Perceptive Presence

Frank Bentley, Konrad Tollmar, David Demirdjian, Kimberle Koile, and Trevor Darrell
Massachusetts Institute of Technology

Perceptive presence systems automatically convey user states to a remote location or application without user input. Our component-based architecture creates presence applications using perceptual user interface widgets.

Ever since computers were first networked together, people have used them for communication. From email—one of the first network applications—to the original MIT Zephyr notification system[1] to the now-popular AOL Instant Messenger System, people use computers to break down physical distances and provide a sense of awareness about another person in another place.

Beyond basic text interaction, research—such as the Portholes project at Xerox PARC—has strived to bring people together by giving them the ability to, for example, share snapshots or videos of themselves in remote places.[2] Although these systems use still shots or video, they still required the human to perform intelligent processing to determine whether people were present, busy, or available to talk.

As interaction with computers moves away from the traditional domain of a fixed keyboard and mouse moving a pointer on a screen, it becomes possible for the computer itself to determine a person's state using perceptive presence applications. This state could include basic information, such as location and head pose, but also more useful information such as whether the person is present or busy. These applications are perceptive because they provide an alternative to the traditional screen-based GUI, using physical presence and gestures to replace the traditional keyboard and mouse. They are also presence applications because they seek to share information about a person or location across some distance. For example, perceptive presence applications can share information about the number of people in an area as well as their location, activity level, or line of site (where they are looking). Applications involving communication, interaction with objects—such as light switches, telephones, or selections on a screen—in a room, or security can use this information.

Currently, many approaches in this domain offer tightly integrated systems that receive information from sensors and process it internally in the application. This approach requires developing new vision modules, integration languages, and architectures for each new application. If multiple applications need to run at once, computation will be redundant, as each application would have to process the raw data from the camera into its own representations. A general approach with a common interface and reusable processing units would greatly reduce computational redundancy, allow for easy integration, and simplify the use of vision interfaces to the level of today's GUIs.

We present a general XML-based interface for perceptive presence applications and a set of widgets that use this interface to infer activity information about a place. The interface connects any number of computer vision modules to perceptually aware applications. Figure 1 shows the implementation's architecture. Although we've interfaced our system only to computer vision widgets, there is no limitation to having other sensing technologies provide information to the widget layer through the interface. Examples could include microphones or touch sensors for locating activity areas.

Scenarios

Perceptive presence applications embody a shift away from traditional modes of interaction with a computer. These perceptual-user-interface-based applications[3] take input initiated by and natural gestures untethered by input devices , location clues, speech, and other modalities. Consider two coworkers, John and Mary, who communicate frequently throughout the course of a day. John's office is located on one side of a large office building and Mary's office is on the other side. Often Mary walks to John's office and finds that he is not present. Mary doesn't like to telephone John each time she wants to ask him something; she doesn't want to interrupt his train of thought or call in the middle of a busy meeting.

The Perceptive Presence Lamp (shown in Figure 2) is a perceptive presence application that could help in this scenario. The lamp connects two physically separated

 Published by the IEEE Computer Society

places by signaling information about the activity at one location to the other. This provides a sense of awareness and connection between the people at the two locations, letting them know when someone is free, or just providing reassurance that a person is where they are expected. Our hope is that this connection will stimulate communication or collaboration and bring people more in tune with the schedules of those around them.

Perceptive presence information can also help applications understand peoples' activities occurring in a space. To study this idea, we created a scenario that uses activity information in a context-aware environment. Consider, for example, John and Mary in a context-aware office: In the morning Mary arrives at work and enters her office. The room lights turn on automatically and the computer screen starts up when she sits down at her desk. While organizing her day and reading her emails, the room turns on the radio, tuning in the morning news. Later that day John walks by Mary's office. Seeing Mary working on her computer reminds him about a presentation that they need to give next week. John opens the door and greets her. Mary swivels her chair around and welcomes him. The volume of her radio goes down and after some small talk they decide to look over the last presentation that they gave on the topic. John sits down in the chair next to Mary's desk and the ambient light in the room increases. Mary asks the room to display the presentation information so that both she and John can see it, and the presentation slides appear on the wall display between them. They then start to work on their presentation. After a short time, the calendar system reminds Mary about a weekly staff meeting, and it informs her that she has one voice mail that was recorded during her meeting.

In each of the transitions in this scenario, information about Mary's activity as expressed through her position, pose, and/or motion are relevant context for environmental controls. We can thus define an activity widget sensitive to patterns observed about Mary's state, which can signal the activity most likely to occur so that widgets governing the environmental controls can react accordingly.

These scenarios have motivated us to create a platform for flexible perceptive presence widgets that we can reuse in a variety of context-aware application contexts.

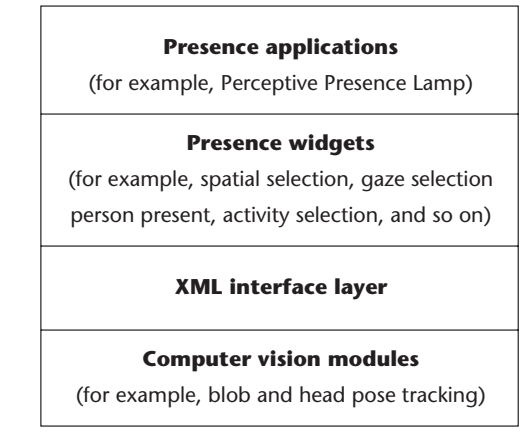
Presence and awareness in context-aware spaces

Researchers have written many papers on both presence and awareness and each provides a slightly different definition. Researchers have long used the terms *telepresence* and *remote presence* to describe a system that allowed a user to see and feel a remote location. The terms' use continued over the years to describe various computer simulations as well as television, radio, and other media. In 1997, Lombard and Ditton performed an extensive survey of presence.[4,5] Their results showed six main dimensions of presence:

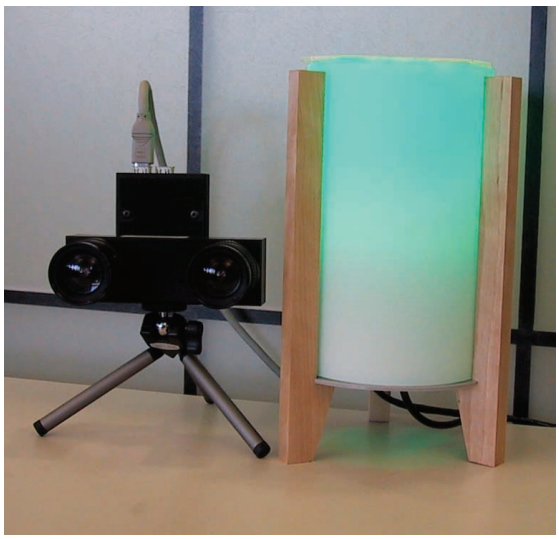
- *Social richness* describes the degree of social warmth or intimacy conducted by the medium. Does the medium invite interaction?
- *Realism* focuses on the perceived reality of the experience that the system provides. Is the user experience similar to a truly present experience?
- *Transportation* signifies that users can feel that they are in an environment and can suspend their disbelief to interact in this created space.
- *Immersion* relates to the feelings coming from a space. Does a person feel involved and engrossed in this space?
- *Social actor within medium* indicates the creation of inappropriate or illogical responses to a medium, such as talking to your television set.
- *Medium as social actor* refers to users thinking that the medium itself is in some way capable of interaction, or is in some way human like.

In our system, we try to create a sense of social richness and realism. We hope that users will feel as if they are interacting directly with the user on the other side of the system (realism), and that the users will interact as they would with an office mate by responding to signals about presence (social richness). Steinfield, Jang, and Pfaff split awareness into five distinct groups. Each group touches on a different aspect of awareness.[6]

Activity awareness is knowing what actions others are



1 Perceptive presence system architecture. The widgets compute perceptive information from data received through an interface layer from computer vision modules.



2 Perceptive Presence Lamp prototype. The lamp uses a stereo camera to determine the positions of people, displaying the result on another color-changing lamp in a remote location.

taking at any given moment. This type of awareness is especially important in computer-supported collaborative work (CSCW) applications where multiple people work together to produce a document or piece of code. Knowing what files others are viewing or editing would be examples of activity awareness.

Availability awareness is the availability of people and their social awareness of each other. Availability awareness is important in knowing whether people are available to talk. This can be important in supporting distance communication and collaboration where the distance makes it impossible to drop in on a colleague to talk, but one user would not wish to interrupt someone while he or she is in a meeting or otherwise occupied. Some systems provide information similar to what you would see walking by an open door, for example, AOL Instant Messenger, ICQ, and EuroPARC's Ravenscroft Audio Video Environment (RAVE).

Process awareness gives people a sense of where their pieces fit into the whole picture, what the next step is, and what needs to occur to move the process along. This form of awareness is useful in applications suited for large group use and managing large projects and was the key to most CSCW projects in the 1990s.

Perspective awareness gives group members information helpful for making sense of others' actions, such as background on team member beliefs and knowledge. Perspective awareness lets people gain information about others' experience. Individuals can provide information on their own work and background so that others can seek their advice or know which tasks to assign to them. This can work as an enhanced phone-book-type system or as a part of a larger system.

Environmental awareness focuses on events occurring outside the immediate workspace that might affect group activity. Environmental awareness is important for applications that provide external information such as news or economic conditions that could impact people's decisions.

In our work, we focus on the areas of activity and availability awareness, as we feel these are the most relevant in conveying information on presence. Knowing when people are in, or when they are busy (for example in a meeting) can convey their state for the purposes of communication.

The "Related Work" sidebar discusses some other recent approaches.

Perceptive presence architecture

A perceptive presence system requires a specific style of program, which differs radically from a GUI-based program in several fundamental ways. First, instead of dealing with discrete and rather precise input, a perceptive presence system needs to deal with input that

- comes from advanced, complex vision/sensory systems that might have many parameters to set,
- contains noisy data due to external factors, and
- could provide data that's far more detailed than what a specific application might need—that is, a 6-degrees-of-freedom (DOF) head tracker providing data for head nod detection.

Due to the complexity of such a system, it's often necessary to run processes in a distributed manner where the input device and the application run on different machines. A perceptive presence architecture should not be bound to a particular perceptive technique, but instead should abstract the required functionality into a set of reusable software components—that is, a widget set for building a perceptive presence system. This widget system should

- receive and filter data from perceptive technology systems,
- infer information from the perceptually grounded information into abstractions that are easier and more convenient to use than the raw perception data (that is, detecting when a person is present), and
- allow for many different and autonomous perceptive technologies trackers that each deal with a specific modality (that is, computer vision systems).

Widgets are the heart of the perceptive presence system. They turn observations passed through the computer vision interface into useful contextual concepts that an application can understand and work with. For example, a widget could indicate the presence of people, their locations, and their movement velocities in a given area. It could communicate that one person is looking at the camera or at an area 2 feet to the left of the camera. The widget could even show a person selecting a virtual object by moving into a given area. Further abstracting the data from the interface makes all these examples possible. To provide a platform-independent widget framework usable in a distributed environment across several devices, we embedded an XML-based communication layer in the widget set. This layer lets widgets request particular data such as the location and velocity of flesh-colored blobs (that is, people) and to receive XML messages with the desired information multiple times per second.

Widgets can feed data back into the computer vision system through the XML interface. For example, a widget could only request face identity information (assuming a face recognition system was available) if it knew that a person was present. Widgets can also pass informative messages back to the computer vision systems to help them run more efficiently. For example, a widget could pass a message back to a person tracker device when it has reason to assume that no one is present, allowing the vision module to reinitialize or update its world model.

