Video-Enhanced Multi-modal Rest Detection

Arjun R. Narayanswamy Larry Rudolph
{arjunrn,rudolph}@lcs.mit.edu

M.I.T. Laboratory for Computer Science
Oxygen Research Group
200 Technology Square
Cambridge, MA 02141

Abstract

Device rest-detection is important for gesture recognition, power conservation, position triangulation and heading drift cancellation. Currently rest is principally detected using static tilt-meters. However tilt-meters are fundamentally insensitive to horizontal translation. A better rest-detection algorithm may be developed by augmenting tilt-meter readings with video data. This paper combines video optical-flow readings with traditional tilt-meter readings to obtain a better estimator of device rest conditions. The algorithm is implemented as a library and API in Familiar Linux for easy integration into applications. Stand-alone rest-detection results are presented. A simple application of rest-detection to Cricket beacon distance estimates is also presented.

1 Introduction

The position and orientation of a device in a 3 dimensional world can be modeled by a 5-tuple $< x, y, z, \theta, \phi >$ where $< x, y, z >$ gives the *location* of an object in 3-D space and $< \theta, \phi >$ gives the *orientation* of the device. When the device is in 'motion' in this three-dimensional world, one or more of x, y, z, θ and ϕ changes appreciably with time, otherwise the device is at 'rest'.[1]

The goal of this paper is to convert rest-information into an application utility. We explore different techniques of robust rest-detection and construct a system that exposes rest-information simply and cheaply to applications. We have found that rest-detection can be made robust by combining traditional tilt-meters with video-based optical flow techniques. Our system is useful for applications in the domains of power conversation, context detection, position triangulation and drift cancellation.

1.1 Power Conservation

Consider a power-constrained device running a deictic (i.e. pointing) application of some sort. In this case, detecting that the device is at rest may imply that the user intends to query or use the object at which the device is pointing. If the process of identifying and querying the object-pointed-at is expensive, then there is a very good argument for delaying such operations until we are reasonably sure that the device is at rest.

To continue this point, *power conservation* is a particularly important challenge for small mobile devices; it is often the single hardest constraint in application design. Also, most query/lookup mechanisms involve an implicit exchange of data with a networked information source- and the first hop of this exchange is made over a wireless network. Wireless exchange

[1] The word 'appreciably' has been chosen judiciously. We well see in Section 6.1 that the notion of 'rest' must be application-defined.

of data is an extremely power-hungry operation, especially for small resource-constrained mobile computers. Thus, using motion cues to reduce our use of power-hungry operations is a worthy exercise.

1.2 Context Detection

Rest-detection is helpful in identifying several gestures (user *contexts*). For example, when we point a device at an object and hold it steady, we indicate and interest in that object. Placing a device on the table, or picking it up may also have meaning. Fundamentally, context-detection applications are translating physical motion of a device into high-level user intentions. Therefore, knowing when a device is at rest is important for these applications.

1.3 Position Triangulation

Not surprisingly, being able to detect rest has applications in the field of location and heading detection. Consider location-detection mechanisms first. While an enormous amount of research has been directed at providing better location support mechanisms, a characteristic of all of these location provision mechanism is error in the location estimates. The cause of the error varies from system to system; and the error may be minimized by improving the location support hardware (often at significant cost), or by designing software algorithms to mathematically remove error. Statistical techniques may be applied either to location estimates collected from a large number of independent sources, or to readings taken from a small set of sources but over a larger interval of time. We call these approaches corroboration and averaging respectively.

Combating error through corroboration is not an easy thing to do. This approach is often circumscribed by power, cost and interference constraints. It may be too expensive to install additional beacons. The power of each beacon may not be sufficient to get sufficient overlap between them. Or it may be technically infeasible to pack beacons too close to each other. Sorting through these interconnected constraints is a tricky design issue- and hence combating error through averaging is an attractive idea.

However, there is a nagging epistemological problem with all averaging approaches. It can be illustrated by the following conundrum:

The 'Assumption of Rest' Conundrum

1. You want to know your position
2. So you assume you are at rest i.e. you assume that your position does not change
3. But how can you assume this?
   Don't you have to know your position in the first place?

In other words, we need some *a priori* method of knowing we are at rest before we can apply our position averaging algorithms. Statistical estimators such as mean, median and standard error are only informative when the underlying distributions they sample have static parameters. They do not react well when the underlying parameter being estimated (location) changes. There is therefore a powerful argument for having an out-of-band mechanism for knowing when we are at rest.

1.4 Drift Cancellation

A different argument applies in the domain of heading detection. Many heading detection systems fundamentally obtain heading information by integrating some measure of change in heading. A consequence of this integration is that error is introduced at every instance of integration and the overall estimate of heading drifts with time. Knowing when the device is stationary may enable us to attempt out-of-band methods of drift cancellation.

2 Previous Work

Three different research communities have previously addressed the problem of rest-detection. These are the mobile robotics, object tracking and human-computer interface research communities.

Determining a robot's location and heading in real-time has been called the *global positioning problem.* A number of researchers have focussed on using video sensor data to enable a robot to detect its location and orientation. The major techniques in this area are landmark-recognition techniques[14, 7], camera-configuration techniques[15], model-based approaches[4], and feature-based visual-map building systems[8]. The last approach is remarkable not only because it does not require any *a priori* map of the world. Chapter 9 of Borenstein et.al.[2] provides an excellent overview of visual positioning systems in mobile robotics.

