AirTouch: Interacting With Computer Systems At A Distance

Daniel R. Schlegel, Albert Y. C. Chen, Caiming Xiong, Jeffrey A. Delmerico, Jason J. Corso
Dept. of Computer Science and Engineering
SUNY at Buffalo
{drschleg, aychen, cxiong, jad12, jcorso}@buffalo.edu

Abstract

We present AirTouch, a new vision-based interaction system. AirTouch uses computer vision techniques to extend commonly used interaction metaphors, such as multitouch screens, yet removes any need to physically touch the display. The user interacts with a virtual plane that rests in between the user and the display. On this plane, hands and fingers are tracked and gestures are recognized in a manner similar to a multitouch surface. Many of the other vision and gesture-based human-computer interaction systems presented in the literature have been limited by requirements that users do not leave the frame or do not perform gestures accidentally, as well as by cost or specialized equipment. AirTouch does not suffer from these drawbacks. Instead, it is robust, easy to use, builds on a familiar interaction paradigm, and can be implemented using a single camera with off-the-shelf equipment such as a webcam-enabled laptop. In order to maintain usability and accessibility while minimizing cost, we present a set of basic AirTouch guidelines. We have developed two interfaces using these guidelines—one for general computer interaction, and one for searching an image database. We present the workings of these systems along with observational results regarding their usability.

1. Introduction

Since the inception of graphical computer systems in April 1981 [8], interaction with graphical computer systems has evolved over the past thirty years to include such interface metaphors as the mouse and keyboard, pen computing, touch, and recently multitouch [22]. These enabling developments have allowed interaction to become more accessible and natural. A recent and notable example is the Apple iPad, which sold more than 3 million units within its first three months [1]. Despite the broad range of technologies in these developments, they all rely on one basic principle: interaction happens on two-dimensional plane.

As an enabling technology, computer vision has great potential to further improve the naturalness of these interaction metaphors and also to take interaction off of the two-dimensional plane constraint. Indeed, multiple attempts have been made over the past decade to create methods for computer interaction using computer vision (e.g., [7, 12, 14, 19, 22–24]), though none of these systems have achieved widespread use. Computer vision systems that allow for human interaction generally use some sort of interface defined by the gestures the user can perform. These gestures must happen within the field of view and range of some camera device. Many systems utilizing a gesture based interface allow for the gestures to happen at many distances before the camera with largely the same effect.

Perhaps the simplest paradigm attempted by some of these systems involves extending the concept of Marking Menus (also known as "pie menus") to computer vision, which only require a users hand to be present in a specific region of the cameras view to perform an action [5]. Extensions of these ideas led to two mature vision interaction systems based on localized interaction within certain "hot spots" in the environment [13, 25]. However, the localized interaction does not allow generalized control. Many other systems attempt to use gestures for general computer use. One example relies on the posture of the hand for gestures [14], a technique used by others as well [19]. Similarly, Toshiba has produced a system for general computer control where the user holds their hand up and pushes it towards the screen to indicate a click [18]. Other systems such as GestureVR [7], which allows the pose and motion of the hand in 3D space to control a virtual reality simulation, are more actively sensitive to the depth of the hand.

These systems have performed well in laboratory settings, but many have not been evaluated or used in general settings for two primary reasons. First, in most cases, when the system is *on*, it is continuously watching for gesturing users such as in [19]. From a usability perspective, this trait yields a limiting scenario in which the user cannot act naturally and assume whatever poses are comfortable for them. Solutions to the "always on" nature of a system which force unnatural postures as shown in the user study for [5], which

978-1-4244-9497-2/10/$26.00 ©2010 IEEE

requires that the user hold their arm away from the body unsupported for long periods of time, are not much of an improvement. Second, many of these systems are difficult to set up, tuned to a specific use case or software package, and can require very expensive hardware. GestureVR [7] requires two cameras configured for stereo-vision, something that may be difficult for general use, while [14] uses only one camera, but uses projectors and a camera inside a robotic head to follow the user for specific kiosk applications.

