Multi-Level Sensory Interpretation and Adaptation in a Mobile Cube

Kristof Van Laerhoven, Nicolas Villar and Hans-Werner Gellersen
Department of Computing
Lancaster University
LA14YR Lancaster, United Kingdom
{kristof, villar, hwg}@comp.lancs.ac.uk

Abstract. Signals from sensors are often analyzed in a sequence of steps, starting with the raw sensor data and eventually ending up with a classification or abstraction of these data. This paper will give a practical example of how the same information can be trained and used to initiate multiple interpretations of the same data on different, application-oriented levels. Crucially, the focus is on expanding embedded analysis software, rather than adding more powerful, but possibly resource-hungry, sensors. Our illustration of this approach involves a tangible input device the shape of a cube that relies exclusively on low-cost accelerometers. The cube supports calibration with user supervision, it can tell which of its sides is on top, give an estimate of its orientation relative to the user, and recognize basic gestures.

INTRODUCTION

In short, our hypothesis can be summarized as: simple sensors combined with machine learning and modeling techniques can pose an improved alternative to a better, but more demanding sensor. This is achieved by taking the same data and interpreting it in multiple ways.

Focusing on Microcontroller-Level Algorithms

Computing components and sensors are becoming increasingly smaller, the power they expect decreases as well, and so does the cost. Because of these factors, designers of small mobile or wearable devices tend to spend a lot of their efforts focusing on the hardware components, 'solving' their problems in hardware. The capabilities of microcontrollers are increasing equally fast, however, up to the point where it has become possible to integrate rather powerful machine learning algorithms and artificial intelligence methods.

Batteries have improved too, but with the rise of wireless devices, power problems still remain challenging, especially in mobile devices. Also other components, like displays for instance, have evolved a great deal over the past years, but they still need a significant amount of power. Both power issues and size constraints lead in our view to realistic incentives for paying more attention to the processing part in a design, and the possibility of starting with implementing more powerful algorithms at microcontroller level.

Potential benefits of this approach include a reduced need for taxing hardware components, such as replacing expensive sensors with simpler ones plus adequate analysis algorithms.

Exploiting Ambiguity in Sensor Signals

Sensors provide de facto ambiguous information in multiple ways; research in sensor fusion and machine learning defines these as:

- *Registration Problem*: what source they observe (e.g., if two cameras in the same room spot a red ball, is it the same object)?
- *Cocktail Party Problem:* multiple sources are being picked up by a single sensor [06] (e.g., focusing on one person when multiple people are talking).
- *Interference and noise:* various causes can distort the signal that is being picked up by the sensor(s), ranging from other sources to imperfect sensor design.

More specific, this paper will concentrate on accelerometers, sensors that measure acceleration relative to the earth's gravitational field. Because of their specific implementation, they measure both acceleration *and* orientation, rooted in a single output signal. This property makes them very suitable for this study.

Normally, applications that use the accelerometer will filter out one of the two, ending up with true acceleration *or* orientation. Here however, our aim is to exploit the nature of this sensor and analyze both sources in the signal.

Application in Mobile Input Devices

The application that we use to illustrate this concept is situated in the area of tangible, mobile interfaces: A small cube, fitted with a low-power microcontroller, a wireless radio transceiver, and

acceleration sensors is used for basic input and navigation [04].

Selection is made by putting the cube with a pre-defined side on top, basic navigation is achieved by rotating the cube left, right, up or down, while additional 'cube-gestures' such as shaking, twisting and knocking the cube enable alternative selection.

Crucial properties that were defined during the design phase of the cube were:

- It is expected to work on the same batteries for long periods (in the order of months).
- No wires are allowed, i.e. it needs to be self-sufficient for power and communication.
- It needs to be accurate enough to enable basic interaction in ubiquitous environments.
- It needs to be able to survive long-term deployment in real-world.

Most of these requirements were directly responsible for rejecting otherwise obvious sensors such as gyroscopes or beacon-based approaches. Implementing the cube with gyroscopes, for instance, would add 120 USD to the cost and about 18 mA (while the accelerometer setup only costs 10 USD, 0.8 mA).

