Waken: Reverse Engineering Usage Information and Interface Structure from Software Videos

Nikola Banovic[1,2]*, Tovi Grossman*[1]*, Justin Matejka*[1]*, and George Fitzmaurice*[1]

[1]Autodesk Research
210 King St. E., Toronto, Ontario, Canada
{firstname.lastname@autodesk.com}

[2]Dept. of Computer Science
University of Toronto, Ontario, Canada
nikola@dgp.toronto.edu

ABSTRACT
We present *Waken*, an application-independent system that recognizes UI components and activities from screen captured videos, without any prior knowledge of that application. *Waken* can identify the cursors, icons, menus, and tooltips that an application contains, and when those items are used. *Waken* uses frame differencing to identify occurrences of behaviors that are common across graphical user interfaces. Candidate templates are built, and then other occurrences of those templates are identified using a multi-phase algorithm. An evaluation demonstrates that the system can successfully reconstruct many aspects of a UI without any prior application-dependant knowledge. To showcase the design opportunities that are introduced by having this additional meta-data, we present the *Waken Video Player*, which allows users to directly interact with UI components that are displayed in the video.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

Keywords: Video; tutorial; pixel-based reverse engineering

INTRODUCTION

In recent years, video tutorials have become a prevalent medium for delivering software learning content [21, 26]. Several recent research projects have attempted to precapture usage meta-data in order to enhance the video tutorial viewing experience by marking up the video timeline [10, 13, 22]. While these types of systems show promise, the immense collection of existing online videos will not possess such meta-data.

We build upon recent work in reverse engineering user interfaces [5, 30], and explore the possibilities and opportunities related to reverse engineering the video tutorials themselves. This is a challenging task, as information from cursor movements, mouse clicks, and accessibility API's are not available to aid recognition. Previous research systems have provided some initial promising results, but have relied on explicit application-dependant rules [21] or templates [26] that may not generalize.

In this paper, we present *Waken*, an application-independent system that recognizes UI components and activities, from an input set of video tutorials. *Waken* does not require any prior knowledge of the appearance of an applications icons or its interface layout. Instead, our system identifies specific motion patterns in the video and extracts the visual appearance of associated widgets in the application UI (Figure 1). Template matching can then be performed on the extracted widgets to locate them in other frames of the video.

Waken is capable of tracking the cursor, identifying application icons, and recognizing when icons are clicked. We also explore the feasibility of associating tooltips to icons, and reconstructing the contents of menus. An initial evaluation of our system on a set of 34 Google SketchUp video tutorials showed that the system was able to successfully identify 14 cursors, 8 icons, and accurately model the hotspot of the system cursor. While the accuracy levels have room for improvement, the trends indicate that with enough samples, all elements could get recognized.

We then present the *Waken Video Player*, to showcase the design opportunities that are introduced by having the meta-data associated with videos. The video player allows users to directly explore and interact with the video itself, almost as if it were a live application. For example, users can click directly on icons in the video, to find related videos in the library that use that particular tool. Users can also hover over icons in the video to display their associated tooltips, or hover over menus to reveal their contents. These techniques are all enabled without any prior knowledge of specific interface layouts or item appearances. We conclude with a discussion of lessons learned and limitations of our work, and outline possible lines of future work.

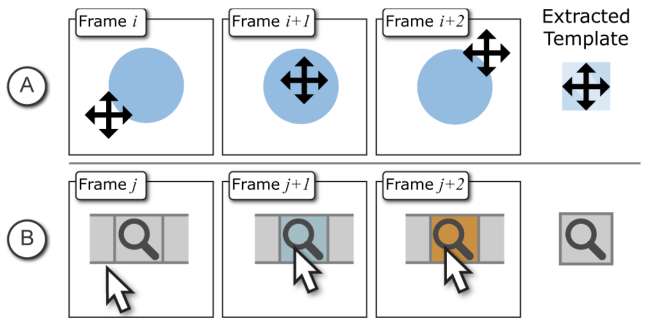

Figure 1. *Waken* uses frame differencing to extract UI elements, such as cursors (a) and icons (b).

RELATED WORK

In this section we discuss previous research on video tutorials, tutorial authoring, and reverse engineering UIs.

Video Tutorials

Video tutorials have become a prevalent instructional aid for learning software [21, 26]. One of the main reasons for the popularity of such tutorials could be due to the benefits of learning software application by directly observing expert use [28]. However, there are also some drawbacks.

One major drawback is navigation issues within longer video tutorials which could lead to misunderstandings of the content [14]. Additionally, users may be unable to keep up with the pace of the instructions, which could lead to reduced knowledge retention [23]. Although Grossman and Fitzmaurice [12] showed performance benefits of short contextual video clips, longer tutorials could still pose significant difficulty for the user.

Past research has shown that navigation of recorded software workflows can be aided by visualizing operation history [22]. Chronicle [13] combines operational history visualizations and timeline-based video summarization and browsing, to aid in exploration of high-level workflows in video tutorials. Ambient Help [21] and Pause-and-Play [26] also provide video timelines based on operation histories.

Our enhanced player provides similar operation history and timeline-based browsing interactions. However, we also extend this literature by exploring how direct manipulation of the user interface in the video tutorial itself could enhance the user experience. While direct manipulation of videos has been previously explored [9] we are unaware of any implementations specific to software tutorial videos. Motivation for such interaction is provided by evidence that users sometimes try to interact with the elements within a documentation article directly [20]. Allowing the user to explore the target application interface directly in the video could also reduce switching between the video and target application [26], and support discovery-based learning [8].

Tutorial Authoring and Usage Data Capture

Video tutorials are typically built from screen capture videos demonstrating usage of the target application. Unfortunately, tutorials authored in this way do not contain additional metadata required for reconstructing operation histories and providing interactive timelines. Existing research has explored ways to automatically capture workflow information [1, 10, 11, 13, 17, 22]. However, those approaches often require special recording tools and instrumentation of the target applications. Additionally, there is already a large collection of tutorial videos on the Internet that were not recorded using such tools. It would be useful to be able to extract workflow information from videos themselves.