In contrast, the AirTouch system we propose overcomes both of these issues while concurrently bringing to bear well-known interaction metaphors like multitouch in a computer vision setting. AirTouch's driving innovation is to create a virtual interaction surface in the space, i.e., the "air," between the user and the computer. The user then "touches" this virtual interaction surface, i.e., this air, and his or her actions are mapped in a natural way to the computer system. The interaction surface acts as an analog to two-dimensional plane metaphors while also incorporating a third dimension for determination of whether the user has "touched" the plane. This limited three-dimensionality removes the restrictiveness of a physical plane while maintaining its usability. As we demonstrate in two example implementations, AirTouch enables natural human-computer interaction at a distance without requiring the user to adapt his or her behavior in any significant way, or spend time calibrating the system. Furthermore, in its simplest form (Section 3.1), it will augment the existing computer system software without requiring any rewrite of existing platform software; the only burden on the user is the availability of a commodity camera (i.e., webcam). Our two example AirTouch systems have been successfully publicly demoed [15].

One existing system has similar qualities to the proposed AirTouch system: GWindows [23]. Wilson's system features an "engagement plane" that is approximately 20 inches from the screen. The user moves his/her hand to the plane then speaks instructions to the computer to control windows on the screen such as "move," "close," or "scroll." The system uses dwell time to engage actions. The user study has indicated the system may be too cumbersome and the cross-modal input was less natural than gesturing alone [23].

In the next section, we describe the design considerations we have incorporated into the AirTouch system. Section 3 then describes two implementations of the AirTouch ideas: one for general computer use (mouse pointing and clicking, and scrolling) and another for intuitive content-based image retrieval (CBIR). In both sections, we incorporate a discussion that includes responses from public user demonstrations.

2. Design Considerations for Vision-Based Interaction Systems at a Distance

Before getting to the specifics of our two implemented AirTouch systems, we describe some design considerations that have driving our research. These considerations are offered in the spirit of our established predecessors, such as Shneiderman's direct manipulation [17]. We divide them into the two key categories for vision-based HCI systems: gesturing and ease of use.

2.1. Gesturing

Gesturing is an extensively studied and important subject. Despite the abundance of gesturing systems (some were covered in the introduction), a few gesture-based interaction metaphor have gained widespread acceptance. Examples include multitouch gestures such as pinch-to-zoom and drag-to-scroll. Other input methods have different gestures—pen computing devices use gestures such as "pen flicks" for turning pages, scrolling, or going back in a web browser. These types of gestures are all unobtrusive, easy for the user to adapt to, and have potential to increase the user productivity. In order to bring these advantages to a vision-based interface we propose a few guidelines.

First, the user must be able to act normally between gestures, which is particularly important as without it the interface is obtrusive and difficult to use. In the AirTouch system manipulative gestures—those that affect the system state—occur on a plane of interaction. This means that when the users are not interfacing with the plane there is no manipulative effect on the system.

Gestures that occur inside the space should correspond to the planar metaphor. The gestures should both require the plane as the place where they should occur and the plane should afford the gestures. This constraint prevents the user confusion from gesture locality issues—e.g., why a gesture can occur in one place versus another or only under certain conditions. Many multi-touch gestures already correspond to this metaphor—using them would minimize the learning curve. Finally, when a gesture has been accepted by the system a cue should appear, whether visual or audible by some other means. Since the user is not physically touching something it can become frustrating if the user performs a gesture and it appears nothing has happened.

2.2. Ease of Use

One advantage of touch screens and pen computing systems is that they only infrequently require calibration and are transparent across different users. Many of these systems are also portable and require little or no setup between sessions. A vision system meant for widespread general use must attempt to mimic these advantages.

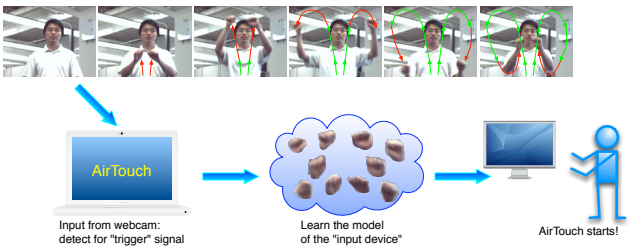


Figure 1. An example calibration gesture where the user motions with both hands.