THE HARDWARE

As mentioned in the introduction, the cube as an object has to remain small and robust enough for the users to handle it, and its operation needs to be accurate and autonomous so it can work accurately for long periods without requiring cabling for power and communication.

The hardware is built around a Microchip PIC microprocessor (PIC18F252), which is small, fast (10 MIPS), consumes little energy (25 µA or 0.2 µA in standby), and is easy to interface to the sensors and communication module. The downside of choosing a microcontroller is that the software for processing the sensor data and broadcasting it via a wireless protocol has to fit in a tiny program memory (32Kbyte) and only has access to a small amount of data memory (1536 bytes).

The PIC 18F252 has fourteen inputs for binary sensors and an analog-to-digital conversion unit that allows five analog sensors to be attached. Our objective, however, to keep the peripheral hardware as simple and low-cost as possible without giving in too much on performance, means that we used only simple sensors:

- Two dual-axis accelerometers (ADXL311, [01]) measure both dynamic acceleration (e.g., vibration) and static acceleration (e.g., gravity) in a plane. The sensors' ability to measure gravity gives us the opportunity to discriminate in contexts where acceleration may be zero (such as different positions of the cube). We used two accelerometers to get acceleration in three dimensions (X-Y and X-Z).
- A set of QT110 capacitive proximity sensors to check whether the user is holding the cube, mainly designed to wake up the microcontroller from its standby mode.

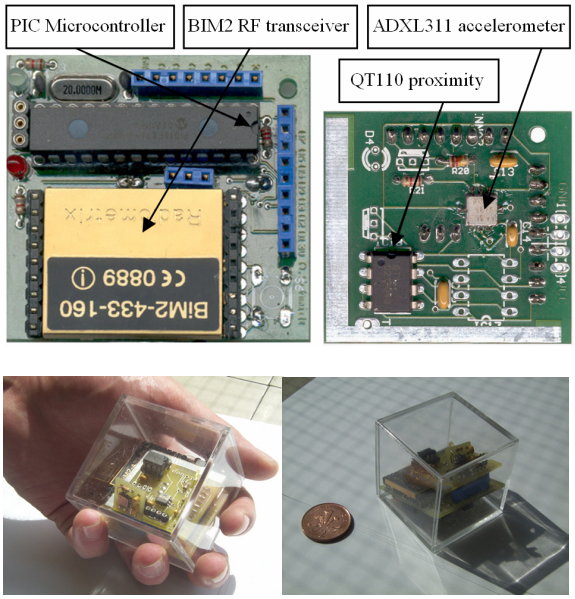

Figure 1. The communications board (top left), the sensor board (top right), and the hardware in a transparent case (bottom).

The communications module is a Radiometrix BIM2 chip that transmits and receives data wirelessly (via FM) over an approximate range of fifty meters indoors. Its relatively low power consumption for an RF transceiver (~8 mA) and considerable data rate (64 kbps) make it an ideal interface between the cube and its surrounding applications.

The hardware used in this paper consists of two boards (as shown in Figure 1) that stack on top of each other: One board makes up the core section, containing the microcontroller, communications module, and a coin-size Lithium cell on the bottom. The second board has the accelerometers and proximity sensor, plus a few empty slots for future sensors, should they be required. The total setup for one cube, including the printed circuit board and all components, costs about 50 USD.

Similar to wireless keyboards and mice, the cube will only output information about itself to its environment when its internal state has changed or when a gesture has been performed, thus preserving the batteries as much as possible. The entire system supports a three volt Lithium coin-cell battery, or two AAA batteries. The latter were used in our experiments and give a lifetime of approximately four months with the current embedded software and normal usage.

THE ALGORITHMS

Three kinds of analysis are implemented as algorithms in the cube: (1) A Gaussian Model for the sensor data for each of the six sides pointing upwards, (2) A Finite State Machine modeled from symmetric properties of the cube, and (3) A simple classifier for gesture recognition (see Figure 2 for an overview). The first and last algorithms are furthermore re-trainable, while the second is derived straight from the symmetric properties from the cube.