Some partial solutions to this problem have been explored through computer visions techniques. Matejka *et al.* [21] reconstructed tool usage information by performing explicit text-based template matching on the AutoCAD command line. While effective for AutoCAD, the technique would not work on other software applications. Pongnumkul *et al.* [26] described a technique to infer icon usage data from existing video tutorials. However, their implementation was also application specific, as it performed template matching on the target application's tool palette. While it may be possible to provide such templates for any application of interest, it would be helpful if such application-dependant templates were not required. We explore an approach, which does not require prior training data.

Reverse Engineering User Interfaces

Advances in computer vision have enabled techniques to identify user interface elements from live applications. Sikuli [30] uses computer vision to identify GUI components in screen captures, search for GUI elements in the interface [31], and automate GUI tasks [2]. However, the underlying computer vision algorithms in Sikuli identify GUI elements based on templates from sample image data.

To minimize the time required for collecting training data, past research [3, 5, 18, 21] explored abstracting identification of different GUI elements and decoupling GUI element representation from predefined image templates. Hurst *et al.* [18] combined a number of useful computer vision techniques with mouse and accessibility API information to automatically identify clickable targets in the interface. Chang *et al.* [3] proposed an accessibility and pixel-based framework, which also allowed for detecting text and arbitrary word blobs in user interfaces. However, when working with video tutorials, mouse information and data from the accessibility API is not available.

Prefab [5] identified GUI elements by using GUI specific visual features, which enabled overlaying of advanced interaction techniques on top of existing interfaces [6, 7]. We build on the insight from Prefab [5] that "it is likely not necessary to individually define each widget in every application, but may instead be possible to learn definitions of entire families of widgets." However, instead of using interactive methods to identify visual features of widgets, we propose an automated application-independent method that learns widget representations from the source video itself.

SYSTEM GOALS AND CHALLENGES

Our work is a first step towards an ultimate goal of automatically constructing interface facades [27], from nothing more than an input set of videos. Previous work on reverse engineering user interfaces rely mainly on "pixels", but inherently have access to other forms of information, such as the cursor position, click and keystroke events, and accessibility information. In contrast, with a video, the only source of information is the pixels of the video itself. Our work provides a first effort at addressing several unique challenges raised by working with videos:

Where is the cursor?

The mouse cursor is a core mechanism for interacting with typical graphical user interfaces. As such, to reverse engineer the activities the user caries out, it is important to know where the cursor is [18]. This stream of information is trivial to access on live applications via global hooks. In videos, this information is absent.

When does the mouse button click?

Related to the cursor position, is understanding when the mouse button clicks. This is another stream of information that can be obtained from a live application, but is absent when working with videos.

Cursor Occlusion

In videos, the screen contents behind the cursor are occluded and unknown. In a live application, taking screenshots by default hides the cursor, so no pixels are occluded. This can be problematic, for example, when trying to understand the appearance of an icon that the cursor has just clicked.

Video Artifacts

When analyzing video screen recordings, there may be artifacts in the video due to lossy codecs being used. The resulting artifacts could be problematic for template matching and frame differencing algorithms.

WAKEN SYSTEM OVERVIEW

Key Insights

The core insight we use in the implementation of *Waken* is that GUI elements possess certain behaviours across applications (Figure 2). Our approach is to use the computer vision technique of frame differencing to identify occurrences of such behaviors, or *motions*, and then extract the associated widgets.

Another insight we build from is that there will be access to a potentially large training set of videos. We can thus use conservative rules to identify target motions. As long as such motion occurs somewhere within the videos, a widget can be extracted. These widgets can then be located in other videos and frames through template matching.

A final insight is that video processing does not need to be accomplished in real time. In contrast, techniques like Prefab [5] need to work in real-time to enable interactive techniques [7]. We thus have the advantage of being able to split the processing of tutorial videos into multiple phases.

Video Processing

Waken processes a set of videos over a sequence of four phases (Figure 3). Phase 1 makes a pass to identify possible cursors from the motions in the video. Phase 2 tracks the cursor in individual frames through template matching. Phase 3 combines the cursor position information with frame differencing to identify and extract clickable icons

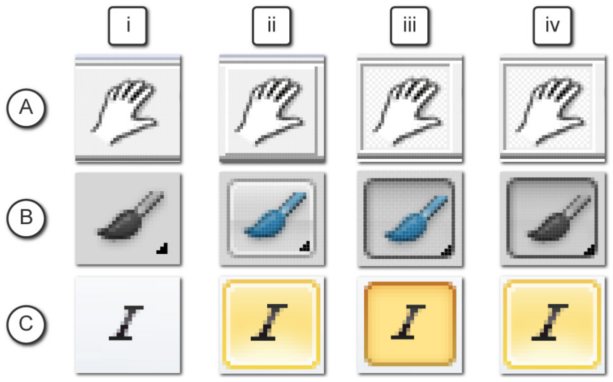

Figure 2. UI Buttons states in a) Google SketchUp, b) Adobe Photoshop, and c) Microsoft Word: i) default, ii) highlighted, iii) clicked, and iv) active.

in the videos. Phase 4 identifies the icons in all other frames and detects when they are clicked. The final phase is also used to identify and associate icons with their tooltips, and menus with their top-level contents.

Source Videos

We make several simplifying assumptions about the source library of videos. First, it is assumed that the input videos are from the same version of a software application. Second we assume videos are captured at a 1 to 1 scale. Finally, we assume that the video is encoded with a reasonable quality level. We discuss the limitations imposed by these assumptions later in this paper.

ELIMINATING CODEC ARTIFACTS

One of the difficulties of reverse engineering videos, in comparison to live applications, is that individual frames may not be pixel perfect representations, due to video compression artifacts. In the presence of a lossy codec, there is a significant amount of noise when comparing consecutive frames (Figure 4), which our algorithms handle.

To compensate for this noise, we filter any small frame-to-frame differences. We first convert the consecutive frames to grayscale and then apply an absolute difference on the two frames. We then apply a threshold function and filter out any difference that is less than 20 on the grayscale (from 0 to 255). This approach preserves the pixels as much as possible, where as a more traditional Gaussian filter would contaminate them.

