HoverFlow: Expanding the Design Space of Around-Device Interaction

Sven Kratz
Deutsche Telekom Laboratories
TU Berlin, Germany
+49 30 8353 58289
sven.kratz@telekom.de

Michael Rohs
Deutsche Telekom Laboratories
TU Berlin, Germany
+49 30 8353 58469
michael.rohs@telekom.de

ABSTRACT

In this paper we explore the design space of around-device interaction (ADI). This approach seeks to expand the interaction possibilities of mobile and wearable devices beyond the confines of the physical device itself to include the space around it. This enables rich 3D input, comprising coarse movement-based gestures, as well as static position-based gestures. ADI can help to solve occlusion problems and scales down to very small devices. We present a novel around-device interaction interface that allows mobile devices to track coarse hand gestures performed above the device's screen. Our prototype uses infrared proximity sensors to track hand and finger positions in the device's proximity. We present an algorithm for detecting hand gestures and provide a rough overview of the design space of ADI-based interfaces.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*input devices and strategies, interaction styles*.

General Terms

Design, Human Factors.

Keywords

Around-device interaction, gestures, mobile devices, wearable devices, proximity sensors.

1. INTRODUCTION

Around-device interaction (ADI) is an emerging research topic in the field of mobile device interaction [4]. Using sensors, the interaction space of small mobile devices can be extended beyond the physical boundary of mobile devices to include the full 3D space around them. Around-device interaction has the potential to be a beneficial addition to standard interface elements of mobile devices, such as keypads or touch screens. This is particularly attractive for very small devices, such as wrist watches, wireless headsets, and future types of wearable devices such as digital jewelry (Figure 1). In these kinds of devices, it is extremely difficult or even impossible to operate small buttons and touch screens. The space beyond the device, however, can easily be used, no matter how small the device may be. Such wearable devices can also serve as easily accessible controllers for appliances in the environment or for wireless communication applications. In a smart home environment, for example, a gesture on the device could dim the light or control the volume of entertainment system.

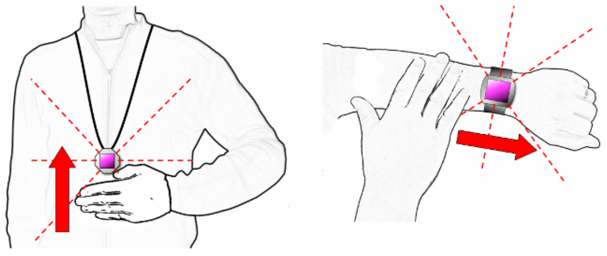

Figure 1. Interacting with very small devices via coarse gestures. The gestures are detected by an array of proximity sensors extending in radial direction from the device.

In mobile use scenarios, an incoming call could casually be forwarded to the voice mailbox or an incoming message could be acknowledged using different gestures. For mobile phones or PDAs - whether handheld, placed on a table, or placed in a cradle in the car - ADI could open up a range of 3D interaction possibilities. Coarse movement-based gestures could control PDA applications, such as turning pages in an electronic book. In a calendar application moving to the next day or month could be controlled by specific gestures, such as sweeping with the palm or with the edge of the hand, respectively (Figure 3). Such coarse gestures do not require the activation of a user interface widget and can be executed without visual feedback. This is especially beneficial for devices for which command selection via visual feedback is difficult, because the device is not in the line of sight, such as digital jewelry or wireless headsets. More fine-grained gestures could have a natural spatial mapping to 3D objects on the screen. Moving the hand closer to the device or rotating the hand could be mapped to zooming along the z-axis or rotating 3D objects. In order to mitigate occlusion, such gestures do not necessarily have to be performed on top of the device display. If infrared proximity sensors are used, they can be oriented in such a way, that the interaction does not occlude visibility of application objects on the screen.

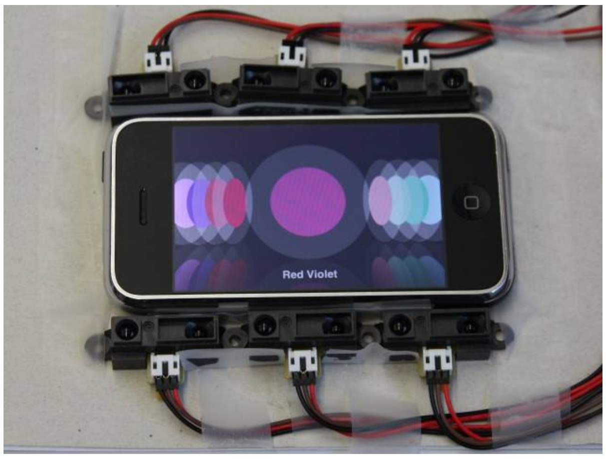

Figure 2. The current set-up of our prototype. Six Sharp GP2D120X IR distance sensors are placed along the long edges of an iPhone mobile device running the HoverFlow application (described in Section 3). Using simple hand gestures, the user can scroll and select colors in the color palette.