In the area of object-tracking, a large number of algorithms and systems have been developed to accurately track a moving object via video data. For our purposes, we may divide these systems into on-line or off-line systems. Off-line systems are not useful to our goal of making rest-detection an application utility. An interesting real-time, video-based position location systems is Project HiBall[17]. Project HiBall instrumented the ceiling space of the user with a dense array of LEDs, and the position of the user was detected via a complex 'ball' that contains six infrared sensors.

Human-computer interface researchers have long appreciated the importance of determining when a device is at rest. The problem of detecting the motion and rotation of the device in 3-D space came to be called one of *context detection.* Rekimoto[11] attached tilt-meters to a Palm device. Harrison et.al.[5], Small & Ishii[13] and Bartlett[1] explored using tilt-meters to navigate. Hinckley et.al.[6] used tilt-meters coupled with a touch sensor and infrared proximity detector to identify when a hand-held device is being used as a voice recorder. The applications require significant reliability from the tilt-meter readings- in one case requiring a very specific relationship between detected x and y angles, and in another identifying tilt angles to within $\pm$ 3 degrees. The TEA project[12] extends the idea of multi-modal context detection both theoretically and practically. They develop and use a customized hardware board fusing data from 8 sensors (photo-diode, two accelerometers, IR proximity detector, temperature, pressure, CO_2 gas, and sound). They offer an insightful distinction between physical and logical sensors and use Kohonen maps to automatically detect contexts as regions of points in 8-dimensional physical sensor data space. Their device was able to use this approach to distinguish between rest, walking and placement in a briefcase.

3 Challenges

The previous work done in this area can be improved on the following grounds: (1) Complexity (2) Cost (3) Rectifying flaws in Tilt-meter Usage and (4) Adding Multi-modality.

3.1 Complexity and Cost

An approach based on extensive analysis means that the technique cannot be performed in real-time, or locally on a hand-held device. This automatically eliminates a number of mobile robotics mapping-based algorithms. Additionally, while the problem of object-tracking via video remains an area of extensive research, these systems are clearly overly complicated for our purposes. Our motivation in rest-detection is not to track objects but merely yo decide if we are at rest or not. Also, systems such as HiBall that require expensive hardware or extensive instrumentation/tagging of the user's work environment are not attractive simply because of the amount of environmental modification they require.

3.2 Tilt-Meters

While tilt-meters have been powerful in certain areas of context detection, there exist two fundamental problems with their use for the problem of rest-detection. Tilt-meter approaches are (i) not sensitive to certain kinds of motion and (ii) not designed to be be an accurate indicator of rest.

Static tilt-meters work by measuring their attitude relative to the Earth's gravity field at a given point (gravity vector for short). If the location and orientation of a device in 3-D space can be described by the 5-tuple $< x, y, z, \theta, \phi >$, then static tilt-meters

are most sensitive to change in ϕ and z. This is because change in these parameters directly affects the relationship of a tilt-meter to the gravity vector at a given point. Therefore, there are changes in x, y and θ that can be made that do not strongly affect readings from a static tilt-meter. The issue will be explored in greater detail in Section 4.2.

Static tilt-meters today are designed to provide pervasive applications with tilt information. They are constructed so that they can quickly tell an application it's general orientation with respect to the gravity vector. The fundamental question they attempt to answer is "Is the device closer to horizontal or vertical?" rather than "Has the device orientation changed in the last few seconds?". Increasing the range of orientations over which quick and accurate comparisons may be made means that tilt-meters cannot be excessively sensitive to small changes from a recent state. Such a sensitivity would distort the ability of the tilt-meter to make macro tilt comparisons.

3.3 Neglect of Multi-modality

This is not to say that tilt-meters are not useful for the problem for rest-detection. They are cheap and elegant solutions for detecting certain kinds of motion, but (as we often see in pervasive computing) they are not the complete answer to the question. We feel earlier efforts at tackling this problem have erred because they did not adequately exploit multi-modal solutions to rest-detection. As sensor-rich devices become cheaper and more widely available, there is a definite case for rectifying the flaws with current sensor approaches by combing them with data from other sensors. This is an idea we will implement in this paper.

4 A Video-Enhanced Approach

Combining video data with tilt-meter data is an approach that promises to meet the challenges raised in Section 3. In this section, we will outline the pros and cons of incorporating video data for rest-detection and then justify the construction of a combined video and tilt approach.

Mobile-robotics research has already shown that video techniques can be used to make position and heading estimations. However, much of this work has not been widely used because it has not been cheap or simple to develop a mobile, sensor-rich, video-capable platform for pervasive computing. Fortunately, this is no longer the case- commercial cell-phones produced by Samsung and Nokia now come with built-in video cameras, and video-capable PDAs are becoming widely available. For example, the research presented in this paper was conducted using a Mercury backpack[3] at the M.I.T. Lab for Computer Science. This device gives a regular off-the-shelf iPaq access to tilt-meters, video and PCMCIA extension slots. It illustrates how video data can now be considered an integral part of the sensor universe of a small, mobile computer.

Video data is high-quality, digital and gravity independent. It is sensitive to horizontal translation motions in a way static accelerometers are not. And it offers the choice of using detection and sensing algorithms that are not handicapped by prior hardware-based design decisions. All of these are attractive arguments for a video-based approach.