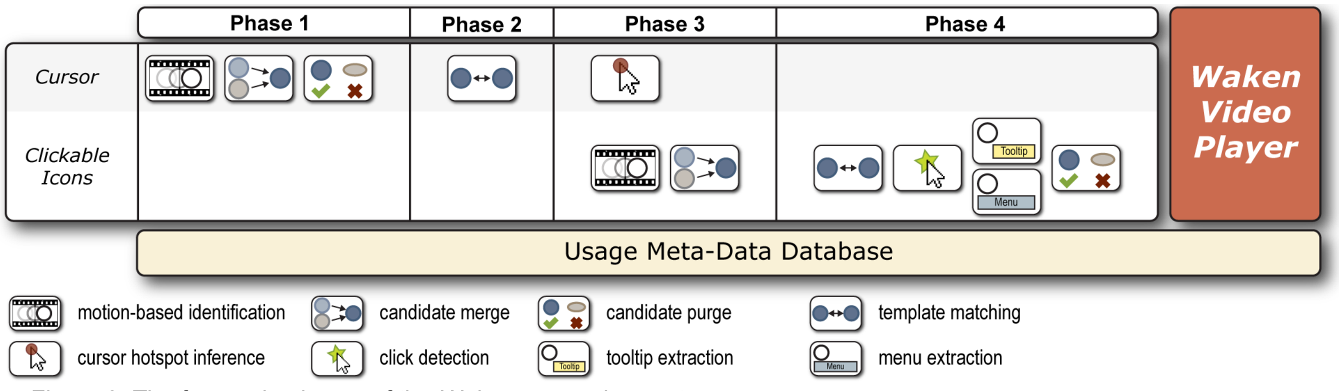

Figure 3. The four main phases of the *Waken* processing system.

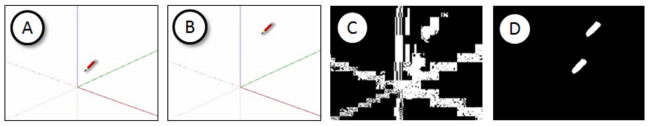

Figure 4. Pixel differences between frames (a) and (b) is show in (c). Applying our filter removes differences due to noise (d).

PHASE 1 AND 2: CURSOR TRACKING

Our first goal is to identify and track the cursor. To remain application independent, we make no assumptions about the cursor appearance, but instead extract the cursor appearance from motion patterns in the video. We do this in a two-phase process described below.

Phase 1: Cursor Identification

In this first phase, we identify the cursor appearance based on specific patterns in frame differences, and then generate a cursor model that can be subsequently used for template matching.

Identifying Cursor Candidates. To identify cursor candidates, we search for frames where only the cursor moves, which creates an easily recognizable pattern (Figure 5). Considering any 3 consecutive frames, if the absolute difference between the first and the second frame, and the absolute difference between the second and the third frame, each contain exactly two blobs, then there is a high likelihood that the blobs identify cursor regions. As an additional confirmation, we compute the contours of the blobs [29] and then match the shapes of the four contours using the OpenCV Hu invariants based [16] shape matching function. If the result of the comparison is greater or equal to 0.95 between all four of the contours then we consider this a candidate cursor match. We store the associated regions from the 3 frames into a new cursor candidate *bucket*. This is repeated for every set of 3 consecutive frames across all videos.

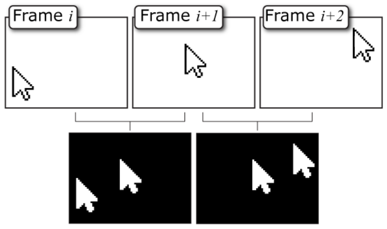

Figure 5. (Top) Consecutive frames and the corresponding blobs in each of the two absolute differences when only the cursor moves (Bottom).

Short Cursor Movements. When the cursor moves by a short distance, the two absolute difference frames might contain only one blob each (Figure 6). In this case, the intersection of the two blobs from the two absolute differences may contain the cursor. We only attempt to identify the cursor in such situations when there are 4 absolute differences between any 5 consecutive frames, each containing a single blob. If this occurs, we calculate the three interactions of the 4 consecutive difference blobs. If the result of comparing the contours of these three intersection blobs is greater than 0.95 we consider this to be three samples of the same cursor in the second, third, and fourth frames and we add the samples to a new *bucket*.

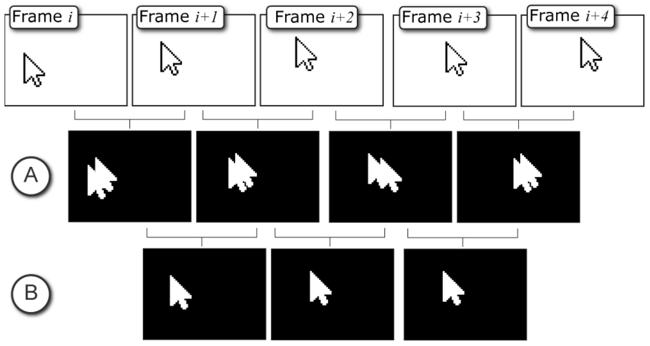

Figure 6. Consecutive frames and a) corresponding single blobs in the absolute difference frames when cursor moves a short distance. b) Cursor shapes in intersection of consecutive differences.

Grouping Cursors Together. In most software applications, there are multiple visual representations of the cursor depending on its location and system mode. We group different cursor *buckets* whose average contour match is equal or greater to 0.95. The templates can then be created from these merged buckets.

Generating Cursor Estimate. The individual instances in each bucket will not be identical. Differences can be due to video artifacts, or transparency in the cursor. In particular, a cursor with high transparency would look much different depending on the background color in the frame it was captured from. Thus, using any single instance for template matching may fail. To determine what the cursors actually look like, we look at all of the captured cursor images for each cursor *bucket*. To build a model of the cursor, we first align all images from a *bucket*, by minimizing the distance between pixels, in RGB space. We then calculate the mean values and variances for the RGB channels for each pixel in the cursor area, across all images in the bucket. This probabilistic template model can subsequently be used for template matching.

To visualize the generated templates, we built a cursor viewer. The cursor viewer shows a 2D representation of the mean pixel values, a 2D representation of the variances, and a 3D representation of the mean pixel values, with height mapped to variance (Figure 7). It can be seen that variance levels are higher when there is transparency or drop-shadows in a cursor.

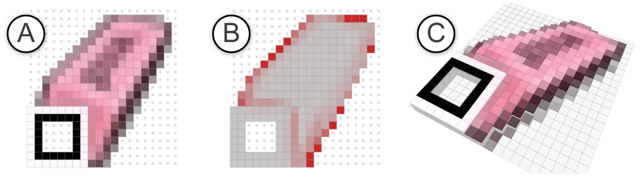

Figure 7. The cursor viewer application visualizes cursor templates we generate. Variance is represented by color in (b) and height in (c).