We implemented these widgets as Java classes that extend a common abstract widget class. The abstract class provides for common functionality, such as connecting with the interface layer and parsing the received XML packets into a more usable data structure. The system also has a mechanism for widgets to inform other widgets or applications about each other. Widgets or applications can subscribe to other widgets and receive information each time their state changes. This state update occurs via an XML text message sent from the widget to all of its listeners. It's not necessary to write widgets in Java, we could just as easily write them in

Related Work

In the past five years, several groups have developed intelligent spaces, or environments that are context aware—that is, they know the activities occurring within them. In general, these intelligent spaces combine multiple sensor modes and are based on agent or widget oriented frameworks where the agents or widgets cooperate with each other to share state information about the environment and to act appropriately.[1,2] Systems like the EasyLiving project at Microsoft integrate input from computer vision, fingerprint readers, and other sensors to provide a large context-aware environment.[3]

The Aware Home project at the Georgia Institute of Technology[4] built a shared widget set that focuses on providing perceptual information. The widgets in our system are much in the spirit of these context widgets and provide many of the same functions, such as determining whether a person is present or moving. However, our widgets go further in providing a general interface and application-level GUI-like tools such as selection based on computer vision techniques. The Digital Family Portraits project[5] conveys awareness information through a digital picture frame that illustrates information about elderly relatives such as whether they ate, socialized, or slept in regular patterns over time. Icons around the actual photo of the monitored person conveyed this information.

Tollmar and Persson's work on the 6th Sense project is a complete application that uses light to signify remote presence.[6] The implemented and field tested version uses motion sensing to determine whether a person is around a lamp. Its structure is similar to our Perceptive Presence Lamp in that there are two lamps connected through a network. If there is motion in one area, the other lamp will glow. This resembles the first half of our initial prototype for the Perceptive Presence Lamp. Tollmar and Persson further field tested the lamp concept and found users received it better than expected.[6] The authors' attribute this to the warm feeling of the light and its nonintrusive nature.

The work on media spaces in the 1990s took the idea of instant messaging beyond text and into the realm of sharing pictures and video. In these systems, people could literally see others and what they were doing, and in most cases determine for themselves whether that person was available.

The Ravenscroft Audio Video Environment (RAVE) at Xerox EuroPARC is a prototypical example of a media space.[7] RAVE placed cameras and monitors at people's work places and allowed them five modes of interaction. First, a user could *sweep* all other locations, or a subset of locations. Sweep let a user peer in on others workplaces for 1-second intervals. The second function was *glance*, which gave users a 3-second view of a workplace, similar to the effect of walking past an open door and glancing in. The *office share* mode allowed for an always-on connection between two locations—as if the users were sharing an office. Finally, *vphone* enabled conversation between two nodes and usually followed a glance to see if a person was available. Again, in this system it was up to the user to determine whether a person was available based on looking at a live scene of a remote location.

A recent entry into this area is the IBM BlueSpace project.[8] It creates an office space that has a large light hanging above it. The user manually controls this status light via a touch screen. It turns red when users are busy and green when they are available. This application illustrates using color to convey presence information through lights. User studies showed that people employ the color information when deciding if someone is free or not to bother the person. The system also provided icons showing a user's state based on sensors and active badges (sitting in front of the computer, standing, or not present). Users preferred these automatic means of obtaining awareness information over time and chose not to manually change their lights as often during a four-week user study.

References

1. M. Coen et al., "Meeting the Computational Needs of Intelligent Environments: The Metaglue System," *Proc. 1st Int'l Conf. Workshop Managing Interactions in Smart Environments* (Manse 99), Springer Verlag, 1999, pp. 201-212.
2. A.K. Dey, *Providing Architectural Support for Building Context-Aware Applications*, doctoral dissertation, College of Computing, Georgia Inst. of Technology, 2000.
3. B. Brumitt et al., "EasyLiving: Technologies for Intelligent Environments," *Proc. Handheld and Ubiquitous Computing*, Springer Verlag, 2000, pp. 12-29.
4. A.K. Dey, D. Salber, and G.D. Abowd, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications," *Human–Computer Interaction (HCI) J.*, vol. 16, nos. 2-4, 2001, pp. 97-166.
5. E.D. Mynatt et al., "Digital Family Portraits: Supporting Peace of Mind for Extended Family Members," *Proc. Conf. Computer–Human Interaction* (CHI 01), ACM Press, 2001, pp. 333-340.
6. K. Tollmar and J. Persson, "Understanding Remote Presence," *Proc. NordiCHI* (Computer Human Interaction) 02, ACM Press, 2002, pp. 41-50.
7. W. Gaver et al., "Realizing a Video Environment: EuroPARC's Rave System," *Proc. Conf. Computer–Human Interaction* (CHI 92), ACM Press, 1992, pp. 27-35.
8. P. Chou et al., *Bluespace: Creating a Personalized and Context-Aware Workspace*, tech. report RC 22281, IBM Research, 2001.

C++ or any other language that supports sending or receiving user datagram protocol packets.

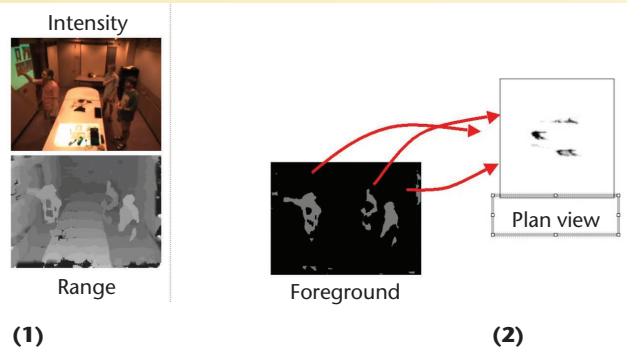
Because particular areas of a room often have different contextual meaning, our widgets operate on particular regions of space. We define an area as a 3D region with a particular, or at least a main, use. For example, the desk and couch areas are usually separate in space and use. Detecting human bodies in different regions—for example, sitting at your desk or on a couch—is a vision-based tracker's primary task. The amount of time that a person spends in a certain area is also a useful piece of information.

Our system can manually define spatial regions, or it can learn them by observation. We developed an activity widget that incorporates algorithms for learning an environment's activity map. Finding spatial clusters of

Vision Tracking

We have developed a set of tracking algorithms that use multiple cameras to observe people and estimate their position and pose in an environment. Our algorithms can provide such information as the number of persons in the space as well as a set of data (location, height, hand position, face pose, and so on) corresponding to each person.

