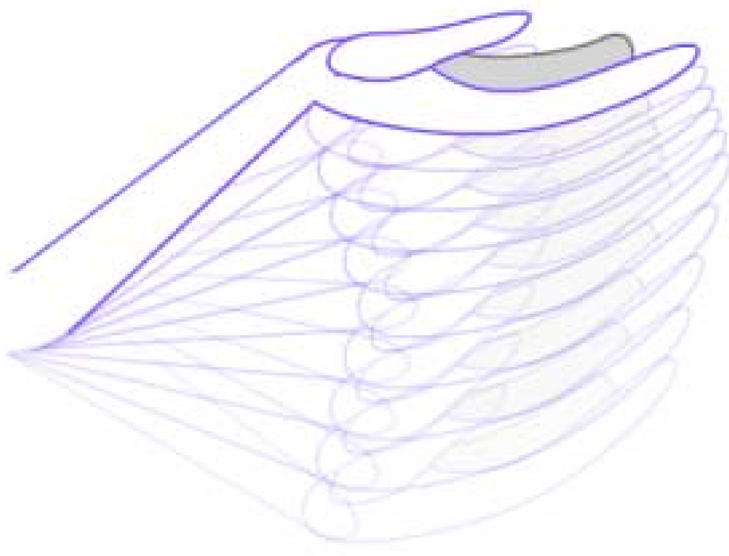
Gesture-Based Control of a Personal Digital Assistant

A Feasibility Analysis of Hand Gestures as User Input

Krispin Leydon, Jonas Akermark, Andrew Jones



Final Report, Engineering Sciences 290
Dartmouth College, March 6th, 2001

Sponsor: Analog Devices
Advisor: John Collier

Table of Contents

1	Introduction & Need Statement	4
2	State of the Art	5
3	Specifications	6
4	Justification of Specifications	7
5	Path to Proposed Solution	8
5.1	Decision Summaries	8
5.2	Summary of Proposed Solution	9
6	Design & Implementation Methodology	10
7	Interaction Design	11
7.1	Mobile Gesture Based Interface Survey	11
7.2	Button Feasibility Survey	11
7.3	Summary of Survey Results	12
7.4	Creation of User Personas	12
7.5	Gesture Selection Process	13
8	Algorithm Development	14
8.1	Algorithm Development Methodology	14
8.2	Algorithm Class Selection	14
8.3	Statistical Envelope Approach	18
8.4	Piece-Wise Algorithm	22
8.5	Threshold Based State Machine	25
9	PDA Development	26
9.1	Hardware: Interfacing the Accelerometer with the Palm III	26
9.2	Software: Implementing the Gesture Recognition Algorithm	27
10	Final Evaluation	29
10.1	Reliability & Consistency	29
10.2	Acceptability	29
10.3	Safety	30
10.4	Recognition Speed	30
10.5	Device Feasibility	30
10.6	Economic Analysis	31
11	Budget	32
12	Future Work	33
12.1	Interaction Design	33
12.2	Software Design	33
12.3	Hardware Design	33
12.4	Prototype testing	33
13	Conclusions	34
14	Acknowledgements	35
15	Works Consulted	36

List of Figures

Figure 1:	Project Methodology	10
Figure 2:	Gesture Selection Process	13
Figure 3:	Algorithm Development Process	14
Figure 4:	A statistical envelope	15
Figure 5:	Threshold-Based State Machines	15
Figure 6:	Neural Networks	16
Figure 7:	Genetic Algorithms	17
Figure 8:	Hausdorf Metric	17
Figure 9:	Splitting the data files	19
Figure 10:	Instances of gesture "Right Shift" raw	19
Figure 11:	Instances of gesture "Right Shift" filtered	19
Figure 12:	All instances "Right Shift" overlaid.	20
Figure 13:	The model constructed for "Right Shift".	20
Figure 14:	Model envelopes for two opposite gestures overlaid.	21
Figure 15:	The piece-wise model approach.	22
Figure 16:	The twelve feature windows used to differentiate gestures	23
Figure 17:	Recognition rate as a function of sampling frequency	24
Figure 18:	Typical "Left Shift"	24
Figure 19:	Typical misinterpretation of "Left Shift"	24
Figure 20:	Modified Hotsync cradle	26
Figure 21:	PDA being wired	26
Figure 22:	Palm III Final Prototype	26
Figure 23:	Palm implementation algorithm	27
Figure 24:	GestRec main screen	28
Figure 25:	GestRec preferences	28
Figure 26:	Accuracy	28
Figure 27:	Puzzle	28

Appendices

Appendix A:	Computational Power & User-Interface Capability Vs. Time	38
Appendix B:	PDA Development Timeline	39
Appendix C:	Modes of Input for Mobile Informational Devices	40
Appendix D:	Multiple Inputs, Parallel Paths	41
Appendix E:	PDA Shapes and Sizes	42
Appendix F:	The "Homunculus"	43
Appendix G:	Graffiti Writing Samples	44
Appendix H:	Graffiti Recognition Rate Benchmark Test Results	45
Appendix I:	Mobile Gesture Based Interface Survey	46
Appendix J:	PDA Use Survey Results	51
Appendix K:	Button Feasibility Survey	57
Appendix L:	Mobile Gesture-Based Interface Survey #2	60
Appendix M:	Use Contexts & User Personas	61
Appendix N:	Generalized Navigation & Selection Commands	62
Appendix O:	Ergonomic Considerations	63
Appendix P:	Decision Matrices	65
Appendix Q:	Gesture Direction Sheets	72
Appendix R:	Gesture Selection Test Form	78
Appendix S:	Gesture Data Acquisition Apparatus	79
Appendix T:	TilePlot & TFPlot	80
Appendix U:	Piece-Wise Gesture Recognition Code Listing	81
Appendix V:	Final Testing of the Piecewise Algorithm	102
Appendix W:	Threshold-Based State Machine #1	103
Appendix X:	Threshold Based State Machine Test Results	105
Appendix Y:	Modified Hot Sync Cradle	106
Appendix Z:	Accelerometer – Palm III Connection Instructions	107
Appendix AA:	Palm Gesture Recognition Code	108
Appendix BB:	Testing of the PDA Prototype	119
Appendix CC:	Price and Percentage Increase in PDA Price	120

1 Introduction & Need Statement

As personal digital assistants (PDAs) grow increasingly powerful, the limitations of their user-input systems become increasingly, frustratingly apparent. Though modern PDAs contain the essential functionality of a networked office PC, this functionality is virtually inaccessible on the fly. Portable electronic address books store and organize thousands of names, yet accessing a single name while holding a briefcase is prohibitively difficult. A doctor may hold the information of a hundred medical texts in the palm of her hand, however this information is inaccessible as she runs down the hallway of an emergency ward. While the computational power necessary to make PDAs extraordinarily useful in mobile use environments exists, interfaces with the required flexibility and usability do not.[1]

In mobile use scenarios, standard interface solutions–the QWERTY keyboard and Window/Icon/Mouse/Pull-down-menu (WIMP) paradigm–fail. The QWERTY keyboard applied to PDAs becomes ineffectively small[2], and its requirements of two hands and a stable typing surface are incompatible with the PDA's potential as a mobile, one-handed platform. The difficulty of actuating precise cursor control in mobile environments hampers the effectiveness of WIMP-style PDA interfaces. Enormous potential exists to harness the existing computational and organizational power of PDAs, and old interface solutions do not meet the challenge. While input methods designed for PDAs, like stylus-based interfaces, are similarly impeded.

Though numerous input modes[3] (and combinations of input modes[4]) have been explored for the purpose of increasing the utility and flexibility of PDA interfaces, hand-initiated PDA motion has not been investigated thoroughly[5]. In the past, technical constraints have prevented such "gestural control" from being a viable source of input; sensors have been too large, signal processing algorithms too computationally expensive.

Recent advances in sensor technology and pattern recognition have removed these limitations. Analog Devices has created sensitive, precise accelerometers that fit the size and power constraints of a PDA. Efforts in speech recognition and airbag deployment have yielded pattern recognition techniques applicable to small digital devices (Kelly).

In order to improve the flexibility and usability of PDA interfaces, developers need access to various modes of user input, and knowledge of the advantages and disadvantage of each. PDA-holding hand movement is a now a possible form of control, and its feasibility, strengths and weaknesses must now be assessed.

[1] See Appendix A for a depiction of how computational power and interface capabilities have changed over time.
[2] For a time line depicting the decreasing size and increasing power of PDAs over the last 20 years, see Appendix B.
[3] For a table describing the advantages and disadvantages of state of the art PDA input modes, see Appendix C.
[4] Appendix D presents two cases where input modes have been used *in parallel* to achieve a more usable PDA.
[5] For a biological justification for the value of investigating hand gestures as a source of user input, see Appendix F.

