areas and code uploading, the visual programming interface incorporating the blocks metaphor provides a considerably more accessible means of creating applications, and is thus applicable to a wider audience.

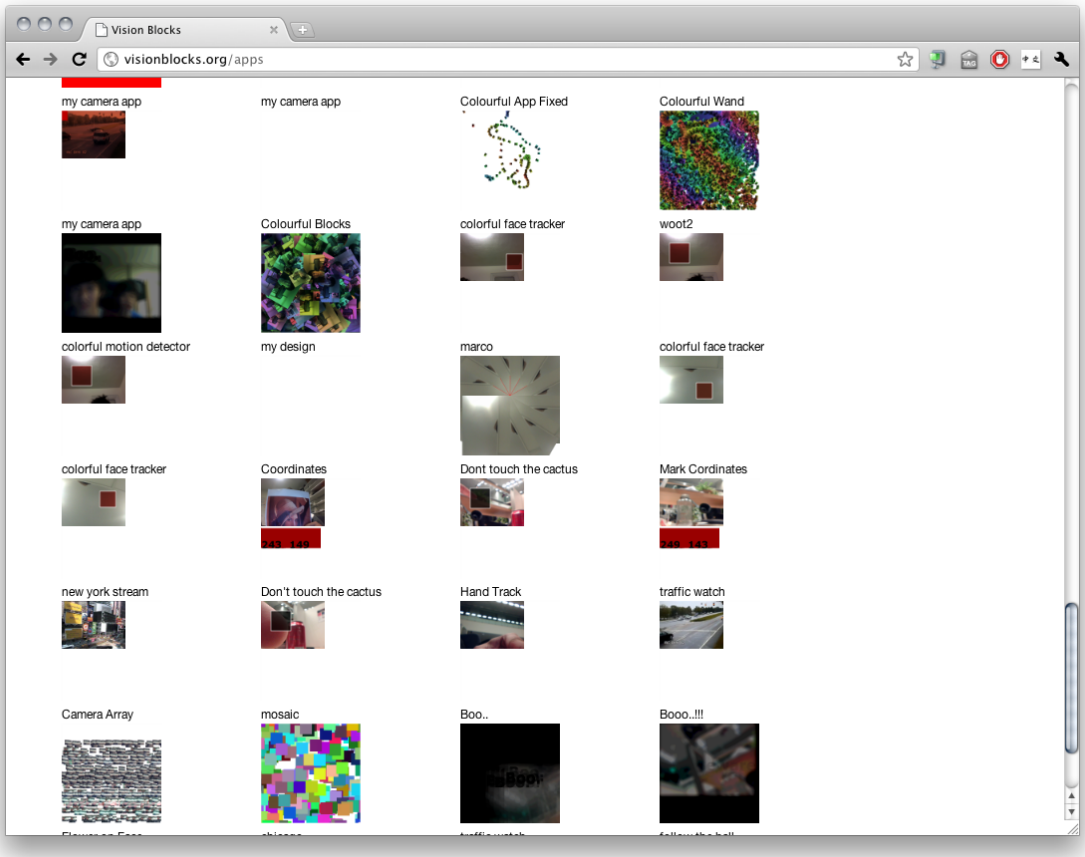

Figure 3-5: A section of the website's browse page featuring several published applications. When a user saves an application, a screenshot of the canvas is saved as well. The saved canvas image, in conjunction with the user-specified title, are presented in the "browse" section of the website. Clicking on an entry loads the corresponding application into the Vision Blocks IDE. Several of the applications pictured here were created by Phu Nguyen and Martin Martinez Rivera.

