IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 8 July 2013; accepted 22 August 2013. Date of publication 9 September 2013; date of current version 4 February 2015.

Digital Object Identifier 10.1109/TETC.2013.2281204

Reducing Energy Waste for Computers by Human-in-the-Loop Control

SIRAJUM MUNIR[1], JOHN A. STANKOVIC (FELLOW, IEEE)[1], CHIEH-JAN MIKE LIANG[2], AND SHAN LIN[3]

[1]Department of Computer Science, University of Virginia, Charlottesville, VA 22903 USA
[2]Microsoft Research Asia, Beijing 100080, China
[3]Temple University, Philadelphia, PA 19122 USA

CORRESPONDING AUTHOR: S. MUNIR (munir@cs.virginia.edu)

This work has been funded, in part, by NSF grant CNS-1239483 and EFRI-1038271.

ABSTRACT Although current cyber physical systems (CPSs) act as the bridge between humans and environment, their implementation mostly assumes humans as an external component to the control loops. We use a case study of energy waste on computer workstations to motivate the incorporation of humans into the control loops. The benefits include better response accuracy and timeliness of the CPS systems. However, incorporating humans into tight control loops remains a challenge as it requires understanding complex human behavior. In our case study, we collect empirical data to understand human behavior regarding distractions in computer usage and develop a human-in-the-loop control that can put workstations into sleep by early detection of distraction. Our control loop implements strategies such as an adaptive timeout interval, multilevel sensing, and addressing background processing. Evaluation on multiple subjects show an accuracy of 97.28% in detecting distractions, which cuts the energy waste of computers by 80.19%.

INDEX TERMS Distraction, energy waste, energy saving, human-in-the-loop, computer, workstation.

I. INTRODUCTION

Cyber Physical Systems (CPS) feature a tight integration of computing resources and physical elements. These systems have played a significant role in helping humans understand and control the environment. To do so, many CPS systems employ humans as an external component, in addition to the control loops. At a high-level, humans loosely couple with the control loops. In some cases, humans have the ability to take over the control loops when necessary or desired. For example, automatic piloting of an aircraft is subject to the pilot's discretion of when to initiate manual control. Another example is a cruise control loop for automobiles that simply maintains constant speed, without taking the driver's behavioral state into consideration.

Moving forward, we believe that CPS systems will have a stronger tie between humans and control loops, or the notion of *human-in-the-loop control*. Moving humans from outside to inside the loop, CPS systems can provide better response accuracy and timeliness. Continuing the example of automobiles, we note that road safety is not just keeping a sufficient distance between two cars, but also taking into account the driver's physiological state (fatigue, anger, drunk, etc.) and behaviors (distraction, erratic steering, etc.). When the driver is unfit to keep the safety or fuel efficiency of the current trip, the automobile can immediately react and signal alarms, or even wrestle control from the driver.

Human-in-loop control introduces challenges in the design of CPS systems. Human behaviors can be unpredictable (or partially predictable), which adds uncertainty to the service guarantee of a tight control loop (e.g., response accuracy and timeliness). Human behavior modeling is the current practice to minimize this uncertainty by predicting from learning. However, tight control loops in CPS systems mean the behavior modeling needs to accurately respond in a short period of time. This stringent requirement suggests needed advances in human behavior modeling and control theory.

This paper systematically incorporates humans into a tight control loop to investigate a fundamental CPS issue. We note that CPS domains have a wide spectrum of issues and problems, so it is necessary to amass examples of human-in-the-loop solutions for multiple domains before general solutions and theory will emerge. To this end, we develop

human-in-the-loop control models and mechanisms for reducing the energy waste of computer workstations. Considering that 72% of the total U.S. electricity consumption occurs in residential and commercial buildings [1] (with 30% of the energy consumed in buildings being *wasted* [2]), any amount of energy saved has an impact on society.

Our user study with 20 human subjects suggests distractions as a major source of energy waste on computers. However, current common practice for detecting distraction in computer usage to reduce energy waste is very coarse-grained. It is usually based on a fixed timer that initiates the sleep mode of the computer after several minutes of inactivity. Our distraction detection addresses human behavioral uncertainty with strategies such as adaptive timeout intervals, multi-level sensing and addressing background processing.

The main contributions of this paper are:

- Collecting empirical data on software development and research professionals to understand their human behavior regarding distractions in computer usage.
- Treating the main issues of computer use and monitoring that includes human interaction as well as background processing of many types, and determining a distraction model of human behavior for computer use.
- Developing a human-in-the-loop feedback control solution for an important category of computer users that detects distraction with 97.28% accuracy and cuts energy waste by 80.19%.

II. PROBLEM STATEMENT

Computers are ubiquitous in homes and offices, and they account for 40–60% of the energy consumption in typical office buildings. Over the years, manufacturers have developed and incorporated various power-saving modes of operation. However, accurately activating the appropriate power-saving modes remains a challenge.

Since the usages of workstations are mostly user-driven, the current practice is providing users with a nob for the sleep timer. In most cases, sleep mode for the monitor and workstation is actuated after a timeout and/or during typical non-working hours. As Section VI shows, this naive approach is coarse-grained and suboptimal. Interestingly, this approach of putting humans as an external element of control loops also exists in many current CPS systems. In this paper we demonstrate the benefits of moving humans inside the loop by considering distractions in computer usage.

We define distractions as periods when the user's current activities do not benefit from running the computer, e.g., answering phone calls, having an office meeting, or restroom breaks. The challenge is to detect the distraction behavior as soon as possible without impacting the user experience by considering the uncertainty of human behavior in a tight energy-saving control loop. We assume that a workstation includes a motherboard, CPU, RAM, hard disk drives, and all other components that usually reside in a computer case. We also assume that a computer consists of a workstation and (one or more) monitor(s). We maintain this definition of "workstation" and "computer" throughout the paper.

III. APPROACH

In this section, we describe the baseline solution and our proposed solution.

A. BASELINE SOLUTION

The current common practice of recognizing distractions is very coarse-grained: a timer that automatically turns off the monitor and puts the workstation into sleep mode after several minutes of inactivity. Similar to bang-bang control, the system switches between an active state and a sleep state. While the timeout interval is configurable, we assume the default configuration of Windows 7 machines, i.e., 10 minutes for the monitor and 30 minutes for the workstation as the baseline solution.

B. OUR SOLUTION

Our solution brings humans into the control loop. In this section, we describe the control architecture and different strategies employed by our control, including an adaptive timeout interval, multilevel sensing, and detection of background processing.

1) CONTROL ARCHITECTURE

In this section, we describe the overall architecture of our human-in-the-loop control based distraction detection system. Our solution enables building a distraction model for research and software development computer users. Intuitively, different users may have their own working habits and attention spans. Our distraction model aims to capture user-specific distraction patterns, so that it can help detect distractions early. Given that users regularly work on computers, their distraction patterns and usage patterns can be learned over time. We note that different users can use a public computer. Here we assume that users must login to their accounts before using the computer.

Our distraction model considers both user activities and system activities, as shown in Figure 1. At the user activity level, the model tracks gaze to detect if the user is distracted. At the system activity level, the keyboard and mouse events, CPU usage, and network activities are monitored. Combining

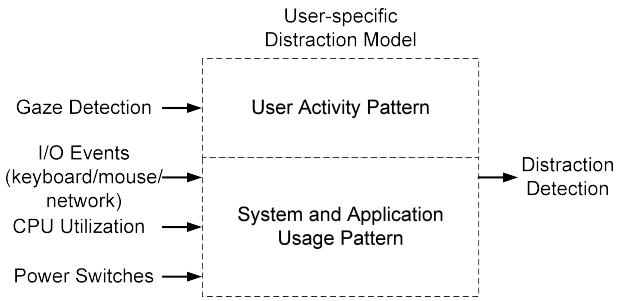


FIGURE 1. Distraction Model.

both types of information enables the model to make a quick assessment of the distraction status of the user.

