Interactive Visual Prototyping of Computer Vision Applications

Dan Maynes-Aminzade
Stanford University HCI Group
Computer Science Department
Stanford, CA 94305-9035, USA
monzy@cs.stanford.edu

ABSTRACT
Cameras are a useful source of input for many interactive applications, but computer vision programming requires specialized knowledge that is out of reach for many HCI practitioners. My thesis is that by using visual examples to train computer vision classifiers, and combining trained classifiers into graphs, people with minimal programming experience, and no specialized knowledge of computer vision, can quickly and easily build a wide variety of useful camera-based applications. I will demonstrate this thesis through the design and evaluation of a prototyping tool for camera-based interaction, which I will build, distribute, evaluate, and incrementally refine over the course of the coming year.

INTRODUCTION
Many compelling systems have used cameras as an interactive input medium, from the pioneering work by Myron Krueger [10] to projects like Light Widgets [3], EyePliances [14], and Gesture Pendant [15], tangible interfaces such as Illuminating Light [16] and the Designers’ Outpost [9], game interfaces [4] including the Sony Eyetoy, and platforms like PlayAnywhere [17], TouchLight [18], and the camera-based SmartBoard.

Digital video cameras are becoming increasingly inexpensive and widespread. Today, the cost of a digital camera is comparable to that of a traditional mass-market input device like a keyboard or mouse, and cameras are already integrated into many of our devices, such as cell phones, PDAs, and laptop computers. While these cameras are typically only used for video conferencing and taking pictures, recent advances in computing power open the door to their use as an additional channel of input in a wide variety of applications, giving “eyes” to our everyday devices and appliances.

Unfortunately, this potential has not been thoroughly explored, in part because of the practical difficulty of implementing computer vision techniques.

Figure 1 The computer vision literature contains a wide variety of sensing and processing techniques that are applicable to HCI. While many of these techniques are easy to understand at a high level, they are often difficult to implement in practice.

PROBLEM DEFINITION
Computer vision is understood by only a select group of technologists and researchers. Many of the techniques used in computer vision, such as optical flow, image moments, pattern matching, and face detection, can be easily explained at the surface level, but to actually use these techniques in the design of an interactive system requires a level of mathematical and technical expertise that many interaction designers do not possess.

If a designer wants to prototype an idea for vision-based interaction, he is often forced to contend with the low-level software libraries of his capture device. Once he has managed to extract raw data from the camera, he is still faced with nothing but a large array of RGB triples that he must process in an intelligent way. Software libraries like OpenCV and the Intel Image Processing Library are extremely helpful in this regard, but these systems require in-depth programming knowledge to use, and their functionality is generally couched in terms of underlying mathematical operations rather than high-level goals. Existing computer vision libraries essentially provide programming shortcuts for users already well acquainted with computer vision techniques.

A similar difficulty existed for many years with physical prototyping. Designers who wished to prototype physical devices that employed novel forms of input or output would have to wrestle with electronics and circuit design.
Fortunately, many tools have evolved to address this difficulty, including Phidgets [5], Papier-Mâché [8], Arduino/Processing [12], d.Tools [7], and Exemplar [6]. These tools abstract and package input and output devices such as servo motors, electrical relays, RFID readers, accelerometers, knobs, and sliders, hiding implementation details and exposing functionality through a well-defined API. This simplifies the construction of physical devices and interfaces, making it comparable in complexity to building a graphical user interface. These tools are used by thousands of students, designers, and hobbyists worldwide to quickly build physical interfaces without soldering together electronics components or designing custom protocols for interfacing with a desktop PC.

For my thesis work, I am taking a similar approach to supporting the design of camera-based interfaces, by developing a rapid prototyping tool for camera-based applications called Eyepatch [11]. Eyepatch will allow application developers to train a variety of different classification algorithms, using visual examples selected from live or recorded video, and compose these classifiers into graphs to specify data flow. In this manner, Eyepatch will allow reusable computer vision and image processing algorithms to be combined in a modular fashion to rapidly prototype camera-based systems without a detailed knowledge of the mathematical complexity behind computer vision techniques. By including a variety of modules for common techniques such as object tracking, pattern matching, optical flow, and face detection, each of which can be customized to the context of the particular application for which they are used, I hope to make programming vision-based interfaces just as accessible as programming graphical user interfaces.