Programming	Vision	Time	Intuitiveness	Utility	Ease of Use	Learning	Satisfaction
no	no	6.88	4	3	2	2	3
no	no	10	2	2	2	1	1
no	no	5.63	2	1	2	1	1
no	no	10.83	4	4	2	3	3
no	no	23.33	3	3	2	1	3
no	no	2.5	3	2	3	2	2
no	no	2.5	5	4	3	3	4
no	no	9.38	2	3	2	1	2
no	no	2.5	5	3	3	3	4
yes	no	2.5	3		3	3	4
yes	no	2.5	3	2	2	3	3
yes	no	2.5	4	4	3	3	4
yes	no	2.5	4	4	2	2	3
yes	no	2.5	3	4	3	3	3
yes	no			1		3	3
yes	no	2.5	3	2	2	3	
yes	no	2.5	4	2	3	2	2
yes	no	15	2	1	1	1	
yes	no	2.5	3	3	2	1	2
yes	no	2.5	3	3	2	2	3
yes	yes	2.5	2	2	2	1	3
yes	yes	2.5	3	4	3	2	2
yes	yes	40	3	2	1	1	3
yes	yes	2.5	4	2	2		1
yes	yes	2.5	3	2	1	1	2
yes	yes	2.5	3	4	2	3	3
yes	yes	2.5	4	3	2	2	2
yes	yes	2.5	4	3	2	3	4
yes	yes	2.5		2			
yes	yes	7.5	3	2		3	4
yes	yes	5	3	4	2	2	3
yes	yes	2.5	4	3	2	3	4
yes	yes	2.5	4	4	3	2	3
yes	yes		1	1	1	1	1
yes	yes	2.5	3	3	2	2	3
yes	yes	2.5	5	5	3	3	4
yes	yes	10	3	4	3	3	3
yes	yes	2.5	4	3	3	3	2
yes	yes	2.5	3	2		2	4

Table 3.1: A numerical representation of survey results. The first 9 users have neither programming nor computer vision experience. The next 11 users have programming experience, but no computer vision experience. The remainder have both programming and computer vision experience.

Programming	Vision	Time	Intuitiveness	Utility	Use	Learning	Satisfaction
no	no	7.65	67	56	78	63	64
yes	no	3.75	64	52	78	79	75
yes	yes	5.5	66	58	77	73	71

Table 3.2: A summary of 39 user surveys. The group sizes are 9 without programming experience, 11 with programming experience but no computer vision experience, and 19 with both programming and computer vision experience. The full survey and numerical mapping are detailed Appendix A.1. Time is in minutes. Numerical values other than time are normalized between 0 and 100, 100 being most positive. The full survey results are available in the supplementary materials.

3.5 User Study

The evaluation consisted of asking users to participate in a three part trial in which they would create applications and provide feedback in a questionnaire. On the front page were three tutorials and corresponding demos the users could use to familiarize themselves with the system. One involving motion detection, another for blob tracking, and a third for face tracking. The tutorials consisted of videos in which the users could watch the demonstration applications being put together one piece at a time. Each action was annotated with a caption describing the purpose of the block being placed, as shown in Figure 3-6. To get visitors, the opportunity to participate in the user study was delivered via email. This user study was approved by the Committee On the Use of Humans as Experimental Subjects (COUHES).

The survey, detailed in Appendix A.1, included questions concerning programming and computer vision background, allowing us to divide the user group into three segments: users without programming background, users with programing background but no computer vision background, and users with both programming and computer vision background.

3.5.1 Results

As shown in Table 3.2, averaged user metrics between the three interest groups were tightly clustered. The users without programming experience tended to face a steeper learning curve and have lower overall satisfaction than the more experienced users. Users without

Visits	6805
Unique Visitors	1490
Application Authoring Attempts	555
Successfully Published Applications	20
Surveys Filled	39

Table 3.3: Participation statistics for the Vision Blocks trial. Notably, of those users who started building applications, less than 4% published their applications.

