Applying Force Fields To Black-Box GUIs Using Computer Vision

Florian van de Camp
Fraunhofer IOSB
Karlsruhe, Germany
florian.vandecamp@iosb.fraunhofer.de

Rainer Stiefelhagen
Karlsruhe Institute of Technology, Fraunhofer IOSB
Karlsruhe, Germany
rainer.stiefelhagen@kit.edu

Abstract

While new input modalities emerge, many established applications have not been adjusted to these new possibilities of interaction. Assisting technologies like force fields have been shown to enhance interaction, but rely on context information that is not provided by existing applications. We present a computer vision based approach that can automatically locate user interface elements to apply assisting technologies to any application without modification. In a user study we show that user interface elements can be detected reliably and allows for automatic placement of force fields that improve interaction in both speed and accuracy. Our approach makes new interaction modalities like pointing gestures usable with existing applications without need for modification or adaptation.

1. Introduction

As new input modalities emerge, specifically tailored user interfaces become more popular which better support the properties of these often less accurate but more natural modalities. Larger user interface elements, more space between elements and visual feedback are only a few of the adaptations that make applications usable for input modalities like touch and pointing. This is necessary, because even if systems that detect pointing gestures will get more accurate, humans are not able to move their arms perfectly accurate or point perfectly steady towards a target, especially from a distance.

Another way of adapting interfaces to work better with these modalities are assisting technologies that have been shown to enhance interaction speed and robustness [2][3][13] but rely on context information. In many cases adaptation or even rewriting of applications is not an option, so the required information about the location of user interface elements is not available. In high security environments such as control rooms, interfaces are often streamed over a network and only mouse and keyboard events are send to the machine. To still be able to make use of these new input modalities, the user input has to be mapped to input events known to the applications, like mouse clicks and movements but still deal with the inaccuracies to make todays applications usable with tomorrows input modalities.

2. Related Work

Assisting technologies have mostly been applied to mouse interaction. Ahlström et al. [2] have shown that force fields improve pointing performance in realistic GUI situations. Force fields warp the cursor towards the center of a target within an area around it with a fixed strength. Since todays GUIs are mostly tailored towards mouse interaction, most user interface elements can be considered rather small for less precise modalities. Especially for such small targets, assisting technologies can significantly improve interaction [3]. With knowledge of the locations of interface elements, several other assisting technologies can be applied [13, 9, 7]. Guiard et al. [8] take this knowledge to the extreme with "Object Pointing", where the cursor always jumps to the closest interface element. While fast, this might be irritating with a direct input modality such as pointing. Analyzing the screen at run time is not uncommon. Yeh et al. [14, 5] developed "Sikuli" a tool that can query databases for documentation using screen shots of windows or dialog boxes using template matching and local features. Instead of using a computer vision approach, both Burschka et al. [4] and Memon at al. [10] rely on querying operating system specific functions to analyse the layout of user interfaces, one for interfacing a visual touch system the other for automated software testing. Dixon at al. [6] analyse screen content with their Prefab system. Unlike our system, they rely on a database of features created offline for a specific application or interface that are compared to the screen content for locating known elements.

3. Design and Implementation

There are two aspects to controlling existing applications with new input modalities while making use of assisting

Figure 1. Cursor icons to indicate click progress.

technologies. First, we need to be able to emulate the kind of input events current applications understand, like mouse movements and clicks. This is easily possible on any major operating system. Secondly, we need to know where user interface elements are. Since existing GUIs are not aware of the input modalities, that information is not available and as windows can be moved and resized, locations can change all the time. We exploit the fact that interface elements are usually quite distinctive and present a template matching based approach to automatically locate user interface elements on the screen. While it is possible to provide the system with widgets up front, the focus lies on extracting and learning the appearance of interface elements on the fly by observing user behaviour. The gained knowledge about widget locations is then used to place force fields that enable the use of pointing interaction for existing applications tailored towards mouse input.