Based on the user distraction model, we design a human-in-the-loop control for reducing waste in computer usage. As shown in Figure 2, this control design consists of two loops: the inner loop within the computing system and the outer loop that incorporates the user.

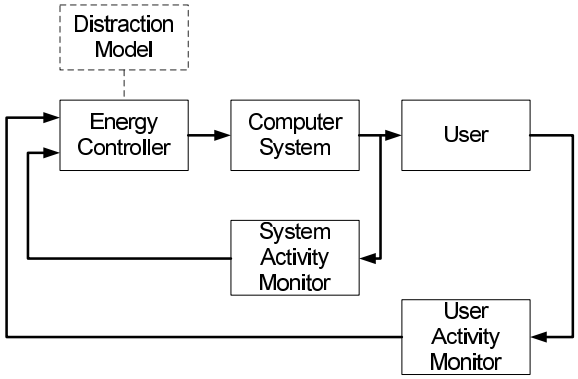


FIGURE 2. Control Architecture for Energy Saving with Human in the Loop.

- The inner control loop monitors the system IO, processing, and networking activities. Recent keyboard/mouse events, high CPU utilization, and networking bandwidth all represent feedback that the system is working effectively. On the other hand, the system is idling if there are no recent keyboard/mouse activities. The system activity monitor also records when the user turns on the computer, which serves as a negative feedback if the user turns on the computer soon after the controller puts it to sleep.
- The outer control loop detects user activities to identify if the user is distracted from using the computer. In this work, we mainly investigate gaze detection. The gaze detection results serve as the feedback to the energy controller.

The inner loop and outer loop are complementary to each other. Existing solutions are mainly based on system information represented by the inner control loop that use same default timeout interval for everyone, which is not efficient in energy saving. Our solution exploits an adaptive timeout interval (c.f. Section 3.2.2) in the inner loop. Also, as a first step towards the human-in-the-loop control, the outer loop takes human behavior into consideration, generating more accurate control designs by tracking eye gaze. Note that we do not need to track the user's gaze all the time. It is only triggered by the outer loop to verify user distraction when the inner loop detects few recent system activities and that's why we call it multilevel sensing (c.f. Section 3.2.3). Also, the inner loop checks if any effective background process (file transfer, computation, etc) is running during human distraction, which serves as negative feedback to the energy controller (c.f. Section 3.2.4).

The energy controller takes the system activities and user behaviors as inputs, and adaptively adjusts the timeout interval as the control output. The control decision is made based on the learned user distraction patterns.

This control design maintains a consistent user experience while reducing energy waste for the system idle duration when the user is distracted. This design represents a tradeoff between energy saving and user experience. Although each user has his/her own requirements, the controller provides different control policies for different users. For users with long attention spans and distractions, the controller employs an energy saving policy. With this policy, the controller aggressively adjusts the timeout interval, so that computer is put to sleep as soon as distraction is detected. Whereas for users with short attention spans who usually get back to work shortly after distraction, the controller employs a policy to improve user experience.

2) ADAPTIVE TIMEOUT INTERVAL

In this section, we describe a mechanism that is implemented in the "System Activity Monitor" of Figure 2 as a part of the inner loop. We call it adaptive timeout interval. The reason for using this strategy is, current common practice basically uses a default timeout interval for all people. Although people can change it, usually they set a conservative timeout interval that wastes energy, but reduces unintended sleep of the workstation. We believe this timeout interval should be learned and adaptive to individuals based on their involvement with the workstation.

We collect data from the computer of ten software development and research individuals for an entire day of their office work, containing an average of 9.43 hours of computer usage per subject. We collect the timestamps of using the keyboard and mouse. We ask the subjects to provide the ground truth of when they are working and when they are distracted (c.f. Section 5).

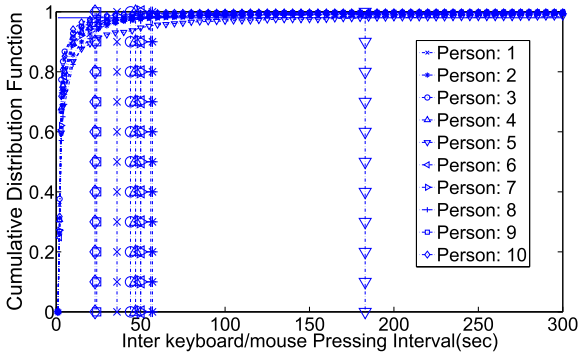


FIGURE 3. Cumulative Distribution Function (CDF) of Inter keyboard/mouse Pressing Interval (*IPI*) of the ten subjects. It also shows the 98-th percentile of their *IPI*.

Let us define *IPI* be the Inter keyboard/mouse Pressing Interval of a user. *IPI* is a random variable and the Cumulative Distribution Function (CDF) of *IPI* of the ten subjects is shown in Figure 3. Let us define t_{IPI}^{98} be the 98-th percentile

of the *IPI*. We mark the t_{IPI}^{98} of the ten subjects in Figure 3. It shows that t_{IPI}^{98} is 23 seconds for person 10, i.e., the *IPI* is less than or equal to 23 seconds with 98% probability. From Figure 3, we observe that (i) t_{IPI}^{98} spans from 23 seconds to 183 seconds for 10 subjects. So, the timeout interval should be adaptive. (ii) For nine out of ten subjects, t_{IPI}^{98} is less than 60 seconds. It means that for most of the subjects, the *IPI* is usually much smaller than 30 minutes with a very high probability.

The problem of detecting distraction boils down to distinguishing the remaining 2% case where *IPI* is greater than t_{IPI}^{98} and the user is either working or distracted. Note that although *IPI* is greater than t_{IPI}^{98} has only 2% probability, since the average number of keyboard/mouse events per day is 3255 per subject, if we ignore this case, the solution will put the computer into sleep 65 times per day on average, which will be a huge disappointment for the users as a significant portion of it will take place while the user is working. To distinguish whether the users are really working and taking more than t_{IPI}^{98} to use their keyboards/mouses or they are distracted, we introduce multilevel sensing.

3) MULTILEVEL SENSING

In this section, we describe multilevel sensing, a technique that is used by the "User Activity Monitor" of Figure 2 as a part of the outer loop. The basic idea is to track the eye gaze of the subjects when they are working but not using the keyboards/mouses for more than t_{IPI}^{98} (c.f. Section 3.2.2). When we collect ten subjects' data of computer usage of an entire day, we instrument each of the ten subject's workstation with ITU Gaze Tracker [3] that uses a webcam to track the subject's gaze. It allows us to know if the subject is looking at the monitor in every second.

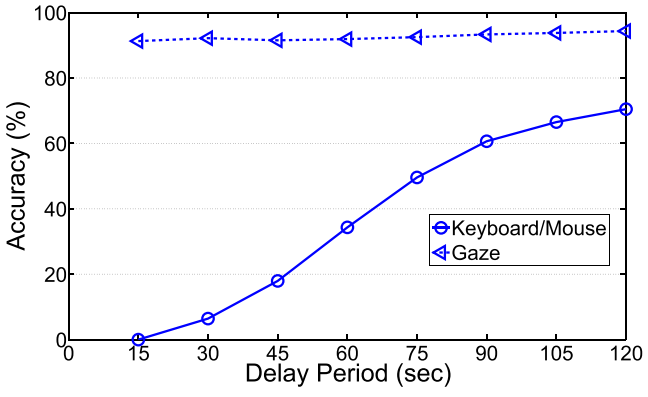


FIGURE 4. Average accuracy of detecting that the subject is working for various delay periods by using only keyboard/mouse or gaze.

Let us define t_d be the delay period for which we monitor user activities after he/she is not using the keyboard/mouse for more than t_{IPI}^{98}. For these ten subjects, Figure 4 shows the average accuracy of detecting if the user is working by using only keyboard/mouse or gaze for various delay periods. It shows that we can detect that the user is working within 15 seconds with 91.27% accuracy if we track the gaze. On the other hand, the accuracy reaches only 70.46% even if we wait for 2 minutes when relying on only keyboard/mouse. This result indicates that tracking gaze enables understanding user involvement with the computer *much quicker with much higher accuracy* than using only the keyboard/mouse.