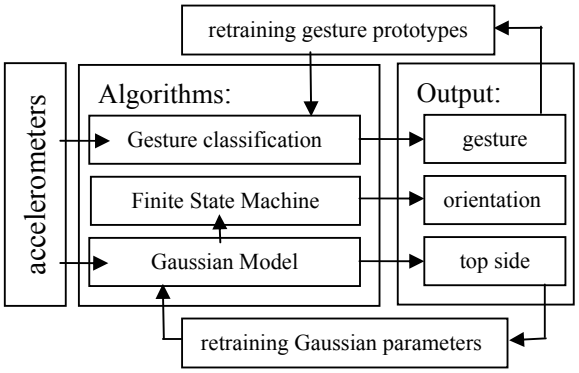

Figure 2. The static acceleration component in the signal provides the input for estimating which of the cube's sides is on top, while the dynamic acceleration component is used for distinguishing gestures.

Multivariate Gaussians

Gaussians in high-dimensional spaces are a common way to model a certain class of sensor data. In [04], it was shown that they are also low-cost and practical to implement on microcontroller-based platforms if implemented as a maximum likelihood estimation.

For the specific case of the cube, six classes need modeling: one for each side being the one that is on top, similar to a die. [08] gives a step-by-step account of how to get to a concise formula to calculate the most likely class, straight from the sensor data:

$$\arg\min_i \left((\bar{x} - \bar{\mu}_i)^T \Sigma_i^{-1} (\bar{x} - \bar{\mu}_i) + \ln|\Sigma_i| \right)$$

where *i* varies from 1 to 6 for each of the sides, x is the vector with the sensor data, and μ_i and Σ_i the average vectors and covariance matrices for side i. In the cube's case, four-dimensional vectors were used (one for each accelerometer, Figure 3 shows typical data in a time-series plot for each of the six sides of the cube being on top), which makes it near-impossible to visualize the Gaussians. Using dimension reduction, however, one can see that the six are, as expected, well-distributed over the input space (see Figure 4).

The retraining of the Gaussian parameters is slightly more complex to implement than the maximum likelihood formula above, mainly because the latter has the inverse and logarithm naturalis of the determinant of the covariance matrices stored as constants. It is exactly these that have to be re-calculated in the re-training phase (see Tables 1 and 2 for the source code).

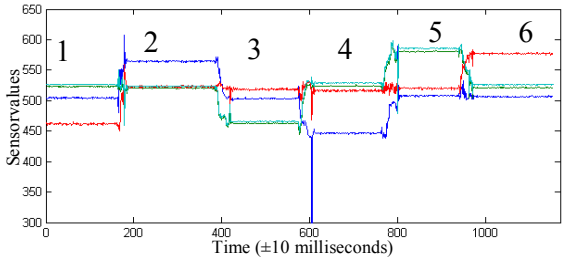

Figure 3. Time series from the cube's sensors, while placed on all six sides successively (labeled in the plot).

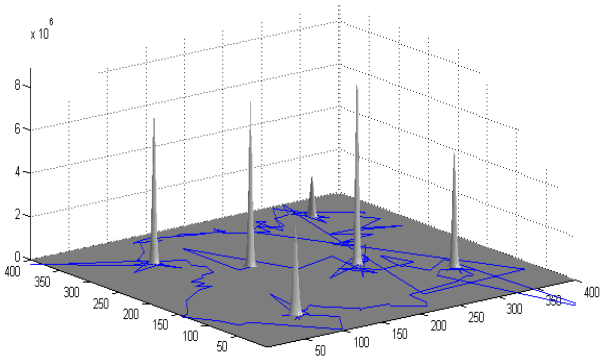

Figure 4. Visualization of the Gaussians for all six sides. The line trace on the bottom is a trace plot of the readings while the cube was turned around from side to side (as in Figure 3).

```
// calculate temporary means
    mx1 = x1 - mean[k][0];    mx2 = x2 - mean[k][1];
    mx3 = x3 - mean[k][2];    mx4 = x4 - mean[k][3];
// calculate 4d Mahalanobis:
    mah1  = mx1*inv11[k]; mah1 += mx2*inv12[k];
    mah1 += mx3*inv13[k]; mah1 += mx4*inv14[k];
    mah1*=mx1;   mah2=mx1*inv12[k]+mx2*inv22[k];
    mah2 += mx3*inv23[k]+mx4*inv24[k];
    mah2 *= mx2;  mah3  = mx1*inv13[k]+mx2*inv23[k];
    mah3 += mx3*inv33[k]+mx4*inv34[k];
    mah3 *= mx3;  mah4  = mx1*inv14[k]+mx2*inv24[k];
    mah4 += mx3*inv34[k]+mx4*inv44[k];
    mah4 *= mx4;
// add ln(det):
    res = mah1 + mah2 + mah3 + mah4 + ln[k];
 // get the minarg:
    if (res < prev_res) {
                prev_res = res;
                temp_min = k + 1;
    }
```

Table 1. Source code for finding the most likely side up.