3.1. Screen scraping and cursor manipulation

The complete screen can be captured very efficiently on any major operating system as demonstrated by the variety of screen sharing applications. Manipulating the mouse cursor position as well as triggering clicks is also possible programmatically on any major operating system. The control of the cursor allows warping of the cursor position depending on the position of interface elements which is the main requirement for most assisting technologies. A popular method for triggering clicks using pointing gestures is a dwell timer, where keeping the cursor still for a certain duration will trigger a click. An important aspect to making this technique intuitive is visual feedback. In many cases it is possible to modify the mouse cursor icon to give the user visual feedback about the click progress as illustrated in Figure 1, even without access to or modification of the target application.

3.2. Template Matching

Most desktop applications are built using GUI toolkits that provide a limited number of user interface elements. These elements always look and work the same as this helps users to understand new applications. This property makes template matching a suitable technology for locating user interface elements. While template matching is not robust enough for most computer vision applications due to sensor noise and scene clutter, this is not the case when working with screen captures. The appearance of an object is identical in each frame and the same types of elements (e.g. buttons or check boxes) are still very similar despite different states or labels. Since the number of common GUI elements is limited and the elements are known, templates can be created that can locate elements via cross-correlation on the current screen capture. While this basic approach works well for many scenarios of well known environments, in the real world we can not expect to know the appearance of all interface elements up front. Skinnable applications and even operating systems as well as the virtually unlimited design possibilities on the web limit any approach that relies on knowing what an interface will look like.

Figure 2. The mouse cursor indicates the click position, the red boundaries show the extracted contour.

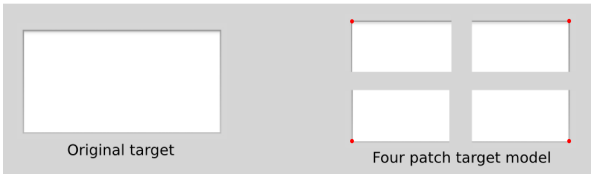


Figure 3. Creation of an initial target model (Anchor points are indicated by the red dots).

We do, however, have another source of information about where user interface elements are located: the user input. It is very likely that whenever a user triggers a click the mouse cursor location is on an interface element. In the next section we describe how we automatically extract previously unknown user interface elements using this information and besides using the template directly to locate more widgets, refine the template over time to create a model that generalizes to widgets that vary in size and appearance.

3.3. Automatic Template Learning

Whenever a click occurs, the surrounding area in the current screen capture is scanned for potential textures to be used as a template by performing canny edge detection on the area around the click in the captured screen. Since user elements are usually designed to stand out and be easily recognizable, the outlines of a widgets are extracted very accurately. Larger contours are found in these outlines using the Teh-Chin chain approximation algorithm [12]. Comparing the location of the click with all found polygons using the point-in-polygon test, we find candidates for widget boundaries. From these candidates the first enclosing, convex contour is picked. Three examples can be seen in Figure 2, the mouse pointer shows the position of the click, the red rectangle the extracted contour.

While it is possible to directly use the extracted area as a template, it will only find identical or very similar widgets. In order to allow the model to match instances of

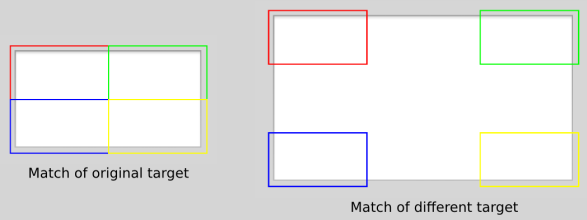


Figure 4. Generalisation of an initial target model.

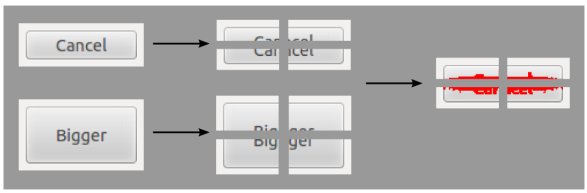


Figure 5. Automatic generalization (transparent pixels are drawn red for visualisation).