In light of this multitude of application possibilities, we think that it is worthwhile to explore the design space of around-device interaction in more detail. In this work, we present a first interface for coarse hand-gesture recognition above mobile devices. We describe the gesture recognition algorithm as well as our hardware prototype. We show a number of movement-based gestures that are suitable for recognition by proximity sensors. Furthermore, we will present a brief characterization of the design space of around-device interaction and identify promising research directions.

2. RELATED WORK

The *Gesture Pendant*, presented by Starner et. al. [21], was a neck-worn device that could recognize hand gestures performed by the user in front of its built-in camera. The advantage of using IR distance sensors, as in our setup, over the use of a camera for hand-gesture recognition is not only the low material cost of the distance sensors compared to a camera, but also ability of our setup to support continuous input using distance data provided by the IR distance sensors. Moreover, getting useful information from distance sensors is algorithmically much simpler than implementing computer vision techniques.

Hinckley et. al. adopted the idea of placing an infrared (IR) distance sensor on a mobile device and investigated technical characteristics of this kind of sensor technology [7]. The infrared distance sensor allowed the device to detect the presence of the user. This idea is used today in a number of digital single-lens reflex (DSLR) cameras that switch off the LCD display on the back when the user looks through the viewfinder.

SideSight [4] is an instance of an around-device interface by locating a series of IR sensors on the long edges of a small mobile device. This technique allows capturing simple multi-touch gestures around the device's perimeter. SideSight focuses on minimizing occlusion problems and is designed for operation while the device is placed on flat surfaces. In SideSight, the field of the distance sensors extends across the

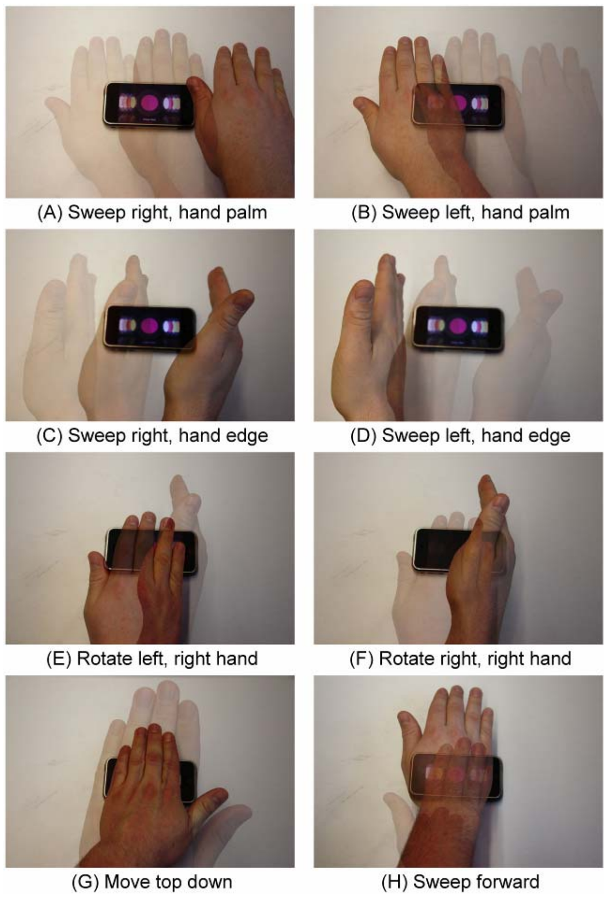
(A) Sweep right, hand palm
(B) Sweep left, hand palm
(C) Sweep right, hand edge
(D) Sweep left, hand edge
(E) Rotate left, right hand
(F) Rotate right, right hand
(G) Move top down
(H) Sweep forward

Figure 3. An overview of the hand gestures currently recognized by our prototype.

display surface to the left and right of the device. In our prototype the sensors are oriented towards the user to allow for handheld interaction (one hand is holding the device, the other hand performs the gesture). In more flexible setups, the distance sensors should be oriented in multiple directions to cover the whole space around the device.

Baudisch and Chu [2] focus on adding pointing input capabilities to very small devices. In order to avoid occlusion they use a touch screen on the back of the device and show that this approach is successful even for display sizes below 1". Since interaction with the nanoTouch device [2] means touching the back of the device, the possible physical extent of input movements is still given by the size of the device. In contrast, around-device interactions are independent of the physical size of the device and can be performed without visual feedback on devices without a touch screen.