While there have been many approaches to tracking people, including several successful monocular approaches, we focus on multicamera systems due to their speed and accuracy. A room tracking system with multiple cameras can perform dense, fast, range-based tracking with modest computational complexity. We can estimate foreground points by comparing estimated stereo images to a stereo model of the background of the scene (see Figure A). When tracking multiple people, we have found that rendering an orthographic vertical projection of detected foreground pixels is a useful representation. A plan view image facilitates correspondence in time since only a 2D search is required. Our approach merges plan-view images for each view when using multiple stereo cameras.



A (1) Intensity, disparity, foreground, and foreground projection images. (2) Spatio-temporal representation of projected foreground points.

Over time, we compute a 3D spatio-temporal plan-view volume. Each independently moving object (person) in the scene generates a continuous volume in the spatio-temporal plan-view volume. When the trajectories of moving objects don't overlap, the trajectory estimation is easily computed by connected-component analysis in the spatio-temporal volume (each component is then a trajectory). When the trajectories of moving objects overlap (for example, two people crossing), we build a graph from a piecewise connected-component analysis where nodes correspond to trajectory crossing and branches to nonambiguous trajectories between two crossings. A color histogram is then estimated for each branch of the graph (using all images associated with this branch). We estimate trajectories by finding in the graph the paths consisting of branches having the most similar color histograms. By tracking people in a space for a long period of time, we can gather a dense set of observed location features $f_i(x, y)$.

We define an activity zone as a connected region where observed location features $f_i(x, y)$ have similar values. An activity zone Z_k is defined by a connected region R_k in the 2D space defined by (x, y) and a characteristic feature $F_k = (h, v, v_{lt})$ representing the typical activity in this area. Since different activities can happen at the same location (x, y), activity zones might overlap as well. (We give further details on this algorithm and describe estimating activity maps in the main body of this article.[1])

We can use this algorithm to estimate people's location; we can also find the location and orientation of face and hand features. We exploit the fact that different people's faces and hands have a relatively similar appearance. In recent years, we've seen extensive research on human skin color and facial appearance detection. All human skin color has an invariant chromatic component that a system can easily learn and detect using a color classifier. A system can learn a discriminative model of frontal face appearance from sets of examples using machine learning techniques (neural networks, support vector machines, boosting, and so on). Such face detectors can find frontal-view faces in images in real time.[2]

We can construct a simple face/hands tracker using human skin color, face detectors, and stereo depth estimates. We find face and hands by first detecting in the image ellipsoidal blobs corresponding to human skin color. Then the detector performs an altering step, rejecting blobs of unusual sizes and shapes (corresponding to misdetected faces and hands). The face detector distinguishes faces from hands. Finally, an image motion analysis step lets us track faces and hands by matching the blobs to blobs detected in previous frames.

We can estimate face orientation approximating the observed rigid motion between subsequent frames, using the frame detected as frontal pose to initialize tracking. Using a scheme to reduce motion-tracking error by comparing new observations to multiple base frames, face pose tracking with stereo depth can be accurate despite large motions and illumination change.[3]

References

1. D. Demirdjian et al., "Activity Maps for Location-Aware Computing," *Proc. IEEE Workshop Applications of Computer Vision*, IEEE Press, 2002.
2. P. Viola and M. Jones, "Rapid Object Detection Using a Boosted Cascade of Simple Features," *Proc. Conf. Computer Vision and Pattern Recognition* (CVPR), IEEE Press, 2001, pp. 511-518.
3. L.-P. Morency et al., "Fast Stereo-Based Head Tracking for Interactive Environments," *Proc. 5th Int'l IEEE Conf. Automatic Face and Gesture Recognition*, IEEE CS Press, 2002, pp. 396-401; http://www.ai.mit.edu/projects/vip/papers/fg2002.pdf.

motion and position features—which tend to correspond to different activities performed in the environment—helps compute an activity map.

Using this framework makes it possible to create many types of widgets. We discuss three of them here. These widgets include a spatial selection widget that lets a user

select actions based on their physical 3D position, a head gaze selection widget that performs a similar task based on position and head angle, and an activity widget that automatically determines areas of certain activities and identifies when a person is in a particular activity region.

Spatial selection widget

The spatial selection widget is a general interface component that can help implement spatial selection tasks. Given a set of areas, this widget will inform its listeners when people enter or leave any of the areas of interest.

We use a depth and color tracking system to detect the presence of people. The "Vision Tracking" sidebar gives details about this system. The user provides a number of areas and the widget looks to see if faces or hands are present in each region, given the data received over the XML interface. When a person enters or leaves a given area, the widget sends an XML message to its listeners, indicating the current state of all areas of interest. Figure 3 shows an example of such a message for areas labeled *desk*, *door*, and *couch*.

We built a simple test application that used this widget to select between four squares arranged in a square on a computer screen. The four areas defined were the

- left half of the camera's field of view with a depth from 0 to 3 feet,
- left half of the camera's field of view with a depth from 3 feet to infinity,
- right half of the camera's field of view with a depth from 0 to 3 feet, and
- right half of the camera's field of view with a depth from 3 feet to infinity.

This application demonstrated that the effort for integrating a perceptive presence widget into an application is similar to that of adding a GUI component. The application also demonstrated the power of the widget set in a visual way.

To analyze this widget's effectiveness, we conducted experiments where seven people of varying age, skin color, and gender performed tasks in a work area. We split the camera range into two regions. Area A covered the left half of the field of view and was approximately 2 meters deep. Area B encompassed the rest of the field of view. We instructed participants to enter the work area, sit down in front of a computer (area A), read a news story of their choice from CNN.com, move away from the desk (into area B), return to the desk, and then leave. We collected data about average position, average velocity, number of flesh-colored blobs detected, and when the widget thought participants were in area A or B or not present at all.

Table 1 shows the average results from these experiments. Note that with the use of position and velocity information, the widget can declare quite quickly (mean 0.61 second, median 0.4 second) if someone has left.

We tested a version of this widget in two offices for the period of one day in each office. We used two offices that were quite different to get a sense of how this widget might work in varying environments. Office 1 was in a common space but with a desk used solely by the participant, although other people worked within the camera's range and people walked past the camera throughout the day. Office 2 was a two-person shared office where the nonparticipant was always within the camera's field of view. In both cases we asked participants to manually log the times that they considered themselves to be present and away.

```
<xml>
    <SpatialSelection>
        <desk>true</desk>
        <door>false</door>
        <couch>true</couch>
    </SpatialSelection>
</xml>
```

3 Example data sent to listeners of a spatial selection widget for a room where a person is at a desk and someone is on the couch.