Note that we do not need to run the webcam for the whole duration of computer usage. We only need to run it in the remaining 2% case when *IPI* is greater than t_{IPI}^{98}. That's why we call it multilevel sensing. Because, we sense user involvement with the workstation using keyboard/mouse first. If we see that the user is not using the keyboard/mouse for more than t_{IPI}^{98}, we turn on the webcam and monitor the gaze for t_d period of time. Although a webcam consumes additional power, its power consumption is much smaller than that of a workstation. A detailed analysis on power consumption is shown in Section 6.3.

We use a threshold $T_{gaze\_tracker}$ to determine if the user is distracted or not at the time of tracking gaze. Recall that we collect 1 gaze sample per second and track gaze for t_d seconds. If the user is constantly looking at the monitor and the gaze tracker is 100% accurate, then the percentage of gaze events where the user is looking at the monitor should be 100%. However, a user may not look at the monitor all the time while he is working and the gaze tracker is not 100% accurate. So, we consider the user is working when the fraction of gaze events where the user is looking at the monitor is greater than or equal to $T_{gaze\_tracker}$.

4) BACKGROUND PROCESSING

In this section, we describe how we address background processing, which is monitored by the "System Activity Monitor" of Figure 2 as a part of the inner loop. Note that detecting user involvement with the workstation in terms of keyboard/mouse activities or gaze at the monitor is not enough to put the computer to sleep. This is because people run background processes. When we detect the user is not directly involved with the computer, but some background processes are being run, we turn off the monitor, but keep the workstation running.

To understand how common it is to run background processes during the office hours, we take a survey at University of Virginia and Microsoft Research Asia. We ask the following question:

During office hours, do you run any processes in the background except music and remote desktop daemon for which you don't want your machine to go to sleep when you leave your workstation?

We get 29 survey responses and 51.7% respond with a 'No'. Although 48.3% respond with a 'Yes', some of them say:

Although I clicked yes, it is actually quite rare. On those rare occasions, I can manually go to the power settings to disable sleep time.

The survey responses are shown in Figure 5. Note that "No Background Processing" in the figure actually means

no background processing except remote desktop daemon and music. The reason for not considering these two cases is because, state of the art techniques are available to address these issues. When we say we put the workstation to sleep, we mean sleep state S3 (suspend to RAM). At this state, the workstation is not capable of responding to remote desktop login requests. SleepServer [4] and Somniloquy [5] allow computers to be responsive to network traffic even if they are in sleep state S3. Monitoring the sound card activity allows us to know if the user is playing music. It is difficult to know if the user is present and listening to music. Location aware power management techniques [6]–[9] can help in this case. We did not implement this feature.

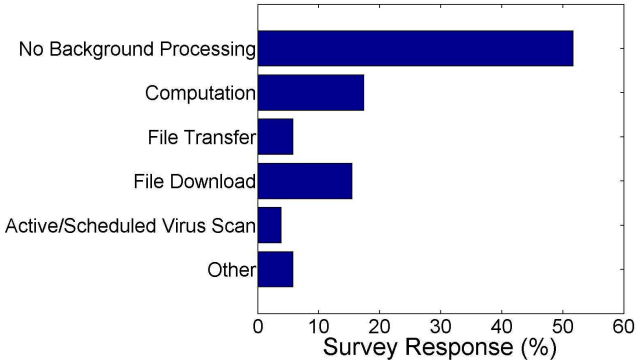


FIGURE 5. Survey results on running background processes during office hours.

Figure 5 shows that most of the background processing are either computation or network activities, e.g., file download and file transfer. To address computation intensive background processing, we monitor CPU utilization of the workstation for the duration of $(t_{IPI}^{98} + t_d)$ before putting it to sleep. If average CPU utilization is less than $T_{CPU\_utilization}$ then we put the workstation to sleep, where $T_{CPU\_utilization}$ is a threshold. A similar strategy should suffice in detecting file transfer and file download related network activities.

Since computers are used in a variety of ways, there are still cases for which we shouldn't put the workstation to sleep. Some of these cases appear in ''Other'' of Figure 5. For example, when people run Instant Messenger, e.g., Skype and wait for someone to call, the keyboard/mouse usage, the gaze, the CPU utilization and network activity are not sufficient indicator to figure out that the workstation should not be put to sleep. Note that if the users are not nearby, we can put their workstations to sleep in some cases. When they return and their workstations wake up, they will receive the messages. One problem with this strategy is that if the workstations go to sleep, the other side of the messenger will see the users' offline. To address this problem, we can create a live messenger server which will show the users online after putting their workstations to sleep. If the users want to be notified whenever new message arrives, the live messenger server can use SleepServer [4] or Somniloquy [5] to wake up their workstations and deliver the message. Alternately, if the users want to be notified when they are nearby, we can use this technique along with a location aware power management technique [6]–[9] to figure out if the users are there. Addressing the messenger related background processing is important as some people use it during office hours and at home. However, addressing this problem is just a matter of implementation and that's why we leave it to future work. At this time, to address the exceptional cases related to background processing, we maintain a list of exceptional processes $P_{exceptional}$. When any process listed in $P_{exceptional}$ runs, we do not put the workstation to sleep. This list is created from user feedback and it is the responsibility of the user not to run these processes unnecessarily to take full advantage of the energy saving. Note that if the user wants to run background processes, then useful work is being done and keeping the workstation on is not an act of energy waste.

5) PARAMETER AND THRESHOLD SELECTION

The performance of distraction detection solution depends on the selection of parameters and threshold values. Our solution learns and adjusts thresholds based on individual's working and distraction behavior. In this section, we describe the strategy for selecting these parameters and threshold values.

When our solution starts, it doesn't know t_{IPI}^{98} of the user. Initially, we assume t_{IPI}^{98} be 60 seconds for an individual as we see in Section 3.2.2 that nine out of ten subject has t_{IPI}^{98} less than 60 seconds. As we get more keystrokes and mouse events, we accurately compute t_{IPI}^{98} and update its value.

We define t_{dt} be the distraction detection time of an individual. It consists of two parts: t_{IPI}^{98} where we monitor keyboard and mouse events, and t_d where we track the gaze. We plan to detect distraction of an individual within two minutes and set t_{dt} to two minutes. As mentioned before, initially t_{IPI}^{98} is set to 60 seconds. So, the initial value of t_d is $120 - 60 = 60$ seconds. As we get more keyboard and mouse events, we update t_{IPI}^{98} accordingly. We keep $t_d = t_{dt} - t_{IPI}^{98}$ and make sure that t_d has at least 30 seconds to track gaze as tracking 30 seconds of gaze allows detecting the subject is working with high accuracy (c.f. Figure 4). When t_{IPI}^{98} increases so much that $(t_{IPI}^{98} + 30$ seconds) exceeds two minutes, we update t_{dt} to accommodate.

Overall we use three thresholds in our solution: t_{dt}, $T_{gaze\_tracker}$, and $T_{CPU\_utilization}$. t_{dt} is initialized to two minutes as mentioned above. $T_{CPU\_utilization}$ is initialized with 0.4 as in [10]. $T_{gaze\_tracker}$ is also initialized with 0.4. When the user is distracted, we do not get any feedback from the user. However, if the user is working and we mistakenly put the computer into sleep, we get a negative feedback. When that happens, we adjust threshold values dynamically to capture individual's distraction behavior as follows.