On the issue of calibration, it must be quick[1], simple and occur at most once per user session. Examples might include calculating a homography by having the users point at on-screen cues, or having a gesture pre-defined for calibration as seen in Fig. 1. Determination of the location of the interaction plane in space should be determined by this calibration as well. Users seem to adapt well to slight calibration errors so extreme attention to detail is not required.

Additionally, the user must be able to leave and re-enter the camera frame with no effect on their ability to utilize the system. If the user moves their hands out of the view of the camera or stands up for a moment it should have no negative effect—when the user's hands are again visible s/he should be able to use the system as s/he would have had they not left the field of view of the camera.

The system must be cost-effective and easy to set-up: built-in webcams or other off-the-shelf cameras. It may be desirable to use more than one camera for stereo vision or another application but the issue of cost and ease must be seriously considered.

In the two sample implementations we have completed (see Section 3) these guidelines have resulted in reportedly pleasant user experiences.

3. Implementations

We have created two AirTouch systems that implement the functionality and basic guidelines given above while performing drastically different tasks: general computer use and CBIR.

3.1. Computer Vision for General HCI

The Computer Vision for General HCI (CVHCI) system allows the user to interact with the generic computer system from afar. In this context the AirTouch system is used to mimic a touch screen floating in space. The user interacts with the system through various pointing gestures. Processing the users interactions happens on two distinct levels—tracking and gesturing—which, when combined, allow for a highly usable system for human-computer interaction.

[1]Quick is, of course, a user-dependent quality. Recall some recent systems took minutes to boot.

3.1.1 Tracking

To reliably track the user's hands in a computationally inexpensive way, we use the CAMSHIFT [4] algorithm, a Kalman filter [9], and require the users to wear gloves with colored fingertips. The use of gloves could be relaxed via skin-color tracking [19], though we use them primarily because of the low computational overhead of the involved algorithms and because our major focus was demonstrating the AirTouch concept rather than developing an efficient tracking system. Colored gloves are used with much success in other real-time applications such as MIT's hand tracking system for virtual manipulation of objects [20]. The CAMSHIFT algorithm allows us to determine not only the X and Y coordinates of each tracked finger, but also the diameter of the tracked area, which we treat as a proxy to the Z-axis using only one camera. The Kalman filter is used to increase the robustness of the tracking and to minimize jitter.

The initial calibration stage requires the user to point at five positions on the screen to establish a homography that allows the conversion of camera to screen coordinates. The finger is recognized using prior histograms and shape data. The user dwells at each point for a very short period of time to indicate selection. The diameter of the finger is noted at the detection point, defining D_{touch}, the "depth" location of the AirTouch plane.

To allow the user to act normally in front of the camera, we use adaptive color histograms to provide the finger tracking with robustness to lighting changes and quick movement. Periodically, the histogram of the tracked region is reevaluated and compared with the histogram on which the tracker's backprojection is currently based; if there is a large enough discrepancy, the histogram is updated to reflect the color of the tracked object in the current environment. The level of match is determined by a sum-of-squares difference metric between the values of the bins of the current histogram and the histogram in the bounding box of the currently tracked object. The histogram from initial calibration is stored and a restriction is placed on the algorithm not to allow drift from the original greater than 30% for any bin.

The histograms we use are in the HSV color space since lighting and camera white balance changes cause variation largely in the saturation and value components with less effect on hue. For this reason, we only use the hue histogram in tracking and we threshold the saturation and value channels to try to maximize the presence of the tracked hue in the backprojection. When the histogram is updated, the saturation and value threshold values are updated automatically using a binary search of the respective spaces to find the best backprojection, the one containing the most positive pixels

in the tracked area. This update assists with not only with lighting changes but also with quick movement by ensuring the track is not lost if the hand blurs with fast motion causing a slightly different histogram. To account for a user's hand leaving the frame, we have configured the tracking algorithm to continually search the lower half of the image for objects matching the color histograms of the size expected of a finger. The lower half has been selected as our search area since the user is most likely to return their hands to the view of the camera from their lap or a desk.