2 State of the Art

There have been relatively few attempts to integrate PDAs and accelerometers for the purpose of gestural control. Notable exceptions include explorations by Sony's Computer Science Laboratory, Compaq's Western Research Lab and independent work conducted by software developer Till Harbaum.

Jun Rekimoto at Sony investigated the feasibility of tilt-based interfaces for handheld devices via workstation simulation. In a paper published in 1996, he postulated that such interfaces would become "much more practical" as motion sensors become more widely used (167). Till Harbaum's work involved integrating an ADXL202 accelerometer with a Palm III PDA. He wrote PDA processor routines for sampling accelerometer data as well as several programs that respond to tilting motions including a marble-maze game called "Mulg". The systems developed by Rekimoto and Harbaum are based entirely on tilt motions, and so far as we know, do not involve the recognition of more complex hand gestures.

The "Rock 'n' Scroll" system under development at Compaq's Western Research Lab is a sophisticated accelerometer-based gestural PDA interface that recognizes a range of hand gestures for the purpose of navigating digital photo albums and other tasks (Bartlett). During our development effort we were unaware of this project. In retrospect, it would have been valuable to have researched its findings more completely.

3 Specifications

A prototype of a successful gesture recognition system would meet the following specifications. The process used to quantify these specifications is described in the next section.

Reliability & Consistency

- The system should recognize three or more gestures.
- When a gesture is performed it should be recognized correctly at least 88% of the time and must not be recognized as a different gesture more than 9% of the time.
- The system should reject all motion when not enabled.

An Acceptable Gesture Set

- The set of actuating gestures will be rated by a group of potential users according to the following criteria: comfortability, intuitiveness, acceptability, recognizability and consistency. Each gesture should score higher than 3 on a scale of 1 to 5.

Safety

- Gestures should be ergonomically safe to perform according to the guidelines outlined by NIOSH, Liberty Mutual (America's largest healthcare insurer), and Dartmouth's ENVHS office. For an outline of these ergonomic considerations see Appendix O.

Cost

- The cost of adding a gesture recognition system to a modern PDA should not exceed $15.

Recognition Speed

- The time between the end of a gesture's performance and the accompanying action should not exceed 0.4 seconds.

Device Requirements

- The memory used by the gesture recognition system should not exceed 200KB of PDA storage memory and 9.6KB of dynamic memory.
- While the sensor is in use, it should not consume more than about 0.5mA.

In addition to developing a prototype, answers to the following questions are necessary:

- In what contexts is hand-held gestural input desirable?
- Who would benefit from hand-held gesture recognition?
- For what uses is gestural control particularly well suited?

Additional Sponsor Dictated Constraints.

- Hardware added should consist of only a single iMEMs accelerometer and associated discrete components.
- The accelerometer must lie in the same plane as the device.
- Project focus should be restricted to designing for right hand users.

4 Justification of Specifications

Quantification of the specifications listed above was accomplished through research and testing of target devices and analogous input systems. The results of this testing and research appear below in question and answer form:

- *What is an acceptable recognition consistency?*

To determine the consistency necessary for PDA input methods, we looked to another widely accepted mode of user input for the Palm Pilot: The Graffiti text-entry system. We measured the consistency with which Graffiti recognized each letter—both in context of use (see "pangram" sample sentences in Appendix G) as well as out of context (repeating each letter 35 times). This test was conducted with three test subjects (the members of the project group) and the results averaged to get a recognition rate of 88%. For the complete results of the test of the Graffiti handwriting recognition system see Appendix H.

- *How much would the addition of gesture recognition add to the price of a device?*

A gesture recognition system would be a convenient addition to a PDA, but is by no means necessary for controlling the device. Thus we do not feel that the addition of this system would justify a large increase in price. Based on informal discussions with potential users we estimate that users would be willing to pay $15 in addition to the base price of the PDA for this feature. Appendix CC diagrams the percent increase in price this addition would cause for different price classes of PDAs. This feature might not be best suited for basic PDA models where it adds 10% to the price, perhaps being more appropriate for more feature rich models where the price increase is between 4% and 6%.

- *What is the maximum delay between gesture completion and the corresponding action performance that users will accept?*

In order to determine a maximum acceptable delay, we performed a test where subjects were asked to press a key and wait for an action (a number representing delay time was printed to a computer screen). As the test progressed, delay time grew longer. Subjects were asked to stop when the delay became "too long". The average delay time from this test was 0.43 seconds, so this value was chosen as our system delay specification. Thus if our gesture recognition system has a delay greater than this we will either have to modify our algorithm to reduce this or find a platform which can execute the algorithm fast enough to meet this requirement.

- *How much memory will be available for our gesture recognition system?*

Present day PDAs have between 2MB and 32MB of storage memory available, and far less dynamic memory (memory allocated to a program at run-time). Since gesture recognition will be taking place in the background of other applications, the memory requirements of a gesture recognition system for PDAs should not exceed 10% of the device's storage memory or 10% of the device's dynamic memory. For the Palm III, this translates to 200KB and 9.6KB, respectively.

5 Path to Proposed Solution

In determining what form our gesture recognition system would take, a number of choices were made. These choices appear below in summary form, with competing alternatives bulleted, and selected alternatives underlined. For a more thorough explanation of our criteria for making choices, see the decision matrices in Appendix P.

5.1 Decision Summaries

5.1.1 Gesture Selection

Given the range of motion possible, what discrete motions warrant further consideration as control gestures for a gesture recognition system?

We began with a brainstorm session yielding 23 candidate gestures (See Appendix Q). These 23 gestures were performed, recorded, and ranked by project team members based on ergonomic criteria and how easily they could be recognized. The top 14 gestures were selected for evaluation by a larger test group. From this second evaluation 8 gestures were chosen for further consideration. A discussion of the gesture selection process appears in Section 7.2.

5.1.2 Gesture Control Scheme

What actions will gestures initiate?

- Text Entry
- Application-Specific Commands
- User-Defined Control Mapping
- <u>Generalized Navigation and Selection Commands</u>

In order to limit the scope of this very broad problem, our effort has been to find the minimum number of gestures that could control a wide variety of digital hand-held tools. The resulting selection of "generalized" control functions appears in Appendix N.

5.1.3 Gestural Interaction Scheme

Does it make sense to include one button to initiate the start, end, or duration of a gesture, or to enable gesture recognition?

- No button. Continuous polling for gestures.
- Button press signals gesture start.
- Button press signals gesture end.
- While button is pressed, system polls for single OR multiple gestures until button release.
- While button is pressed, system polls for a single gesture until button release
- <u>Button activates/de-activates gesture recognition system</u>

5.1.4 Primary Simulation Strategy

- Create PC-based application that reads & analyzes accelerometer data in real time.
- <u>Create PC-based application that reads & analyzes accelerometer data from a file.</u>

Reading in data from files rather than obtaining it directly from the accelerometer simplifies the comparison of different algorithms as multiple algorithms can be tested with the same data.

5.1.5 Gesture Recording System

- Develop New Data Acquisition System
- Use Signal Quest Data Acquisition System
- <u>Use Crossbow Software with Crossbow Data Acquisition Board</u>

We were given the crossbow system, and aside from some problems with maintaining a consistent sampling rate, it performed well, so we opted to use it.

5.1.6 Motion Sensor

- MEMS Accelerometer
- Piezoelectric Accelerometer
- Variable Resistance Accelerometer
- Gyroscope
- Variable Capacitive Accelerometer

The use of Analog Devices accelerometers was a sponsor-dictated constraint, however we researched alternatives, and found them inferior for our application. Gyroscopes only provide rotational information, and compared to Analog's iMEMs accelerometers non-iMEMs capacitive accelerometers are larger, variable resistance accelerometers more power-hungry, and piezoelectric accelerometers more costly and less accurate at low frequencies (Motus Bioengineering, PCB Piezoelectronics, Endevco).

5.1.7 Proof-of-Concept PDA Platform

- Windows CE Palmtop Computer
- <u>3Com Palm Pilot</u>
- Handspring Visor

Initially we planned to use the Handspring Visor as our PDA development platform because its "SpringBoard" system of interfacing the PDA with auxiliary components seemed ideal for our application. After several months, we realized that the development of a springboard module would take more time than connecting an accelerometer to a Palm Pilot serial port as per the web-posted instructions of Palm developer Till Harbaum, described in Section 9.1.

5.2 Summary of Proposed Solution

To experimentally investigate the merits and limitations of a gesture recognition system as an input method for mobile hand-held devices, the team planned to create a system demonstrating the ability to distinguish between five actions, as five actions allow control over a wide variety of applications. Algorithms for gesture recognition would be developed iteratively through simulation using recorded gesture data. The most promising algorithm(s) would then be implemented as real time gesture recognition system(s) on a PC. Finally, time allowing, the final gesture recognition system would be ported to a Palm Pilot. The gesture control system will be composed of PalmOS software together with one MEMS accelerometer and one pushbutton.