Phase 2: Cursor Template Matching

In the Phase 2, we iterate through every frame for which a cursor has not already been found, and search for the cursor. We use a probabilistic template matching algorithm against the templates generated in Phase 1. A brute force approach searches for every cursor in every frame.

We consider an individual pixel match when the RGB value of a pixel is within 4 standard deviations of the mean for the corresponding pixel in the cursor model. We consider a template match if 95% of the individual pixels match.

If multiple matches are found, we keep the cursor with the highest template match score. If any cursor achieves a match score of 100%, we keep that cursor and move to the next frame. To improve processing time, we first look for the same cursor as the previous frame, in the proximity of the previous cursor position. If that old cursor is found with a 95% match, we keep that cursor and move to the next frame. If a frame contains no cursor matches, then we assume that the cursor is not present in that frame.

A filtering heuristic is used to account for potential false positives when static icons take on the appearance of a cursor. For each cursor, we generate the distribution of (x, y) coordinates that it was found at. If the number of occurrences at any single (x, y) coordinate is more than 4 times the standard deviation of this distribution, then those matches are discarded, and the template matching is repeated to find a new cursor in the associated frames.

PHASE 3 AND 4: ICON IDENTIFICATION

The technique described above allows us to infer the location of the cursor. In this section, we describe how the cursor information can be leveraged to identify clickable icons in a user interface video.

As with cursor tracking, icon identification is accomplished across two phases, Phase 3 and 4 of our algorithm. We first identify candidate icons based on consecutive frame image differences. We then use this information to locate candidates in other frames of the video.

Phase 3: Icon Identification

Identifying Candidate Icons. As with cursor identification, we identify icons using a conservative set of rules. Imposing such rules reduces the number of icons which will be identified in Phase 3, and remaining icons will be located in Phase 4 through template matching.

We identify candidate icons from instances when they are clicked in the video. We look for a visual pattern that is common across typical user interfaces, where the cursor: enters the icon region causing the region to highlight, briefly pauses, clicks the icon causing the icon region to switch to a "depressed" state, and exits the icon.

To identify such patterns, we start by searching across all instances where there cursor location changes across 3 consecutive frames, *i*, *i+1*, and *i+2*. We then look at the differences between frames *i* and *i+1*. We consider the movement a potential candidate if there is a blob that intersects the cursor position from frame *i+1*, that does not intersect the cursor position from frame *i*. This indicates that the cursor has potentially moved into an icon, and the icon has highlighted. We then require that blob to *not* be present in the difference of frame *i+1* and *i+2*, which ensures that the cursor has remained within the icon. This pattern identifies a highlighted icon, and the associated region (Figure 8).

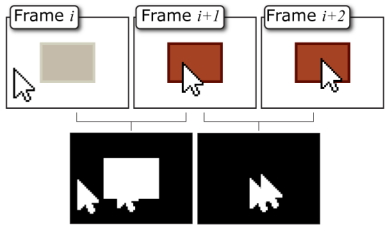

Figure 8. Typical cursor movement when approaching and acquiring an icon and resulting frame differences.

We then test to see if the icon which was highlighted gets clicked. As long as the cursor remains within the highlighted region, we search for 3 consecutive frames, *j*, *j+1*, and *j+2*. To detect a click, we require the cursor to be stationary, within 2 pixels, between frames *j* and *j+1*, and fully stationary between frames *j+1* and *j+2*. In addition, there must be a new difference blob between frames *j+1* and *j+2*, which intersects the highlight region. This indicates the icon has been clicked (Figure 9). This identification process will identify root level menu clicks as well.

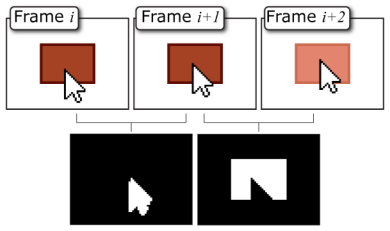

Figure 9. Typical cursor movement when clicking an icon and resulting frame differences.

To extract the appearance of the icon, we store the image of the highlight region from frame *i* (before it was occluded by the cursor), and from the first frame after the click which the cursor is no longer occluding the region.

Grouping Icons Together. Each time we extract an icon, we check if it matches the appearance of previously detected icons, using OpenCV's standard template matching algorithm, with a 0.95 threshold. If there is no match, the icon is added. If it matches with a previous icon, they are grouped together. The final icon templates are generated by averaging the pixel values of all icons in a group.

Inferring the Cursor Hotspot. We also use Phase 3 to infer the hotspot of the cursor. Each time an icon is highlighted, we can detect which pixels of the cursor intersect the highlighted region. We define the hotspot as the region of pixels which are in the highlight region at least 95% of the time an icon is highlighted. We use this hotspot to test when the cursor enters the area of an icon in Phase 4.

Phase 4: *Detecting Icons, Tooltips, and Menus*

Icon Template Matching. In Phase 4 we use template matching to find instances of the icons extracted from Phase 3. For simplicity, in our implementation, we only search a small area around the initial location where we found the icon. However, this could be extended to the whole frame to account for customized layouts, but this would increase processing time.

Icon Click Detection. We can now identify icon clicks by using knowledge of the cursor hotspot and icon positions. While the cursor hotspot is in an icon, we look for 3 consecutive frames, *i*, *i+1*, and *i+2*, where the cursor remains stationary. To detect a click, there must be a difference blob between frames *i* and *i+1*, which intersects the icon region, and that blob cannot be in the difference between frames *i+1* and *i+2*. When this occurs, we record a click in frame *i+1*. Blob larger than the icon region we classify as a menu.

Purging Icons. We developed two simple mechanisms to reduce occurrences of false positives. First, any icon that has only a solid color is dismissed. Second, we remove icons that were rarely identified through template matching: if an icon is identified is Phase 3 95% of the time, relative to the total number of times it is identified between Phases 3 and 4, we discard it.

Tooltip and Menu Identification. In addition to identifying clickable icons, we can begin to reconstruct additional UI information associated with the icons. An ultimate goal would be to automatically construct interface facades [27] from the videos. We take an initial step in this direction, and infer the tooltip information associated with icons, and the top-level contents associated with menus.