On the flip side, the use of video images means that a video-based rest detection algorithm will always be susceptible to background motion. It is not reassuring for our rest-decisions to be solely dependent on who walks across our camera at any given time. Additionally, video based techniques are notoriously computationally intensive - which make them hard to implement on small, mobile, resource-constrained devices.

The arguments for and against a pure video-based rest detections system are summarized in Table 1. Given this discussion, we can see that a more robust approach is to combine information from both video and tilt sensors to produce a better rest-detection system.

4.1 Statistical Measures

The video component is based on an Optical Flow algorithm which is simple, computationally inexpensive, provides a good measure of overall image drift,

Table 1: Pros and Cons of Video-Based Approaches to Two-Dimensional Rest Detection.

Pros	*Cons*
1. Sensitive to image change	1. Susceptible to back-ground motion
2. Improvable via software	2. Potentially computationally expensive
3. Independent of Earth's gravity field	

and resiliant to variations in image color, brightness and minor movement.

Consider a given frame P. Let $shift(P, h, v)$ be the same frame except with each pixel displaced horizontally by h columns and vertically by v rows. Let D be a binary difference measure that operates across *frames* then. For any two consecutive frames P_n and P_{n-1}, we define:

$$\textit{optical-flow} = < h, v > \text{ where } D(P_{n-1}, shift(P_n, h, v)) \text{ is minimum.}$$

For the purposes of rest-detection, the vertical-flow component v is more of a liability than a benefit. The information it provides is easily corrupted by scan lines seen in standard video feeds. Additionally the information it provides is redundant with pitch and tilt information obtainable from a tilt-meter. Therefore we choose to ignore the vertical flow component and instead define *MEAN-FLOW* where:

$$\textit{MEAN-FLOW} = AVERAGE(h_1, h_2...h_k) \text{ where } (h_1, h_2...h_k) \text{ is a window of horizontal flow readings from an optical-flow algorithm.}$$

For the tilt-meter component, we use a window of tilt-readings from a tilt-meter. Unlike the optical-flow measure, the mean of the window is not informative in this case because all it tells us is the current tilt of the device. We are not interested in the current tilt, but in the change in tilt. Therefore we define *TILT DEVIATION*, where:

$$\textit{TILT DEVIATION} = STD\ DEV(r_1, r_2...r_k) \text{ where } (r_1, r_2...r_k) \text{ is a window of consecutive tilt readings from a tilt-meter.}$$

We hope to build a system where we recieve new video and tilt-readings at approximately 10Hz. Therefore we arbitrarily choose a window size of 1 second so that our MEAN-FLOW and TILT-DEVIATION measures are obtained from at least 10 readings. Readings from the two measures are never compared to each other numerically, hence we have no need to normalise the two types of readings.

To guide the design of a video-based rest detector, we performed thrtee experiments with the hand held device (1) at rest in the palm of a user, (2) translating horizontally across a room and (3) staying in place but pointing to different objects in the room.

4.2 Experiments

1. Hand-held Rest

   Tthe device is at rest in the hands of the user. The user holds the device steady, as if attempting to use it as a pointing device. Over the course of this experiment, the user made tapping motions on the iPaq, as if attempting to select portions of the screen or input data. This models the device being used as a pointing device.

2. Horizontal Translation

   In this experiment the user carries the device in the palm of his hand and walks back and forth across a room. Over the course of a 50-second trial, the user walked a distance of 30 metres. This experiment models the common case when a user is using the device, but is not stationary.

3. Pitching and Rolling

   In this experiment, the device was kept in one place but randomly pointed at different objects in the room. This models the case when the device exhibits a lot of motion.

Table 2: Performance of MEAN-FLOW and TILT-DEVIATION in different rest conditions. The sliding window size is 1 second, and the sampling frequency is 15Hz.

MEAN-FLOW

	Average Reading	*Standard Deviation*
hand-held rest	0.19	0.18
horizontal translation	9.87	8.30
pitching and rolling	30.06	6.86

TILT-DEVIATION

	Average Reading	*Standard Deviation*
hand-held rest	7.57	1.38
horizontal translation	7.74	1.70
pitching and rolling	52.68	9.01

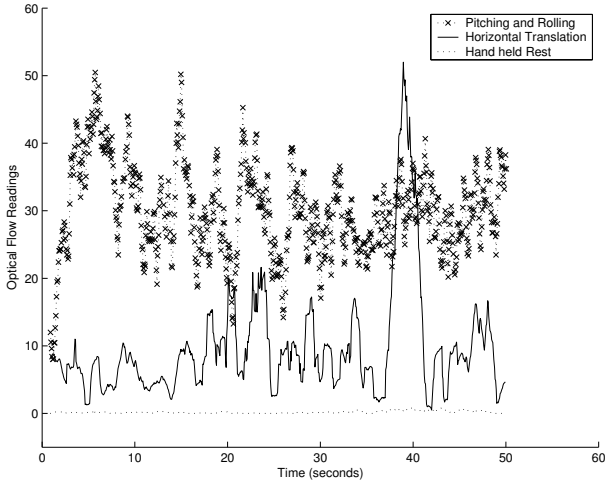

Figure 1: A 50-second trace of MEAN-FLOW with a sliding window size of 1 second and a sampling rate of 15Hz. A clear difference can be seen between readings at rest and in motion.