With this first algorithm in place, it is not only possible for the cube to estimate on which side it has been placed, but also to retrain itself with help from the user.

At the moment, re-training is done by sending an explicit command to the cube, after which the cube will prompt the user six times to place it on a side. One can imagine however, that the triggering of the re-train function might be implemented with gestures, which would not require a 'debug' tool.

```
// calculate means and covariance matrix
    mean[current_class][j] = 0;
    for(j=0; j<4; j++){
      for(k=j; k<4; k++) cov[current_class][j][k] = 0;
      for(i=0; i<MAX_SAMPLES; i++){
              mean[current_class][j] += samples[i][j];
              for(k=j; k<4; k++)
              cov[current_class][j][k] += samples[i][j]*samples[i][k];
      }
      for(k=j; k<4; k++) {
        cov[current_class][j][k] /= MAX_SAMPLES;
        cov[current_class][j][k]+=((mean[current_class][j]
          *mean[current_class][j])/MAX_SAMPLES);
      }
      mean[current_class][j]=(mean[current_class][j]
           /MAX_SAMPLES);
    }
// calculate the determinant of the covariance matrix:
    calc_det(current_class);
// calculate the ln of the determinant (using Borchardt's):
    ln[current_class] = 6*(detcov-1)/(detcov+1+4*sqrt(detcov));
// calculate the adjoint (=transpose of cofactor matrix):
    calc_adj(current_class);
// calculate the inverse of the covariance:
    calc_inv(current_class);
```

Table 2. Main source code structure for retraining the maximum likelihood parameters.

Modeling the cube's state

The cube is known in mathematics as a hexahedron, from the group of polyhedrons, or more precisely isohedrons, consisting of six square sides, 8 vertices, 12 edges, and having numerous interesting properties (see [09] for a concise introduction). In this paper we have the additional properties of a die: the cube has opposite faces, which are labeled by number to always sum to seven. This gives two possible mirror image arrangements in which the numbers 1, 2, and 3 may be arranged in a clockwise or counterclockwise order about a corner; we chose the clockwise arrangement.

We define the state of our cube by two sides: the top side (the side of the cube that faces upwards) and the south side (the side of the cube that points towards *the user*). Figure 5a shows a sketch of a state, with the other sides marked as well. Using the cube's symmetry, *knowing these two sides is sufficient to know the orientation of all other sides*.

We define the transition from one state to another as: left, right, forward, or back by turning the cube left, right, away from the user, or towards the user. Figure 5b shows a sketch with these 4 transitions in the form of rotational arrows.

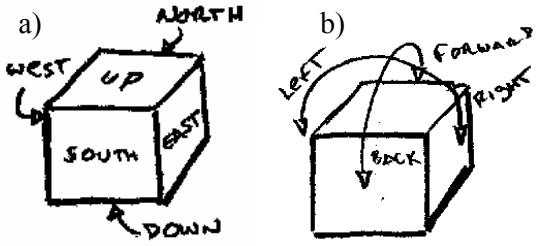

Figure 5. Illustrations for the definitions of the state (left) and the transitions (right) of a cube.

With these definitions in place, we can model a finite state machine, as depicted in Figure 6. The most important property of this model is that we need to know only *the previous state and the current top state* to detect which transition was made and which is the current state. This allows us to keep track of the state at all times by just searching for the most likely top side, and by starting with a known state!

Table 3 shows how the finite state machine was actually implemented in the microcontroller, using two look-up tables: one for linking states to a *(top side, south side)* tupel, and another one for storing the actual transitions.