Usually CPU utilization is higher when the user is working and lower when the user is distracted (and so not doing any background processing). We need to update $T_{CPU\_utilization}$ in a way that when the CPU utilization is higher than this threshold, we can assume that the user is working. To do that,

we keep track of the average CPU utilization of the computer when the user is working and when he/she is distracted. We set $T_{CPU\_utilization}$ to be the value that separates the two average values most, i.e., the value that lies in the middle of these two averages. Similarly, we keep track of the average fraction of gaze events where the user is looking at the monitor when the user is working and when the user is distracted. We set $T_{gaze\_tracker}$ be the value that separates these two average values most.

Note that it is possible to have cases when CPU utilization is very low and almost no gaze event is detected, but the user claims to be working. In these cases, changing $T_{CPU\_utilization}$ or $T_{gaze\_tracker}$ is not going to help much in differentiating the distraction state from the working state. Instead, we need to increase distraction detection time t_{dt}. To address this issue, we increase t_{dt} by two minutes at a negative feedback. It has a significant impact on energy waste. So, instead of increasing it every time we get a negative feedback, we increase it every other time.

IV. ENERGY WASTE MODEL

In this section, we design a model that analyzes the energy waste of a generic distraction detection algorithm. A specific algorithm may use several parameters to capture distraction. For example, as mentioned in Section 3.2, we use several parameters to capture an adaptive timeout interval, multilevel sensing, and background processing to detect distraction. This section attempts to abstract away the internal details of a distraction detection solution and defines some parameters that captures its performance in energy waste from a high level.

A distraction detection algorithm may take an arbitrary amount of time to detect distraction and use additional power for it, e.g., an algorithm may take 2 minutes of time to detect distraction with 5 W of additional power. The question is, is it going to waste energy by detecting distraction this way? If it does, how much energy is wasted per person for an entire day?

To answer these questions, we design a model that takes into account the following factors:

A. HUMAN DISTRACTION PATTERN

The energy waste on distraction largely depends on people's distraction pattern. We define two parameters that capture the subject's distraction pattern of an entire day. Let $\{t^{WS}_{sleep_1}, t^{WS}_{sleep_2}, \ldots, t^{WS}_{sleep_n}\}$ be the durations of time when the workstation could be put into sleep and $\{t^{MN}_{off_1}, t^{MN}_{off_2}, \ldots, t^{MN}_{off_m}\}$ are the durations of time when the monitor could be turned off in an entire day.

B. HARDWARE POWER

We define P^{WS}_{on}, P^{WS}_{sleep}, and P^{MN}_{on} to be the power consumption of the workstation when it is on, the power consumption of the workstation when it is put into sleep, and the power consumption of the monitor when it is on, respectively. Empirical results with multiple monitors at our lab show that monitors consume negligible amount of power when they are off. So, we ignore that power consumption from our analysis.

C. ALGORITHM PARAMETERS

We assume that a distraction detection algorithm takes t_{dt} amount of time and $P_{additional}$ amount of power to detect distraction. Note that t_{dt} can be different for the monitor and the workstation depending on the algorithm. For example, the baseline solution takes 10 minutes for the monitor and 30 minutes for the workstation to detect workstation with 0 W of additional power in both cases. We also assume that $P_{additional}$ is consumed only for the duration of $t_{additional}$ in an entire day of computer usage.

For simplification, the model assumes that monitors and workstations can be put into sleep and awakened instantaneously. We list all the model parameters in Table 1.

TABLE 1. Parameters for modeling the energy waste of a distraction detection algorithm. The first two parameters describe human distraction behavior, the next three parameters are specific to hardware power, and the last three are related to the distraction detection algorithm.

Notation	Parameter Definition
$\{t^{WS}_{sleep_1}, t^{WS}_{sleep_2}, \ldots, t^{WS}_{sleep_n}\}$	Durations of time when the workstation could be put into sleep mode.
$\{t^{MN}_{off_1}, t^{MN}_{off_2}, \ldots, t^{MN}_{off_m}\}$	Durations of time when the monitor could be turned off.
P^{WS}_{on}	Power consumption of the workstation when it is on.
P^{WS}_{sleep}	Power consumption of the workstation when it is put into sleep.
P^{MN}_{on}	Power consumption of the monitor when it is on.
t_{dt}	Time to detect distraction.
$P_{additional}$	Additional power consumption to detect distraction.
$t_{additional}$	Duration of time when $P_{additional}$ is consumed.

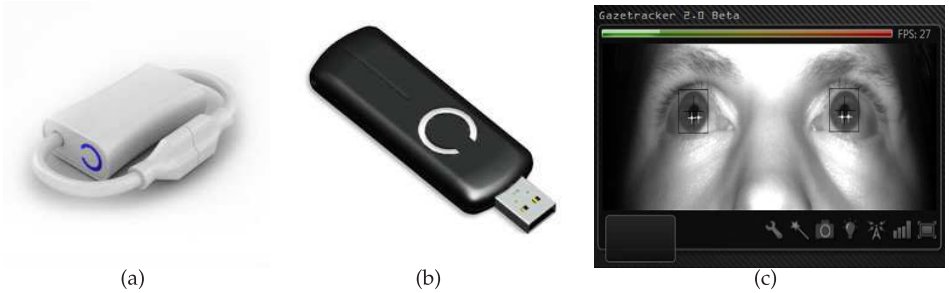
We define the energy waste of an algorithm as the energy it fails to save, as compared to the optimal solution. We assume that the optimal solution detects distraction immediately without consuming any additional power, i.e, the solution with $t_{dt} = 0$, $P_{additional} = 0$, $t_{additional} = 0$. Although the optimal solution does not exist in practice, i.e., we do not actually have a system that can detect distraction immediately without any additional power, but measuring energy waste with respect to the optimal solution gives us an estimation about how much room we have for the improvement.

Total energy waste of a distraction detection algorithm is,

$$E_{waste} = E^{monitor}_{waste} + E^{workstation}_{waste} + E^{additional}_{waste} \quad (1)$$

Where, $E^{monitor}_{waste}$, $E^{workstation}_{waste}$ are the energy waste due to running the monitor and workstation, respectively. $E^{additional}_{waste}$ is the energy waste for taking additional power. Now we estimate each of these three terms.

A distraction detection algorithm wastes energy on the monitor for running it for the time the user is distracted. If the



(a) (b) (c)

FIGURE 6. Aeon Labs (a) Energy Plug Load and (b) Z-Stick. (c) Gaze Tracker is tracking gaze.

distraction time ($t_{off_i}^{MN}$) is smaller than detection time (t_{dt}), then this distraction remains undetected and energy is wasted for running the monitor for the distraction time. Otherwise, the distraction is detected and energy is wasted only for the time of detecting distraction. So, energy waste for running the monitor is,

$$E_{waste}^{monitor} = \sum_{i=1}^{m} min(t_{off_i}^{MN}, t_{dt}) * P_{on}^{MN}$$

Similarly, energy waste on the workstation for running it for the time the user is distracted instead of putting it into sleep is,

$$E_{waste}^{workstation} = \sum_{i=1}^{n} min(t_{sleep_i}^{WS}, t_{dt}) * (P_{on}^{WS} - P_{sleep}^{WS})$$

The algorithm takes $P_{additional}$ additional power to detect distraction and wastes energy for $t_{additional}$ time. The additional energy consumption by the distraction detection algorithm is,

$$E_{waste}^{additional} = P_{additional} * t_{additional}$$

This model is very helpful in measuring energy waste. It also gives us a deeper understanding about various components that affect energy waste and helps us to figure out the areas to improve. It enables the direct comparison between two distraction detection algorithms. For example, it allows us to compute the energy waste of the baseline solution, i.e., the solution with $t_{dt} = 10$ minutes for the monitor and 30 minutes for the workstation, and $P_{additional} = t_{additional} = 0$. Now we can answer questions like, is it going to waste less energy than the baseline solution if we can detect distraction within 5 minutes, but consume 2.5 W of additional power? We use this model to compare our solution with the baseline solution in terms of energy waste in Section 6.3.