The widget performed well in both situations by using areas that were limited in depth to only log a person sitting at the desk located closest to the camera when determining if that person was present. Table 2 shows the results of the two-day study. We determined accuracy by the percentage of the total seconds that the system was incorrect in determining if a person was present at their desk. We calculated the average accuracy as the average over all time for all participants.

Common errors included declaring someone had left for a minute or two while a person was on the phone or otherwise looking away from the camera (no flesh color in view). All these errors were rare, as shown by the total system accuracy of 91.7 percent. We later extended this widget to have a short delay before stating that a person had left, allowing for times when the person was just looking away or gone for a few seconds. This enhanced widget had an accuracy of 95.6 percent on the same data.

Gaze selection widget

The gaze selection widget is another example of an application-centric widget. It's similar to the spatial selection widget, except it uses head pose to select where a person is looking. This widget suits applications that require navigating via head gestures such as looking at objects to activate them (for example, a light switch or a heating control) or in certain directions to prompt computer interfaces.

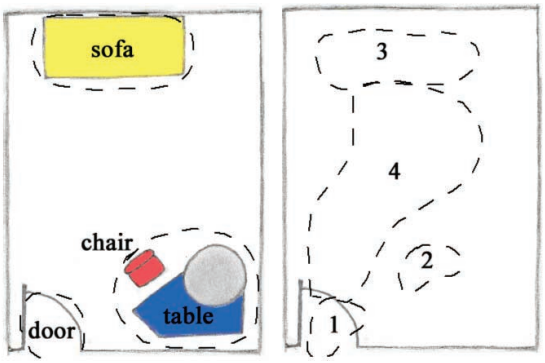
The gaze selection widget uses a 6 DOF head-pose tracker.[7] We implemented our tracker using a rigid stereo-motion algorithm, which can estimate the trans-

Table 1. Average results of the spatial selection widget experiment (more than 35 trials for seven users of varying age, skin color, and gender).

Task	Mean Time (seconds)	Median Time (seconds)	Standard Error (seconds)
Entering	1.43	0.8	0.40
Leaving	0.61	0.4	0.15

Table 2. Results from a study of the spatial selection widget.

Office	Accuracy (%)	Enhanced Widget Accuracy (%)
1	92.5	95.4
2	90.6	95.9
Average	91.7	95.6



4 Conceptual example of an activity map. Regions numbered 1 to 4 represent activity zones.

lation and rotation of objects moving in space (including heads, which are rigid except for expression change.) We developed an algorithm that can build a model of object appearance while it tracks, so that long-term accurate tracking is possible without error accumulation (see the "Vision Tracking" sidebar for details).

A gaze selection widget listens to data about head pose through the vision interface. Just as with the spatial selection widget, the user provides the areas. Every time the widget receives new blob information, it calculates the vector from the blob in the direction of the head pose. The widget then determines if this vector enters any of the areas of interest and returns an XML message to all of its listeners.

Activity widget

Simply considering the instantaneous 3D location of users is useful, but sometimes alone is insufficient as context information. Applications must generalize context information from previous experience, and therefore, an application writer might like to access categorical context information, such as what activity a user is performing. While location cues alone can't fully determine which objects or tasks are used during a particular activity, we found that activities are often correlated with location cues. By defining an activity map—which divides a physical space into regions corresponding to activities—we can infer activity behavior by looking for patterns in these locations. While the previous widgets have partitioned space based on simple proximity or relied on user-specified maps for regions, the activity widget learns location regions from observed activity, including motion and shape cues as well as position.

Figure 4 shows a conceptual example of an activity map for an office floor plan. We would ideally find the zones shown on the right: zone 1 is a region in which someone stands. Activity zone 4 shows a person moving—for example, a user walking between regions of a room. Finally, activity zones 2 and 3 illustrate a person sitting in a relatively still position—for example, reading or writing at a desk. Since a person can perform different activities that might require different movement at the same location, zones might overlap.

Based on 3D person tracking techniques, our algorithm generates an activity map by clustering spatio-temporal data gathered using a 3D person tracker.[8] Later we use the activity map to determine the user's location context. The widget packages the identified activity as an event for the application (that is, *working* can represent the state when a single person is sitting by a desk and typing on a computer, or *meeting* can describe the state when two people are in one specific area of the room and talk for more than five minutes). Many applications can take advantage of this activity information including the Perceptive Presence Lamp or, for example, a simple desk lamp that would automatically turn on when a person is working at his or her desk.

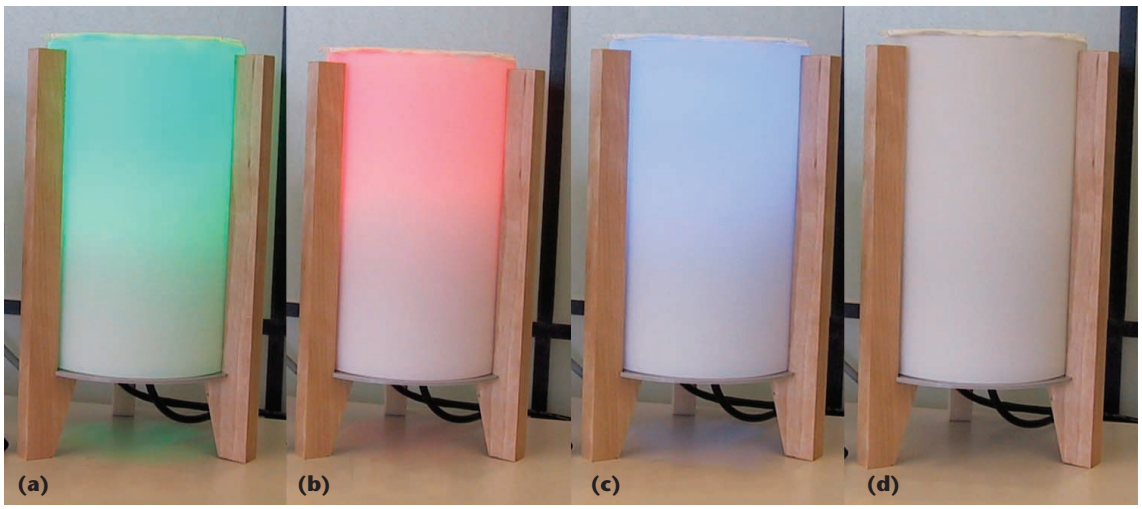
Example applications

We implemented and evaluated two example applications. These applications use the widget architecture to easily monitor perceptual data.