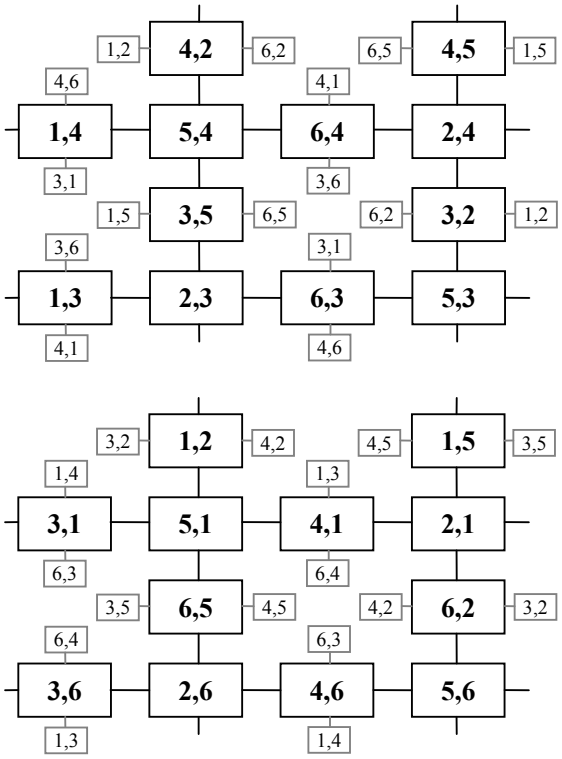

Figure 6. All possible state transitions in two interlinked graphs: Links between a state (rectangles) to the left, right, up, and down another state indicate a transition after rotating the cube to the left, right, forward, and backward respectively. A cube's state X,Y is characterized by the top side X, and the south side Y. The smaller states specify links between the two graphs. Open links should be linked to the state on the other side of the graph: a left transition from 3,1 (in the upper-left of the second graph) links to 2,1 for example (upper-right of the same graph).

```
// get the state id from the top side and the south side
signed int trans_state(int prev_top_side, int prev_south_side) {
  int previous_state = 0;
  int1 found = 0; // int1 is just one bit
  // calculate previous state id:
   previous_state=0;
   while ((previous_state<NUM_STATES) && (found==0)) {
      if ((prev_top_side==states[previous_state][SD_TOP]   )&&
      (prev_south_side==states[previous_state][SD_SOUTH])){
            found = 1;
       }
       previous_state++;
   }
   if (previous_state == NUM_STATES)
       return -1; // bad!!
   else return previous_state; // good!
}
```

```
// estimate the transition using the finite state diagram
int get_state(int previous_state, int top_side) {
  int ret = TR_BAD;
  // calculate the top side of ...
   if ( states[ trans[previous_state][0] ][SD_TOP] == top_side)
       ret = TR_UP; else
   if ( states[ trans[previous_state][1] ][SD_TOP] == top_side)
       ret = TR_DOWN; else
   if ( states[ trans[previous_state][2] ][SD_TOP] == top_side)
       ret = TR_LEFT; else
   if ( states[ trans[previous_state][3] ][SD_TOP] == top_side)
       ret = TR_RIGHT; else ret = TR_BAD;
  return ret;
}
------------------------ in main(): ----------------------------------------
// get the complete state of the cube (using the transitions):
          previous_state = trans_state(previous_top_side,
                                       previous_south_side);
          if (previous_state != -1) {
            transition = get_state(previous_state, top_side);
            if (transition != TR_BAD) {
                current_state = trans[previous_state][transition];
                south_side = states[current_state][SD_SOUTH];
            }
          }
 ------------------------------------------------------------------------------
```

Table 3. Source code for finding the current state and the last transition, using 2 look-up tables (states, trans).

Distinguishing gestures

So far, the accelerometers have been used to just measure the cube's orientation. However, dynamic acceleration can provide additional information: a basic set of gestures (shaking the cube, twisting it, and knocking on it) that give distinct signal patterns is used as additional means of input.

Note that these gestures do not pose any restrictions on the orientation or position of the cube: An important requirement for the gestures is that they must be independent of the orientation of the cube itself, i.e. the gestures must be recognized regardless of the cube's states (although they might be combined for the application). The other restrictions that applied in previous sections due to the microcontroller-based platform, such as limited memory, power, and processing resources, are valid for the gesture recognition algorithms as well.