After each frame from the camera is processed by the tracker, the relevant data is extracted into an *observation* O such that

$$O_t = (X_{scr,i}, Y_{scr,i}, X_{rel,i}, Y_{rel,i}, D_i, V_i) \qquad (1)$$

where i is the finger's ID, X_{scr},Y_{scr} are the screen coordinates referenced by the finger, X_{rel},Y_{rel} are the relative distances to the first (pointer) finger, D is the diameter of the finger, and V is its velocity in relation to the previous observation O_{t-1}. Only O_t and O_{t-1} are maintained by the system at any time. The observation is submitted to the gesture processing system after it is generated.

3.1.2 Gesturing

The system currently allows for three gestures for interaction with the computer: mouse movement, mouse clicking, and scrolling (see Figure 2). All three gestures are variations on pointing at the screen. To move the mouse the user moves their pointer finger in front of the camera at a distance outside of the AirTouch plane. This can be likened to how a Tablet PC or Wacom tablet handles mouse movement—then pen can hover above the interface and still change the location of the mouse pointer. Mouse clicking is actually two gestures—moving the pointer finger into the plane to activate mouse down, and removing it to mimic mouse up. In other words, we have a virtual touch screen that allows for click-and-drag type actions. Finally, the scrolling gesture is accomplished by moving both the pointer and middle fingers into the interaction plane and moving them vertically, which is common on multitouch trackpads. Recognition of a finger i's containment in the plane is computed by taking the difference of D_{touch} from the calibration and D_i from the observation.

All of these gestures result in calls directly to the windowing system that would otherwise be made by the mouse. We therefore require none of the client software be modified for our system to work. We have also implemented an API for applications to plug-in to AirTouch and add their own gestures, but the details are beyond the scope of this paper.

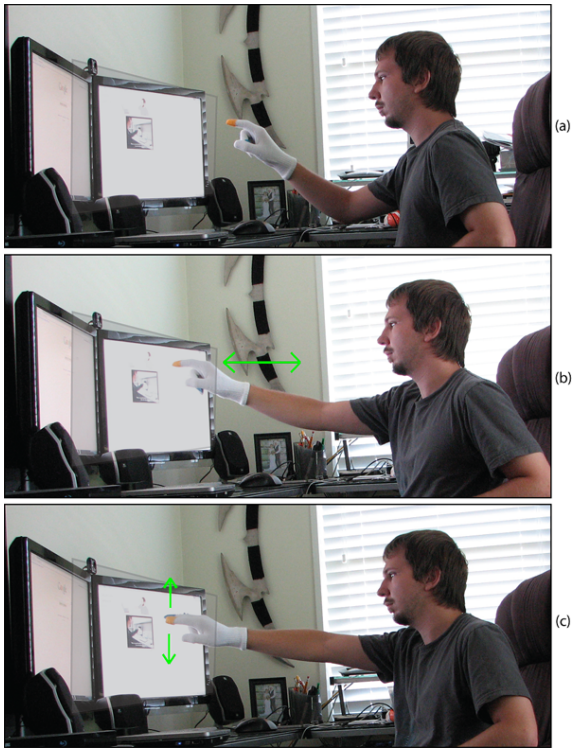


Figure 2. The CVHCI Gestures. (a) shows moving the mouse around the screen. (b) shows the user moving their hand in to and out of the plane with their pointer finger to click. (c) shows the user using two fingers in the plane moving vertically to scroll.

3.1.3 Discussion

The system has undergone informal user testing, and initial results appear to be promising. Two anticipated issues—users' inability to adapt to an invisible plane and that hand fatigue would quickly set in—appear to be absent, but will require further study.

An early version of the system was demoed with approximately 15 students and faculty attempting to use the system. While at the time the system was too rough to get much data, we observed that with only minimal time (tens of seconds) using the system users tend to adapt to the location of the plane even if there are slight calibration errors or the system had been calibrated by another person, which suggests a one-time "factory" calibration may be possible.

Users have also reported after using the system for significant amounts of time (for example a 3-hour demo given at CVPR [15]) the phenomena known as "Gorilla Arm" does not readily occur. We believe this stems from the user being able to sit in a relaxed position, and not being required to constantly have their hands in the view of the camera. Fatigue can occur after extended periods of time, but more testing is required to determine where this threshold is and what the implications might be.