Perceptive Presence Lamp

The Perceptive Presence Lamp is a set of two lamps and cameras that virtually connect two places by conveying awareness information about each location.

This lamp, as shown in Figure 5, changes color depending on the state of the person or people at the remote location. If the lamp is off, no one is present. If



5 States of the Perceptive Presence Lamp: (a) present, (b) busy, (c) interacting, and (d) away.

Table 3. States of the widget-based Perceptive Presence Lamp and XML messages for updating the state.

State	Color Change or Lamp Action	XML Message
Person present at his or her desk	Green	<xml><update>present</update></xml>
Multiple people present; the person is busy	Red	<xml><update>busy</update></xml>
Person interacting with his lamp, trying to get the user's attention	Blue	<xml><update>interact</update></xml>
No one present	Off	<xml><update>away</update></xml>

it's green, someone is present and working at his or her desk. When guests are present, the lamp turns red to signify that the user is busy. Finally, when the user moves directly in front of his or her lamp, the other lamp turns blue signifying that the user wishes to get the remote user's attention. Table 3 summarizes these states.

The color changing in our system is similar to the status lights in the BlueSpace/myTeam system,[9] which turn green when a user sets his or her state to *available* and red when the state is set to *busy*. However, this system automatically computes all awareness information, and it requires no user interaction.

The Perceptive Presence Lamp is a software application using perceptive presence widgets, so the user could override the automatic sensing from the widgets. The user could set the lamp application with a manual override switch to explicitly set his or her state to busy, or to turn off the lamp's sensing for privacy. Ultimately, the application itself determines how to act on the inputs received from the presence widgets.

Implementation. We implemented the Perceptive Presence Lamp using the spatial selection widget. This widget defines three areas:

- The desk region consists of the space directly in front of a person's desk where the user would sit while working.
- The visitor area consists of all the other space in the office that the camera can view.
- The interacting area consists of the space directly in front of the camera/lamp and is where the user goes to interact with the lamp to get the remote user's attention.

The spatial selection widget determines whether a person (or multiple people) is present in a given area over time and passes this information to the application each time a person enters or leaves an area.

The lamp's vision processing software runs on a PC hidden under the desk and sends XML messages to the remote lamp's computer every time the state changes. Table 3 shows the messages sent. The application assumes that a given state exists until it receives a new state.

When a lamp application receives a message, it updates the color of its lamp accordingly. We use Color Kinetics color-changing light-emitting diode lights and place them into the body of a lamp bought from IKEA and control it through a Color Kinetics Smart Jack USB interface. A color square on a liquid crystal display monitor shows feedback on the local user's state, but we could easily integrate this into the lamp in the form of an LED. The feedback lets the user see what the other person is seeing to help relieve some privacy concerns.

Table 4. Results from a Perceptive Presence Lamp study.

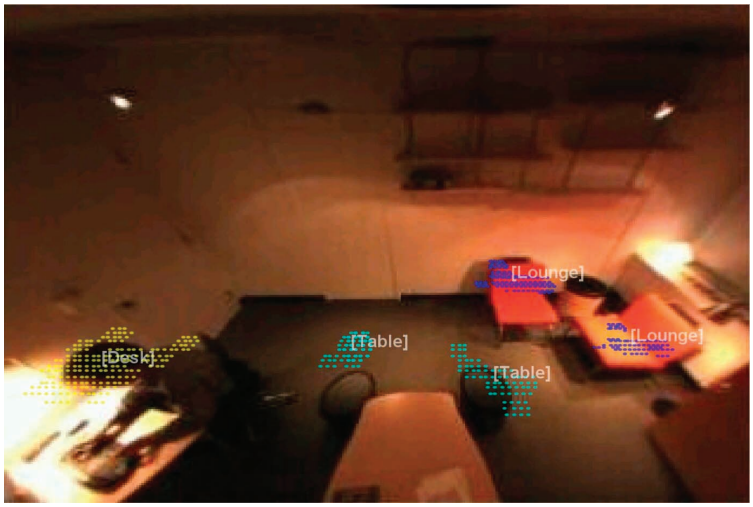
Office	Lamp Accuracy (%)	Usage Length (hours)
1	95.4	7
2	95.9	7
3	97.8	16
4	99.0	16

Evaluation. Since the Perceptive Presence Lamp uses the spatial selection widget, its technical performance is identical to this widget's performance, including the time it takes to identify that a person is present or has left.

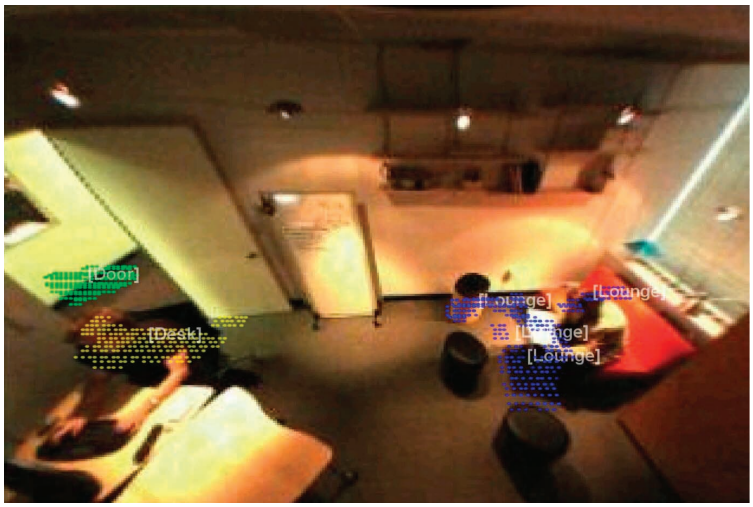
We performed a series of user studies involving six people over one or two days each. We collected more than 46 hours of data from varying office conditions including shared work areas, shared offices, and areas where only the user was visible in the camera's field of view. For each space, we manually defined a set of areas to explicitly mark off the work, visitor, and interaction areas in three dimensions. Table 4 summarizes the results and shows an average accuracy of 97.6 percent. Accuracy is the percentage of time the state of the lamp agreed with the state of the user.