Ideally, the gestures should be straightforward and familiar enough to a person that is handed the cube for the first time and asked to perform gestures like shaking, twisting or knocking on the cube, hence the choice for these three particular gestures.

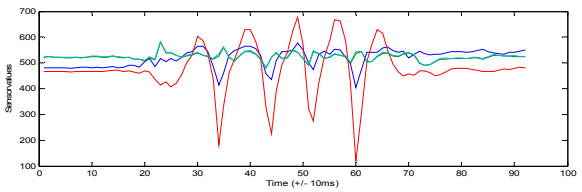

Figure 7a. An example of accelerometer data while the cube is being shaken.

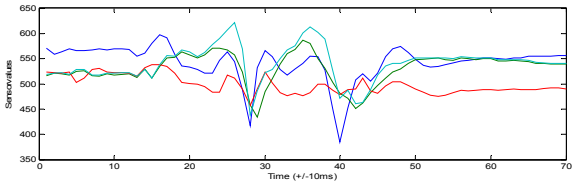

Figure 7b. An example of accelerometer data while the cube is being twisted.

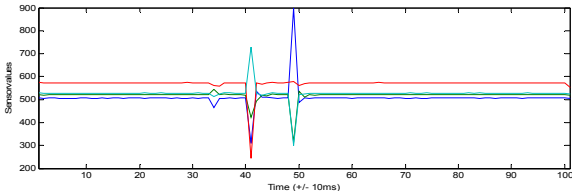

Figure 7c. An example of accelerometer data while the cube is being knocked on.

The patterns of these three gestures have certain characteristics that are distinguishable enough from one another, and from a non-gesture (i.e. normal use of the cube without performing an actual gesture). The three examples in Figure 7 show a high variation across the signals whenever a gesture is carried out, which validates the use of variance or standard deviation as a feature. A main difference between the shaking gesture and the twisting gesture is the number of sensors that give a strong variation, as shaking tends to be more unidirectional. The variance over a shorter time span is also key to distinguishing the knocking on the cube.

Microcontroller-specific implementation

Certain choices can reduce the amount of required memory significantly. Apart from the obvious need for well-chosen data types (8 bit integer, floating point, etc.), there are a few things that allow implementation of more interesting algorithms and a swift execution of the microcontroller program.

One of these is the use of Borchardt's formula to approximate the logarithm function:

$$\log(x) \cong \frac{6.(x-1)}{x+1+4.(\sqrt{x})}$$

which proved to be accurate enough in our case. Another one is the use of a running sum and a running sum of squares instead of calculating the variances each time over the whole window of *n* past samples. This works because:

$$\mathrm{var}(x) \sim \sum_{i=1}^{n} x_i^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}$$

By implementing the required algorithms in such an efficient way, all three discussed algorithms fit effortlessly in the 18F525 microcontroller.

APPLICATIONS

Profile Switching in Ubiquitous Environments

The Innovative Interactions Lab at Lancaster University is a 'living lab' casual meeting area which is fitted with several large-screen displays, sensors, and a high-end 8-channel audio system.

Figure 8. The equipment for controlling the audio system in Lancaster's innovative interactions lab, which generally requires expertise to operate.

The audio system is controlled through a mixer that is hooked up to a number of devices, such as a TV, a net-meeting environment, MiniDisc player, CD player and several input sockets positioned around the lab space. The audio system is located in a dedicated room, and using it for any of these applications means going to the mixing panel, switching the appropriate channels and setting the sliders to the required volume on the mixer. In practice, this requires expertise with not only the mixer settings, but also the entire audio system (including a dedicated PC, and a rack of audio equipment, see Figure 8).

The cube is used here as a simple, tangible interface: anybody in the lab can set the audio system to one of the six most popular settings by placing the cube so that the side that relates to the desired appliance faces upwards. The cube's output is sent to a base station that redirects and converts the packets to a MIDI stream that controls the mixing panel (see Figure 9 for a picture of the lab setup). This application does not use the abilities the cube to its full extend, but is deployed for a period of several months and is being used extensively by people on a daily basis.