To detect a candidate tooltip, we consider consecutive frames in which the cursor is within an icon and does not move. If the difference between consecutive frames contains a single blob that does not intersect with the highlighted area of the icon, it is considered a tooltip. To capture the tooltip appearance, we take the region of the blob in the last frame that blob is detected. This ensures the tooltip is captured at full opacity, to account for user interfaces where tooltips gradually fade-in. We calculate the mean value of RGB for each pixel in the tooltip from all occurrences for that particular icon. This gives us the final appearance of the tooltip that we use.

We use a similar approach to detect the top-level contents associated with a menu. When we detect the click of an icon that has been classified as a menu, we capture the image of the top-level menu items, defined as the blob area, with the icon area subtracted. As with tooltips, we take the image from the last frame which the blob occurs, and we average the RGB value across all occurrences, to get its final appearance.

One limitation of this approach is that it does not handle menu or tooltip associations changing due to application context. Inferring the correct context is a more challenging task, which requires additional research.

IMPLEMENTATION AND INITIAL RESULTS

We implemented *Waken* using the C++ implementation of OpenCV[1] version 2.3 on Windows 7. The system used a Postgres[2] version 9.1.3 database to store and access processed usage data. The system ran on a 64-bit Windows 7 Enterprise with Intel Xeon 2 quad-core 2.53GHz processors and 12GB or RAM.

We ran *Waken* on a library of 34 Google SketchUp[3] videos downloaded from the Google SketchUp training website[4]. The videos were in the Apple QuickTime format with *H264 – MPG-4 AVC* codec. The resolution of the videos was 640×480, with a 1:1 scale, and frame rates ranged from 8fps to 24fps. The videos lasted from 1-8 minutes, and totaled 2 hours and 22 minutes of playing time.

Total analysis time for all videos was approximately 50 hours. The large majority of time was spent searching for cursor templates in Phase 2. Although processing time is high, the system implementation is highly parallelisable, and the process time could decrease with more processors.

Below we provide some of the results and observations from this test set on the core features of our algorithm: cursor tracking, icon identification, and icon click detections. Refining and evaluating the UI reconstruction techniques (*i.e.* tooltip and menu recognition) is left for future work.

We do not expect to be able to reconstruct all cursors and icon clicks in this test, with this relatively small input set of videos. Our goal in this initial evaluation is to get an indication of how well the system will work, and identify limitations that can guide future work and improvements.

Results

We discuss the results and observations for each of the phases of the *Waken* processing system. In order to obtain the ground truth, one of the authors manually coded all cursor changes and icon click events in the videos. Phase 1 was performed on all 34 videos. However, we removed 18 videos as outliers due to the large number of frames that were not at 1 to 1 scale, due to zooming effects. In Phases 2 to 4, we ran on the remaining 16 videos. We present the results based on the data from these 16 videos.

Phase 1: Cursor Identification and Template Generation. The total number of frames that contributed to motion-based identification of cursors was 1,847 frames out of 96,223 (1.9%). This provides a rough guideline as to how often cursors will exhibit the required behavior to be tracked by our algorithm.

The system was able to successfully recognize 14 different cursors, leaving 5 cursors unrecognized. There were no false positives. In most cases, the unrecognized cursors were rarely used. This would be alleviated by larger data sets, which would increase the chances of each cursor being recognized.

[1] http://opencv.willowgarage.com
[2] http://www.postgresql.org/
[3] http://sketchup.google.com/
[4] https://sites.google.com/site/sketchupvideo/

The quality of the template cursors depended on how frequently the cursor was observed. Figure 10 shows mean pixel RGB values and variance for the cursor with the greatest number of templates (N=656). A side by side comparison with the actual cursor icon shows that the templates can be extremely accurate with enough samples to build from.

Overall we feel these results are impressive, given that we had *no* a priori knowledge of cursor appearance.

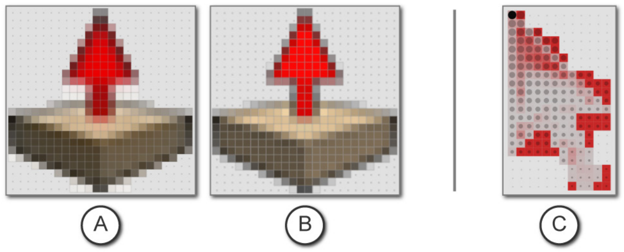

Figure 10. a) Cursor reconstructed using our algorithm on the test data. b) Actual cursor icon. c) System cursor with hotspot estimate (solid black dot indicates 95% confidence).

Phase 2: Cursor Tracking Results. Attempting to track the cursors in every individual frame resulted in an accuracy of 77.28% (SD=13.19) and recall of 72.67% (SD=14.48) (Figure 11). There were no false positives. Accuracy for individual cursors varied greatly, depending on the number of samples used to build its template (Figure 12). When removing the 8 cursors whose templates were generated from less than 10 samples, the average accuracy increased to 81.39% (SD=5.98) and the average recall increased to 81.28% (SD=12.78) (Figure 11).

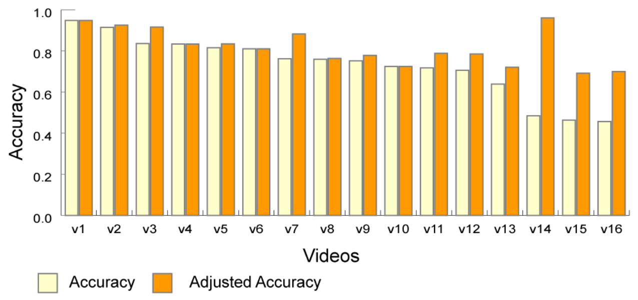

Figure 11. Cursor tracking accuracy by video. Adjusted accuracy shows results when the low-sampled cursors are removed from the data set.

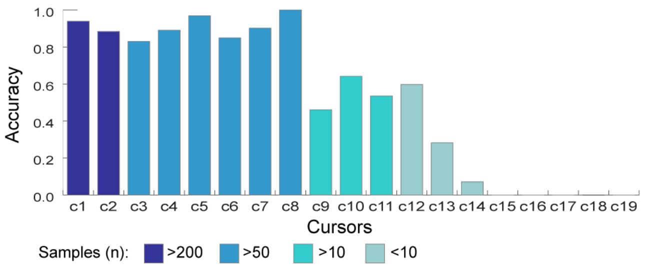

Figure 12. Cursor tracking accuracy by cursor.