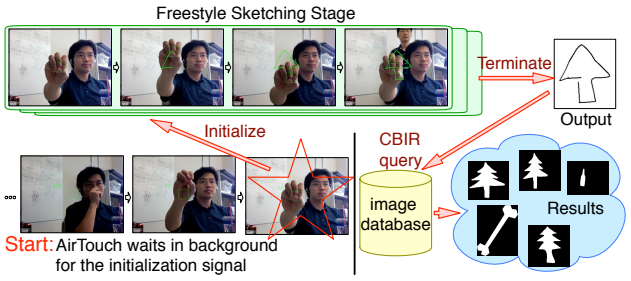


Figure 3. Overview of the AirTouch-based Freestyle Sketching system.

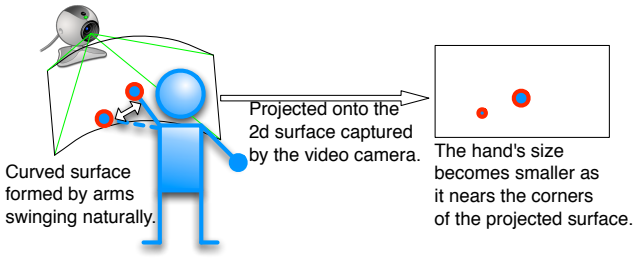


Figure 4. When the user thinks that he or she is drawing on a 2D virtual plane vertical to the camera, its often a curved one due to the way human arms rotate.

Users reported an affinity for the use of the touchscreen-type interaction metaphor as it is one that they were already familiar and comfortable with. Many people have commented that they would like to see a system like this utilized for computer control in areas other than at a desk—ranging from a 10-foot interface for interacting with large-scale data, to cell phones, and even for use with automotive computers. Our approach's credibility as an usable interface is bolstered by a usability study that found users liked the idea of an invisible "engagement plane" and prefered gesturing to manipulate the system rather than voice commands [23]. Presumably users like the sensation that they are actually manipulating the on-screen controls using their hands.

3.2. AirTouch-based Freestyle Sketching for CBIR

In this *AirTouch-based Freestyle Sketching for CBIR* system, we demonstrate that the true power of AirTouch is not limited to gesture-based HCI input; it is more of a general-purpose, unrestrained way of interacting with computers. In short, the user performs freestyle sketching on the self-defined virtual plane.[2] Anything from the user's finger and fist to a special gesture, or even a mug can be specified as the input device during the initialization stage. The input device is then continuously detected and tracked; when it intersects with the virtual plane, it is treated as if a pen is touching the surface of a paper, and the trajectory is added to the canvas. When a termination gesture is detected, the canvas is outputted to trigger a Content-based Image Retrieval (CBIR) query. Figure 3 is an overview of how the AirTouch-based Freestyle Sketching system triggers a CBIR query.

3.2.1 System Setup and Initialization

For the goal of requiring only minimum calibration effort and portability, a single webcam setup is again used in the *AirTouch-based Freestyle Sketching* system. An initialization process similar to those of the *Computer Vision for General HCI* system could be used to learn the size of the user-chosen input device, where multiple corners of the virtual plane need to be specified. However, this Freestyle Sketching system doesn't require as detailed distance measurement as the HCI system demonstrated previously. Therefore, the initialization process is simplified to just touching one corner, with the additional assumption and constraint that the virtual plane is strictly orthogonal to the webcam. A robust optical-flow-based transition model is developed in Sec. 3.2.2 to deal with the curvatures in the virtual surface while determining the relative distance of the input device to the surface.

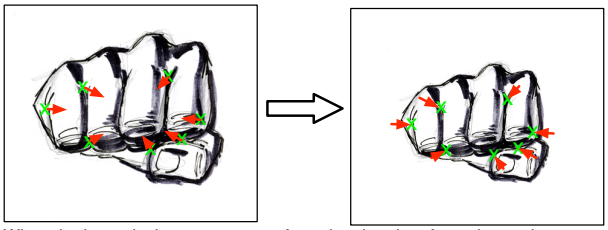
The user-chosen input device is learned during the initialization stage, and is continuously detected and tracked during the sketching stage. Since a simple contour-based CBIR system is used in this initial implementation, the user is limited to sketching the contour of the to-be-queried object with one continuous sketch. This restriction is to be relaxed in future AirTouch-based Freestyle Sketching subsystems as the CBIR subsystem gets more sophisticated (refer to Sec. 3.2.3.)