Errors tended to be more false negatives than false positives. Some common errors included people considering themselves present when they were outside of their immediate work area (sitting on a couch or getting coffee), and the lamp stating that a person had left after the person had his or her back to the camera for more than a minute (no flesh-colored regions showing). We can improve on the first error by having more complex areas or by learning areas through a system like the activity widget and associating certain areas as present, away, busy, and interacting. We can improve the second error by having multiple cameras or a model other than flesh color for identifying people.

Overall, the users liked the concept of the lamp and used it to judge the availability of the other person. They frequently used the lamp to determine when a person returned from a meeting or lunch and saved wasted trips down the hall to check if he or she had returned. Most importantly, it served as a successful test of an application using the spatial selection widget.



6 First experimental office configuration, with a desk, meeting table, and reading area (here the door is closed). The workspace is shown from the camera's perspective, overlaid with clustered tracker data for the three zones; labels are user supplied.



7 Second experimental office configuration. The two tables are joined to form a large desk and the door is open. The workspace is shown from the camera's perspective, overlaid with clustered tracker data for three zones; labels are user supplied.

Generalization. We can generalize the concept of the Perceptive Presence Lamp to other devices.

Our perceptive presence display uses a projector to shine images onto a wall, table, or divider. The user can configure different images to correspond to different states, or even different users that might use the lamp. A small bar on the bottom of the image displays the current local state as viewed at the remote location(s).

Another version of the lamp, the Perceptive Presence Beacon, uses the same color-changing light used in the lamp, but instead shines it onto a wall, corner, or workspace. This might create more of an effect of being involved with the person on the other side of the connection as the light will cover a larger area and is not confined to a physical device.

Finally, this concept can be applied to traditional, screen-based awareness applications to create an AOL Instant-Messenger-style application that automatically updates when people enter, leave, or are busy.

Context-aware environments

In addition to experimenting with the automatic generation of activity zones, we have begun testing our system in an intelligent environment. We created a simple scenario (mentioned in the "Scenarios" section) that illustrates a context-aware environment. We used this scenario to implement a prototype application. Previous work on activity zones, in particular private and public zones, inform these scenarios.[10]

Our prototype application focuses on three tasks: control of light and audio, and display of information in an office.

Implementation. We created an application for these tasks using our activity widget to generate an activity map for an office. The application attached preferred lighting and audio settings to particular activity zones. It then used the widget to gather context information for people working in the offices and to receive notification of when particular activities were taking place.

We can compute an estimation of an activity as follows. The person tracker provides a history of 3D information for each person in the observed space (see the "Vision Tracking" sidebar). The activity widget receives the information about each person's location, velocity, and height from the person tracker through the XML interface. Tracking people in a space for a long period of time helps gather a dense set of observed location features. We then create activity zones representing regions of a physical space in which observed activity features—location, motion (represented as velocity), and shape (represented as height)—have similar values. Ideally, each zone corresponds to a region in which a person is likely to engage in similar activities. A relatively still person sitting at a particular location, for example, might be reading, writing, or typing. Finally, we match the activity map and the real-time data from the person tracker to estimate a person's activity, and the widget provides location context for applications in a pervasive computing environment—in our case, for the control of lamps, audio, and displays in a room.

Evaluation. We conducted two experiments in which our system automatically generated activity maps for different environments.

In a one-person office we created two furniture configurations. We equipped the office with a single stereo camera mounted on the wall in a standard surveillance camera configuration. For each experiment, we recorded tracking data over a long period of time and estimated activity maps offline using the approach described in the "Activity widget" section (due to the high number of data points, the segmentation algorithm took several minutes to run). In all of the experiments, we set the initial number of classes, N, for the clustering step to 15, and at the end of clustering the observed heights and velocities, we removed regions corresponding to small numbers of location features.

Figures 6 and 7 show results of the two setups. In each experiment, the automatically generated activity maps segment the space into zones related to structures in the environment (chairs, desk, file cabinet, corridors, and so on).

After estimating activity in the first configuration, we found activity areas. The meeting table zone is the union of smaller zones that the tracker identified around the stools. The tracker found a total of 11 zones, which represent four functionally separate areas. The four activity areas include the zones around the desk, meeting table, and lounge chairs (shown in Figure 6), plus an access corridor through the middle of the room.

Figure 7 also contains four major activity zones. Zone 1 corresponds to the access and walking context, zone 2 corresponds to the working context (desk), and zones 3 and 4 correspond to the reading and informal meeting context. As in Figure 6, Figure 7 doesn't show the access zone. These figures also don't show the location features (velocity and height) corresponding to the different zones. However, we observed that they correspond to their expected values—that is, regular standing heights in door and access zones, low heights in desk and lounge zones. Velocities were large in the access zone, medium in the door zone, and small in desk and lounge zones.

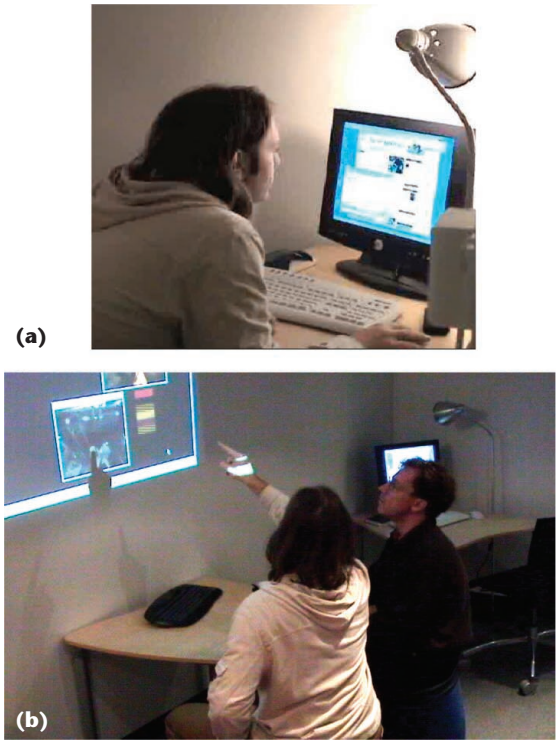
Figure 8 shows two scenes of our prototype system in use. Figure 8a illustrates the light automatically turning on when someone is working in zone 2. Figure 8b shows the automatic choice of display (computer screen or projector) at the meeting table.