6 Design & Implementation Methodology

The project can be broken down into four areas: Interaction Design, Algorithm Development & Simulation, PDA Control and Planning & Documentation. Figure 1 shows a flow chart diagramming necessary tasks in each of the four areas.

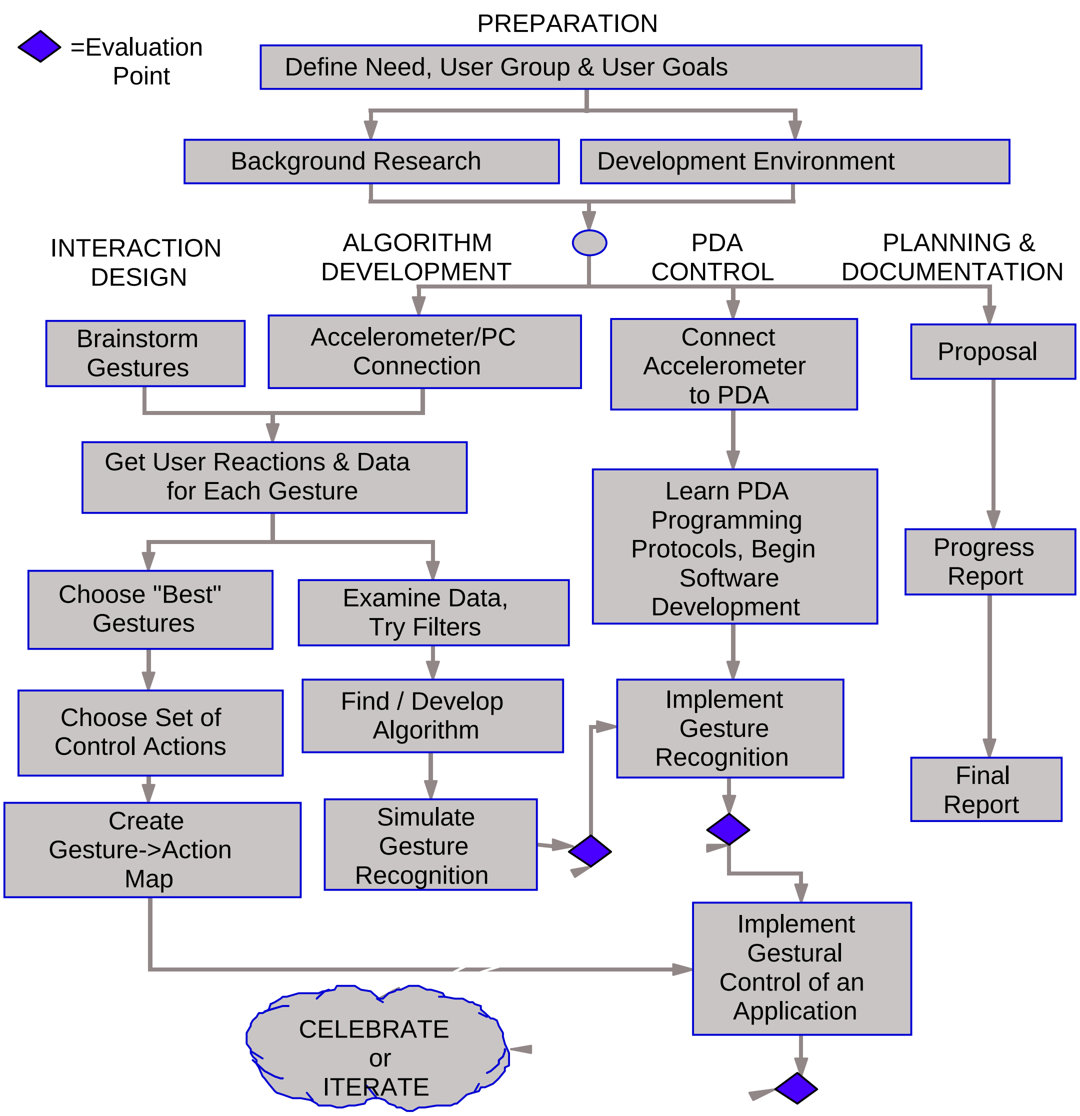


Figure 1: Project Methodology

7 Interaction Design

7.1 Mobile Gesture Based Interface Survey

In order to verify a need for gesture recognition and gauge public interest, we conducted a survey. The first part of the survey gathered information concerning the frustrations people have with present-day PDA interfaces. The second half of the survey obtained answers to the questions: "In what contexts could gestural control of a PDA prove useful?" and "For what tasks would gestural control of a PDA be well suited?" The survey was distributed via the web to numerous people in the northeast United States and Stockholm, Sweden. The complete survey appears in Appendix I, and the results appear in Appendix J.

We received a total of 85 responses, mostly from students at Dartmouth's Thayer School of Engineering and the Royal Institute of Technology in Stockholm. Just under half of the respondents owned PDAs, the majority of which were PalmOS devices.

The most popular PDA applications cited were address books, date books, and to-do lists – the date book the most popular of the three.

The most common problems people had with their PDAs were with handwriting recognition, small screen size, and low screen resolution. Other complaints concerned stylus use and general remarks about the interface being inadequate. Respondents rated the application navigation systems of their devices as useful and intuitive, but not particularly so.

While most of the suggestions survey participants made concerning improvements to the PDA interface were not related to our project's focus, respondents did request additional modes of data entry and an input method requiring just one hand. Many of the situations in which people reported not being able to use their PDAs and wanting to do so—while driving, talking on the phone or walking—were situations where only one hand was available. Slightly over half of the respondents thought that using motion to control their PDA would be helpful.

Participants also pointed out that gestural control could be useful in poor lighting conditions, when seeing a PDA is difficult. Most survey participants thought a gesture recognition system would be particularly well suited for the tasks of information-system navigation, shortcuts to basic PDA functions, and scrolling through text.

7.2 Button Feasibility Survey

A second survey was conducted to answer one question neglected in the first survey: "At what size do push-buttons become an inadequate form of user-input for a personal digital assistant?" Answering this question was an important part of our effort to determine when hand gestures constitute the *best* form of user input for a PDA. This second survey appears in Appendix K and the corresponding results appear in Appendix L. The survey found that when PDA size falls below $3.5cm^2$, buttons become an inconvenient form of user input. Appendix E shows a diagram showing a few of the shapes and sizes mobile computing devices are available in today. Three of these examples have less than $3.5cm^2$ of surface area.

7.3 Summary of Survey Results

Through surveys and informal conversations, we determined that gestural control over a PDA is useful in the following contexts:

- When the user has only one free hand, eliminating stylus use.
- When the device has a surface area on the order of 3.5 cm^2 (credit card sized) or less, eliminating button use.
- When there is ambient noise, background conversation, or the need to speak while controlling PDA functions, eliminating the use of voice recognition.

The tasks that lend themselves to gestural control include navigation through information systems (file-systems, data-bases, maps, etc), moving back and forth quickly between basic PDA functions, and scrolling through text. We determined that gestural control was *not* well suited to the task of data entry.

7.4 Creation of User Personas

After articulating the contexts and tasks appropriate for gestural control, we developed a list of user personas; a cast of fictional characters who would be interested in a gesture recognition system used to illustrate real user-groups with real needs. This list of personas appears in Appendix M. From this cast of characters, we selected “Jaimie”, an interface designer for a web-enabled mobile phone company, as our “primary persona”. This character represented the group of users whose goals we would keep in mind when making design decisions. “Jaime” was selected because she had a compelling need in the present for a gestural interface system, and had the professional experience necessary to provide good feedback concerning how the system could be improved. This feedback would be invaluable in early stages of testing a gestural control system.

After selecting Jaimie as our primary persona, we contacted several real life “Jaimies”—interface designers at Cognetics Corporation, Cooper Interaction Design, Research in Motion and the Human Computer Interaction Institute. All demonstrated an interest in gestures as a means of controlling personal digital assistants.

7.5 Gesture Selection Process

To select the gestures we would attempt to incorporate into we began by conducting a brainstorming session that yielded 23 candidate gestures. These gestures were performed, recorded, and ranked by team members. Our criteria and the ranking appear in Appendix P. The nine worst ranked gestures were eliminated. A test group of thirty people of varied age, dexterity and size performed the remaining 14 gestures and filled out the form shown in Appendix R, ranking the gestures based on comfortability, intuitiveness, and acceptability. See Appendix P for the results of the test group survey. From this second evaluation, eight gestures were chosen for further consideration. A diagram of this process is shown in figure 2.