FreeDigiter [13] is a proximity-sensing mobile phone headset that allows for the entry of digits via finger gestures. It uses a single infrared proximity sensor and is thus limited to very simple gestures involving counting 1-4 fingers crossing the field of the proximity sensor. The system uses an explicit clutching gesture, which is implemented with a dual-axis accelerometer. To begin gesture recognition the user has to briskly nod his head to the right. Another nod turns the recognition system off again.

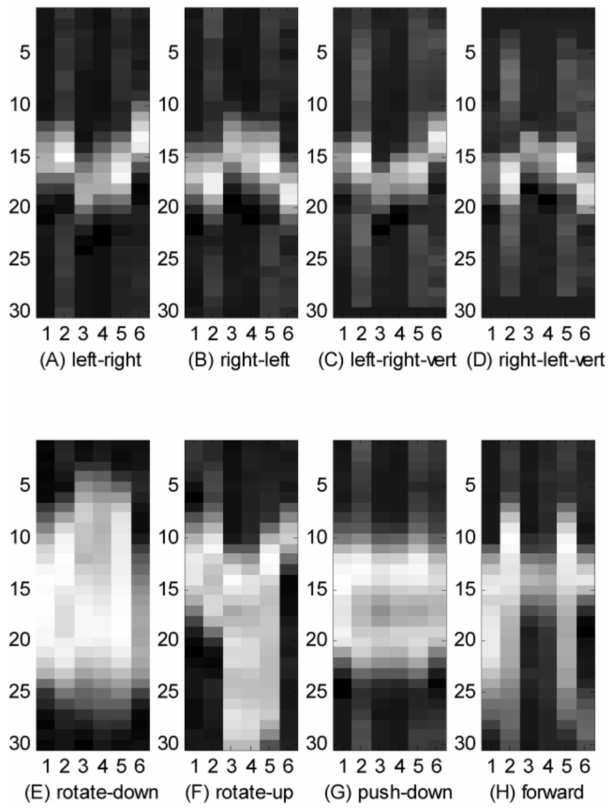

Figure 4. Image maps of the six IR distance sensor readings against time. The bright areas signify the proximity of an object. Notice the staggering of the peaks in the sample data, which is one of the distinctive features by which the gestures can be identified.

LightGlove [9] is a wrist-mounted input device that senses reflections from IR light beams when the fingers are bent. As an alternative to optical sensing, electric field sensing has been applied for a long time for touch detection and sensing hand position in space.

The *Theremin* [22], for example, was one of the earliest music instruments that used electronic field sensing of hand positions in space. Electronic music instruments have since also been implemented with IR proximity sensing [6]. Smith et al. [19] apply low frequency electric fields to implement a number of applications in stationary settings. The approach can be unreliable and susceptible to noise and it is not clear whether it is suitable for mobile use cases.

ThinSight [8,10] is a thin form-factor multitouch display that can detect fingers and objects touching an LCD screen. ThinSight uses a 2D grid of IR optosensors that are located directly behind a modified LCD panel. The system is primarily designed for (multi-)touch interaction and cannot detect hand gestures at a distance beyond 10 mm. However, the technology is suitable for detecting IR light sources even at larger distances from the surface and in principle supports bi-directional data transfer via IR as well.

3. HOVERFLOW

HoverFlow is an example application for the Apple iPhone that demonstrates the use of a sensor-based interface for detecting coarse hand-gestures above small mobile devices. The implementation of our application is partially based upon the *CoverFlow Example* by Sadun [14]. HoverFlow allows the user to select colors from a color palette through hand. Possible gestures are moving the hand across the device, presenting a number of hand postures, or by moving a hand rapidly towards or away from the device. Figure 3 shows an overview of all gestures currently implemented in our system.

3.1 Supported Gestures

The *CoverFlow View* provided by the iPhone's iPod application inspired the visual layout of HoverFlow. Thus, we decided to map the user's movements in the following way: if her hand moves across the device from left to right (Figure 3A), the color palette scrolls from left to right, and vice-versa (Figure 3B). A hand-edge movement from left to right (Figure 3C) makes the color palette scroll 5 colors to the right and vice-versa (Figure 3D). A color is selected when the user moves her hand swiftly towards the device (Figure 3G). A color is deselected when the user moves her hand rapidly away from the device. Rotating the hand towards the left (Figure 3E) or right (Figure 3F) permits the user to scroll directly to the beginning or end of the palette, respectively.

3.2 Interface Implementation

3.2.1 Sensing

To capture simple hand movements and gestures, our prototype system uses six Sharp GP2D120X IR [17] distance sensors, placed around the device's edges and facing vertically away from the device. Figure 2 shows the current sensor configuration of our prototype.

An Arduino BT [1] microcontroller board captures the distance readings provided by the sensors. The sensors supply 256 discrete range readings allowing them to detect distant objects from 4 to 30 cm away. The sensor update rate is 25Hz. A PC processes the sensor data, and handles the gesture recognition. In future versions of HoverFlow, we aim to conduct all processing on the mobile device, by establishing a direct link between the Arduino board and the mobile device via RS-232 or Bluetooth.