V. DEPLOYMENT AND DATA COLLECTION

We deploy our distraction detection system to ten individuals' computers and collect data for an entire day of their office work. The individuals represent an important class of computer users as all of them are involved in research and software development. The collected data contains an average of 9.43 hours of computer usage per subject. We ask the subjects to provide the ground truth of when they are working and when they are distracted in two ways: (i) they write the ground truth on a paper log and (ii) they use a software to log the ground truth. This two level of ground truth collection reduces the chances of forgetting to log the distraction episodes. We cross check these two versions of ground truth and find consistency.

Our deployed system consists of three components: energy load sensing, computer activity tracking, and human activity recognition.

Energy Load Sensing: We use Aeon Labs Smart Energy Plug Load [11] to measure the workstation and the monitor energy consumption (c.f. Figure 6). The Plug Load sends a power reading to the Z-Stick USB dongle [12] every 10 seconds. We log the energy consumption during both the active use and the sleep state.

Computer Activity Tracking: We run a key logger program at the subject's computer to record the time of keyboard and mouse activities. We record the CPU utilization every five seconds by averaging 10 samples collected at an interval of 500 milliseconds.

Human Activity Recognition: To detect if the subject is looking at the monitor, we use ITU Gaze Tracker [3], an open source gaze tracking software. The software uses a webcam to estimate the coordinate of the monitor display area where the subject is looking at every second (c.f. Figure 6(c)). We run the Gaze Tracking software on a separate computer to prevent introducing noise to CPU utilization monitoring.

VI. EVALUATION

In this section, we evaluate the performance of our human-in-the-loop control based distraction detection system. We start with a user study on 20 people that shows tremendous prospects in reducing energy waste by early detection of distraction. Then we show the accuracy of our distraction detection system. Next we compare our solution with the baseline solution in terms of energy waste. Finally, we show the loss of comfort in using our solution.

A. USER STUDY RESULTS

Our first evaluation step is to demonstrate the potential gain of our system over the baseline solution by providing both

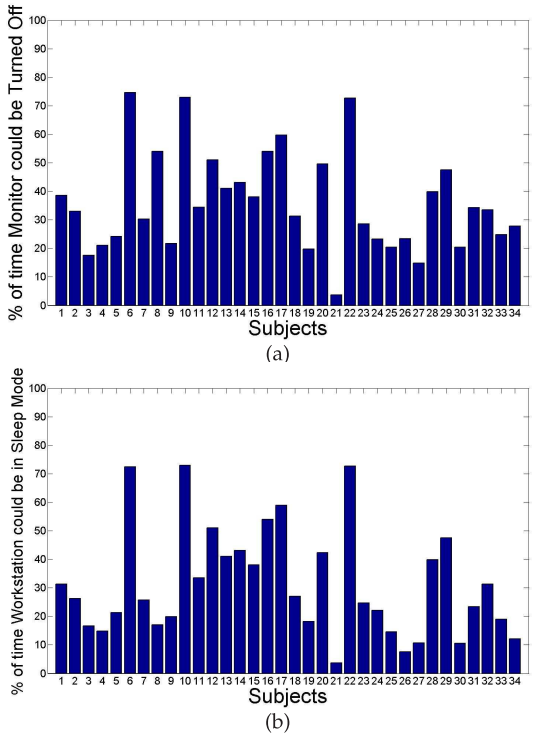


FIGURE 7. Percentage of time during a day users indicate that monitors or workstations are not needed. (a) Monitors. (b) Workstation.

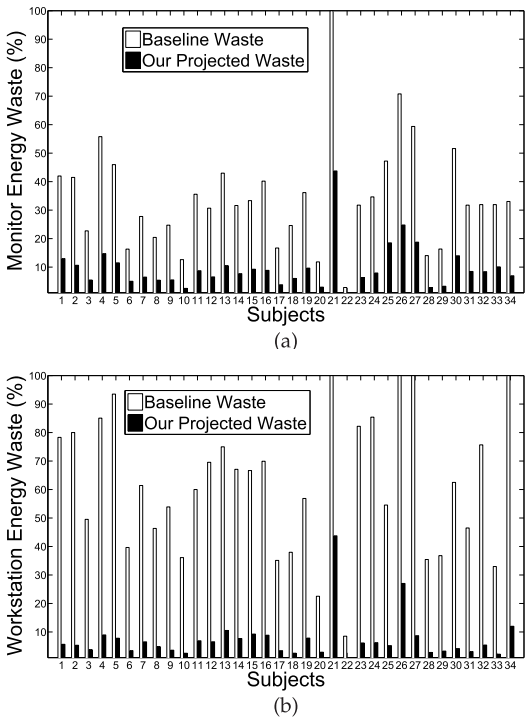


FIGURE 8. Comparison of energy waste if computers adopt the baseline solution and our proposed solution. (a) Monitors. (b) Workstation.

solutions with real-world user behaviors. We collect real-world data traces by conducting a user study among researchers and software developers at University of Virginia and Microsoft Research Asia. We instruct subjects to record their daily computer usage for several days. And, for each distraction, the logs include both the length and whether turning off monitors and/or putting the workstations into sleep mode is appropriate. In total, we receive 34 responses from 20 people with an average of 9.41 hours of computer usage per day.

Figure 7 shows the percentage of time during a day that users indicate that they do not need monitors or workstations to stay on. In fact, the results show a significant space for energy saving. Specifically, monitors and workstations are not needed 36.04% and 31.35% of the working hours, respectively.

Next, we evaluate how well the baseline solution and our solution perform for the ground truth in Figure 7. Intuitively, since our solution not only monitors computer activity, but also monitors user activity by bringing a human into the loop, it can detect distraction much earlier and turn off monitors and put the workstations into sleep much more quickly.

Figure 8 shows the energy waste of the baseline solution and our solution based on the data trace collected from the user study. Here, our proposed solution takes two minutes to detect distraction accurately from the past behavior of the user and puts the workstation into sleep immediately. The reason for choosing two minutes is because, although the timeout interval is adaptive to user behavior, we find that t_{IPI}^{98} is less than one minute for nine out of ten subjects (c.f. Section 3.2). Therefore, two minutes seems to be a reasonable interval for detecting distraction for the large majority of the people in our sample. For some people, two minutes is not enough and our adaptive solution adjusts parameters depending on their behavior and reduces less energy waste from their computers as shown in Section 6.2.

Figure 8(a) shows that the baseline solution fails to save about 34.43% energy waste for the monitor on average. For some subjects, all the distraction durations are less than 10 minutes causing the baseline solution to fail to save 100% energy waste. And, this figure reduces by a factor of 3.56 (to about 9.67%) with our solution. Much of the improvement comes from early detection of distractions by bringing humans into the loop. Similarly, Figure 8(b) shows an energy waste reduction by a factor of 8.43 (from 61.92% to 7.34%) on workstations with our solution. The reason for better performance for the workstation is because the baseline timeout interval for the workstation is 30 minutes which is larger than that of the monitor (10 minutes).

B. ACCURACY OF DISTRACTION DETECTION

In this section, we evaluate the performance of our human-in-the-loop based distraction detection in terms of *accuracy* (% of time we correctly classify distractions and non-distractions), *false positive rate* (% of time we classify the subject distracted while he/she is actually working) and

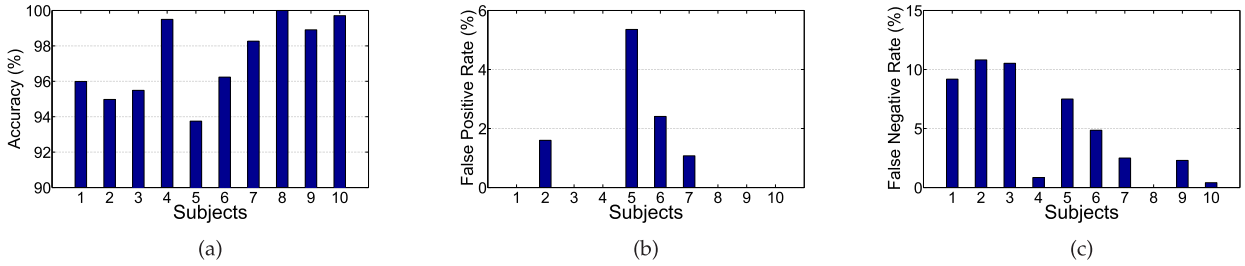


FIGURE 9. (a) Accuracy, (b) False Positive Rate and (c) False Negative Rate of the Human-in-the-loop based Distraction Detection Algorithm.

false negative rate (% of time we classify the subject is working while he/she is actually distracted). To evaluate the performance of our system, we instrument ten human subjects' workstations and monitors for an entire working day as mentioned in Section 5. This experiment requires subjects to specify the ground truth in two levels (c.f. Section 5) of when they are distracted and whether their workstations should stay on during each of the distractions at a minute level of granularity. The collected data contains an average of 9.43 hours of computer usage per subject.