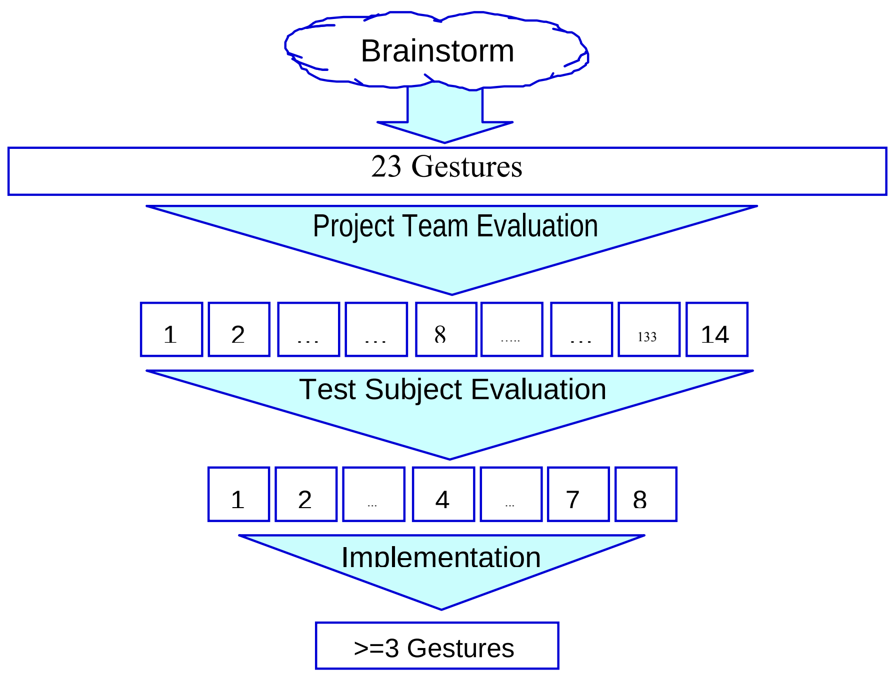
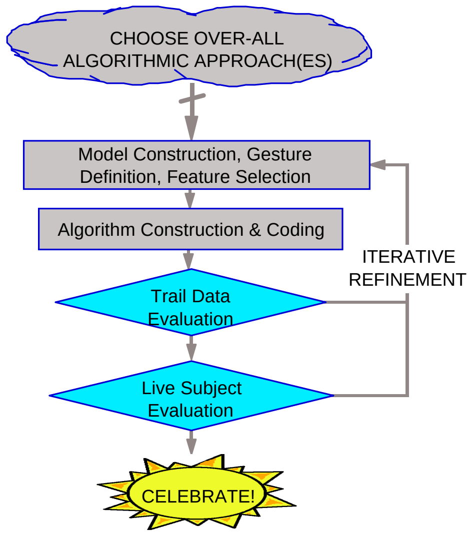


Figure 2: Gesture Selection Process

8 Algorithm Development

8.1 Algorithm Development Methodology

Development of an appropriate pattern recognition algorithm was a key aspect of our project. The flow chart in figure 3 illustrates our process for this challenging task.



Model Construction & Refinement: Developing models for each gesture; determining what the "ideal" Tilt Snap Gesture (for instance) might be, through visual comparison and statistical means.

Algorithm Construction & Coding: Creating an instructional framework in which acceleration data can be monitored and features recognized.

Trial Data-Based Evaluation: "Feeding" gesture trial data to the algorithm, and recording the algorithm's over-all performance.

Live Subject Evaluation: Determining how well the algorithm recognizes "live" gestures from a group of test users.

Figure 3: Algorithm Development Process

Another path we could have taken, instead of implementing our own recognition algorithm was to try to use Palm's built in Graffiti system to recognize gestures. However, we decided not to do so as it would forever limit our system's usefulness to PalmOS-based PDAs. In addition, relying on Graffiti as a "black box" would prevent us from learning anything about pattern recognition. The full decision matrix addressing this decision is supplied in Appendix P.

8.2 Algorithm Class Selection

Algorithm development began with a survey of the classes of algorithms that could be (or have been) applied to the problem of pattern recognition & classification. These classes include: Statistical Envelopes, Threshold-Based State Machines, Neural Networks, Genetic Algorithms, Application of the Hausdorf Metric, and Hidden Markov Models. An overview of these algorithm classes is presented below.

8.2.1 Statistical Envelopes

<u>Explanation</u>: Point by point, check to see if a trial signal matches the statistical envelope created by a mean value +/- a certainty interval.

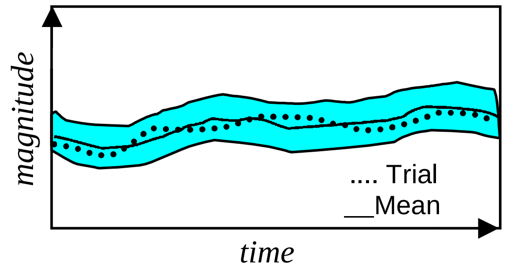


Figure 4: A statistical envelope

<u>Applications</u>: Statistical envelopes are used in the sciences to determine if experimental results match theoretical predictions within expected error bounds. This methodology is not used widely to solve categorization problems.

Advantages:	Disadvantages:
• Simple to implement	• Lacks characterization power -- can distinguish between relatively few gestures. • Highly dependent on consistent trial data

8.2.2 Threshold-Based State Machines

<u>Explanation</u>: Identify patterns through state-space search. Transitions between states occur when threshold conditions are met.

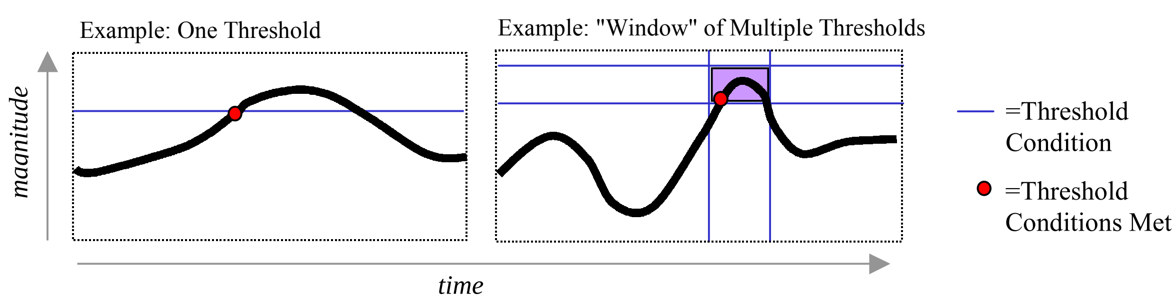


Figure 5: Threshold-Based State Machines

<u>Typical Applications</u>: Airbag deployment, fall-detection for the elderly, rate responsive pace-makers.

Advantages:	Disadvantages:
• Simplicity • Not processor intensive	• Requires a thorough understanding of the signals being examined. • Solutions are very problem specific.

8.2.3 Neural Networks

Explanation: Artificial neural networks (ANNs) are a form of classification based on observed brain behavior. The fundamental building block of an ANN a unit called a "perceptron". A perceptron adds weighted linear inputs, then compares the sum to a threshold to produce a discrete output. Through automated processes, input weights and the threshold level can be adjusted so as to achieve desired input-output behavior. Perceptrons are capable of approximating a number of linearly separable functions, and can be linked together in networks to approximate more complex functions. The behavior of ANNs with multiple perceptrons is not understood analytically at present, but ANNs can still be employed to solve categorization problems (Aslam).

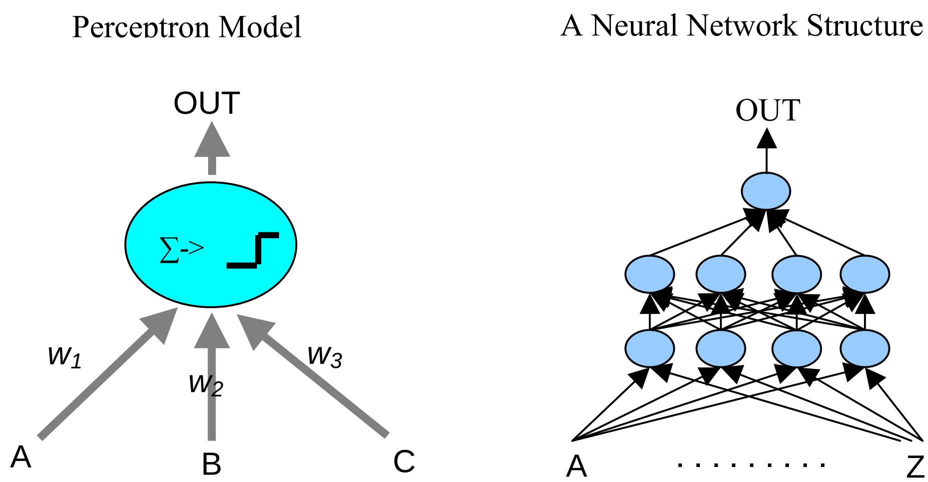


Figure 6: Neural Networks

Typical Applications: Speech & Character Recognition

Advantages:	Disadvantages:
• Flexibility & Extendibility; ANNs can be applied to almost any categorization problem	• Complex to implement • Depends on good training examples • Processor intensive • Time intensive to "train" an ANN • Solves problems in a "black box" manner; solution tells you nothing about solving the problem.