3.2.2 Gesture Detection and Recognition

To smooth the raw sensor data, it is passed through a Savitzky-Golay filter [16] in an initial processing step. The filtered data is then added to a queue containing the differences of the last 16 sensor readings, i.e. the difference $\Delta D = D_t - D_{t-1}$. We use the difference values instead of the absolute values in order to make gesture recognition independent of the distance between the user's hands and the device. The queue is updated every time the Arduino provides a new sensor reading. The window length of 16 was chosen because the sampling rate of the distance sensors is 25 Hz, which means that the system constantly keeps a history of the last 640 ms of interaction. This window length provides us with enough samples do discern user gestures in a meaningful way while at the same time assuring a response time from the system within a acceptable time interval (<1000 ms).

An advantage of the method we implemented is that it does not require any clutching mechanism to detect the start and end of a gesture, which is often required for accelerometer-based gesture recognition. When no IR-reflective object is present in the range of the distance sensors, they will provide a noise floor of values close to zero. Gestures can be distinguished from operation on the touch screen, by checking whether the screen was touched after the distance sensors detect an object in range. If a screen touch event occurs then this activity is interpreted as touch input and the gesture is discarded. Otherwise the activity is treated as a gesture. Accelerometers constantly provide sensor data as the user moves. It is therefore much harder to distinguish between moves that are part of a gesture and those that are not.

To determine if a significant user movement has been detected, the Euclidean norm of the oldest element of the readings queue is constantly calculated. If this norm surpasses a predefined threshold, the remaining 15 sensor readings are analyzed to determine the end of the sequence representing user input. Interaction with HoverFlow is designed to take place within a certain distance range around the device, so this threshold is set to the value the sensor array provides when a large object is held in front of them at a distance of about 5-7 cm away from it. Figure 4 shows image maps of several gestures supported by our system. In each graph, time progresses from top to bottom. The numbers on the y-axis show the sample index. 30 samples are shown, which corresponds to a time span of 1200 ms. The x-axis shows one column of data for each of the six sensors. As can be seen from this visualization, states of inactivity (low-amplitude noise) can easily be distinguished from a gesture entry by looking for a point in time from which on the amplitude of the signal rises significantly.

3.2.3 Gesture Classification

Once the bounds of the sequence containing user activity have been detected, a best-matching gesture template from a set of prerecorded user inputs is estimated using Dynamic Time Warping (DTW). A good overview of how DTW functions can be found in [12, 15]. Gestures and templates are represented as 16-by-6 matrices of sensor value deltas.

DTW performs well in cases where the captured sample and the matching template are distorted in time, but have similar values. In our case, using DTW allows the recognition of gestures that are similar in movement to but are performed at different speeds than the pre-recorded templates. In our prototype, we achieved acceptable recognition rates using only 2 to 3 training samples per gesture, with a gesture vocabulary of up to 9 gestures.

DTW-based approaches generally need less training samples than other methods, such as Hidden Markov Models [23]. Thus we do not require an extensive corpus of gestures to be available in order for our prototype to function correctly. A possible drawback of the DTW algorithm, its time and space complexity of $O(n^2)$, is not an issue due to the small size of the sampling window, which results in a maximum size of the distortion matrix of 256 elements. (Entry (i,j) of the distortion matrix contains the DTW-distance between samples 1 to i of the gesture and samples 1 to j of the template. Entry (16,16) thus contains our measure of similarity between gesture and template. The distortion matrix is built up from entry (1,1) using a dynamic programming approach.) Because of the small size of the distortion matrix, CPU and memory requirements should not present a constraint for our algorithm if it is run on modern mobile devices. If, however, sensors with a higher sampling rate were to be employed, which would result in larger data sets to be processed at a time, it may be likely that further optimizations of the DTW algorithm, such as FastDTW [15], will be required.

Gesture	A	B	C	D	E	F	G
A	0.775	0	0.225	0	0	0	0
B	0.025	0.925	0.025	0.025	0	0	0
C	0.1	0	0.9	0	0	0	0
D	0	0.175	0.025	0.825	0	0	0
E	0	0	0	0	0.875	0	0.125
F	0	0	0	0.025	0.025	0.95	0
G	0	0	0.025	0	0.025		0.95

Figure 5. This confusion matrix shows the actual gesture entries (row) and the predictions (columns), as the number of the predictions for each gesture class divided by the total number of entered gestures for that class. The average (correct) gesture recognition rate was 88.6%. (The indices A-G correspond to the indices in Figure 3.)