RELATED WORK

My greatest inspiration for this work comes from the Crayons design tool for camera-based interaction [2], which allows the interactive training of pixel classifiers using a “paint, view, and correct” process. Its authors proposed a variety of ways of extending the Crayons model, many of which I will incorporate into my work, such as supporting additional feature types and taking motion into account.

I also draw inspiration from Exemplar [6], which provides an example-based approach to training classifiers for recognizing sensor input. The data classified in Exemplar is multiple channels of one-dimensional sensor input, but the approach of building a model through examples, viewing the state of the created model, and adjusting it for better results, is very similar to the approach I will adopt.

The Papier-Mâché toolkit [8] provides a high-level programming abstraction that allows users to extract certain events from a camera without worrying about the underlying details of computer vision algorithms. Its event model is designed to parallel an RFID reader, and can trigger an event when particular objects are added to or removed from the camera view. It also provides the ability to extract basic parameters from the objects, such as color and image moments. This makes it very easy to program certain types of computer vision applications, but the single classifier used constrains the type of applications it can produce, and its event model does not adapt well to applications that require dynamic tracking at interactive frame rates.

The Cambience system [1] allows users to select regions of a video frame in which they want to detect motion, and then map the motion in each region to a different sound effect. The system also incorporates certain types of control over data flow and filtering, for example to map the intensity of motion to the volume of a sound. However, Cambience focuses on only one type of input (motion) and one type of output (ambient soundscapes).

The Lego Mindstorms “Vision Command” Kit was one of the first systems to attempt to make computer vision accessible to novice programmers. Its visual programming interface allows users to test for simple conditions in a camera image: when a certain color or a certain brightness threshold is seen in one of five regions of the frame, an event can be triggered. This highly simplified model works such as supporting additional feature types and taking motion into account.

I also draw inspiration from Exemplar [6], which provides an example-based approach to training classifiers for recognizing sensor input. The data classified in Exemplar is multiple channels of one-dimensional sensor input, but the approach of building a model through examples, viewing the state of the created model, and adjusting it for better results, is very similar to the approach I will adopt.

The Papier-Mâché toolkit [8] provides a high-level programming abstraction that allows users to extract certain events from a camera without worrying about the underlying details of computer vision algorithms. Its event model is designed to parallel an RFID reader, and can trigger an event when particular objects are added to or removed from the camera view. It also provides the ability to extract basic parameters from the objects, such as color and image moments. This makes it very easy to program certain types of computer vision applications, but the single classifier used constrains the type of applications it can produce, and its event model does not adapt well to applications that require dynamic tracking at interactive frame rates.

The Cambience system [1] allows users to select regions of a video frame in which they want to detect motion, and then map the motion in each region to a different sound effect. The system also incorporates certain types of control over data flow and filtering, for example to map the intensity of motion to the volume of a sound. However, Cambience focuses on only one type of input (motion) and one type of output (ambient soundscapes).

The Lego Mindstorms “Vision Command” Kit was one of the first systems to attempt to make computer vision accessible to novice programmers. Its visual programming interface allows users to test for simple conditions in a camera image: when a certain color or a certain brightness threshold is seen in one of five regions of the frame, an event can be triggered. This highly simplified model works
well for certain tasks, but many events and objects cannot be recognized by these simple classifier types alone.

DESIGN APPROACH
My overall research strategy is based on iterative prototyping. After creating each version of Eyepatch, I provide it to designers, and observe what they can achieve with it and where they encounter problems. This provides insights that help me improve subsequent versions.

I wrote the first version of Eyepatch as a collection of ActiveX Controls in Visual Basic, each control custom-built to solve a particular computer vision problem, such as tracking faces or finding laser pointer dots. I deployed this first version to students in an HCI class at the University of Tokyo (Figure 4), but found that the highly specific nature of the ActiveX Controls constrained the students to a very narrow range of possible applications, and the ActiveX architecture prevented students from using the camera data in other prototyping tools.