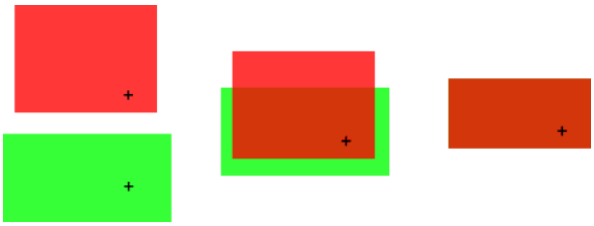


Figure 6. Patch alignment for refinement.

the same widget with different sizes, we represent widgets by parts that can move with respect to each other. We define these parts as follows. The extracted area is expanded by 5 pixels in all directions and split into 4 parts of equal size by splitting the area along the horizontal and vertical axes (Figure 3). This way, the distinctive gradients are preserved and the four image patches are saved along with an anchor point for each. The anchor point for each patch is the point on the gradient furthest away from the original widgets center. This constitutes a first target model which can be used to find widgets by matching all four patches on the screen. Taking their layout into account, correct targets can be found in the list of matches. This means, for an example, that the top right patch always has to be right of the top left patch but on the same vertical position. Also, there can not be any other matches in between a top left match and a top right match. Since user interface layouts are very structured and matches are very reliable because of the consistent appearance of interface elements, this heuristic allows to filter out any incorrect match combinations. For some widgets the split of the original template already allows for some generalization. For an example, empty line edits or text boxes of most sizes will be found after a first line edit or text box has been extracted (See Figure 4).

However, other widgets differ more from each other and are usually not found using the model created from the first extracted target. To not only find widgets that are identical to widgets that have already been selected (allowing support from every second click on) we automatically generalize the existing model as more widgets are extracted. To successfully generalize an existing model using a new extraction, we need to make sure the widgets are of the same type. To check for matching, existing target models we align the corresponding patches of the extracted model with those of the existing models. There have to be at least 20 connected pixels identical in both patches for all four corner patches to consider an existing model to match the newly extracted one. If the extracted model matches an existing one, the existing model is refined. The refinement is repeated for each pair of corresponding patches of the two models. After alignment of the patches at their anchor points the patches are reduced to the overlapping area (Figure 6). The patches are then compared pixel for pixel and pixels that differ are set to transparent. Transparent pixels are simply ignored when the patch is later used for template matching on the screen. Figure 5 shows how the original patches of the first model (top) are generalized to match the second target (bottom) as well. The initial constraint of identical pixels in both patches avoids extreme over generalization. In addition the refined patch has to contain edges to avoid ending up with a homogeneous patch that would not allow for sufficient distinction. If this condition is satisfied for all four refined patches the refinement is considered successful and applied to the existing model. The newly extracted model is then deleted as it becomes redundant. If the refinement does not succeed, however, the original model is not modified and the newly extracted model is kept. This guarantees that the new target and any identical targets will be found at the price of increasing the runtime slightly.

3.4. Runtime considerations

Considering display resolutions of large displays and video walls common for pointing interaction, shifting the templates over the whole image can be time consuming. The generalizing models that are build on the fly avoid the need for separate templates for every individual target and therefor provide a significant speed up. Since we know where a user is pointing, there is no need to enhance every interface element at all times. Instead, templates are only matched against a region of $500px \times 500px$ around the pointing direction. A target can be matched against the $500px \times 500px$ region in $70ms$, with a common set of 10 targets this still allows an update rate for force field positioning of $1.4fps$ which is sufficient considering that user interface elements are mostly stationary. It is important to point out that this “spotlight” technique results in a constant runtime, independent of the actual size of a display. Considering the display sizes common for interaction technologies

Figure 7. Sample of web buttons for automatic widget extraction.

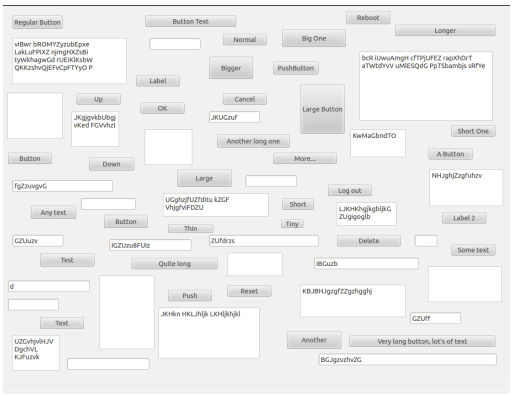


Figure 8. Variety of widgets to test model refinement.

like gestures and pointing, this property is essential for real world use.