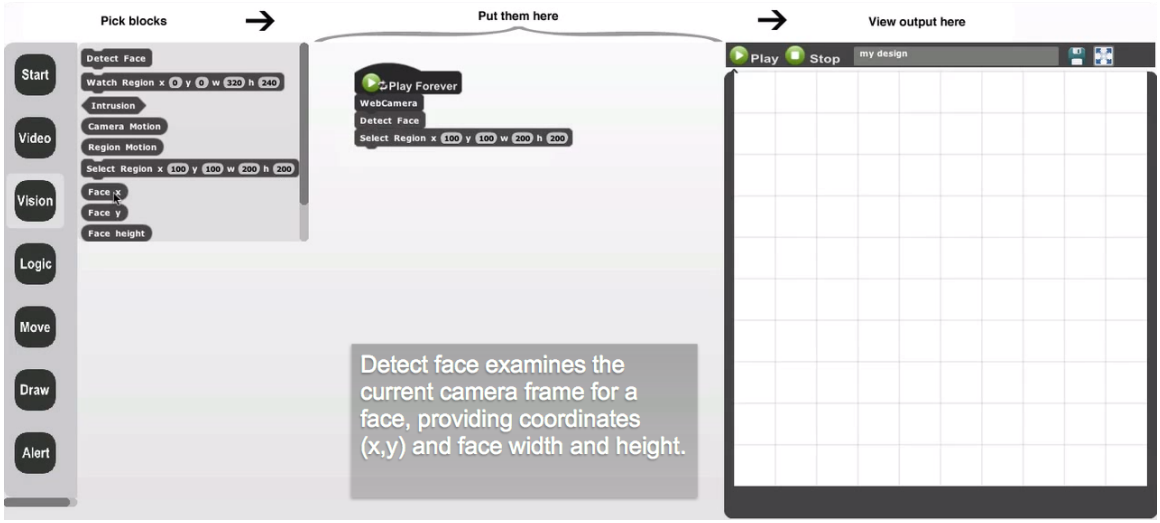

Figure 3-6: A screenshot from a tutorial video used in the user study for Vision Blocks. In this image, the annotations explaining the face detection block are displayed. Tutorial videos are available in the supplementary materials.

programming experience also took the longest to complete the tasks. Notably, users with programming experience but no computer vision experience completed the tasks the fastest. Full survey results are available in the supplementary materials.

3.5.2 Observation and Feedback

As part of the survey, users were allowed to respond freely with any thoughts they had concerning Vision Blocks. Overall feedback was a mix of praise, problems, and suggestions for improvement. Below, similar responses are grouped together and organized in accordance to relevance with regards to the interface, the curriculum, technical issues, and the overall

approach.

Interface

Although most users grasped the notion that the blocks were meant to snap together, others became easily confused and attempted to create programs in which blocks were not attached to one another.

In a few cases, there was confusion or negative opinions over behaviors inherited from Scratch or with visual programming in general.

- "I dislike the drag&drop style. maybe it's pretty and cool for demos to non-users, but it is ultimately a slow and inflexible way of doing things."
- "If I draw a line of a certain length, then reduce the length, there is a residual line where the last one I drew was."

More advanced users saw the system as a convenient rapid prototyping tool that would be more useful if it allowed users to convert the applications into code.

- "It could be great if you generate a matlab or python code from the 'algorithm' that you draw so more advanced users can do some more hardcore stuff. In that case your app would be more appreciated from a far bigger community."

Curriculum

A common suggestion was that there should have been narration over the tutorial videos. At least one user had to watch the videos multiple times before understanding what was going on.

- "The demo videos would have been much more explanatory if there was voice over explaining the application."

- "I think there should be some kind of narrative over the demo videos."

Additionally, users found a greater need for detailed documentation or more of an existing canonical literature to draw from.

- "There should be better documentation..."
- "...it would be nice to at least have tool-tips..."
- "...it needs a bit more documentation, OR, more templates to be inspired from (I emphasize on this last one). because that could help people get it in few minutes."

Technical Issues

In some cases, users faced technical difficulties. In several cases, this was due to Flash.

- "I was unable to do the baby and face demo because I could not select Adobe Flash Player Settings to allow camera and mic access."
- "The webcam block doesn't work."

In other cases, this was due to programming errors in the IDE or incompatibility with certain users' systems even though the functionality was present.

- "Stop button doesn't work."
- "Full screen button is broken."
- "The boxes should snap in place and allow easy movement if the user is unhappy with the current position."

Overall

In most cases, users understood the purpose of the system and enjoyed what the system had to offer.

- "This was cool. I liked the face tracking one the most."
- "I like the whole concept of module blocks that can help you make a program..."
- "...a great way of introducing young kids to the fascinating field of computer vision and the kind of cool things automated processing techniques can do."