Phase 3: Icon Identification Results. Phase 3 was not able to identify any icons in the videos with 8 and 15fps, suggesting that the frame rate could affect the identification accuracy. We therefore continued our phase 3 analysis on the 4 videos that had 24fps frame rate.

In these 4 videos, the system was able to successfully identify 8 icon buttons. Five buttons were not recognized, and there was one false positive. However, any button that was clicked at least four times was accurately recognized. The system successfully identified the cursor that clicked on the icons, and accurately estimated its hotspot (Figure 10.c).

Phase 4: Click Tracking Results. While we didn't manually calculate tracking accuracy of the icons which were identified in Phase 3, our observations indicated that accuracy was above 99%.

The precision and recall of click tracking was 81.79% and 29.48% respectively. However, for videos with 24fps the precision was 87.97% and recall was 66.04%. This suggests that the click usage data can only be usable from videos with larger frame rates, but on the other hand the user interface reconstruction works with videos with very low frame rates.

Lessons Learned

Given this is a first effort in reconstructing UI elements from videos, without any application dependant knowledge, we are encouraged by these results. *Waken* was able to successfully recognize 14 cursors, 8 icons, and accurately model the hotspot of the system cursor. While the accuracy levels have room for improvement, the trends indicate that with enough samples, all elements could get recognized.

More importantly, the analysis has identified a number of challenges leaving room for future improvements.

We found some cursors were not recognized because they were rarely used. However, one of the cursors appeared frequently in the videos, but was rarely recognized in Phase 1. It turned out that this cursor was used in drawing content and almost was always associated with other movements in the working area. It would be helpful to enhance Phase 1 so that it could identify the cursor, even when other content in the frames are changing.

Low frame rate was also problematic preventing icon click identification. The problem is that key frames required for our rules may be skipped when the video has a low frame rate (for example, the cursor remaining stationary for one frame after a click event). To overcome this problem, we could attempt to generate templates for "active" states of the buttons. We could then perform template matching to determine if an icon has taken on its active state, similar to the strategy used in Pause-and-Play [26].

Finally, the lack of false positives in the tests suggests that heuristics we use to identify and track cursors and icon buttons describe the motion well. However, there is a potential to, in the future, relax some of the rules and test how that affects the overall accuracy of the system.

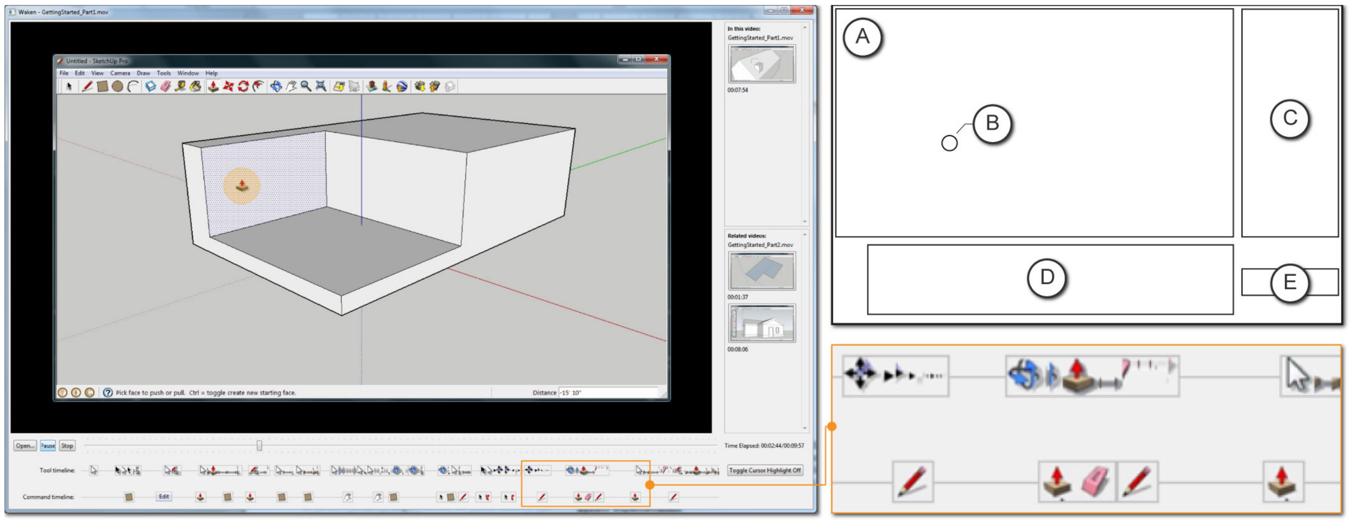

Figure 13. The *Waken Video Player* user interface components. a) The playback area. b) Highlighted cursor. c) Navigation panel. d) Event based timelines. e) Cursor highlight toggle.

WAKEN VIDEO PLAYER

We developed a video player that serves as a demonstration of some of the design opportunities that our video analysis system enables. The *Waken Video Player* brings the videos to life, by allowing users to directly interact with them.

System Implementation

The video player was implemented on the Java 6 SE platform, using Xuggle, a free open source video manipulation library. A Postgres version 9.1.3 database was used to store and access the tutorial video usage meta-data.

Main Player Components

Figure 13 shows the main components of the video player. These components include a playback area (Figure 13a), event-based timelines (Figure 13d), a navigation panel (Figure 13c), and a toggle button for cursor highlighting (Figure 13e). Users can open video files, which automatically loads the associated meta-data from the database.

Cursor Highlighting

The player contains a toggle to highlight the cursor. This is an effect in many screen captured videos, but must be hard-coded into the video. Viewers thus have no capability to enable or disable it. By leveraging the cursor location data generated by *Waken*, we can allow users to dynamically enable or disable cursor highlighting (Figure 13b).

Enhanced Timeline Navigations

Icon Track. Similar to previous systems [13, 21, 26] we mark up the timeline with the icons the tutorial author clicked on. Users can click on the icons to navigate to associated time in the video. While the design is similar to previous systems, the technique to obtain this data, without any prior knowledge of the UI appearance, is novel.