3.2.4 Update of Mobile User Interface

Once a gesture has been detected, the user interface of the mobile device running the HoverFlow application needs to be updated. In our prototype, the PC sends XML-RPC calls to the mobile device to signal interface updates when new gestures have been detected.

3.3 Evaluation of Gesture Recognition

As the initial gesture recognition performance of our prototype was acceptable, we decided to perform a small user study to further evaluate our interface. Firstly, we wanted to gain insight into the impact that our choice of gesture vocabulary makes on gesture recognition. We were especially interested in identifying gestures possessing similar features with respect to the gesture recognizer, i.e. leading to false positive recognitions. Secondly, we wanted to get a basic overview of our gesture recognizer's recognition rate.

3.3.1 Test Layout

We invited four experienced users to take part in our evaluation. Each user was given a brief description of our system and of the gestures it can recognize. In an initial training phase the users were asked to train the system with three samples of all the gestures shown in Figure 3 except the "sweep forward" gesture, which is a total of 7 gestures (A-G in Figure 3). After the training phase, the users were asked to enter each gesture of the training set ten times. For each gesture entry by the test participants, we recorded which gesture the interface recognized.

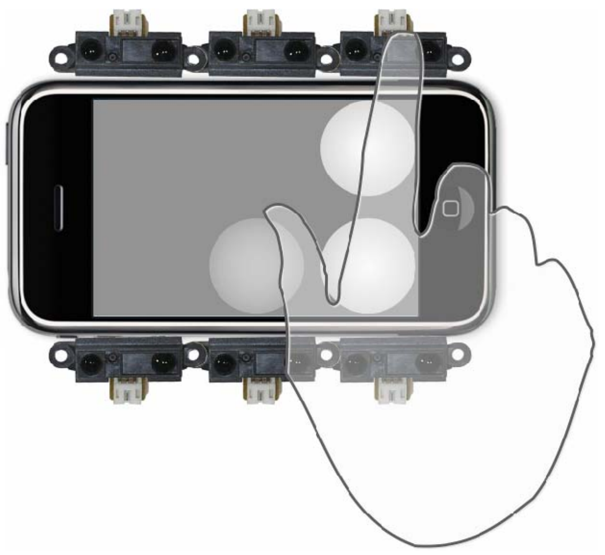

Figure 6. A possible method of visual feedback for interfaces using hand-gesture detection. The sensor values are mapped to semi-transparent globes that change their brightness according to the proximity values measured by the sensor. In this example, the sensors on the right measure full proximity (bright globes), whereas the user's thumb (bottom center) is not in full view of the sensor, leading to a somewhat reduced measurement value of the bottom center sensor (dim globe).

3.3.2 Test Results

In order to determine which gestures are prone to false recognition, we determined a confusion matrix using the data obtained from our study. The confusion matrix, as shown in Figure 5, reveals that gestures A, C ("sweep right flat hand", "sweep right hand edge") and B, D ("sweep left flat hand", "sweep left hand edge") are prone to be confused by the recognizer. The similarity of these gestures can be observed in the example plots in Figure 4 A-D. Gestures E and F ("rotate left / right"), however, are recognized by the system in a much more stable way.

A possible explanation for this behavior is the relative similarity of gestures A,C and B,D. When gestures A or B are quickly executed, they may "look" similar to gestures C and D to the system. This can be explained by considering the search strategy of the DTW algorithm, which aims to compensate feature differences in the time domain. In general, though, the average gesture recognition rate of 88.6% was fairly good considering only 3 samples were recorded to train the system for each user and, more importantly, the low number of sensors used by our prototype. If more densely spaced sensors are used, then gestures A and B would cover more sensors at a time than gestures C and D, which should be easily distinguishable by the recognizer. Of course, the study presented here is not representative of the general user population due to the limited number of participants and their high relative level of expertise. In spite of this, we gain some indication of the prototype's performance under controlled conditions. More significantly, the confusion matrix allows us to identify those gestures which are likely to be falsely recognized by the system, which is useful for the design of the gesture vocabularies of future systems employing a similar sensor configuration and using DTW for gesture recognition.

3.4 How to Improve HoverFlow

Several aspects of our design have the potential for improvement, which could lead to the stable detection of even finer hand gestures.

The current gesture recognizer could be improved to adjust for changing conditions by using an adaptation strategy, for example as described in [24].

As described in Section 3.1, our current prototype only allows the mapping of the user's hand gestures to discrete actions. Thus the gestures detected are always "iconic" in nature. Though this is a limitation of our current software implementation it is not a limitation of our interface, as continous action can be easily implemented using the distance readings provided by the IR sensors. We intend to explore continous, Theremin-like input in future revisions of or prototype.