8.2.4 Genetic Algorithms

Explanation: Genetic Algorithms (GAs) model the process of natural selection. In GA's, solutions to a given problem are represented as boolean strings. Each boolean "gene" corresponds to the presence or absence of a particular attribute that might be part of a good solution to the problem. Through selection, crossover, and mutation, generations of potential solutions are created. The assumption is that later generations will contain stronger solutions than earlier generations. (Orchestrating this in practice is an art).

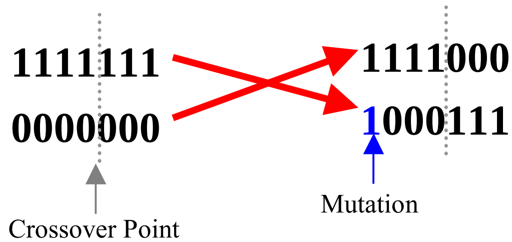


Figure 7: Genetic Algorithms

Typical Applications: Scheduling, path planning.

Advantages:	Disadvantages:
• Simple to implement • Flexibility & Extendibility; GAs can be applied to almost any problem	• Time intensive, thousands of generations may be necessary.

8.2.5 Hausdorf Metric

Explanation: The Hausdorf metric is a measure of the similarity between two point-maps (pictures, graphs, etc). Mathematically, the Hausdorf metric is the "maximum minimum distance" between points in a model and points in an actual image (Rus).

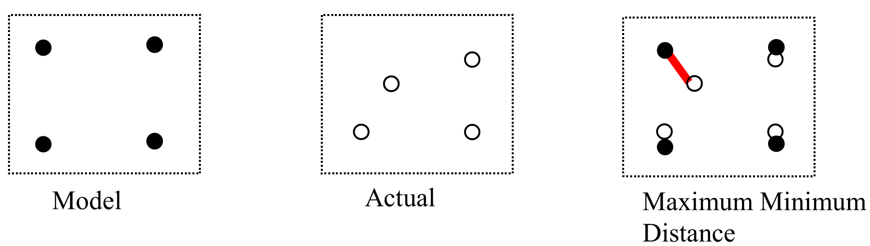


Figure 8: Hausdorf Metric

Typical Applications: Used in computer vision, visual pattern recognition.

Advantages:	Disadvantages:
• Good for matching arbitrary patterns • Relatively simple to implement	• Processor intensive • Slower than other forms of pattern recognition

8.2.6 Hidden Markov Models

Explanation: A Hidden Markov model is implemented as a state machine consisting one begin state and multiple possible end states. Each state has an associated probability distribution. Transitions between states are governed by these probability distributions (Warakagoda).

Typical Applications: Speech and handwriting recognition.

Advantages:	Disadvantages:
• Quick; lends itself to real-time signal analysis	• Processor intensive if implemented poorly • Complex to implement

NOTE: Of all the methods discussed, Hidden Markov Models were investigated the least thoroughly.

After researching these algorithm classes, we decided to investigate two in an in-depth fashion: Statistical Envelopes and Threshold-Based State Machines. For the matrix corresponding to this decision, see Appendix P.

8.3 Statistical Envelope Approach

Model Construction & Refinement

After choosing a set of gestures to investigate, each gesture was modeled to create a quantitative, statistical basis for deciding if a given motion was a gesture or simply random motion.

Model development began with making several simplifying assumptions and choosing an equation. Our initial simplifying assumptions were:

- The “start” and “end” points of gestures are detectable
- All acceleration signals begin “zeroed” at the origin
- Quantified variations in noise and the way a gesture is performed both fit standard normal distributions.

These assumptions led to the following equations:

$$Ax_{experimental}(t,s) = Ax_{mean}(t) \;+/\!-\; E_x\,(t,s)$$
$$Ay_{experimental}(t,s) = Ay_{mean}(t) \;+/\!-\; E_y\,(t,s)$$

where *A* is acceleration, *t* is time, *s* is a given test subject, and +/- E is a “envelope” accounting for variation.

To determine the values of each variable in the equation from out gesture data, gesture files were split, cropped, translated, normalized, resampled, filtered and graphed in order to create visual models for each gesture. This process is illustrated below using “Right Shift” as an example the gesture. The MATLAB applications shown in Appendix T were used for graphing and manipulating the data files.

First, data files were separated and cropped so that each file contained only one gesture. Shown below in figure 9 are three trials of the gesture "Right Shift".

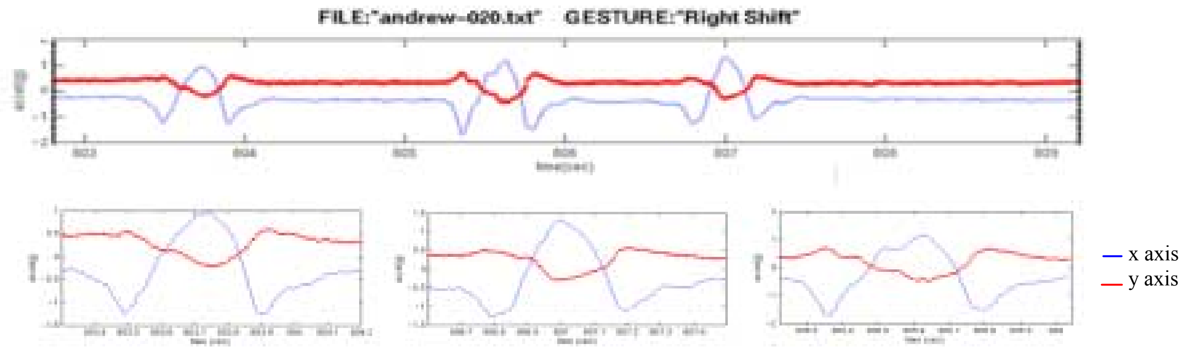


Figure 9: Splitting the data files

Next, trial data for each gesture was plotted side by side for comparison. Figure 10 shows unfiltered plotted data for "Right Shift", while figure 11 shows the same graphs after filtering. The filter selected was a low-pass digital filter designed to have a cutoff frequency of 4Hz – below the frequency range associated with hand tremor (8-12Hz), yet above the frequency range of gestural motion (Krause).

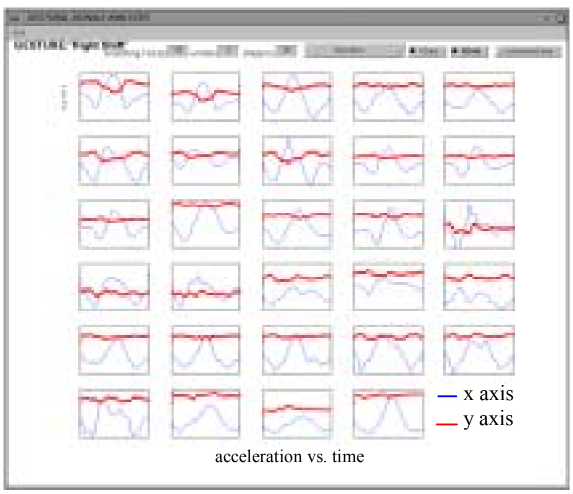


Figure 10: Instances of gesture "Right Shift" raw

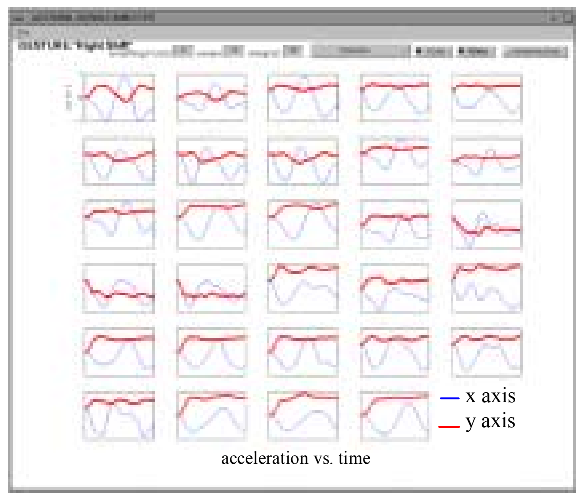


Figure 11: Instances of gesture "Right Shift" filtered

After separation and smoothing, data for a "mean" gesture with 95% certainty bounds was created by making the following substitutions in our model equations:

$$Ex\ (t,s) = 1.96*[standard\_dev(Ax_{experimental}(t,s)]$$
$$Ey\ (t,s) = 1.96*[standard\_dev(Ay_{experimental}(t,s)]$$

Our data for "mean" gestures and their 95% certainty envelopes was then plotted to obtain a statistical model for each gesture.