4. Evaluation

For the evaluation of our system we conducted several experiments to to analyze the three main aspects: target extraction, target matching and target model generalization, individually. Those are described along with with the results in section 4.1. In addition, we applied our system to automatically place force fields on an application to enable pointing interaction with small targets on a large video wall. We conducted a user study in which participants were asked to try three variants of a pointing system. First the pointing system was used without any assisting technologies (NFF), second, force fields were placed based on the detected locations of the template matching system using manually created templates (MTFF) and third, force fields were placed based on templates learned from user clicks on the fly as described above (ATFF). We did not use a variant with manually annotated force field locations as the manually created templates detected every button of the interface and therefore annotated force fields would have been identical.

4.1. Template learning

The extraction of targets based on a single observed click worked for any of the standard widgets of the major operating systems (Windows 7, OSX Lion, Ubuntu 12.04 (all with their default theme)) without failure. Because especially widgets that are not common are interesting for the automatic extraction as those can not be supplied up front, we evaluated the extraction of a wide variety of buttons from the web (see Figure 7). To make sure that the position of a click did not influence the results we had four different users click all 209 buttons 10 times. In average, 198 of the 209 buttons were extracted successfully after a single click on each button. After the first click on a target an initial target model is created as described above. Matching this target or any identical target works without failure as they match pixel for pixel.

The model refinement is especially interesting for widgets that vary in size and appearance. To evaluate how well and how fast a target model can be refined to match all widgets of the same type, we created a test interface with 38 buttons, 15 text fields and 15 line edits of varying sizes and labels for each of the major operating systems. An example of such an interface can be seen in Figure 8. We then asked ten users to randomly click on the buttons, line edits and text fields and without the users knowledge, logged to how many buttons the target models generalized after each click. In average it took 3.4 clicks on buttons, 4.4 clicks on text fields and 3.9 clicks on line edits for the respective model to generalize to all elements.

4.2. Application to pointing interaction

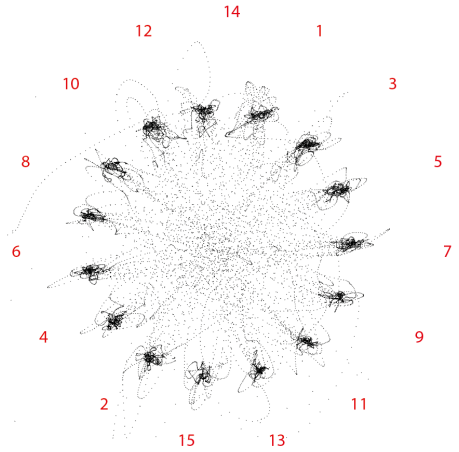
The experimental setup consisted of a pointing gesture recognition system that creates a 3D reconstruction of persons using multiple, overlapping video cameras and provides the intersecting point of the pointing direction of an extended arm with a display (see [11] for details). For the display we used a $4m \times 1.5m$ back projection video wall (resolution $4096px \times 1536px$). Using the pointing system, users are able to move the mouse cursor across the video wall by simply pointing towards the desired location. The position is updated with 30Hz and a click is triggered using a dwell timer described above with a trigger duration of $1sec$. The cursor icon was changed to three different icons to indicate the click progress (Figure 1). For an application, we used a simple layout (depicted in Figure 9) with buttons arranged in a circle to satisfy the ISO 9241 [1] requirements. The application had no interface to our system but accepting mouse input. The application was in no way aware of the pointing system or any assisting technologies. Ten (9 male) users aged between 22 and 32 participated. Eight of those were right handed and all used their primary arm for the entire experiment. Eight of the users had prior experience with pointing gesture interaction. All 10 participants had normal

Figure 9. User interacting with pointing gesture.