We divide the computer usage data of a whole day of each subject into a number of time slots. The length of each slot has a duration of t_{dt}, which is the distraction detection time. At the time of computing the accuracy, false positive rate, and false negative rate of our solution, we compute the performance in classifying these time slots. We choose thresholds as described in Section 3.2.5. Since none of the subjects use any background processing while they are distracted, we don't have the opportunity to evaluate the solution on background processing.

Figure 9 shows average accuracy, false positive rate, and false negative rate of the distraction detection algorithm. It shows an average accuracy of 97.28%, which suggests that the system can detect users' working and distracted states with high accuracy. The average false positive rate is only 1.04%. The reason for this false positive is because of two reasons. First, after interviewing a subject, we come to know that there are times when it is really hard to say whether someone is distracted or not. Because, sometimes the subject was reading (that does not involve monitor) and using the computer intermittently. During the whole time, he/she recorded that he/she was working with the computer in the ground truth. But in actual practice, he/she was reading sometimes. These types of incidents increase false positive rate. Second, sometimes the gaze tracker can not track eyes, because the subject is not looking at the monitor directly, or due to environmental factors, e.g., if the eye comes too close or goes too far from the webcam. The average false negative rate is 4.89%. The reason for this false negative rate is due to the fact that when the subject is distracted, the gaze tracker can still track eyes, because sometimes the subject is still there, and other times the subject has left, but the gaze tracker mistakenly classifies other objects as eyes, e.g., some black object at the background, or other people's movement. Although 4.89% false negative rate is non-negligible, it helps keeping the false positive rate low and enables reducing significant energy waste at the cost of low loss of comfort as shown in the next two sections.

C. ENERGY WASTE REDUCTION

In this section, we compute the energy waste of the baseline solution and our solution for the 10 subjects based on their computer usage of an entire day using the model specified in Section 4. We choose two variations of our solution. In one variation, we assume that we can detect distraction perfectly within 2 minutes. In another variation, we use the actual human-in-the-loop based feedback control system to detect distraction.

The model parameters are chosen in the following way. There are 3 types of model parameters: human distraction pattern, algorithm parameters, and hardware power. For the human distraction pattern, we use the ground truth of the distraction durations of the 10 subjects for the baseline solution and our solution that assumes perfect distraction detection. For the actual case of our solution, instead of using the ground truth, we use our solution to detect distraction. For algorithm parameters, for the baseline solution, we set $t_{dt} = 10$ minutes for the monitor and 30 minutes for the workstation, and $P_{additional} = t_{additional} = 0$. For the perfect detection case of our solution, we set $t_{dt} = 2$ minutes for all the subjects. For the actual case of our solution, we initialize $t_{dt} = 2$ minutes for all the subjects and update it based on their behavior as described in Section 3.2.5. We measure the hardware power of the computers of each subject using techniques mentioned in Section 5. We use a Lenovo USB webcam that draws 480 mA current from the USB port. The USB port provides a 5V supply. So, the webcam consumes $480 * 5/1000 = 2.4$ W of power, which is $P_{additional}$. Since we use multilevel sensing, we only turn on the webcam when the user is not using the keyboard/mouse for more than t_{IPI}^{98} and keep it on for t_d duration. Let's say such events occur k times per day. We compute k from the data trace and set $t_{additional} = k*t_d$.

We compute the energy waste of the baseline solution and two variations of our solutions using equation (1) and show it in Figure 10. We see that the baseline solution wastes 698.12 KJ of energy on average per day per computer

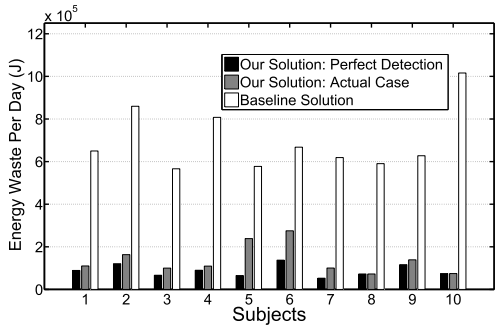


FIGURE 10. Comparison between our solution and the baseline solution for energy waste.

where our solution with human-in-the-loop control wastes only 138.28 KJ of energy per day per computer, which is about 19.81% of the baseline solution. So, our solution cuts energy waste by 80.19% of the baseline solution on average, which can have a huge impact on the society. For example, in 2003, 77 million people in the U.S. used a computer at work [13]. Assuming 8 hours of computer use per work day and 250 work days per year (excluding weekends and 10 holidays), our projected annual savings would be 2.54 billion kWh nationwide. It will be even more if we consider distractions in computer use at home.

TABLE 2. Comparison between subject 3 and subject 10 for their distraction behavior in computer usage.

Criteria	Subject 3	Subject 10
Total computer on time per day	9.05 hours	11.53 hours
Total distracted time per day	3.9 hours	8.16 hours
Total number of distraction episodes per day	9	6
Average distraction episode length	25.98 minutes	81.65 minutes
Standard deviation of distraction episode length	28.15 minutes	57.24 minutes
Average monitor power	29.26 Watt	22.07 Watt
Average workstation power	30.68 Watt	85.53 Watt

In Figure 10, we also see that subject 3 has the smallest energy waste whereas subject 10 has the largest energy waste for the baseline solution. We use the model specified in Section 4 to figure out the underlying reasons of this difference. Table 2 shows some major differences. We see from Table 2 that subject 10 spends more time with his/her computer per day as well as gets distracted for longer periods. Also, subject 10 uses a desktop computer whereas subject 3 uses a laptop with an external monitor. Thats why the average workstation power of subject 10 is much larger than that of subject 3. The combined effect of all these differences constitutes a larger energy waste for subject 10.

Note that there is a difference in energy waste between the two variations of our solution. The solution with perfect distraction detection wastes 88.49 KJ of energy per day per computer on average whereas our actual solution wastes 138.28 KJ of energy per day per computer. For subject 5, this difference is very clear. The reason is, for subject 5, the perfect detection version of our solution takes 2 minutes to detect distraction. On the other hand, our actual solution starts with two minutes of detection time and based on the subject's behavior the detection time increases to 7 minutes. So, our actual solution fails to save 5 minutes of power consumption in every subsequent distraction episode. Similarly, for subject 6, distraction detection time increases to 5 minutes in our actual solution. For some subjects, e.g., for subjects 3 and 9, there is a difference between the two versions of our solution although the distraction detection time remains two minutes in the actual solution. The reason for this difference is because of the 4.89% false negative rate. Note that the solution with perfect detection does not exist. The energy that we sacrifice for the false negative rate pays off in terms of comfort of the user as shown in the next section.