4.3 Observations

Figure 1 and Figure 2 compare the performance of vision and tilt techniques in each of these situations. From the graphs, we can make the following observations:

I. **The MEAN-FLOW technique makes a sharp distinction between cases at rest and in motion.**

The readings for MEAN-FLOW when the device is in motion are almost always higher than the maximum values attained for MEAN-FLOW when the device is at hand-held rest. For horizontal translation, MEAN-FLOW exhibits momentary low readings that are caused by natural moments of rest in the gait of the user. This suggests that the usage of an initial wait-period of a short duration will be required to filter out conditions of momentary rest. Table 2 provides additional statistics on these experiments.

II. **TILT-DEVIATION cannot detect horizontal translation**

From Figure 2 and Table 2, we see that is difficult to differentiate between horizontal motion and rest based on tilt-meter readings alone. This means that tilt-meters are not useful in identifying rest when the user is using the device and walking around. However, TILT-DEVIATION is effective in robustly detecting gross changes in device orientation.

5 Integration of Video and Tilt

Given our preliminary experiments, it is possible for us to define a two-sided range LOW for readings from both MEAN-FLOW and TILT-DEVIATION that corresponds to readings when the device is at

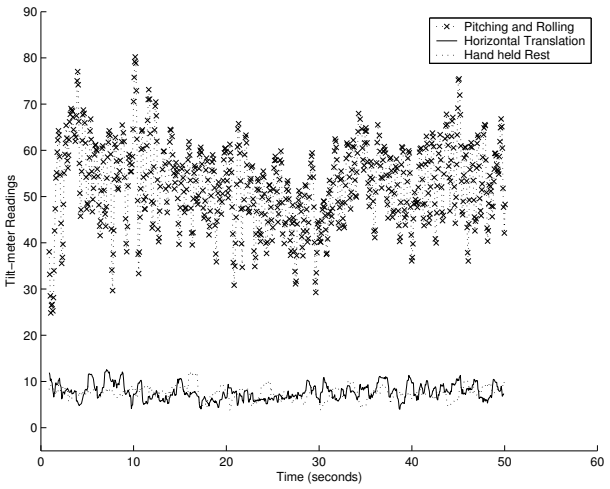

Figure 2: A 50-second trace of the output from TILT DEVIATION with a sliding window size of 1 second window and a sampling rate of 15Hz. It is not easy to differentiate between rest and horizontal translation.

hand-held rest. Given our definition of LOW, we can define a one-sided range called HIGH (more accurately, NOT-LOW) that contains all values greater than the LOW range. Therefore, at all times our algorithm is given video and tilt readings that are from {low, high} X {low, high} and asked to decide if it is at *instantaneous rest* or *instantaneous motion*. The algorithm believes itself to be at overall rest after a prolonged period of instantaneous rest.

The cases where both video and tilt data corroborate each other is easy to label. If both MEAN-FLOW and TILT-DEVIATION readings are in the low range, then it is likely that we are in a period of instantaneous rest; if both readings are in the high range, then we are are in a period of instantaneous motion. But what happens, when video and tilt readings do not agree with each other? Can we choose one over another?

5.1 Case: MEAN-FLOW low, TILT-DEVIATION high

Let us consider the case where MEAN-FLOW is low, but TILT-DEVIATION is high. Initially, we imagined that such a situation could be produced by 'hoodwinking' the video by presenting it with a dark or featureless surface, However, we found that the design of the optical flow algorithm is such that the output of MEAN-FLOW in this case is not zero but random. Intuitively this means that the algorithm cannot reliably pick one value of flow over another, and hence becomes sensitive to very minute differences in image quality. Therefore, switching off the lights or blocking the camera lens does not produce low values for MEAN-FLOW. Similarly, we found that even apparently featureless surfaces such as blank whiteboards contained enough variation in pixel values for the algorithm to detect motion.

In fact, the only way we were able to fool MEAN-FLOW into making a false positive error for rest detection was to affix a small highly-detailed image (in our case, a circuit board) a short distance in front of the camera lens. The effort required to fool the camera in this way suggests that this situation is unlikely to occur often. However, if we did face the case MEAN-FLOW low, TILT-DEVIATION high, are we to trust the video data and assume rest, or are we to trust the tilt data and assume motion?

We observe that, unlike a camera, it is hard to artificially force the tilt-meter to generate high readings without actually moving the device. While the a low reading from TILT-DEVIATION may be ambiguous, a high reading from TILT-DEVIATION generally means that the device is undergoing acceleration (and hence motion). Hence the correct decision in this situation is to trust the tilt-meters and assume that we are in instantaneous motion.

5.2 Case: MEAN-FLOW high, TILT-DEVIATION low

What happens when MEAN-FLOW readings are high, but TILT-DEVIATION readings are low? Are we at rest, or are we in motion?

Horizontal translation is an example of an instan-

taneous motion situation where TILT-DEVIATION readings are low but MEAN-FLOW readings are high. Since horizontal motion models the case where the user is walking and using the device, we cannot assume that this is an unlikely situation. Thefore this set of readings may mean that we are in motion.

However, MEAN-FLOW readings may also go high due to momentary occlusions by parts of the user's body; or by large moving portions of the background. One may attempt to filter out background motion by using more complicated techniques than optical-flow, but at the most fundamental level this is a decision that is undecidable visually. Video is also sensitive to background noise. Therefore this set of readings may also mean that we are at rest.