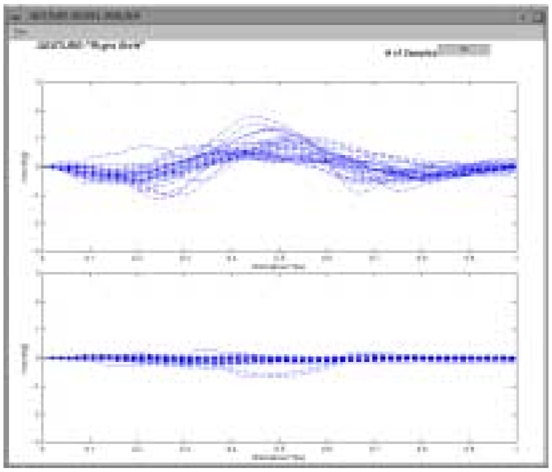


Figure 12: All instances "Right Shift" overlaid.

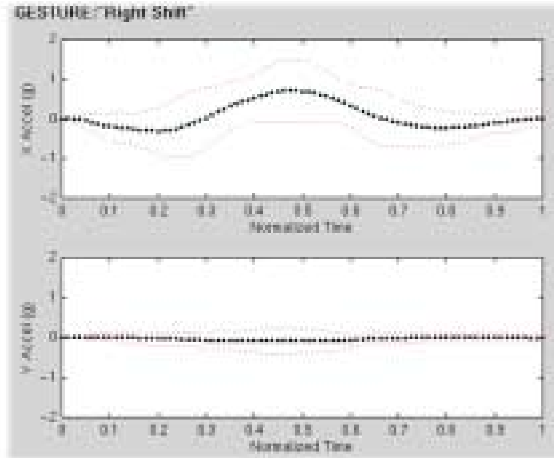


Figure 13: The model constructed for "Right Shift".

Gesture Definition & Feature Selection

This analysis of the gestures allowed us to verbally articulate the behavior of the x and y signals for each gesture. A summary of our English descriptions appear below:

Gesture	Written Description of A_x Waveform	Written Description of A_y Waveform
Tilt Left:	Goes down, stays down & flat, goes up.	A flat line close to A_y=0.
Tilt Right:	Goes up, stays up & flat, goes down.	A flat line close to A_y=0.
Tilt Up:	A flat line close to A_x=0.	Goes up, stays up & flat, goes down.
Tilt Down:	A flat line close to A_x=0.	Goes down, stays down & flat, goes up.
Tilt-Snap Left:	Goes down, goes up.	A flat line close to A_y=0.
Right Shift	First Valley<0, Peak>0, Second Valley<0. |Peak|>|First or Second Valley|.	A flat line close to A_y=0.
Left Shift	First Peak>0, Valley<0, Second Peak>0. |Valley|>|First or Second Peak|.	A flat line close to A_y=0.
Out and In:	A flat line close to A_x=0.	First Peak >0, Valley<0, Second Peak>0. |Valley|>|First or Second Peak|.
In and Out:	A flat line close to A_x=0.	First Valley<0, Peak>0, Second Valley<0. |Peak|>|First or Second Valley|.

Is the Statistical Envelope Approach Feasible?

To test the hypothesis that gesture recognition could be accomplished by comparing trial data with each statistical model, we overlaid model graphs to determine how much overlap there was. Through inspection we quickly found that the probability intervals were too wide for successful gesture recognition, as the graphs shown below illustrate. The graphs in figure 14 show the envelopes for two gestures, overlaid. The top graphs show x-acceleration versus normalized time, while bottom graphs show y-acceleration vs. normalized time. The blue areas in the graphs to the right show the union of each the two gestures' envelopes. Since it is possible to find a path from start to finish in the blue area for both x-acceleration and y-acceleration graphs, the envelopes are not unique enough to enable recognition of these two opposite gestures.

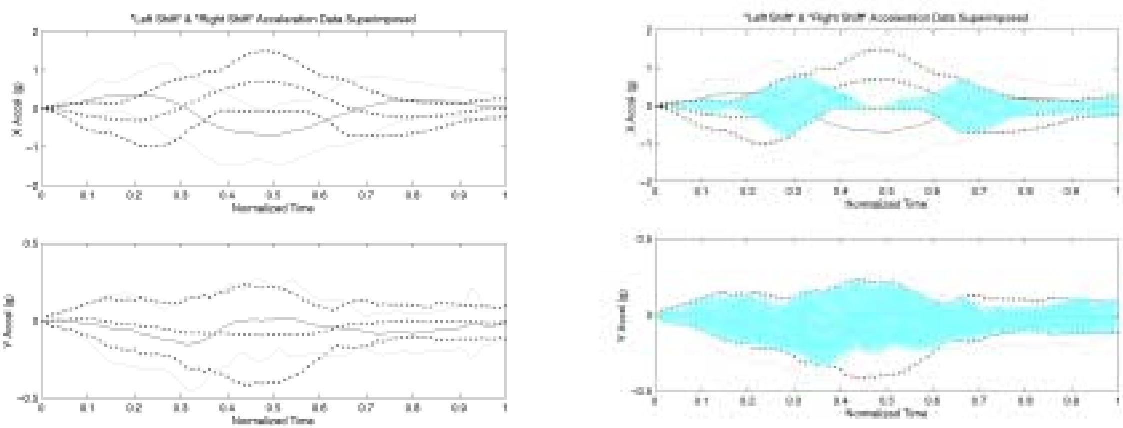


Figure 14: Model envelopes for two opposite gestures overlaid.

It must be noted that the 95% probability mentioned above is point-wise. It is the probability that one point in an experimental gesture data vector is within the certainty interval for a given sample time. This point-wise probability does not take into account the fact that many experimental data points must fall between upper and lower bounds. Given the “envelope” approach, a hand motion is a gesture if every data point falls within its respective certainty interval. We could have made the additional assumption that probabilities for each sample point were independent, however this assumption creates a much wider envelope (of no use to us) and is inaccurate — if one point falls within its respective 95% bounds, it is actually quite likely that neighboring points will fall within their respective bounds as well.

The envelope model is limited in that it does not treat time is a statistical variable. A more successful approach would focus on obtaining both magnitude *and* time probability windows for specific events within a gesture, rather than simply magnitude probability for the whole gesture.

8.4 Piece-Wise Algorithm

After discovering that a statistical envelope approach lacked sufficient categorization power, we decided to investigate a piece-wise model. This model defines a "start" region, an "end" region, and numerous "Feature Windows". We defined a feature window as the set of conditions (min/max normalized time, min/max signal magnitude, min/max slope) between which all instances of a specific feature, like a peak or valley, fell. By determining which feature windows the signal passed through and comparing this with known features in each gesture we can determine which gesture was performed.

Figure 15 shows an illustration of the piece-wise model.

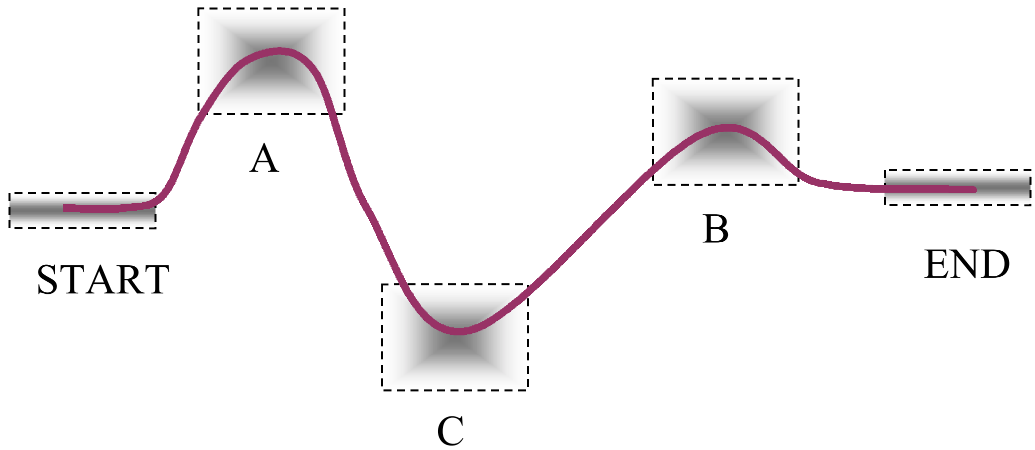


Figure 15: The piece-wise model approach.

The applicability of this model was tested in MATLAB. Trial data was examined to determine how many feature windows were necessary and what the slope, magnitude and time parameters for each should be. The windows were made as small as possible initially, and made larger as necessary to correctly recognize as many trial gesture files as possible.

At this stage, some trial data for each gesture was thrown out. Trials of a given gesture that differed significantly from the majority of the trials for that gesture were assumed to be bad data. This was necessary since many of our test subjects did not adequately practice performing the gestures before recording them or did not perform them correctly. We were unsure to what extent learning would effect a person's ability to reproduce a given gesture consistently, and our decision to eliminate certain trial data was based on the assumption that people could learn to do gestures with greater consistency than was reflected by our trial gesture data.