Chapter 4

Conclusion and Future Work

We have presented a system that provides a means for users with neither programming nor computer vision knowledge to create and share computer vision applications. Users visiting our website are not required to install any extra applications to either create or consume online computer vision applications.

This brings computer vision capabilities to a wider audience including those who might not have the skills traditionally thought of as necessary for creating and distributing computer vision applications.

In its first iteration, our system required advanced programming knowledge to create even the most basic online computer vision applications. Our publishing interface, a minimalistic web form, provided a means to program, but the interface was too minimal and not suitable for non-expert use. It was usable by advanced users or users with access to an external development environment. Additionally, we found that the server-side component of the majority of proposed computer vision applications were unnecessary.

For the second iteration, we concentrated on improving the user interface for creating applications and building a more robust client application that did not rely on a server for the majority of tasks. However, despite utilizing a visual programming environment, we still found that a minority of visitors decided to contribute applications to the system.

During our user study, we were surprised to find that users became easily frustrated with the interface and more often than not failed to produce any applications after the tutorial. Users faced a large number of technical difficulties, many of which were dependent on their particular system configuration. Also, out of 555 trials, only 20 users published their creations. In future work, it would be interesting to compare our user application publication

rates with the application publication rates of other websites that are built upon user contributions, such as Scratch[23], to understand what we can expect with regards to creator participation levels.

Our progression of studies show a need to further simplify the interface for end users while providing additional code-based interfaces for advanced creators. A new effort, currently titled CloudCam.tk is a project working towards a simpler interface for end users and more flexibility for publishers. It mixes the concepts presented in Vision on Tap and Vision Blocks to provide a split environment in which consumers maintain drag and drop simplicity without having to understand complex program control flow involving loops or conditional statements. Unlike Vision Blocks, the blocks in CloudCam.tk will be monolithic and will not feature any control flow outside each monolithic block. There will only be a single executable path, in the fashion of Apple Automator[3]. Additionally, user-created applications will be exportable as code that can be easily reused by advanced users.

Appendix A

Vision Blocks User Study Survey

A.1 Survey

The user survey was designed by Abhijit Bendale and Ramesh Raskar, who also served as Principle Investigator. Users were recruited with the help of Kshitij Marwah. Facilitators were certified by COUHES before the administration of this user study.

The survey is replicated below. A portion of the questions provided multiple choice answers, which are included in-line. A numerical mapping, indicated in parenthesis before each answer, is assigned in order to further analyze the results in a uniform manner.

Have you programmed a computer vision algorithm before?

yes / no

Do you have previous programming experience?

yes / no

Age

Time taken to build the Motion Tracking example

(2.5) less than 5 minutes
(7.5) 5 - 10 minutes
(15) 10 - 20 minutes
(40) 20 - 60 minutes

(did not finish) 60 or more minutes

Time taken to build the Face Detection example

(2.5) less than 5 minutes

(7.5) 5 - 10 minutes

(15) 10 - 20 minutes

(40) 20 - 60 minutes

(did not finish) 60 or more minutes

Time taken to build the Video Streaming example

(2.5) less than 5 minutes

(7.5) 5 - 10 minutes

(15) 10 - 20 minutes

(40) 20 - 60 minutes

(did not finish) 60 or more minutes

Time taken to build the Personal Demo example

(2.5) less than 5 minutes

(7.5) 5 - 10 minutes

(15) 10 - 20 minutes

(40) 20 - 60 minutes

(did not finish) 60 or more minutes

Are you concerned about your privacy in the context of Vision on Tap

yes / no / don't know

Were the "vision blocks" intuitive to use?

least intuitive – 1 2 3 4 5 – most intuitive

For the given set of demos, would you have considered the following user interface / software?

OpenCV / Matlab / Scratch / Processing / Other:

How many friends did you invite to use this website?

none / 1-5 / 5-10 / 10-20 / More than 20

Did you build on a Vision on Tap program created by someone else for your personal app?

Yes / No

What do you think about the usefulness of the website?