So unlike the MEAN-FLOW low, TILT-DEVIATION high case, we cannot argue for trusting one device over another. How do we resolve this situation? Essentially, our system can either decide that the device is at rest (and run the risk of making a *false positive* error) or decide that the device is in motion (and thereby risk a *false negative* error).

Looking at the examples given in Section 1, it can be seen that the cost associated with a false positive error is greater. Erroneously deciding that the device is at rest could lead to unnecessary usage of expensive resources, wildly inaccurate position estimations, and failed calibrations. In contrast, assuming that we are in motion means that higher level applications are forced to make looser assumptions about their context. Hence it is preferrable to err on the side of detecting too much motion rather than detecting too much rest. Hence, the system in this situation should decide that it is in instantaneous motion.[2]

The complete integration of video MEAN-FLOW and tilt-meter TILT-DEVIATION readings is therefore as given in Table 3. The text in bold tells us whether our algorithm interpreted the event as instantaneous rest or instantaneous motion.

[2]Our system could also potentially ignore this conflict between video and tilt and do nothing at all. However the argument for conservative rest detection means that we are safer believing ourselves to be in motion.

5.3 Algorithm

Let $< m, t >$ be instantaneous readings of the *MEAN FLOW* and *TILT DEVIATION* metrics. Given these, we can at every instant of time determine if our *instantaneous-state* is instantaneous-rest or instantaneous-motion. Our rest-detection algorithm is a simple state machine that exists on one of three states - *MOTION*, *WAIT* and *REST*. We assume that the algorithm is initially in the *MOTION* state, but transitions to the *REST* state after being in instaneous rest for an application-specified period of time. The algorithm exits the *REST* state when it encounters an instance of instantaneous motion. We maintain a state variable called *rest-counter* that counts the number of at-rest intervals the device has experienced since rebooting. Applications that poll this system can use the current state of the rest-algorithm and the value of the rest-counter to decide if they have been at rest since the last poll.

The algorithm is outlined in Figure 3.

Figure 3: Algorithm

```
rest-counter = 0
SWITCH instantaneous-state
        CASE instantaneous-motion:
                state = MOTION
        CASE instantaneous-rest:
        SWITCH state
                CASE MOTION:
                        state = WAIT
                CASE WAIT:
                        IF waited< init. wait period
                                state = WAIT
                        ELSE
                                state = REST
                                rest-counter++
                        ENDIF
                CASE REST:
                        state = REST
        END
END
```

Table 3: Integration of Video and Tilt Readings. Given tilt (TILT-DEVIATION) and video (MEAN-FLOW) readings, this table tells us whether our algorithm treats the situation as instantaneous rest or instantaneous motion. The algorithm transitions into overall rest if and only if it is in instananeous rest for an application-specified period of time.

		MEAN-FLOW	
		low	high
TILT-DEV.	low	**REST**	**MOTION** *(Conservative rest detection)*
	high	**MOTION** *(Video is being 'hoodwinked')*	**MOTION**

6 Implementation

We intend for the implementation of our rest-detection system to meet the following high-level goals:

6.1 Goals

- Application Programmability

  The notion of 'rest' varies from application to application. An energy conservation application would consider itself at rest when unused for a few seconds. A screensaver application may consider itself at rest only if unused for a few minutes. Hence it is important that applications be important to define what "rest' means to them.

- Ease of Development

  Our aim is that the notions of rest and rest-detection be simply and clearly defined, and that applications be written easily against an rest-detection mechanism. Hence it is important the rest-detection mechanism be simply and easily integrated into applications.

- Local Code

  On the hand-held computers of today power, CPU and memory are scarce resources. Many visual algorithms are resource hungry. Extensive computation may also not be shipped to a remote server because wireless network access is a power-hungry operation. Hence it is important that our visual rest-detection system be small, local and computationally inexpensive.

- Push/Pull Mechanism

  In certain applications, it may suffice to query a system variable to determine if the device is at rest. Other applications may require intimation when a device is at rest. Therefore our rest-detection mechanism should support both polling and call-back mechanisms.

The rest-detection system is written in C, with the video devices being accessed via a Video4Linux API. The test hardware platform is an iPaq 3650 running Familiar Linux over an ARM processor. The system defines a single object file with a simple header file *rest.h* that defines the API. The API consists of a struct called a RestDetector and simple functions that allow an application to control sampling rates, define rest-detection parameters and utilise the call-backs and thread-safe polling mechanisms provided by the system. The principal component of the system is a daemon that reads video and tilt data from the underlying Linux file-system and executes the algorithms discussed earlier in this section. We have been able to run this daemon at speeds of upto 15Hz on our hardware platform.

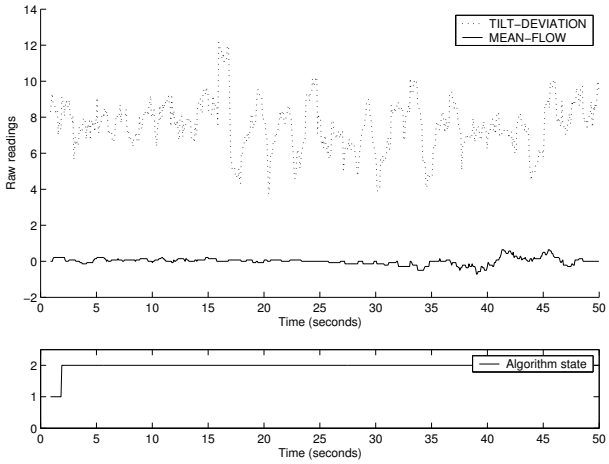

Figure 4: A 50 second trace of the performance of the multi-modal algorithm when the device is held in the palm of a user. The inital wait period is set at 1 second. The sliding window size and the sampling frequency for MEAN-FLOW and TILT-DEVIATION are set at 1 second and 15Hz respectively.