Cursor Track. We also visualize the cursor representation in the timeline, to show the user when cursor changes occur. This may help users locates areas of the video where a mode changes or a specific type of action occurred that did not result from an icon click.

Direct Video Navigation

Users can interact with the individual icons that are recognized in any frame of the video. When a user pauses the video, bounding box overlays are rendered on top of the video indicating any recognized component. Users can then click directly on an icon to search for other moments in the video when that icon is used. This is similar to previous direct video navigation techniques [9], but is novel for software videos. When the user clicks on an icon, a panel on the right shows thumbnails representing all the times in the video when that icon was used. A second section shows instances of when that icon was used in other videos. Thus, a user can select a tool in the current video and see a list of frames from related videos which also use the tool. Clicking on a related video opens the video and seeks to the frame when that tool is being used. This can be thought of as event-based interactive hyperlinks between videos.

UI Exploration

In our player, we can show tooltips within the video canvas, even if the video itself contains no tooltips. To see a tooltip, the user simply hovers over the corresponding icon of interest while the video is paused. A bounding box is drawn around the tooltip to distinguish it from actual content from the video. Tooltips within the video allow users to get simple information about tools, just like they would in a live user interface (Figure 14a). We also added the ability to expand and collapse top-level menu contents, by hovering over the root menu. This adds further abilities for a user to explore an interface directly from the video itself. As with tooltips, a bounding box is drawn to distinguish it from an actual menu in the video (Figure 14b).

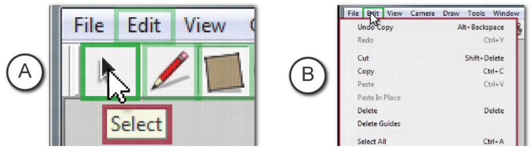

Figure 14. a) A tooltip is rendered over the video b) Menu contents are rendered over the video.

DISCUSSION AND LIMITATIONS

We have demonstrated an application independent system that is able to reconstruct events which occur in video tutorials. We have also shown how this data can allow for a set of novel, direct manipulation interactions, on the videos themselves. Results from an initial test of 16 videos provided some encouraging results, but there are still a number of ways our work can be extended and improved.

Waken has some inherent limitations. For example, we cannot capture interactions resulting from keyboard usage. Furthermore, if, across the entire library of videos, an icon is *never* clicked, or a tooltip is *never* shown, then it will not be recognized. Other limitations are addressable. For example, we did not try to account for repositioning or customization of palettes, but searching the entire frame for template icons would likely address this challenge.

Our study showed that the performance of *Waken* is dependent on the quality of the videos. Today's CPU capabilities and network infrastructure do make it easier to record and share high-quality videos. As such, limitations regarding video quality will not likely be an important issue in the years to come. That being said, it would still be desirable for the system to work well on lower quality videos.

Our initial tests with loseless videos showed very high accuracy rates. Our evaluation was performed on a set of videos that had been compressed with a lossy codec, and accuracy rates were still acceptable. However, as the amount of loss increases, more codec artifacts will begin to impact accuracy. While such artifacts can be overcome by tuning the thresholds we used, at some point, alternative strategies would need to be considered. Similarly, at 8 frames per second, accuracy levels begin to degrade. Refinements to account for low-quality videos should be explored.

Our work was also limited to videos that were recorded at a 1 to 1 scale. The main impact of analyzing downscaled videos would be on the quality of the cursor templates our system generates. Generating templates for multiple resolutions could be a strategy to overcome the scale issue [26]. This type of technique could also be used to account for pan and zoom operations that may occur in the videos.

One of the main advantages of our work is that there is no application-specific information required for the algorithms to work. However, if the end goal is to run our system on a large library of videos, from multiple applications or application versions, there would need to be a way to distinguish what application was in the video. Otherwise, the system would try to match icons from one application to another. In our study, the system was run on a library of videos from a single application. However, future work could look at techniques to automatically distinguish or group applications from the video itself.

Our system depends of finding consecutive frames in a video that have specific patterns in their differences. If a region of the video is changing throughout the entire video, this could prevent any recognition from occurring. For example, if an animation or video is playing within the video itself. One way to address this could be to mask areas of the video that are changing for large periods of time.

A strength of our system is that information from all videos in the library are combined. For example, no cursors may be found in a video from Phase 1, but in Phase 2, the cursor can be tracked through template matching. Similarly, users can view tooltips in videos which they never appeared in.

While our system does not require any application-dependant rules or templates, it does depend on some high-level rules. For example, the recognition of clickable elements requires those elements to become highlighted when the cursor enters, and to provide an additional highlight when the element is clicked. Although this type of behavior is prevalent in most of today's user interfaces, there will be instances of widgets that do not elicit such behaviors.

Finally, while *Waken* was designed to be application-independent, our test was performed only for one application. However, our informal tests on videos for Microsoft Word and Adobe Photoshop found comparable identification and tracking accuracy for cursors and clickable icons to results from our study (Figure 15). Unless the UI mechanics are much different, we expect our system to work for most software applications.

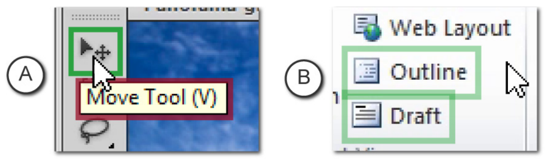

Figure 15. Cursor and clickable icon recognition from Adobe Photoshop (a) and Microsoft Word (b).

FUTURE WORK

Our work opens up a number of interesting opportunities for future research.

With respect to the recognition system, new techniques to address the limitations described in the previous section could be developed. In particular the ability to automatically recognize software and versions, or the recording resolution, could help improve the robustness of the system. Also, some motion pattern recognition rules may have been stricter than necessary, although our current implementation was intentionally biased towards avoiding false positives. One of our goals for future work is to explore how to relax some of these rules without sacrificing identification accuracy. Further investigating and addressing the issue of codec artifacts would also increase the scope of videos where our work would be applicable.

Our work was guided by numerous previous systems that have performed recognition of on-screen activities. We used a combination of frame differencing and template matching. However, our algorithms could likely be improved by integrating other concepts that have been seen in prior art, such as interpreting interface hierarchy [6].