The twelve feature windows selected and their parameters appear in figure 16.

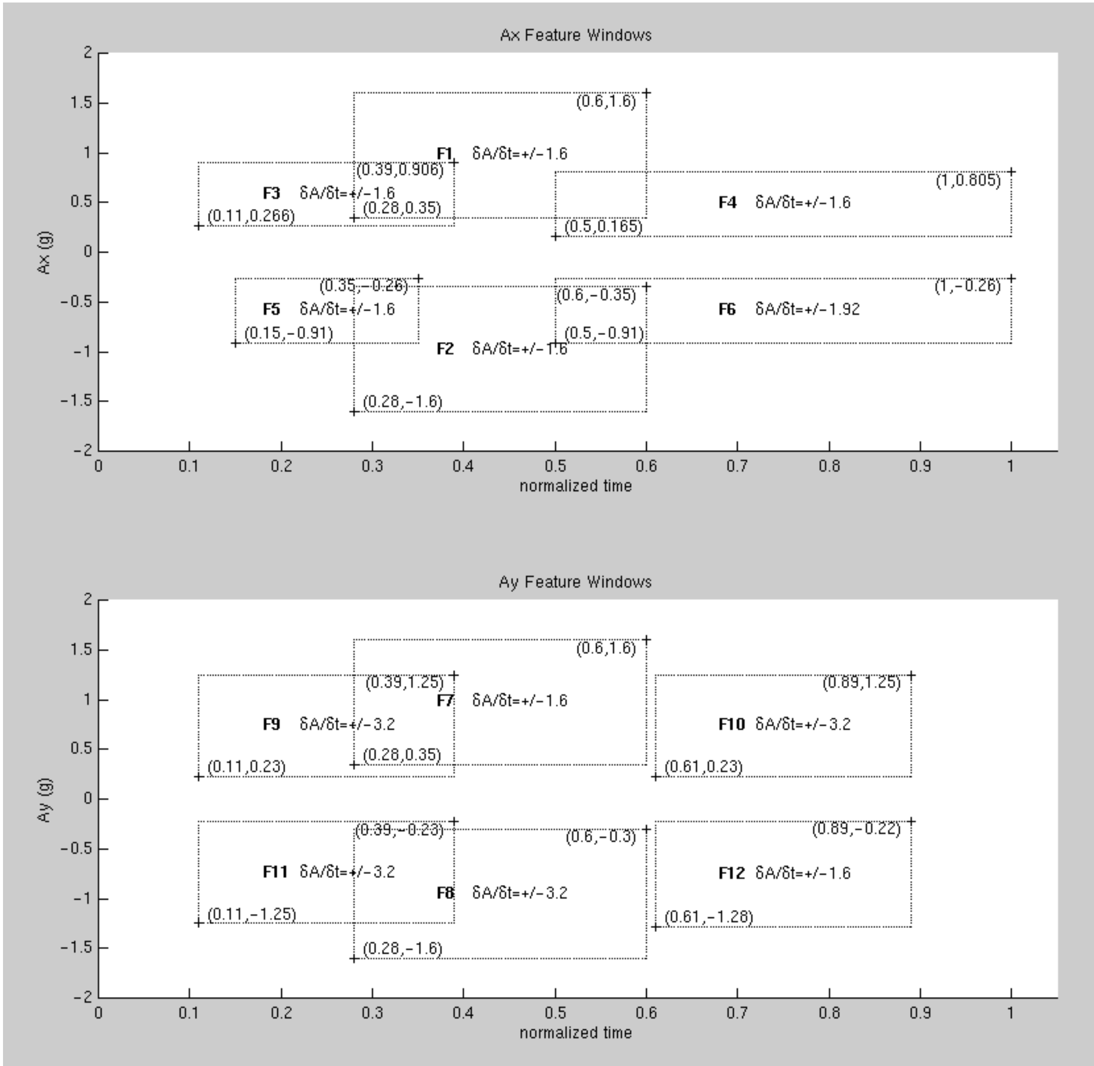


Figure 16: The twelve feature windows used to differentiate gestures

Once the MATLAB script could correctly identify all remaining gesture trial data, a program was created in C++ to test the piecewise algorithm in real time. The pseudo-code for this program appears below. A full code listing appears in Appendix U.

Define "Feature Windows": sets of conditions that must be met to test for a feature's presence or absence

1) Find Gesture Start
2) Begin recording & filtering accelerometer data
3) Find Gesture End (Or Time-Out)
4) Stop recording
5) Classify gesture, based on Feature Windows whose conditions are satisfied

After the program had been created and debugged, we tested its ability to recognize gestures. Our test apparatus was a Unix workstation and an accelerometer test board taped to the back of a Visor PDA. A diagram of the apparatus is given in Appendix S. Three test subjects (the project group) performed each of the five gestures the system could recognize in two tests. In the first test each gesture was repeated 35 times, the second test involved performing random gestures until each gesture had been performed 10 times. The results from this test were somewhat disappointing. The program was able to achieve an average recognition rate of 69% -- far below our specification of 88%. (For complete test results, see Appendix V.)

The real time implementation of the piecewise algorithm performed poorly for three reasons: 1) the sampling rate was too slow 2) the algorithm could not find gesture start and end reliably and 3) more trial data was necessary to optimally define feature windows. The graph below in figure 17 shows the initial MATLAB script's recognition rate of the gesture "Out & In" as a function of sampling rate. At a (resampled) 80Hz, the MATLAB script is able to recognize all our representative trials of "Out and In". As the (resampled) frequency is lowered to 10Hz, the recognition rate drops to below 40%.

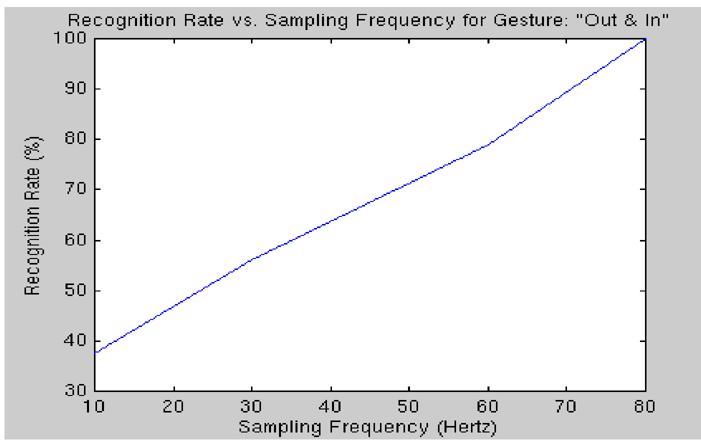


Figure 17: Recognition rate as a function of sampling frequency

The program's inability to find starting and end points consistently is evident from the graphs below. Figure 18 the left shows a typical "Left Shift" while figure 19 shows a typical misinterpretation of "Left Shift" by the real-time program:

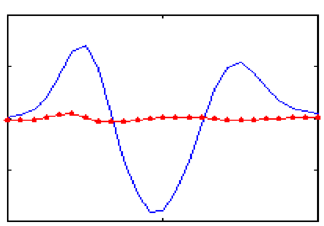


Figure 18: Typical "Left Shift"

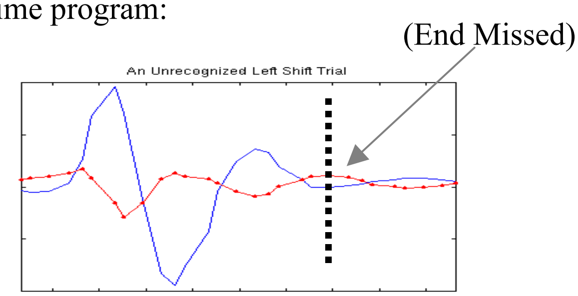


Figure 19: Typical misinterpretation of "Left Shift"

In summary, our piece-wise approach performs well given a sampling rate on the order of 100 Hertz and the assumption that gesture start and end points can be detected. Since we were not able to meet these conditions in our real-time PC testing, we decided to explore a different approach.

8.5 Threshold Based State Machine

In parallel with our work on the statistical approach to pattern recognition, we worked on implementing a threshold based state machine. We took an alternate approach to developing this algorithm, starting with the simplest implementation and iteratively adding features and refining it. See Appendix W for diagrams of this algorithm.

This algorithm works by identifying features as the data comes in, and checking to see if the order in which features have occurred matches known gesture feature orders. The features that are recognized are peaks and tilts. Features are defined as any time the accelerometer signal goes above or below preset threshold values. When the signal is out of the threshold range for a short time, it is a peak. Tilts are when the signal stays above or below a threshold for a sustained period of time.

This approach was able to recognize certain gestures, like tilts, very well, but had trouble when it tried to recognize more complicated gestures which consisted of multiple features. In addition, it frequently confused similar gestures. For example both tilt-snap left and right shift begin with positive peaks in the x-axis, causing a high misinterpretation rate between the two. Our error rates were inflated also by the poor quality of much of our test data. See Appendix X for the results of a preliminary test of this system.