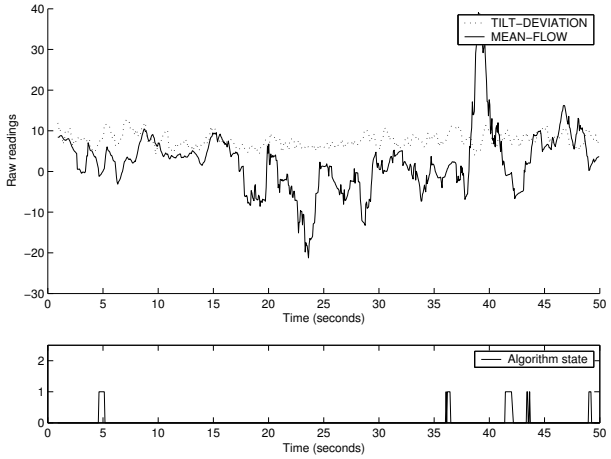

Figure 5: A 50 second trace of the performance of the multi-modal algorithm when the device is held in the palm of a user walking about a room. The user walked a distance of 30m over the course of this trace. The inital wait period is set at 1 second. The sliding window size and the sampling frequency for MEAN-FLOW and TILT-DEVIATION are set at 1 second and 15Hz respectively.

7 Results

Figure 4 and Figure 5 model the performance of our rest-detection algorithm on the sensor traces introduced in Section 4.2. The lower graph plots the state of the rest-detection algorithm, which may be 0 - *MOTION*, 1 - *WAIT* or 2 - *REST*.

The performance of the rest-detection algorithm when the device is at rest in the palm of the user is stellar (Figure 4). Both MEAN-FLOW and TILT-DEVIATION readings are always in the low range. Hence the algorithm waits for the inital period of 1 second, and then transitions into REST state. The algorithm is resiliant to jitter caused by the user's hands and the tapping motion of the stylus.

When looking at the performance of the algorithm in the case of horizontal translation (Figure 5), it becomes clear that the size of the initial wait-period is important. There are several instances over the course of the user's walk when he is momentarily at rest- these show up in our algorithm state trace as periods when the algorithm is in the *WAIT* state. The application controls which of these periods of instantaneous rest count as periods of overall algorithm *REST*. In this experiment we see that a one-second wait period means that the algorithm never settles into *REST* state.

The performance of the multi-modal algorithm when the device is being pointed at different objects in the room (Figure 6) is also stellar. Both MEAN-FLOW and TILT-DEVIATION readings are high. Consequently the algorithm does not experience even a single moment of instantaneous rest, and the overall state of the algorithm remains firmly pegged in the MOTION state.

8 Application: Rest-Aware Position Triangulation

As outlined in Section 1, a standard approach to position triangulation is to assume that the device is at

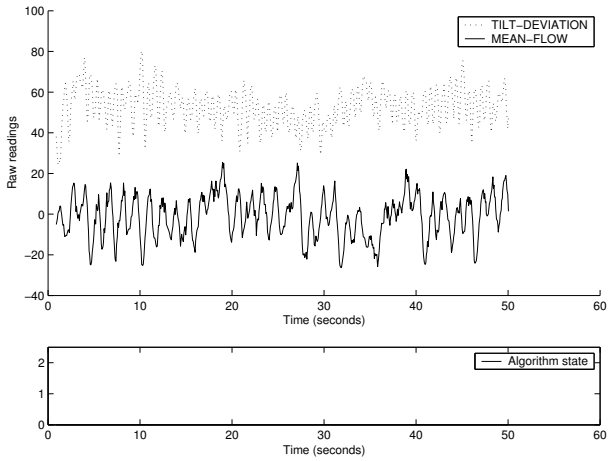

Figure 6: A 50 second trace of the performance of the multi-modal algorithm when the device is being pointed around the room (extreme motion). The inital wait period is set at 1 second. The sliding window size and the sampling frequency for MEAN-FLOW and TILT-DEVIATION are set at 1 second and 15Hz respectively. The algorithm never deviates from the MOTION state.

rest, collect distance-estimates from multiple beacons and then triangulate the position of the device in 3-D space. Popular systems that implement such triangulation algorithms are Active Bat[16] and Cricket[10]. However, there are a couple of problems with this approach.

Firstly, there currently exists no reliable way of knowing when we are actually at rest. In current implementations, the user manually signals to the position triangulation mechanism that she intends to be stationary for the next few seconds. She does this in order to give the triangulation algorithm time to collect the distance readings it needs. The algorithm has no way of ensuring that this contract is met, and therefore cannot decide if the readings it collects are corrupted by user motion.

Secondly, the distance estimates collected by the device exhibit error. To mitigate the overall positioning error, positioning algorithms can either resort to collecting multiple samples from each beacon, or to collecting individual samples from multiple beacons. The latter approach involves designing our system with a greater beacon density; since this is technically and economically challenging, we would like to avoid this approach if possible. However, the cost of using approaches that collect multiple readings from each beacon is application latency- we have to wait longer to collect the readings that we need. Since latency determines the end usability of a positioning algorithm, location support systems today are designed to bound latency and only then minimize cost.