(5) It is perfect! It does everything I would expect it to do.

(4) It is great! I could save time.

(3) It is useful. It meets my needs.

(2) It has some flaws

(1) I didn't like it

Does this website provide you with fast enough results in terms of speed of execution?

Yes / No / Other:

Is the website easy to use?

(3) I like it. It is easy and simple to use.

(2) It could be better.

(1) I didn't like it.

Other:

Please explain

Is the website easy to learn? (Minimum effort needed to use)

(3) I can use it without written instructions.

(2) It requires the fewest steps possible to accomplish what I want to do with it.

(1) It has a steep learning curve

Are you satisfied with this method of doing computer vision?

(4) It is wonderful. I would recommend it to a friend.

(3) It is fun to use. I feel I need to have it.

(2) I am satisfied with it. It works the way I want it to work.

(1) It is the worst thing invented

Any other comments and feedback

Bibliography

[1] *U.S. Photo Industry 2010: Review and Forecast Report*. PMA Marketing Research, 2010.

[2] Cycling 74. Max msp jitter. http://cycling74.com/products/maxmspjitter/, 5 2011.

[3] Apple. Automator. http://support.apple.com/kb/ht2488, 5 2011.

[4] Apple. Quartz composer. http://developer.apple.com/graphicsimaging/quartzcomposer/, 5 2011.

[5] Sinan Ascioglu. Openprocessing. http://openprocessing.org, 5 2011.

[6] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 1st edition, September 2008.

[7] Steve Branson, Catherine Wah, Boris Babenko, Florian Schroff, Peter Welinder, Pietro Perona, and Serge Belongie. Visual recognition with humans in the loop. In *European Conference on Computer Vision (ECCV)*, Heraklion, Crete, Sept. 2010.

[8] Kyle Buza and Takashi Okamoto. Code sharing on the web - two approaches. *Workshop on Visual Interfaces to the Social and the Semantic Web (VISSW)*, 2009.

[9] Jeffrey Byrne and Jianbo Shi. Visym. http://visym.com, 5 2011.

[10] A. Camurri, B. Mazzarino, and G. Volpe. Analysis of expressive gesture: The eyesweb expressive gesture processing library, 2003.

[11] K. Chiu and R. Raskar. Computer vision on tap. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 31–38. IEEE, June 2009.

[12] Paul Dietz. Mayhem. Personal Communication, http://makemayhem.codeplex.com, 5 2011.

[13] Ivan Alex Games and Kurt Squire. Design thinking in gamestar mechanic: the role of gamer experience on the appropriation of the discourse practices of game designers. In *Proceedings of the 8th international conference on International conference for the learning sciences - Volume 1*, ICLS'08, pages 257–264. International Society of the Learning Sciences, 2008.

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

[15] Caitlin Kelleher, Randy Pausch, and Sara Kiesler. Storytelling alice motivates middle school girls to learn computer programming. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1455–1464, New York, NY, USA, 2007. ACM.

[16] Brian Krausz, Scot Kennedy, and Joe Gershenson. Gazehawk. http://gazehawk.com, 6 2010.

[17] Zach Lieberman and Watson Theo. Openframeworks, http://openframeworks.cc.

[18] Feng Liu, Yu H. Hu, and Michael 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.

[19] Thomas Lochmatter, Pierre Roduit, Chris Cianci, Nikolaus Correll, Jacques Jacot, and Alcherio Martinoli. SwisTrack - a flexible open source tracking software for multi-agent systems. *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 4004–4010, 2008.

[20] Matt MacLaurin. Kodu: end-user programming and design for games. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, FDG '09, pages 2:xviii–2:xix, New York, NY, USA, 2009. ACM.

[21] John Maloney, Leo Burd, Yasmin B. Kafai, Natalie Rusk, Brian Silverman, and Mitchel Resnick. Scratch: A sneak preview. In *C5*, pages 104–109. IEEE Computer Society, 2004.

[22] Dan Maynes-Aminzade, Terry Winograd, and Takeo Igarashi. Eyepatch: prototyping camera-based interaction through examples. In Chia Shen, Robert J. K. Jacob, and Ravin Balakrishnan, editors, *UIST*, pages 33–42. ACM, 2007.