3.2.2 Freestyle Sketching Stage

Prior to the sketching stage, the input gesture and device is detected occasionally at less than 1fps. However, once the sketching stage starts, the system works at 15fps in order to let the user have a real-time response. Therefore, once the input device intersects with the virtual surface, we reduce the computational load by only sparsely selecting feature points and calculating the optical flow within the image region that contains the input device. The classical *Shi and Tomasi* algorithm [16] is used to select the feature points, and *Pyramidal Lucas Kanade* [3] is used to compute the optical flow between the frames. Due to the occasional inaccuracy caused by optical flow estimations, a commonly used region-based flow-averaging approach as in [2] is used to estimate the motion of the whole input device and a Kalman filter [21] is used to further stabilize the tracking results.

During the sketching stage, the distance between the in-

[2]This "plane" could be some more complex segment of space such as a quadratic or a patch from a sphere, rather than a strict plane.



When the input device moves away from the virtual surface, the motion vectors inside the tracked input device would move towards the center of the object. Red arrows are the motion vectors and green X's are the feature points selected.

Figure 5. Estimating the relative change-of-distance between an object and the video camera via the movements of the optical flow vectors within the object. Fist figure courtesy of artist Loston Wallace (http://lostonwallace.deviantart.com/).

put device and the single video camera needs to be continuously estimated. However, notice that the virtual surface formed by the user freely swinging their hands is often not a flat 2D surface when being projected into the video capture device, as shown in Fig. 4. This would cause the captured image of the hand to become smaller as it nears the corners of the image. To remedy this problem, multiple samples of the input device need to be captured at different corners of the image during the initialization stage, which would contradict our goal of minimizing system calibration. Luckily, with frames being processed at real-time rates, an easy way to determine if the input device have left the virtual surface exists by simply looking at where the tracked feature points are moving: when the object moves away from the camera, the optical flows within the object flow towards the center of the object, as shown in Fig. 5; when it moves towards the camera, the optical flows flow away from the center.

Based on this observation, we record the change-of-distance between the optical flow vectors and the distribution of their angles for all optical flow vectors originating within the image region of the user-chosen input device. We use the entropy $E(\cdot)$ of the angle distribution X^t (at frame t) to estimate the coherency of the motion vector orientations. It is approximated through the discrete entropy of an n bin histogram:

$$E(X^t) = -\sum_{i=1}^{n} f(x_i^t) \log f(x_i^t) \ . \tag{2}$$

When the input device is moving on or near the virtual surface with little noise, X^t peaks at bins that correspond to the moving direction, which results in relatively smaller $E(X^t)$ values. When the input device is moving either towards or away from the virtual surface, X^t's distribution is spread out covering multiple bins, which results in relatively larger $E(X^t)$ values. The two distributions are learned non-parametrically offline and the threshold τ for determining the two cases is fixed throughout run-time.

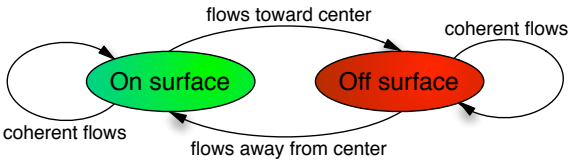


Figure 6. The state-transition model used in for determining when the input-device touches the user-defined virtual surface.

When $E(X) > \tau$, we know that the motion vectors are not moving towards the same direction, *i.e.* the input device is either moving towards or away from the camera, or performing non-rigid transformations. Non-rigid transformation of the input device is beyond the scope of this discussion, since even sophisticated models with multiple calibration points are not guaranteed to handle the situation. As for discriminating between "moving towards" or "moving away" from the camera, the average distance from the individual motion vectors to the center of the input-device is compared from frame to frame:

$$\Delta = \sum_{j=1}^{m} D(v_j^{t+1}, o^{t+1})/m - \sum_{j=1}^{m} D(v_j^t, o^t)/m \tag{3}$$

$D(\cdot)$ is the euclidean distance between any two points on the image, v_j^t is the origin of the j^{th} of m motion vector within the region of the input-device in frame t, and o^t is the center of the input device, tracked and updated by the Kalman filter.