Continuing, the standard approach to bounding latency is to *a priori* choose a triangulation window of a fixed, static size so that worst-case latency is bounded. However this approach is suboptimal because it ignores the observation that users of a pervasive devices are at rest often, and for varying periods of time. Using a fixed window size means that we impose high-level behavioral restrictions on the user. We also lose access to optimizations that we might have been able to perform had we chosen a larger triangulation window. Both of these are expensive failings.

Therefore, the rest-detection system that we have developed will allow a location-support system to (1) validate that all distance measurements are taken at rest and (2) dynamically change the triangulation window to exploit optimizations enabled by unusually long periods of rest. We will demonstrate both of these advantages using the Cricket Location Support System as a case-study. Our hardware platform is a sensor-rich Mercury backpack connected to a Cricket listener.

8.1 Case Study: Crickets and CricketNav

In "Design and Implementation of an Indoor Mobile Navigation System" [9], Miu conducted a comprehensive investigation into the Cricket beacon distance estimates and the accuracy of 2-D position triangulation using Crickets. He defined the *sample frequency* (k) of a position estimate to be the number of distance estimates collected from each beacon. He also defined the *beacon multiplicity* (m) of a position estimate to be the number of distinct beacons from which

distance estimates are collected. Over the course of their investigation, he found that:

1. A MODE distance estimate produces more accurate readings than a MEAN estimate
2. MODE does not being to take effect until $k > 5$
3. A least-squares method of position triangulation effectively reduces error, especially when k is large
4. For $k < 5$ and $m \geq 5$ it is better to assume that the speed of sound is unknown. Otherwise it is better to assume that the speed of sound is known
5. For $k = 1$, Cricket is accurate to within 30cm, 95% of the time.

After conducting this investigation into the positioning accuracy of Cricket, the author then used the Cricket positioning information to construct an indoor mobile navigation system called CricketNav. While designing this system, author ran up against a tough application constraint - latency. He found that in order for their navigation application to respond promptly to the user, the triangulation window had to be bounded to be less than 5 seconds. Since increasing k increases the latency of the application, this meant that the CricketNav application could only use position estimates with $k = 1$ and $m = 3, 4, 5$ etc. An entire body of optimizations [bullets 1 - 4 of the 5 bullets presented above] was rendered inaccessible to the CricketNav application. How could this have been avoided?

8.2 A Simple Rest-Aware Cricket Distance Estimator

Using the RestDetector designed and built in Section 6.1, we constructed a simple application that tracks the distance to a beacon. The application is rest-aware in the sense that the sample frequency k increases when the application discovers itself to be at rest. As in CricketNav, the initial value of k is 1.

Upon receipt of a new reading from the beacon, we check our rest-detector to ensure that we have been at rest since the receipt of the last reading. If so, we add the reading to our buffer and increment k by 1. If not, we flush the buffer, add the new reading and set $k = 1$.

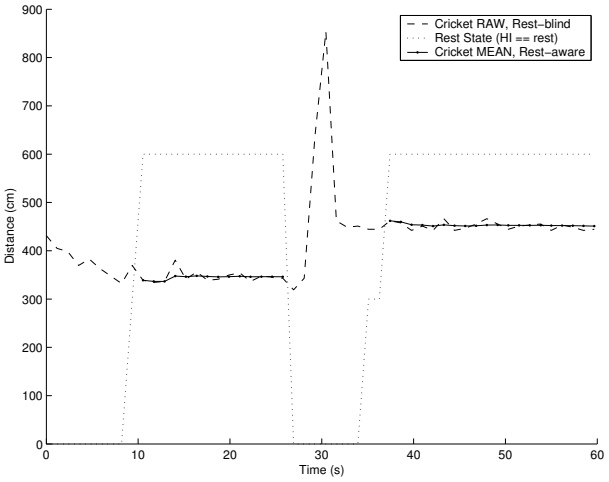

Figure 7: A 60-second trace of the distance readings from one Cricket beacon as the user moved from one desktop to another. The spike in the distance readings as the user moves is likely due to a reflected ultrasound pulse.

Figure 7 plots the distance estimates that we receive for the simple case of the user walking from one desktop to another. The graph labeled 'Cricket RAW, Rest-blind' are the current distance estimates that CricketNav would use from the beacon. The graph labeled 'Rest State' gives us the state of the Rest-Detector - high implies we are at rest, low implies we are in motion.

The graph labeled 'Cricket MEAN, Rest-aware' is the the smoothed distance estimates obtained by dynamically incrementing k at rest. Only the rest-portions of this graph are shown; when the device is in motion the readings from this technique are identical to 'Cricket RAW, Rest-blind'. From looking at Figure 7, we can make the following observations:

I. We do not have to impose end-user mobility restrictions

We do not require that applications enter into a 'no-motion-for-n-seconds' contract with the end

user. Alternatively, if such a contract is necessary, we can now check to see if the contract was met. This allows our applications to adapt to user behaviour, and not vice versa.

II. Our distance readings may be guaranteed to belong to the same physical location

Having an out-of-band rest detection mechanism means that we have some fundamental guarantees about the quality of our distance readings. We know when it makes sense to use statistical estimation mechanisms (when we are at rest), and when it does not make sense to do so (when we are in motion).