[23] Andrés Monroy-Hernández. Scratchr: sharing user-generated programmable media. In *Proceedings of the 6th international conference on Interaction design and children*, IDC '07, pages 167–168, New York, NY, USA, 2007. ACM.

[24] Joseph Mundy. The image understanding environment program. *IEEE Expert: Intelligent Systems and Their Applications*, 10:64–73, December 1995.

[25] J. Oreg, Max Wolf, Sebastian Gregor, and Sebastian Oschatz. vvvv. http://vvvv.org/, 5 2010.

[26] Ken Perlin, Mary Flanagan, and Andrea Hollingshead. The rapunsel project. In Gérard Subsol, editor, *International Conference on Virtual Storytelling*, volume 3805 of *Lecture Notes in Computer Science*, pages 251–259. Springer, 2005.

[27] P. Perona. Vision of a visipedia. *Proceedings of the IEEE*, 98(8):1526 –1534, aug. 2010.

[28] Tom Preston-Werner. Github.

[29] Casey Reas and Benjamin Fry. Processing: programming for the media arts. *AI Soc.*, 20(4):526–538, 2006.

[30] Mitchel Resnick, Amy Bruckman, and Fred Martin. Pianos not stereos: creating computational construction kits. *interactions*, 3:40–50, September 1996.

[31] Mitchel Resnick, Mary Flanagan, Caitlin Kelleher, Matthew MacLaurin, Yoshiki Ohshima, Ken Perlin, and Robert Torres. Growing up programming: democratizing the creation of dynamic, interactive media. In *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, CHI EA '09, pages 3293–3296, New York, NY, USA, 2009. ACM.

[32] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. Scratch: programming for all. *Commun. ACM*, 52:60–67, November 2009.

[33] Everett M. Rogers. *Diffusion of Innovations, 5th Edition*. Free Press, original edition, August 2003.

[34] Bryan C. Russell, Antonio Torralba, Kevin P. Murphy, and William T. Freeman. Labelme: A database and web-based tool for image annotation. *Int. J. Comput. Vision*, 77:157–173, May 2008.

[35] D Skocaj, A Leonardis, A Jaklic, and F Solina. Sharing Computer Vision Algorithms Over the World Wide Web. Technical report, University of Ljubljana, 1998.

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

[37] Alexander Sorokin and David Forsyth. Utility data annotation with amazon mechanical turk. *Computer Vision and Pattern Recognition Workshop*, 0:1–8, 2008.

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

[39] Stephan Steinbach, Vincent Rabaud, and Serge Belongie. Soylent grid: it's made of people! 2007 2007.

[40] Antonio Torralba, Rob Fergus, and William T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30:1958–1970, 2008.

[41] Troikatronix. Isadora. http://www.troikatronix.com/isadora.html, 5 2010.

[42] Luis von Ahn and Laura Dabbish. Labeling images with a computer game. pages 319–326, 2004.

[43] Luis von Ahn, Ruoran Liu, and Manuel 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.

[44] Luis von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. reCAPTCHA: Human-Based Character Recognition via Web Security Measures. *Science*, 321(5895):1465–1468, September 2008.

[45] Eric von Hippel. *Democratizing Innovation*. The MIT Press, April 2005.

[46] Eric von Hippel and Ralph Katz. Shifting Innovation to Users via Toolkits. *MANAGEMENT SCIENCE*, 48(7):821–833, July 2002.

[47] Peter Welinder, Steve Branson, Serge Belongie, and Pietro Perona. The multidimensional wisdom of crowds. In *Neural Information Processing Systems Conference (NIPS)*, 2010.

[48] Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-ucsd birds 200. Technical Report CNS-TR-201, Caltech, 2010.

[49] P. Wlinder and P. Perona. Online crowdsourcing: rating annotators and obtaining cost-effective labels. *CVPR*, 2010.

[50] IOhannes M. Zmoelnig. Pure data gem. http://puredata.info/community/projects/software/gem, 5 2010.