The placement of the distance sensors is very important to correctly capture the movements of the user. If the sensors do not cover areas that the user typically interacts with, unreliable or even incorrect gestures may ensue. In order to increase the fidelity of the distance measurements, more sensors could be used, or, alternatively, distance sensors with wide-angle coverage, such as Sharp 2Y3A001 [18] (25° coverage) could be employed. Such wide-angle sensors have the advantage of increasing the detection area without the cost of adding additional hardware and wiring.

The IR distance sensors employed in our design are prone to noisy readings. Applying adaptive filtering techniques on the incoming data from the sensors could improve the quality of the distance measures, and lead to better recognition results. We aim to explore the effects of adaptive filtering on our recognition algorithm in the near future. Moreover, the sensors we employed have a relatively low update rate of 25 Hz. Future HoverFlow-like interfaces would benefit greatly from distance sensors with a higher refresh rate, as this would allow the interface to extract gesture information with a much finer temporal granularity.

It is very likely that Time-of-flight (TOF) depth-sensing cameras [20] will in the future be used as an alternative to IR distance sensors in Around-Device Interaction applications. The realatively high (depth) resolution provided by TOF cameras enables very fine-grained user gesture recognition [3]. At present, though, TOF cameras are still prohibitively expensive and too large to be incorporated into mobile devices.

The HoverFlow application itself could also be improved by adding visual feedback in order to help the user to coordinate his hand gestures more precisely. For instance, a certain region of the screen could be coupled to the sensor readings of a particular sensor. This technique, which is illustrated in Figure 6, would then allow the user to verify if his movement is being tracked around the particular region of the screen that is highlighted. In this case, occlusion isn't likely to be a major problem, as the highlighted screen areas are large enough to be seen even if partially occluded and the interaction takes place at some distance from the screen, thus the screen contents remain viewable (albeit at an angle) at all times.

4. AROUND-DEVICE INTERACTION

As demonstrated by the HoverFlow example application presented in this paper, Around-Device Interaction (ADI) shows promising potential as a complimentary interface to existing mobile device interfaces.

ADI allows for quick and coarse interaction with the devices, in cases where the desired actions are so simple that fine-grained interaction with the device's keypad or touch-screen is not necessary.

For example, simple hand gestures may present an alternative to clicking the *back*, *forward* or *reload* buttons in the device's web browser. Similar functionality could be implemented to control playback of songs or movies with the device's media player. The detection of coarse hand gestures can also be beneficial in cases where the user needs to deliver a very quick input to the device, such as muting the device or answering a call. A simple hand gesture here is presumably quicker than getting the device's screen into focus, locating the appropriate button and coordinating the button press.

The occlusion problem on small device displays is at least partially solved by ADI, as implemented in HoverFlow. Because the user interacts with the device at a certain distance from its screen, a gap opens up between the user's hand and the display, allowing the user to see the display's contents at an angle.

Although ADI breaks the metaphor of direct manipulation [4], quick hand gestures may be particularly useful for tasks of an immediate and direct nature. Also, situations where visual interaction is not preferable, for example when driving vehicles, may benefit from interfaces that allow the input of simple commands using rough hand gestures. However, a formal study of the efficiency of ADI interfaces for such and other tasks still needs to be conducted.

4.1 The Design Space of Around-Device Interaction

In the following, we shall characterize some of the elements of the design space of ADI-Based interfaces. We use the term *design space* in a very broad sense, including elements that we deem to be important to the fidelity, usability and development of ADI-based interfaces.

4.1.1 Sensors

IR distance sensors are a popular choice to sense user proximity in mobile interfaces. The advantage of such sensors over ultrasound range finders, for example, is that multiple IR distance sensors working in unison show much less interference than ultrasound sensors. On the other hand, the coverage area of IR sensors is usually narrower than that of their ultrasound counterparts.

Obviously, the fidelity of ADI increases with the number of sensors. Technological miniaturization may in the future allow for the development of very small sensors. Placed in significant numbers on the device, such miniature sensors would allow mobile device to gain a relatively high-resolution "image" of its surroundings. Covering the device in printed organic distance sensor circuitry has also been envisioned [4].

However, since energy consumption on mobile devices must be kept at a minimum, each increase of the amount of sensors will come with a cost. Not only do the sensors themselves consume energy, the higher the number of sensors that are mounted on the mobile device, the more CPU cycles will have to be devoted to processing the influx of data from the sensors.

Sensor placement is thus an important design decision. Sensors should be placed on locations on the device that allow them to optimally track the features (i.e. hands) of the user that are used for interaction with the device. As demonstrated by Butler et. al., one possible useful placement of IR distance sensors is on the edges of the device facing outwards. This allows the device to track the presence of the user's fingers when the device is placed on a flat surface. The current paper demonstrates a set-up using sensors facing upwards from the device allow it to track the motion of the user's hand using information from only six IR distance sensors. An even more significant advantage of HoverFlow-like interfaces is that they do not require the device to be placed on a flat surface in order to operate correctly. LucidTouch by Wigdor et. al. [21] demonstrated a technique enabling a mobile device to track the presence of the user's fingers to the rear of the device.