D. LOSS OF COMFORT

The above evaluation shows significant reduction in energy waste by using a human-in-the-loop solution. However, it is important to assess if this gain is at the cost of loss of comfort. Measuring comfort is extremely challenging since the notion of comfort varies from person to person and even with a single person, it varies over time. In this work, we define loss of comfort to be when a computer is put to sleep, but this is incorrect for the user. Note that although it is said in the literature [10] that it takes about 10 seconds to wake up a computer from the sleep state, based on our deployment experience with 10 computers, we find that it takes only about 2–5 seconds to wake up modern windows 7 machines. We suggest that these 2–5 seconds of wakeup time are not significant and not a discomfort if the user was really distracted and experiences this wakeup time when he/she returns to work from a true distraction. On the other hand, a put to sleep event will cause a major discomfort if the computer is put to sleep when the subject is still working. In this section, we compute the number of instances where we put the computer to sleep while the subject is working and call these instances as *discomfort events*.

To estimate the number of discomfort events per day, we use the same data and keep the parameter values the same as described in Section 6.2. We divide the entire day of computer usage of each subject into a number of time slots, each having t_{dt} duration, and compute the number of time slots where our solution would put the computer to sleep. To assess the effectiveness of the keyboard/mouse and gaze techniques, we separately evaluate them. We use three variations of our solution. In one variation, we only use keyboard, mouse and do not use anything else. In another variation, we use keyboard, mouse, and gaze. The third variation is our actual solution that considers keyboard, mouse, gaze, and CPU utilization altogether as in Section 3.2.

The result is shown in Figure 11. It shows that subject 9 does not really need a gaze tracker as he/she types very frequently while working. Subject 7 uses the keyboard, mouse infrequently and using gaze tracker reduces the number of discomfort events from 29 to 1 per day! Comparing with

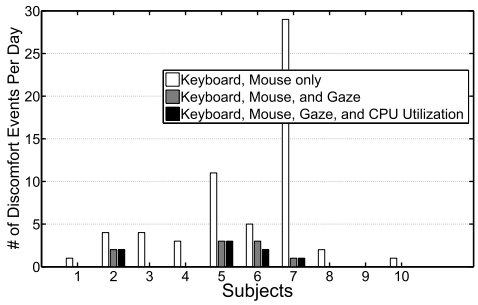


FIGURE 11. Loss of comfort in using our distraction detection solution. It also shows the advantage of tracking eye gaze.

Figure 9(c), we see that several subject's (e.g., subject 1, 2 and 3) high false negative rate allows us to focus more on comfort and reduce their discomfort events. Subject 5 and 6 experience 3 discomfort events per day when our solution relies on keyboard, mouse, and gaze. Subject 5 uses his/her computer for 8.82 hours and all the 3 discomfort events take place within first 7% time of his/her computer usage. During these time slots, no keyboard/mouse event is detected. The fraction of gaze events looking at the monitor is 0 in two of the three time slots and in the other slot it is so low that does not pass the initial value of $T_{gaze\_tracker}$ threshold. These discomfort events perform as a negative feedback to our solution. After these events, the parameters are adjusted as described in Section 3.2.5 and the subject does not observe any discomfort event in the remaining 93% time of his/her computer usage. However, subject 6 uses his/her computer for 11.90 hours and the discomfort events take place during the first quartile, second quartile and the fourth quartile of his/her computer usage when we just use keybaord, mouse, and gaze. Using CPU utilization saves from the last discomfort event, thus reducing the number of discomfort events to 2 per day.

Figure 11 shows that the gaze tracker is very effective in reducing discomfort events for most of the subjects. On average, the solution that relies on keyboard and mouse causes 6.0 discomfort events per day, whereas including the gaze tracking reduces discomfort events to only 0.90 per day. Including CPU utilization reduces it to 0.80 such events per day. This shows that using the human-in-the-loop solution is not only very effective in minimizing energy waste, but does so with minimal discomfort.

VII. RELATED WORK

Human-in-the-loop is not a new concept. It has been studied in the human computer interaction (HCI) area [14]–[16]. However, these works focus on human computer interface designs. Our work focuses on energy saving for CPS systems.

Control designs with a human as part of the loop have been used in physiological control systems [17], [18], mobile sensing and computing systems [19]–[21], thermal control systems [22]–[24], and robotic systems [25]. These works demonstrate that feedback control can be effectively used to control computing systems with feedback directly from the user. In this work, we apply human in the loop control to save energy in the context of user distractions, which is the first work in this direction to the best of our knowledge.

Monitoring energy consumption to identify potential source of energy saving has been an active area of research [26]–[28]. But most of the saving is achieved by detecting occupancy and learning appliance usage patterns, and using these patterns to turn off appliances automatically when they are not in use [22], [23], and [29].

Location-aware power management techniques [6]–[9] exploit users' location information for power management decisions of the computer. These techniques either require very accurate location estimation using ultrasonic systems, or require users to carry some devices. Also, these techniques do not differentiate monitor from workstation for power management decisions, and may miss the opportunity to save energy when the user is nearby, but not using the computer. SleepServer [4] allows end hosts to utilize low power sleep modes without sacrificing availability. [10] uses a power management software to put the computer into sleep mode after 10 minutes of idleness with CPU utilization less than 40%.

VIII. DISCUSSION

Bringing humans into the control loop is a challenge for CPS applications as it requires modeling of complex behavioral, psychological, or physiological aspects of human beings. The level of modeling for each of these aspects depends on application requirements. One very important behavior to model for many applications is human distraction. For example, detecting a driver's distraction is important for safety. Here the distraction itself and the underlying behavioral and physiological aspects of the driver may also be necessary for accurate and timely detection of distraction. On the other hand, modeling of distraction alone is enough for detecting distraction in computer usage as the goal is to save energy. Many other applications are also affected by human distraction. A human distraction in a home health care application may cause someone to forget to take his/her medication or to turn a stove off. In an industrial plant a distraction may cause injury to a human operator. While each of these applications where models of human distraction may prove useful may seem different, there are also similarities. For example, learning user specific timing thresholds for when something is a distraction, adjusting any parameter over time as individuals change their behavior, and using different sensing modalities (e.g., gaze) to best detect the distraction are central to all solutions. We believe that it is necessary to begin to create individual human-in-the-loop solutions for distraction behaviors before general principles will emerge. In this paper, we start with designing a human behavioral model that captures distraction in computer usage and saves energy by employing a human-in-the-loop feedback control system for one important category of computer users that are involved in research and software development. As we find solutions to various other CPS applications involving humans, we will be better

able to find general principles that are common across these CPS applications.

IX. LIMITATIONS AND FUTURE WORK

In this section, we discuss two limitations of our user study. First, the number of subjects involved in the user study is very limited and so it is hard to draw a statistically significant conclusion from the study. However, the class of people we consider in the study represents an important class of computer users, i.e., people involved in research and software development. In the future, we will consider other computing professionals to cover other categories of computer users. For example, people involved in the wall street business enterprises may have different distraction patterns, but the same model can be used to capture their distraction behavior. Also, we plan to conduct a larger user study in the future. Second, for obtaining ground truth, we rely on the self reporting of the subjects about when they are working and when they get distracted. Using a camera offers a more reliable way for collecting ground truth. However, using a camera in the computing environment is privacy invasive. Alternately, taking surveys is a widely accepted approach in the medical field for collecting ground truth and we find it accurate enough in the user study.

X. CONCLUSION

We believe that human-in-the-loop control can offer better response accuracy and timeliness. As a first step towards controlling energy consumption for computers, this work presents a control algorithm based on users' distractions. Our experiments reveal different distraction patterns from different users. Such observations guide us to design user-specific models to detect distractions. Each user's distraction model is learned from a combination of system level events (keyboard, mouse, etc) and gaze monitoring of that user. Different from existing approaches, the modeling and monitoring of user behavior put users into a tight control loop. Therefore, the derived energy control algorithm detects distraction early and significantly reduces energy waste when the user is distracted.