A simple two-state transition model is used to determine when the user has interacted with the virtual surface, as shown in Fig. 6. When

$$E(X) > \tau \ \ \&\& \ \ \Delta < 0 \tag{4}$$

the input device is moving from *on surface* to *off surface*, and when

$$E(X) > \tau \ \ \&\& \ \ \Delta > 0 \tag{5}$$

the input device is moving from *off surface* back to *on surface*. This method for determining the intersection of the input device and virtual surface not only requires minimal calibration, but is also very robust to virtual surface deformations and adds little additional computation cost. This system, implemented in mixed C/C++ runs at 30+ fps on Intel Core 2 Duo 2.2Ghz machines.

3.2.3 CBIR Query Stage

Current CBIR systems roughly fall into two categories: (1) sample query image based systems that retrieve similar images in terms of low-level features such as color, texture, and salient points; (2) semantic based systems that retrieve

images by query keywords such as "find images with sky and sea". The first type of CBIR system is prone to returning completely irrelevant images with analogous low-level features, while the second type is constrained by the limited amount of keywords/classes the system recognizes. Another drawback of the sample query image based CBIR system is the hassle of finding a sample query image itself.

Inspired by Marr's theory that primal sketches [6] alone provide enough information to recognize the object and scene, we have developed a CBIR system that retrieves images containing objects similar to what the user has sketched. The CBIR subsystem uses the AirTouch-based Freestyle Sketching subsystem's output as input. The retrieval system itself consists of the following parts:

1. For each image in our query database, we construct an edge map which is seen as our 'primal sketch' by using the boundary detector proposed by Martin et al. [10]. We then trace the boundaries, vectorize them, and compute the vector field for each image by using orientation diffusion [11].

2. When an image query is performed, after extracting and vectorizing the boundaries of the image, we calculate the distance between the query image and images in the database by calculating the difference of the orientation of points in the boundary of query image with the corresponding points' orientation of the image in the database. Then we rank the images by the distance.

3.2.4 Discussion

In the same way as the CVHCI system, the AirTouch-based Freestyle Sketching for CBIR system has been openly demonstrated with more than 15 outside users testing the system. The flow-averaging technique we employed showed robustness towards the individual flow calculation errors, which is often caused by the unintended rotation of the input device. The two-state transition model provides a robust way of estimating the relative distance between the user-chosen input device and the video camera. As opposed to methods that are trained on skin color or equipped with specific hand-detection models, humans entering or leaving the background (as shown in the last frame of the *freestyle sketching stage* in Fig. 3) have little effect on the system's accuracy or efficiency.

Issues that our test users reported are often related to the simple contour-based CBIR subsystem. As shown in the *results* of Fig. 3, our contour-based image queries are prone to returning semantically-different, yet contour-wise similar images. A more sophisticated sketching model that is capable of sensing multiple-levels of pressure applied to the virtual surface (like the Wacom styluses) would allow the users to draw more realistic images, therefore allowing the CBIR system to perform better queries. However, with the goal and restriction of having as simple calibration as possible, the accuracy of the pressure-estimation process would decrease as the number and level of pressure-levels it senses increases. Another possible route would be to develop a natural and easy-to-use drawing mechanism for color and even texture.

4. Conclusion

The issue of natural and comfortable interaction between humans and computers has received much study in recent years. On several occasions vision systems have been proposed in an attempt to create a natural method for interacting with machines while not directly touching them. These systems have primarily been restricted to lab settings, likely due to robustness problems, difficulty of set-up, and cost issues. In contrast, we have focused our efforts on a vision-based interaction system that uses standard hardware and extends already well-known interaction metaphors.

The concept of using a virtual plane inhabiting the space in front of a camera to delimit the regions in which a gesture can occur has shown to have promise as an interface metaphor. The methods described here allow for more complete use of the capabilities of simple consumer-grade cameras to perceive depth and use this to augment the gesture systems which are common in research that do not extensively utilize the capabilities of the z-axis.

We believe that this interface metaphor is one which has nearly limitless applications in offices, labs, households, industry, and on the move. We hope this system will be adopted by others and used to promote efficient and natural methods of human-computer interaction.