We informally observed people working in the space. We noted that the activity widget correlated well with particular activities such as typing at the keyboard in the desk zone. Most people thought the automatic control of lights, radio, and projector was novel and useful. In particular, within a work environment filled with devices and gadgets, the changes in environment state (light and music) and information display state proved useful even in our preliminary user studies.

However, at this stage, we perceive a need to edit the maps—for example by clustering several zones into one or by labeling zones to provide the user with semantic meaning. For this reason we added a graphical tool that let a user or developer easily configure the parameters through a visual user interface, similar to the way you would set up areas for the spatial selection widget. A lesson here is that even though widgets provide advanced and complicated abstractions, they might require support for appropriate configuration and use.

Future directions

While our widgets and examples in this article exploited cues sensed by cameras and computer vision algorithms, we could extend our framework to other types of sensing modalities including audio, ultrasound, or radar. Widgets could combine inputs from this broad array of sources to declare more properties about a given environment or interaction. Just as the development of GUI abstractions made raster display devices ubiquitous interaction devices, abstractions for perceptual presence and context will play a key role in making perceptual interfaces usable by everyday applications. ■



8 (a) A user sitting down at the desk automatically turns on the light and computer screen. (b) Information displayed on the wall at the meeting table.

References

1. C.A. DeUafera et al., "The Zephyr Notification Service," *Proc. USENIX*, USENIX Assoc., 1988, pp. 213-220.
2. P. Dourish and S. Bly, "Portholes: Supporting Awareness in a Distributed Work Group," *Conf. Proc. Human Factors in Computing Systems*, ACM Press, 1992, pp. 541-547.
3. M. Turk and G. Robertson, "Perceptual User Interfaces," *Comm. ACM*, vol. 43, no. 3, 2000, pp. 32-34.
4. M. Lombard and T.B. Ditton, "At the Heart of it All: The Concept of Presence," *J. Computer Mediated Comm.*, vol. 2, no. 3, Sept. 1997.
5. M. Lombard et al., "Measuring Presence: A Literature-Based Approach to the Development of a Standardized Paper-and-Pencil Instrument," *Presence 2000: Proc. 3rd Int'l Workshop on Presence*, 2000; http://astro.temple.edu/~lombard/p2000.htm.
6. C. Steinfield, C.-Y. Jang, and B. Pfaff, "Supporting Virtual Team Collaboration: The Teamscope System," *Proc. Int'l ACM Siggroup Conf. Supporting Group Work* (Group 99), ACM Press, 1999, pp. 81-90.
7. T. Darrell et al., "Face-Responsive Interfaces: From Direct Manipulation to Perceptive Presence," *Proc. Ubiquitous Computing* (UbiComp 2002), G. Borriello and L.E. Holmquist, eds., Springer-Verlag, 2002, pp. 135-151.
8. D. Demirdjian et al., "Activity Maps for Location-Aware Computing," *Proc. IEEE Workshop Applications of Computer Vision*, IEEE Press, 2002, p. 79.
9. P. Chou et al., *Bluespace: Creating a Personalized and Context-Aware Workspace*, tech. report RC 22281, IBM Research, 2001.

10. K. Tollmar and S. Junestrand, "Video Mediated Communication for Domestic Environments—Architectural and Technological Design," *Proc. 2nd Int'l Workshop Cooperative Buildings—Integrating Information, Organizations, and Architecture* (CoBuild 99), LNCS 1670, N. Streitz, et al., eds., Springer-Verlag, 1999, pp. 176-189.

Frank Bentley *is a senior software engineer at Motorola Labs in Schaumburg, Illinois. His research interests include presence, software architecture, multimedia, and user-centered design. Bentley has SB and master's of engineering degrees in electrical engineering and computer science from the Massachusetts Institute of Technology.*

Konrad Tollmar *is a postdoctoral lecturer in the Vision Interface Group at the MIT Artificial Intelligence Laboratory. His research interests include human–computer interaction, computer vision, and interactive graphics. Tollmar has a PhD from Stockholm University.*

David Demirdjian *is a researcher at the MIT Artificial Intelligence Laboratory where he has pursued research on the analysis of human motion and multimodal interfaces. His research interests include articulated body tracking, activity segmentation, gesture recognition, and multimodal dialogue systems. Demirdjian has an engineering degree from Ecole Nationale Supérieure de Techniques Avancées, Paris, and a PhD in computer science from the Institut National Polytechnique de Grenoble.*

Kimberle Koile *is a postdoctoral lecturer in electrical engineering and computer science at MIT, where she also teaches AI and researches in the Agent-based Intelligent Reactive Environments (AIRE) research group. Her research interests include ubiquitous computing, knowledge-based systems, human–computer interaction, and computer-aided design. Koile has an SM and PhD in computer science from MIT.*

Trevor Darrell *is an associate professor of electrical engineering and computer science at MIT. He leads the Vision Interface Group at the Artificial Intelligence Laboratory. His research interests include computer vision, interactive graphics, and machine learning. Darrell has a BSE from the GRASP Robotics Laboratory at the University of Pennsylvania, and an SM and PhD from the MIT Media Lab.*

Readers may contact Trevor Darrell at the MIT AI Laboratory, Room NE43-V608, 200 Technology Square, Cambridge, MA 02139; trevor@ai.mit.edu.

For further information on this or any other computing topic, please visit our Digital Library at http://computer.org/publications/dlib.