Figure 9. The television area of Lancaster's innovative interactions lab, with the cube in front of it. Placing it so that the side with the TV-icon is on top will switch the audio system to the TV-channels.

Basic Navigation

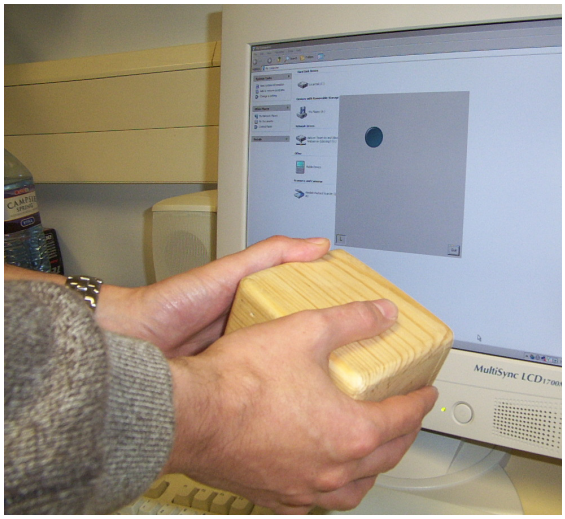

Figure 10. The demonstrator in action, where a dot on the screen can be moved by turning the cube. If a gesture is detected, a notification is displayed.

At this point, basic navigation (up/down/left/right) of a dot on the screen (see Figure 10) with recognition of the aforementioned basic gestures, exists as a demonstrator to show the full abilities of the cube. We hope to further this research by combining case studies and experiments in user experience, as started in [07].

ACKNOWLEDGEMENTS

This research was funded by Equator IRC (GR/N15986/01 - "Technological Innovation in Physical and Digital Life") and under grant GR/S08848/01 ("Multi-Sensor Perceptive Interfaces in Wearable and Ubiquitous Computing"), both by the EPSRC.

REFERENCES

[01] Analog Devices Inc. ADXL311 dual axis accelerometer datasheet, 2003. http://www.analog.com/Analog_Root/productPage/productHome/ADXL311.html

[02] A.Y. Benbasat and J. A. Paradiso. "An Inertial Measurement Framework for Gesture Recognition and Applications". In Ipke Wachsmuth, Timo Sowa (Eds.), Gesture and Sign Language in Human-Computer Interaction, International Gesture Workshop, GW 2001, London, UK, 2001 Proceedings, Springer-Verlag Berlin, 2002.

[03] M. Brand, N. Oliver, and A. Pentland. Coupled hidden Markov models for complex action recognition. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, pages 994–999, San Juan, Puerto Rico, 1997.

[04] O. Cakmakci, J. Coutaz, K. Van Laerhoven, and H.-W. Gellersen. "Context Awareness in Systems with Limited Resources". In Proc. of the third workshop on Artificial Intelligence in Mobile Systems (AIMS), ECAI 2002, Lyon, France. 2002. pp. 21-29.

[05] K. Camarata, E. Y.-L. Do, B. R. Johnson, and M. D. Gross. "Navigational blocks: navigating information space with tangible media". In Proceedings of the 7th international conference on Intelligent user interfaces, pp. 32-38. ACM Press, 2002.

[06] E. C. Cherry, "Some experiments in the recognition of speech, with one and two ears". Journal of the Acoustical Society of America, 25, pp. 975-979. 1953.

[07] J. G. Sheridan, B. W. Short, G. Kortuem, K. Van Laerhoven and N. Villar. "Exploring Cube Affordance: Towards A Classification Of Non-Verbal Dynamics Of Physical Interfaces For Wearable Computing". In Proceedings of the IEE Eurowearable 2003; ISBN 0-85296-282-7; IEE Press. Birmingham, UK, pp. 113-118.

[08] K. Van Laerhoven, N. Villar, A. Schmidt, G. Kortuem and H.-W. Gellersen. "Using an Autonomous Cube for Basic Navigation and Input". In Proceedings of ICMI/PUI 2003. ACM Press. Vancouver, Canada. 2003. To appear.

[09] Weisstein, E. The Cube. World of Mathematics. Online web resource. http://mathworld.wolfram.com/Cube.html August 2003.