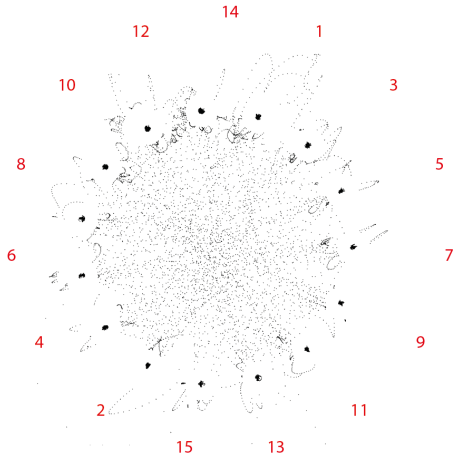
or corrected to normal sight. The average height of participants was $176.8cm$ while buttons were placed at heights from $108cm$ to $205cm$ from the ground.

For evaluation, participants had to perform a multi directional pointing task as suggested in part 9 of the ISO 9241 standard [1]. Circles of $1000px$ diameter and $600px$ diameter with 15 buttons each (sized $80px \times 30px$) were used to fulfill the ISO standard suggestion of varying the task parameters. The order of buttons following the ISO standard was indicated by numbered labels on the buttons. In addition, the current target was labeled in green while all other buttons were labeled in red (The order can be seen in the red annotations of Figures 10(a)-10(c)). Users were encouraged to experiment with the pointing system before the start of the experiment. Users were then consecutively presented with the two circle layouts for each of the three variants of the pointing system. The order of variants was randomly chosen for each participant. Because pointing gesture interaction in front of a large video wall can be tiring, users were free to take breaks in between circles. With a small mark on the floor for users to stand on (at $92cm$ from the video wall), we minimized any influence of the distance from the wall it might have had on the accuracy of the pointing system. Users were encouraged to select the targets as fast as possible but asked to balance speed and accuracy.
As in [2] we used a force field with the strength 0.8, as well as a force field size of $60px$.

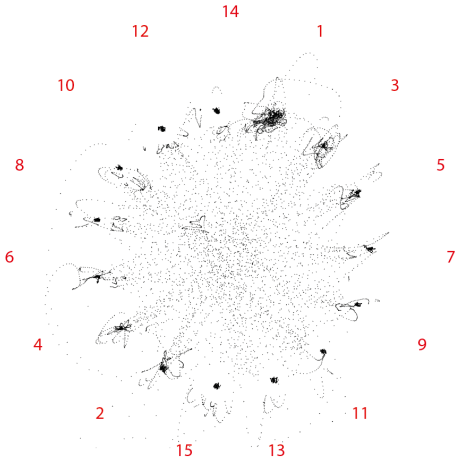
During the experiment all cursor movements as well as clicks, clicked buttons, the current target and the offset of a click to the button center were recorded. From this data we calculated the time it took to perform clicks as well as the accuracy with which buttons were clicked. To compare the time it took to click a button with each technique, the time between clicks on targets was measured. To be able to compare these durations despite the different layouts (small and large circle layout), we normalized the values to a distance of $500px$. In addition, users were given a short questionnaire to compare the participant's impressions to the measured results. The fastest (average time between



(a) No force fields



(b) Force fields from manually created templates



(c) Force fields from automatically created templates

Figure 10. Heat maps of cursor positions.

clicks) variant was MTFF ($3.5sec$) which shows a significant improvement over NFF ($4.66sec$). But not only force fields placed using manual templates improved the interaction time, the ATFF variant also showed a similar improvement ($3.66sec$). The same is true for the accuracy (offset between click and button center) which with buttons were hit (NFF:$17.12px$, MTFF:$5.25px$, ATFF:$8.59px$). These measured results are also reflected in the answers to the question which variant was perceived as the fastest: NFF:0 , MTFF: 4 , ATFF: 5 , MTFF+ATFF: 1. All participants could imagine using the pointing system with exiting desktop applications. However, only two persons could imagine this with variant NFF (NFF:2, MTFF: 9, ATFF: 8) (Multiple choices were allowed for all questions).