4.1.2 Mapping of Sensor Data to Interface Actions

Useful mappings of the sensor information to interface elements need to be discovered. In this work, we demonstrate the use of IR distance sensors to recognize simple hand gestures. In our application, the data was input into a gesture recognizer in order to recognize simple hand gestures. However, for other applications alternative mappings may be more advantageous. Currently our system can only effectively discern a single user action from sensor noise. An interesting but difficult improvement of HoverFlow would be the capability to identify a sequence of gestures performed in a single user action (for instance the gestures "rotate-right" followed by "rotate-left" and "sweep left" performed in close succession, appearing as a single gestureal *phrase* to the user [5]). A further example of a useful sensor mapping is the one used by Butler et. al. in SideSight. They use the sensor readings of their prototype to enable multi-touch like user inputs on the sides of a mobile device. They map their sensor readings to a one-dimensional bitmap, from which the finger position and estimated distance can be inferred. In LucidTouch, the user's fingers are located on the rear of the device. The camera image of the fingers is mapped to the device's screen in the form of finger shadows.

With a very high sensor density, the mobile device could even obtain a 3D representation of the part user's fingers or hands facing the sensor area. It may become possible to identify very detailed hand gestures, enabling interesting possibilities for 3D manipulation of on-screen objects or for gaming, for example.

4.1.3 Feedback

Since the direct manipulation metaphor is broken due to the interaction taking place away from the device, feedback plays an important role for the usability of the interface, as it will help users operate ADI interfaces more effectively.

Apart from the visual feedback mechanism proposed in Section 3.4, it may be feasible to use vibrotactile feedback (i.e. using the mobile device's built-in vibrator motor), if the device is held in one hand while the other hand performs the interaction.

In a similar way, auditory feedback could provide feedback on the status of the gesture recognition, i.e. playing back a notification when a gesture has been recognized.

4.1.4 Expanding Existing User Interface Frameworks

The majority of the interface frameworks of existing mobile devices do not yet support ADI. However, as can be seen from the recent addition of accelerometer support in UI frameworks in Apple's iPhone or the Nokia S60 series of mobile devices, the palette of supported sensors is being continuously expanded and may well support IR distance sensors in the near future.

The general advantage of integrating sensor support into mobile interface frameworks is that this allows the actual processing of the sensor data to be abstracted away, allowing developers to focus on the core benefits provided by the additional sensing. An example of such an abstraction can be found in the device orientation frameworks on iPhone or Nokia S60 mobile devices. The device orientation frameworks use the devices' built-in acceleration sensor to provide device orientation information (i.e. face-up, face-down, portrait, landscape) to the applications running on the device. The actual sensor data is abstracted by the orientation framework and thus not seen by the applications using it.

Beneficial abstractions need to be included into existing mobile UI frameworks to leverage the numerous capabilities offered by ADI. Developing such abstractions will require careful identification and evaluation of the most useful functionalities that are provided by ADI.

5. SUMMARY

We presented a user interface prototype that allows mobile devices to track coarse hand gestures using a small number of infrared distance sensors. We implemented an example application, HoverFlow, which allows the user to select colors from a "flowing" palette. Hoverflow conceptually demonstrates how interfaces that expand the interaction area of mobile devices beyond their physical boundaries can enhance mobile user interfaces. We classify this type of interaction as *Around-Device Interaction* (ADI). We describe the implementation of HoverFlow, a sensor-based ADI interface and present an initial study of the system's gesture recognition performance. The results of this study identify those types of gestures that may generally lead to false positive recognitions in interfaces using configurations similar to the one used in our prototype.

We provide a rough overview of the design space of ADI-based interfaces and highlight the areas that merit further research activity. High-quality sensors are the most important building block of ADI-based interfaces. The amount and the resolution of the sensors employed directly influence the expressiveness of ADI. The way the sensor data is mapped to the user interface highly influences the interface's "character" and also usefulness for everyday operation. Additionally, Feedback mechanisms need to be explored in order to keep the user "in the loop" and to make ADI-based interfaces easier to use and understand by unskilled users. For developers, toolkit integration will be an important factor for the incorporation ADI techniques in future mobile interfaces. Only if useful abstraction layers for the ADI sensor information are provided by popular mobile interface toolkits will we see ADI techniques implemented in applications outside of the research community.

6. REFERENCES

[1] Arduino: prototyping boards and development environment, http://www.arduino.cc .

[2] Baudisch, P. and Chu, G. 2009. Back-of-device interaction allows creating very small touch devices. In Proc. of CHI '09.