III. The latency of the application is not degraded

When the distance estimation algorithm detects itself to have moved, it quickly drops the size of the readings buffer to the default setting. Therefore, the worst case latency of the rest-aware averaging mechanism is no worse than that obtained at $k = 1$. Additionally, the theoretical latency of the algorithm when we are at rest is *zero*. This is because knowing that we are at rest means that we do not need to wait for additional readings - we can just re-use the last computed estimate.

IV. Extra long sequences of readings enable the deployment of more sophisticated optimizations

As outlined in Section 8.1, there are a number of optimizations that we can perform given a large number of readings from a single beacon. The MODE distance metric may now be costlessly deployed. An exponentially-weighted moving average may be developed to reduce the jitter of the distance readings during motion. Additional techniques may also be developed and implemented.

9 Conclusion and Future Work

There is a need for a system that makes rest-information simply and cheaply available to higher level pervasive applications. In this paper we have constructed a rest-detection system that combines video data with traditional tilt-meter data for greater robustness. We have found that this algorithm is an improvement over traditional tilt-meter based approaches, and we have outlined a potential use for this technology in the domain of position triangulation.

Future work for this project would involve the development of more complicated algorithms that we can call upon in cases where video and tilt-data donot agree with each other. However such algorithms must stay computationally inexpensive to fulfill the goal of being an application utility. Secondly, the window size for our MEAN-FLOW and TILT-DEVIATION measures is arbitrarily set at 1 second. This may be too short a window at low sampling rates. The 'high' and 'low' mark for each measure may also be configured dynamically, rather than hardcoded as it is now. We can imagine a short user configuration routine that sets the values of these parameters dynamically.

Additionally, there is a need for better access to the Linux device file-descriptors. While the Video4Linux API is well-defined, the API for accessing the tilt-meter is obscure and platform-specific. The rest library also does not gracefully share the file descriptors; current applications that use the video or the tilt-meter must be rewritten to access that data. An implementation that just snoops on the device files without locking them would be a better implementation.

References

[1] Joel F. Bartlett. Rock 'n' scroll is here to stay. *IEEE Computer Graphics and Applications*, 20(3):40–45, May/June 2000.

[2] J. Borenstein, H.R. Everett, and L.Feng. *Navigating Mobile Robots: Systems and Techniques.* A.K. Peters, Wellesley, Mass., 1996. This is available as a 'Where am I' report from the University of Michigan.

[3] Compaq Cambridge Research Laboratories, Cambridge, MA. *Project Mercury.*

[4] C. Fennema, A. Hanson, E. Riseman, J. R. Beveride, and R. Kumar. Model-directed mobile robot navigation. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(6):1352–1369, November-December 1990.

[5] Beverly L. Harrison, Kenneth P. Fishkin, Anuj Gujar, Carlos Mochon, and Roy Want. Squeeze me, Hold me, Tilt me ! An exploration of manipulative user interfaces. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI '98) : Making the Impossible Possible*, pages 17–24, New York, April 18–23 1998. ACM Press.

[6] Ken Hinckley, Jeffrey S. Pierce, Mike Sinclair, and Eric Horvitz. Sensing techniques for mobile interaction. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '00)*, pages 91–100, New York, 2000. ACM Press.

[7] Eric Krotkov. Mobile robot localization using A single image. In *Proceedings 1989 IEEE International Conference on Robotics and Automation*, pages 978–983. IEEE, 1989.

[8] Larry Matthies and Steven A. Shafer. Error modeling in stereo navigation. *IEEE Journal of Robotics and Automation*, RA-3(3):239–248, June 1987.

[9] Allen Miu. Design and Implementation of an Indoor Mobile Navigation System. M.S. thesis, Massachusetts Institute of Technology, Electrical Engineering and Computer Science Department, Jan 2002.

[10] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The Cricket Location-Support System. In *Proceedings of the 6th Annual ACM International Conference on Mobile Computing and Networking (MobiCom '00)*, pages 32–43, New York, August 6–11 2000. ACM Press.

[11] Jun Rekimoto. Tilting operations for small screen interfaces. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '96)*, Papers: Interaction Techniques (TechNote), pages 167–168, 1996.

[12] A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. Van Laerhoven, and W. Van de Velde. Advanced interaction in context. *Lecture Notes in Computer Science*, 1707:89–??, 1999.

[13] David Small and Hiroshi Ishii. Design of spatially aware graspable displays. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI '97)*, volume 2 of *SHORT TALKS: Devices*, pages 367–368, 1997.

[14] K. Sugihara. Some location properties for robot navigation using a single camera. *Computer Vision, Graphics and Image Processing*, 42:112–129, 1988.

[15] R. Y. Tsai. An efficient and accurate camera calibration technique for 3D machine vision. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, (CVPR '86)*, IEEE Publ.86CH2290-5, pages 364–374. IEEE, June 22–26 1986.

[16] Andy Ward, Alan Jones, and Andy Hopper. A new location technique for the active office. *IEEE Personal Communications*, 4(5):42–47, October 1997.

[17] Greg Welch, Gary Bishop, Leandra Vicci, Stephen Brumback, Kurtis Keller, and D'nardo Colucci. The HiBall tracker: High-performance wide-area tracking for virtual and augmented environments. In Mel Slater, editor, *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST '99)*, pages 1–10, New York, December 20–22 2000. ACM Press.