No force fields at non button locations were created due to erroneous template matches with either MTFF or ATFF during the experiment. From the cursor positions over the course of the experiment, we created heat maps to analyze the movement properties of the different variants in more detail. Comparing Figures 10(a) and 10(b) the effect of the force fields can be clearly seen. In Figure 10(b) the mouse positions are perfectly concentrated around the button centers. Figure 10(c) shows the heat map for the automatically acquired target models. Looking at the annotated button layout, the very different accuracies around buttons are not surprising. For the first button, there was no information yet where buttons were located and therefore no force field active. With every click it became more likely for the model to generalize to the other targets to locate all other buttons, which leads to a gradual increase in accuracy following the order in which buttons were clicked.

5. Conclusion

In this paper we presented an approach to extracting context information from any application without modification, which can be used to utilize assisting technologies like force fields. We presented an algorithm that automatically extracts target user interface elements by observing user behaviour, and creates target models without any prior knowledge about the targets and without any manual intervention. With a spotlight approach to the matching algorithm the runtime of the system remains constant even for large video walls which makes it suitable for the use with novel input technologies that often rely on assisting technologies because of their often lower accuracy. It allows the use of new input modalities with established real world applications. In an evaluation we have shown that the system works for a wide variety of widgets and integrated it into a pointing system to evaluate real world use. The results show that the automatically placed force fields improve interaction in both speed and accuracy. The proposed technologies enable the use of pointing gesture systems to control existing real world applications without modification of target applications and are able to bridge the gap between current advancements in interaction technology and the reality of established, mouse and keyboard centric applications.

References

[1] *ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs) – Part 9: Requirements for non-keyboard input devices*. 2000. 4, 5

[2] D. Ahlström, M. Hitz, and G. Leitner. An evaluation of sticky and force enhanced targets in multi target situations. In *4th Nordic conference on Human-*, number October, pages 14–18, 2006. 1, 5

[3] S. B. Andy Cockburn. Multimodal feedback for the acquisition of small targets Multimodal feedback for the acquisition of small targets, 2005. 1

[4] D. Burschka, G. Ye, J. J. Corso, and G. D. Hager. A practical approach for integrating vision-based methods into interactive 2d/3d applications. Technical report, The Johns Hopkins University, 2005. 1

[5] T.-H. Chang, T. Yeh, and R. C. Miller. Gui testing using computer vision. In *International conference on Human factors in computing systems*, CHI '10, pages 1535–1544, New York, NY, USA, 2010. 1

[6] M. Dixon and J. Fogarty. Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 1525–1534, New York, NY, USA, 2010. ACM. 1

[7] T. Grossman. The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area. *of the SIGCHI conference on Human*, (c):281–290, 2005. 1

[8] Y. Guiard and R. Blanch. Object pointing: a complement to bitmap pointing in GUIs. *Proceedings of Graphics*, pages 9–16, 2004. 1

[9] P. Kabbash and W. A. S. Buxton. The "prince" technique. In *Proceedings of the SIGCHI conference on Human factors in computing systems CHI 95*, pages 273–279, 1995. 1

[10] A. Memon, I. Banerjee, and A. Nagarajan. Gui ripping: Reverse engineering of graphical user interfaces for testing. In *Working Conference on Reverse Engineering*, WCRE '03, pages 260–, Washington, DC, USA, 2003. IEEE Computer Society. 1

[11] A. Schick, F. v. d. Camp, J. Ijsselmuiden, and R. Stiefelhagen. Extending touch: towards interaction with large-scale surfaces. In *ACM International Conference on Interactive Tabletops and Surfaces*, pages 117–124. ACM, 2009. 4

[12] C. H. Teh and R. T. Chin. On the detection of dominant points on digital curves. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(8):859–872, Aug. 1989. 2

[13] A. Worden, N. Walker, K. Bharat, and S. Hudson. Making computers easier for older adults to use: area cursors and sticky icons. ACM Conference on Human Factors in Computing Systems, pages 266–271. ACM New York, NY, USA, 1997. 1

[14] T. Yeh, T.-H. Chang, and R. C. Miller. Sikuli: using gui screenshots for search and automation. UIST '09, pages 183–192, New York, NY, USA, 2009. ACM. 1