We leveraged the information collected by the recognizer to provide an enhanced video player. However, the information we collect from the recognition opens up several alternative design opportunities. For example, it would be interesting to investigate the feasibility of automatically translating a recorded video into a different language, similar to what has been explored for live user interfaces [6].

Finally, the menu reconstruction provides an initial step towards automatically reconstructing an entire user interface façade [27]. It would be interesting to explore how close to a full reconstruction could be obtained by extending our work. This would require modelling a much deeper set of application components, such as application states, hierarchal menus, and floating palates.

CONCLUSION

We have presented *Waken*, a new system that leverages computer vision techniques to extract usage information from tutorial videos, such as the movement of the cursor, the icons which are clicked, and the tooltips associated with interactive elements. A main contribution of this system is that it does not require any application specific rules or templates. We have demonstrated a number of novel interactions that are enabled, such as allowing the user to interact directly with the video itself. Results from our initial test indicate a number of fruitful areas for future work.

REFERENCES

1. Bergman, L., Castelli, V., Lau, T., and Oblinger, D. 2005. DocWizards: a system for authoring follow-me documentation wizards. *ACM UIST*, 191-200.
2. Chang, T., Yeh, T., and Miller, R. C. 2010. GUI testing using computer vision. *ACM CHI*, 1535-1544.
3. Chang, T., Yeh, T., and Miller, R. C. 2011. Associating the visual representation of user interfaces with their internal structures and metadata. *ACM UIST*, 245-256.
4. Cheng, K., Luo, S., and Chen, B. 2009. SmartPlayer: user-centric video fast-forwarding. *ACM CHI*, 789-798.
5. Dixon M. and Fogarty, J. 2010. Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure. *ACM CHI*, 1525-1534.
6. Dixon, M., Leventhal, D., and Fogarty, J. 2011. Content and hierarchy in pixel-based methods for reverse engineering interface structure. *ACM CHI*, 969-978.
7. Dixon M., Fogarty, J., and Wobbrock, J. 2012. A general-purpose target-aware pointing enhancement using pixel-level analysis on graphical interfaces. *ACM CHI*, 3167-3176.
8. Dong, T., Dontcheva, M., Joseph, D., Karahalios, K., Newman, M. W., and Ackerman, M. S. 2012. Discovery-based Games for Learning Software. *ACM CHI*, 2083-2086.
9. Dragicevic, P., Ramos, G., Bibliowitcz, J., Nowrouzezahrai, D., Balakrishnan, R., and Singh, K. 2008. Video browsing by direct manipulation. *ACM CHI*, 237-246.
10. Fernquist, J., Grossman, T., and Fitzmaurice, G. 2011. Sketch-sketch revolution: an engaging tutorial system for guided sketching and application learning. *ACM UIST*, 373-382.
11. Grabler, F., Agrawala, M., Li, W., Dontcheva, M., and Igarashi, T. 2009. Generating photo manipulation tutorials by demonstration. *ACM SIGGRAPH*. 66. 9 pages.
12. Grossman, T. and Fitzmaurice, G. 2010. ToolClips: an investigation of contextual video assistance for functionality understanding. *ACM CHI*, 1515-1524.
13. Grossman, T., Matejka, J., and Fitzmaurice, G. 2010. Chronicle: capture, exploration, and playback of document workflow histories. *ACM UIST*, 143-152.
14. Harrison, S. M. 1995. A comparison of still, animated, or nonillustrated on-line help with written or spoken instructions in a graphical user interface. *ACM CHI*, 82-89.
15. Hategekimana, C., Gilbert, S., and Blessing, S. 2008. Effectiveness of using an intelligent tutoring system to train users on off-the-shelf software. *Society for Info. Tech. and Teacher Education Int'l Conf.*, AACE.
16. Hu, M. 1962. Visual pattern recognition by moment invariants. *IRE Trans. Inf. Theory* IT-8, 8, 1409-1420.
17. Huang, J. and Twidale, M. B. 2007. Graphstract: minimal graphical help for computers. *ACM UIST*, 203-212.
18. Hurst, A., Hudson, S. E., and Mankoff, J. 2010. Automatically identifying targets users interact with during real world tasks. *ACM IUI*, 11-20.
19. Kelleher, C. and Pausch, R. 2005. Stencils-based tutorials: design and evaluation. *ACM CHI*, 541-550.
20. Knabe, K. 1995. Apple guide: a case study in user-aided design of online help. *ACM CHI*, 286-287.
21. Matejka, J., Grossman, T., and Fitzmaurice, G. 2011. Ambient help. *ACM CHI*, 2751-2760.
22. Nakamura, T. and Igarashi, T. 2008. An application-independent system for visualizing user operation history. *ACM UIST*, 23-32.
23. Palmiter, S. and Elkerton, J. 1991. An evaluation of animated demonstrations of learning computer-based tasks. *ACM CHI*, 257-263.
24. Petrovic, N., Jojic, N., and Huang, T. S. 2005. Adaptive Video Fast Forward. *Multimed. Tools Appl.* 26, 3.
25. Pongnumkul, S., Wang, J., Ramos, G., and Cohen, M. 2010. Content-aware dynamic timeline for video browsing. *ACM UIST*, 139-142.
26. Pongnumkul, S., Dontcheva, M., Li, W., Wang, J., Bourdev, L., Avidan, S., and Cohen, M. F. 2011. Pause-and-play: automatically linking screencast video tutorials with applications. *ACM UIST*, 135-144.
27. Ramesh, V., Hsu, C., Agrawala, M., and Hartmann, B. 2011. ShowMeHow: translating user interface instructions between applications. *ACM UIST*, 127-134.
28. Shneiderman, B. 1983. Direct Manipulation: A Step Beyond Programming Languages. *Compt.* 16, 8, 57-69.
29. Suzuki, S. and Abe, K. 1985. Topological structural analysis of digitized binary images by border following. *Comput. Vision Graph.*, 30(1), 32-46.
30. Yeh, T., Chang, T., and Miller R. C. 2009. Sikuli: using GUI screenshots for search and automation. *ACM UIST*, 183-192.
31. Yeh, T., Chang, T., Xie, B., Walsh, G., Watkins, I., Wongsuphasawat, K., Huang, M., Davis, L. S., and Bederson, B. B. 2011. Creating contextual help for GUIs using screenshots. *ACM UIST*, 145-154.