5. Acknowledgements

We are grateful for the support provided by NSF CAREER IIS-0845282 and NSF CNS-0855220.

References

[1] Apple Inc. Apple sells three million ipad in 80 days. http://www.apple.com/pr/library/2010/06/22ipad.html.

[2] X. Bai, J. Wang, D. Simons, and G. Sapiro. Video SnapCut: robust video object cutout using localized classifiers. In *ACM SIGGRAPH 2009*. ACM, 2009.

[3] J. Bouguet et al. Pyramidal implementation of the lucas kanade feature tracker description of the algorithm. *Intel Corporation, Microprocessor Research Labs, OpenCV Documents*, 1999.

[4] G. R. Bradski. Computer vision face tracking for use in a perceptual user interface. Technical report, Intel Technology Journal, Q2 2008.

[5] L. Bretzner, S. Lenman, and B. Eiderbck. Computer vision based recognition of hand gestures for human-computer interaction. Technical report, University of Stockholm, 2002.

[6] C. Guo, S. Zhu, and Y. Wu. Towards a mathematical theory of primal sketch and sketchability. In *Proceedings of IEEE International Conference on Computer Vision*, 2003.

[7] S. K. Jakub Segen. Video-based gesture interface to interactive movies. In *Proceedings of the Sixth ACM International Conference on Multimedia*, pages 39–42, 1998.

[8] J. Johnson, T. Roberts, W. Verplank, D. Smith, C. Irby, M. Beard, and K. Mackey. The Xerox Star: A Retrospective. *Computer Graphics*, 22(9):11–26, 1989.

[9] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASMA; Journal of Basic Engineering*, 82:35–45, 1960.

[10] D. Martin, C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using brightness and texture. In *Proc. of NIPS*, pages 1255–1262, 2002.

[11] P. Perona. Orientation diffusions. In *cvpr*, page 710. Published by the IEEE Computer Society, 1997.

[12] E. Petajan. Vision-based HCI applications. In B. Kisacanin, V. Pavlovic, and T. S. Huang, editors, *Real-Time Vision for Human-Computer Interaction*. Springer, 2005.

[13] C. Pinhanez. The everywhere displays projector: A device to create ubiquitous graphical environments. In *Proceedings of UBICOMP*, volume LNCS 2201, pages 315–331, 2001.

[14] P. Robertson, R. Laddaga, and M. V. Kleek. Virtual mouse vision based interface. In *Proceedings of IUI'04*, pages 177–183, January 2004.

[15] D. R. Schlegel, J. A. Delmerico, and J. Corso. Armchair Interface: Computer Vision for General HCI. In *IEEE Conference on Computer Vision and Pattern Recognition: Demo Track*, 2010.

[16] J. Shi and C. Tomasi. Good features to track. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 593–593. Citeseer, 1994.

[17] B. Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer*, 16(8):57–69, 1983.

[18] Toshiba Research Europe Ltd. Cambridge Research Laboratory. Projects: Gesture user interfaces, May 2010.

[19] C. von Hardenberg and F. Bérard. Bare-hand human-computer interaction. In *Proceedings of the ACM Workshop on Perceptive User Interfaces*, November 2001.

[20] R. Wang and J. Popovi. Real-time hand-tracking with a color glove. *ACM Transactions on Graphics*, 2009.

[21] G. Welch and G. Bishop. An introduction to the Kalman filter. *University of North Carolina at Chapel Hill, Chapel Hill, NC*, 1995.

[22] W. Westerman, J. G. Elias, and A. Hedge. Multi-touch: A new tactile 2-d gesture interface for human-computer interaction. In *Proceedings of the Human Factors and Ergonomics Society 45th Annual Meeting*, pages 632–636, 2001.

[23] A. Wilson and N. Oliver. Gwindows: Towards robust perception-based ui. In *IEEE Workshop on Human Computer Interaction at Conference on Computer Vision and Pattern Recognition*, 2003.

[24] C. R. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-Time Tracking of the Human Body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, 1997.

[25] G. Ye, J. J. Corso, D. Burschka, and G. D. Hager. VICs: A Modular HCI Framework Using Spatio-Temporal Dynamics. *Machine Vision and Applications*, 16(1):13–20, 2004.