REFERENCES

[1] (2008). *U.S. DOE Buildings Energy Data Book* [Online]. Available: http://buildingsdatabook.eere.energy.gov/

[2] U.S. DOE Energy Information Administration. (2003). *Commercial Buildings Energy Consumption Survey*, Washington, DC, USA [Online]. Available: http://www.eia.doe.gov/emeu/cbecs/

[3] *ITU Gaze Tracker* [Online]. Available: http://www.gazegroup.org/

[4] Y. Agarwal, S. Savage, and R. Gupta, "Sleepserver: A software-only approach for reducing the energy consumption of PCs within enterprise environments," in *Proc. USENIX Annu. Tech. Conf.*, 2010.

[5] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl, and R. Gupta, "Somniloquy: Augmenting network interfaces to reduce PC energy usage," in *Proc. NSDI*, 2009, pp. 365–380.

[6] Z.-Y. Jin and R. Gupta, "RSSI based location-aware PC power management," in *Proc. HotPower*, 2009, pp. 1–5.

[7] R. K. Harle and A. Hopper, "The potential for location-aware power management," in *Proc. UbiComp*, 2008, pp. 1–10.

[8] C. Harris and V. Cahill, "Exploiting user behaviour for context-aware power management," in *Proc. WiMob*, Aug. 2005, pp. 122–130.

[9] C. Harris and V. Cahill, "An empirical study of the potential for context-aware power management," in *Proc. UbiComp*, 2007, pp. 235–252.

[10] X. Jiang, M. Van Ly, J. Taneja, P. Dutta, and D. Culler, "Experiences with a high-fidelity wireless building energy auditing network," in *Proc. SenSys*, 2009, pp. 113–126.

[11] *Aeon Labs. Smart Energy Plug Load* [Online]. Available: http://www.aeon-labs.com/site/products/view/5/

[12] *Aeon Labs. Z-Stick* [Online]. Available: http://www.aeon-labs.com/site/products/view/2/

[13] *Bureau of Labor Statistics*, U.S. Department of Labor, Washington, DC, USA.

[14] A. Waibel and R. Stiefelhagen, *Computers in the Human Interaction Loop* (Human-Computer Interaction Series). New York, NY, USA: Springer-Verlag, 2009.

[15] D. Tennenhouse, "Proactive computing," *Commun. ACM*, vol. 43, no. 5, pp. 43–50, 2000.

[16] P. Rani, N. Sarkar, and J. Adams, "Anxiety-based affective communication for implicit human-machine interaction," *Adv. Eng. Inf.*, vol. 21, no. 3, pp. 323–334, 2007.

[17] D. Arney, M. Pajic, J. M. Goldman, I. Lee, R. Mangharam, and O. Sokolsky, "Toward patient safety in closed-loop medical device systems," in *Proc. ICCPS*, 2010, pp. 139–148.

[18] I. Lee, O. Sokolsky, S. Chen, J. Hatcliff, E. Jee, B. Kim, A. King, M. Mullen-Fortino, S. Park, A. Roederer, and K. K. Venkatasubramanian, "Challenges and research directions in medical cyber–physical systems," *Proc. IEEE*, vol. 100, no. 1, pp. 75–90, Jan. 2012.

[19] A. D. Wood and J. A. Stankovic, "Human in the loop: Distributed data streams for immersive cyberphysical systems," *ACM SIGBED Rev.*, vol. 5, no. 1, pp. 20:1–20:2, 2008.

[20] Y. Xin, I. Baldine, J. Chase, T. Beyene, B. Parkhurst, and A. Chakrabortty, "Virtual smart grid architecture and control framework," in *Proc. IEEE Int. Smart Grid Commun.*, Oct. 2011, pp. 1–6.

[21] X. Liu, H. Ding, K. Lee, L. Sha, and M. Caccamo, "Feedback fault tolerance of real-time embedded systems: Issues and possible solutions," *ACM SIGBED Rev.*, vol. 3, no. 2, pp. 23–28, Apr. 2006.

[22] J. Lu, T. Sookoor, V. Srinivasan, G. Gao, B. Holben, J. Stankovic, E. Field, and K. Whitehouse, "The smart thermostat: Using occupancy sensors to save energy in homes," in *Proc. SenSys*, 2010, pp. 1–14.

[23] Y. Agarwal, B. Balaji, S. Dutta, R. K. Gupta, and T. Weng, "Duty-cycling buildings aggressively: The next frontier in HVAC control," in *Proc. 10th Int. Conf. IPSN*, Apr. 2011, pp. 246–257.

[24] Y. Fu, N. Kottenstette, Y. Chen, C. Lu, X. D. Koutsoukos, and H. Wang, "Feedback thermal control for real-time systems," in *Proc. 16th IEEE RTAS*, Apr. 2010, pp. 111–120.

[25] D.-J. Kim and A. Behal, "Human-in-the-loop control of an assistive robotic arm in unstructured environments for spinal cord injured users," in *Proc. 4th ACM/IEEE Int. Conf. HRI*, Mar. 2009, pp. 285–286.

[26] Y. Agarwal, T. Weng, and R. K. Gupta, "The energy dashboard: Improving the visibility of energy consumption at a campus-wide scale," in *Proc. 1st ACM BuildSys*, 2009, pp. 55–60.

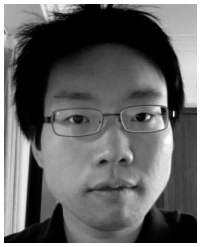
[27] G. Bellala, M. Marwah, M. Arlitt, G. Lyon, and C. E. Bash, "Towards an understanding of campus-scale power consumption," in *Proc. BuildSys*, 2011, pp. 1–6.

[28] J. Z. Kolter and J. Ferreira, "A large-scale study on predicting and contextualizing building energy usage," in *Proc. AAAI*, 2011, pp. 1340–1356.

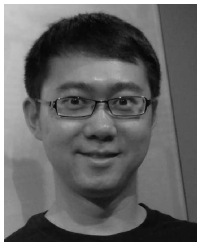
[29] A. Marchiori and Q. Han, "Distributed wireless control for building energy management," in *Proc. BuildSys*, 2010, pp. 37–42.

SIRAJUM MUNIR received the B.Sc. degree in computer science and engineering from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 2007, and the M.S. degree from the Department of Computer Science, University of Virginia (UVA), Charlottesville, VA, USA, in 2010, where he is currently pursuing the Ph.D. degree under the supervision of Prof. J. A. Stankovic. His research interest lies in the area of cyber physical systems, smart home management, and ubiquitous computing.

JOHN A. STANKOVIC is the BP America Professor with the Computer Science Department, University of Virginia, Charlottesville, VA, USA. He served as a Chair of the department for eight years. He is a fellow of the ACM. He received the IEEE Real-Time Systems Technical Committee's Award for Outstanding Technical Contributions and Leadership. He won the IEEE Technical Committee on Distributed Processing's Distinguished Achievement Award (inaugural winner). He has seven Best Paper Awards, including one for ACM SenSys in 2006. He has an h-index of 97. He has won Distinguished Faculty Awards at the University of Massachusetts and the University of Virginia. He has given more than 30 keynote talks at conferences and many Distinguished Lectures at major Universities. He was the Editor-in-Chief for the IEEE TRANSACTIONS ON DISTRIBUTED AND PARALLEL SYSTEMS and was Founder and Co-Editor-in-Chief for the *Real-Time Systems Journal*. His research interests are in real-time systems, wireless sensor networks, wireless health, and cyber physical systems. He received the Ph.D. degree from Brown University, Providence, RI, USA.



CHIEH-JAN MIKE LIANG received the Ph.D. degree from Johns Hopkins University, Baltimore, MD, USA, in 2011. He is currently a Researcher with the Mobile and Sensing Systems Group, Microsoft Research Asia, Beijing, China. His current research interests surround the system and networking aspects of sensory and mobile computing.



SHAN LIN received the Ph.D. degree in computer science from the University of Virgina, Charlottesville, VA, USA, in 2010. He joined Temple University, Philadelphia, PA, USA, as an Assistant Professor with the CIS Department. His primary funded research areas are in the areas of cyber physical systems, networked information systems, and wireless sensor networks. He has published papers in major conferences and journals.