[3] Breuer, P. and Eckes, C. and Muller, S., Hand Gesture Recognition with a Novel IR Time-of-Flight Range Camera-A Pilot Study, Lecture Notes in Computer Science, Vol. 4418, pp. 247, Springer, 2007.

[4] Butler, A., Izadi, S., and Hodges, S. 2008. SideSight: Multi-"touch" interaction around small devices. In Proc. of UIST '08. ACM, 201-204.

[5] Buxton, W. A. 1995. Chunking and phrasing and the design of human-computer dialogues. In Human-Computer interaction: Toward the Year 2000, R. M. Baecker, J. Grudin, W. A. Buxton, and S. Greenberg, Eds. Morgan Kaufmann Publishers, San Francisco, CA, 494-499.

[6] Franco, I. 2004. The AirStick: A free-gesture controller using infrared sensing. In Proc. New Interfaces For Musical Expression (NIME), 248-249.

[7] Hinckley, K., Pierce, J., Sinclair, M., and Horvitz, E. 2000. Sensing techniques for mobile interaction. In Proc. of UIST '00. ACM, 91-100.

[8] Hodges, S., Izadi, S., Butler, A., Rrustemi, A., and Buxton, B. 2007. ThinSight: Versatile multi-touch sensing for thin form-factor displays. In Proc. of UIST '07. ACM, New York, NY, 259-268.

[9] Howard, B. and Howard, M.G.: Ubiquitous Computing Enabled by Optical Reflectance Controller. Whitepaper. Lightglove, Inc., http://lightglove.com/White%20Paper.htm . (Visited on 25.06.2009).

[10] Izadi, S., Hodges, S., Butler, A., Rrustemi, A., and Buxton, B. 2007. ThinSight: Integrated optical multi-touch sensing through thin form-factor displays. In Proc. of the 2007 Workshop on Emerging Displays Technologies: Images and Beyond: the Future of Displays and interacton. EDT '07, vol. 252. ACM, 6-9.

[11] Izadi,S., Butler, A., Hodges, S., West, D., Hall, M., Buxton, B., Molloy, M. Experiences with Building a Thin Form-factor Touch and Tangible Tabletop. Proc. of TABLETOP 2008, 193-196.

[12] Kruskall, J. & M. Liberman. The Symmetric Time Warping Problem: From Continuous to Discrete. In Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison, pp. 125-161, Addison-Wesley Publishing Co., Reading, Massachusetts, 1983.

[13] Metzger, C., Anderson, M., and Starner, T. 2004. FreeDigiter: A Contact-Free Device for Gesture Control. In Proc. of the Eighth international Symposium on Wearable Computers. ISWC. IEEE Computer Society, 18-21.

[14] Sadun, E., iPhone Developer's Cookbook, Pearson Education, November 28, 2008, ISBN 0321555457.

[15] Salvador, S. and Chan, P., FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space, KDD Workshop on Mining Temporal and Sequential Data, 70-80, 2004.

[16] Savitzky, A., Golay, M.J.E. 1964. Smoothing and Differentiation of Data by Simplified Least Squares Procedures. Analytical Chemistry, 36 (8), 1627-1639.

[17] Sharp GP2D120 Distance Sensor, http://www.sharpsma.com/Page.aspx/americas/en/part/GP2D120/

[18] Sharp 2Y2A001 wide-angle IR distance sensor, http://sharp-world.com/products/device/lineup/data/pdf/datasheet/gp2y3a001k_e.pdf

[19] Smith, J., White, T., Dodge, C., Paradiso, J., Gershenfeld, N., and Allport, D. 1998. Electric Field Sensing For Graphical Interfaces. IEEE Comput. Graph. Appl. 18, 3 (May. 1998), 54-60.

[20] SwissRanger 4000 Time-of-Flight Camera, Mesa Imaging AG, http://www.mesa-imaging.ch/dlm.php?fname=pdf/SR4000_Data_Sheet_rev1.5.pdf

[21] Starner, T. and Auxier, J. and Ashbrook, D. and Gandy, M, The gesture pendant: A self-illuminating, wearable, infrared computer vision system for home automation control and medical monitoring, International Symposium on Wearable Computing, 2000, pp. 87-94.

[22] Theremin, L.S. 1996. The Design of a Musical Instrument Based on Cathode Relays. Reprinted in Leonardo Music J., No. 6, 1996, 49-50.

[23] Wigdor, D., Forlines, C., Baudisch, P., Barnwell, J., and Shen, C. 2007. Lucid touch: A see-through mobile device. In Proc. of UIST '07. ACM, 269-278.

[24] Zhen Wang, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan, uWave: Accelerometer-based personalized gesture recognition, Technical Report, Rice University and Motorola Labs, June 2008.