I recently completed and evaluated a second version of Eyepatch [11] (Figure 5), which was designed to allow users more creativity in developing their own computer vision algorithms. I wanted to enable designers to attack a broader range of problems, and use a wider variety of prototyping tools. At the same time, I tried to keep things as clear-cut as possible by making certain simplifying assumptions in the framework. I assumed that the input came from a camera or recorded video, the sequence of frames was passed through one or more binary classifiers, and the only output was the set of image regions yielded by each of the active classifiers. My expectation was that this framework would still be too limited for some real-world applications. However, by examining what users could build and what was out of their reach, I began to gain an understanding of what features were needed in a camera-based interaction design tool in order for it to achieve widespread use.

GOALS AND METHODOLOGY
My thesis is that by using visual examples to train computer vision classifiers, and combining these classifiers into graphs, people with minimal programming experience, and no specialized knowledge of machine learning, can build a wide variety of useful applications. This thesis statement can be divided into three hypotheses, which I will discuss individually:

H1. Training classifiers using visual examples is a simple enough approach that it can be used effectively by people without extensive technical knowledge or programming experience.

To evaluate this hypothesis, I will deploy Eyepatch to students for use in class projects, and observe what they are capable of building using its model, and where the limitations of the model lie. I believe that students will be capable of building many vision-based applications, without needing to know the full details of the computer vision and machine learning algorithms upon which they are based. By iteratively refining Eyepatch based on observations and student feedback, I will learn which details of the computer vision algorithms are necessary to expose to the user, and which details are best left hidden. In addition, I will learn what features are most important to users who are designing camera-based applications.

H2. Example-based training of classifiers, combined with the composition of classifiers into graphs, covers a broad enough space of applications that it can be used to build a substantial subset of camera-based applications in the HCI literature.

To evaluate this hypothesis, I will conduct a broad survey of HCI systems that use computer vision, and demonstrate how these systems can be built using my approach. I will replicate several representative applications of different types to demonstrate the power and versatility of Eyepatch. For those applications that cannot be replicated, I will characterize the nature of the problem, to better understand the limitations of my approach. This will allow me to precisely distinguish the space of applications that can and cannot be built using the Eyepatch model.

H3. In some cases, vision classifiers built using a human-in-the-loop training approach can perform

Figure 4 Sample projects created by students at the University of Tokyo using the first version of Eyepatch.

Figure 5 Eyepatch in training mode. Here the user is training a color classifier; the pane at the right shows the examples she has chosen, and the internal state of the classifier is represented by hue histogram in the lower pane.
more effectively, and be trained more quickly, than fully automated offline learning.

To evaluate this hypothesis, I will study various machine learning techniques to see which can benefit from human-in-the-loop training, and which work best through fully automated learning. In particular, I plan to build a system for gesture selection that reduces false positives by pre-screening proposed gestures, and warning the user if a proposed gesture is detected erroneously in background data.

PRELIMINARY RESULTS AND DISCUSSION

I began evaluating Eyepatch to see which types of applications it could support, which it could not, and what capabilities were needed for a tool of this nature to be useful to designers. For the purpose of this evaluation, I offered a one-quarter course on computer vision for HCI called “Designing Applications that See.” In a series of hands-on workshop sessions, I introduced the students in the class to a variety of computer vision tools, including MATLAB, OpenCV, JMyron (a Processing [13] extension), and Eyepatch. They were also given a general introduction to the common techniques used in computer vision and image processing, without delving into any complex mathematical details. After five weeks of being introduced to the tools, I asked the students to spend five weeks building working prototypes of camera-based applications, using the tools of their choice.

Deploying Eyepatch to a project class in a longitudinal study was an excellent way to gain insight into its strengths and weaknesses. I will continue to refine and evaluate Eyepatch using an iterative prototyping process. Although I learned that Eyepatch still has much room for improvement, I believe that it represents an important step towards making camera input accessible to interaction designers. Eyepatch allows designers to create, test, and refine their own customized classifiers, without writing any specialized code. Its diversity of classification strategies makes it adaptable to a wide variety of applications, and the fluidity of its classifier training interface illustrates how interactive machine learning can allow designers to tailor a recognition algorithm to their application without any specialized knowledge of computer vision.

REFERENCES