After a few initial PC implementations of this algorithm, we decided to try to implement this on the PDA. See section 9.2 for a further discussion of this approach.

9 PDA Development

9.1 Hardware: Interfacing the Accelerometer with the Palm III

There are two ways of connecting the accelerometer and the Palm III. The first is to sample data through the Hotsync serial port. Unfortunately, since the Hotsync and infrared ports share the same hardware, the accelerometer and infrared cannot be active at the same time on a device configured in this way. A more preferable solution is to connect the accelerometer directly to the PDA's processor, avoiding IR-port interference. We chose to implement the first approach for our prototype because it required less time and was easier to construct. Good "hands-on" instructions of how to implement this were provided by software developer Till Harbaum.

The accelerometer we used was part of an ADXL202EB evaluation board. To this we added a 120kΩ resistor to set the accelerometers duty cycle to 1ms and two 1 μF capacitors (one for each channel x and y) to set the bandwidth of each channel to 50Hz.

Before directly modifying the Palm III, we wanted to verify that our approach worked. To do so we modified a Hot Sync Cradle and used that for a removable connection between accelerometer and PDA (see Fig. 20 and App. Y). Once we had verified the operation of the accelerometer-Hotsync port connection, we soldered the accelerometer directly to the PDA, as seen in figures 21 and 22. See Appendix Z for further pictures and schematics.

Figure 20: Modified Hotsync cradle

In addition to connecting the accelerometer and PDA, we also added a button at the top of the backside of the PDA to facilitate one-handed use of the device. We wired this button in parallel with the Address Book button included on the Palm Pilot. This button has a number of potential uses, such as zeroing the accelerometer or starting and stopping recognition.

Figure 21: PDA being wired

Figure 22: Palm III Final Prototype

9.2 Software: Implementing the Gesture Recognition Algorithm

9.2.1 Recognition Algorithm

Since our gesture recognition program would have to be running while other programs are running we were interested in limiting the total amount of memory and processor power used by our algorithm. To do this we implemented the simplest possible solution by taking our simple state machine implementation described in section 8.5 and making it even simpler. By defining the gesture set that we wanted to recognize as the gestures, which are comprised of single features only, i.e. tilts and tilt-snaps, we eliminated the need to remember sequences of features and were able to pare our algorithm down to the state machine only. Figure 23 shows a diagram of this algorithm.

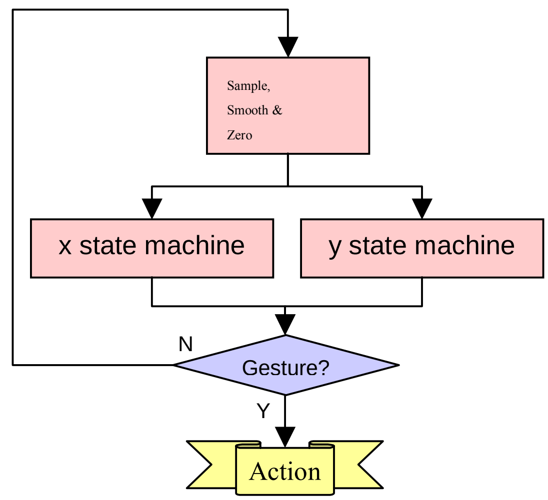


Figure 23: Palm implementation algorithm

Once we had implemented this algorithm and were able to able to test the prototype, one problem we encountered was that when performing gestures the "home" hand position that the user returned to varied. Thus the user might after a while be inadvertently performing a tilt gesture just due to hand position drift. To compensate for this we modified our algorithm to filter out constant components of the signal. The first thought of how to do this was to zero the signal using the average of the last 20 readings that occurred when no gesture was being performed. However, we did not want to have to keep track of that many points, so instead we simulated an average by adding a $1/20^{th}$ of the current reading to a variable while subtracting $1/20^{th}$ of the average of that variable from it. Using this value to zero the device allows the user to move gradually without having tilts eventually be recognized. Appendix AA shows the core code necessary for this implementation of a gesture recognition system.

9.2.2 Applications

GestRec

This program was written to facilitate the development of the recognition algorithm. The main window, shown in figure 24, graphically displays the x and y acceleration curves while printing the x and y values on the bottom of the screen. Any gesture that is recognized is reported along the bottom line. Both the zeroed lines and the direct readings from the accelerometer can be displayed.

Figure 24: GestRec main screen

Figure 25: GestRec preferences

Through the preferences screen (Fig. 25) the user can set the threshold levels for the x and y state machine and the sensitivity of the floating zero. The user can also choose to have the floating zero and to turn of the displaying of the non-zeroed line.

Accuracy

This application, shown in 26, was written to facilitate testing of the device. In this program, the user indicates the start of a gesture by tapping a button onscreen. Once the system recognizes a gesture it pops up a dialog to verify its interpretation. Statistics are kept for each gesture and reported onscreen in a table. Above the table, the x and y acceleration waveforms are displayed.

Figure 26: Accuracy

Figure 27: Puzzle

Puzzle

For a demonstration of our gesture recognition system, we modified the standard puzzle application to allow the tiles to be moved using tilt-snaps. To provide feedback to the user a graphical box was added in the lower left hand corner. A line originating from the center points in the direction the user is tilting the device, with its length reflection the magnitude of the acceleration. This can be seen in figure 27.

10 Final Evaluation

10.1 Reliability & Consistency

- *The system should recognize three or more gestures.*

Our final gesture recognition system could recognize tilts and tilt-snaps in four directions for a total of eight gestures.

- *When a gesture is performed it should be recognized correctly at least 88% of the time and must not be recognized as a different gesture more than 9% of the time.*

We did not have time to extensive test our device in a large test population, but we were able to get an idea of our system's abilities through a test performed by us three group members. This test was designed similarly to our test of the graffiti writing system's accuracy. Each group member performed each gesture 35 times in a row to get a general idea of the ease with which each gesture could be correctly performed. We then performed each gesture in a more realistic mixed order with each gesture being performed a total of 10 times. We were able to achieve a very high rate of 97.3% with the gesture with the worst recognition rate, tilt-snap left, still above 88% with a recognition rate of 89.5% and a misinterpretation rate of 8.6%. See Appendix BB for the full test results.

- *The system should reject all motion when not enabled.*

Since the accelerometer data is not being sampled and the accelerometer is not powered when the gesture recognition system is not enabled, there is no chance that a motion will be interpreted as a gesture unless the system is enable.

10.2 Acceptability

- *The set of actuating gestures will be rated by a group of potential users according to the following criteria: comfortability, intuitiveness, acceptability, recognizability and consistency. Each gesture should score higher than 3 on a scale of 1 to 5.*

All the gestures our system recognizes do not meet this specification as tilt-snap up and tilt-snap down were rejected in the project group's initial evaluation. However, the other six gestures passed this specification. Since we were able to recognize more than three gestures that passed this specification, we feel we were able to fulfill this requirement.

10.3 Safety

- *Gestures should be ergonomically safe to perform according to the guidelines outlined by NIOSH, Liberty Mutual, and Dartmouth's ENVHS office.*

Since our gesture recognition system allows the user to dynamically adjust recognition requirements we feel that we have followed the guidelines as stated in Appendix O. The user's ability to dynamically zero the x and y signal's allow the user to choose a posture which is most comfortable for them and to maintain a neutral wrist alignment. The floating zero allows the user to change their position while using the device. Since the user can adjust the threshold at which gestures are recognized they can choose a setting that does not make the gestures require excessive force or awkward motion. Also, our gesture selection process outlined in section 7.2 prevented us from selecting gestures which were awkward or would cause pain.

10.4 Recognition Speed

- *The time between the end of a gesture's performance and the accompanying action should not exceed 0.4 seconds.*

Since our algorithm does not perform any analysis of the signal once the end of a gesture has been recognized, and since the end of a gesture is recognized at the end of the performance of a gesture, there is no delay between the end of the gesture performance and the accompanying action.

10.5 Device Feasibility

- *The memory used by the gesture recognition system should not exceed 200KB of PDA storage memory and 9.6KB of dynamic memory.*

When gesture recognition control was added to the Puzzle program, the application size increased from 4.92KB to 8.08KB. We can conclude that the base size of the gesture recognition software takes about 3.16KB of storage memory.

To evaluate the amount of dynamic memory used, the following chart lists each data type used along with the total number of instances and total memory used by the algorithm.

Data Type	Instances	Memory Per Instance	Total Memory
int	19	2 bytes	38 bytes
UInt16	2	2 bytes	4 bytes
UInt32	3	4 bytes	12 bytes
Total			54 bytes

This shows that the total amount of dynamic memory used is also far less than the 9.6KB allowed.

- *While the sensor is in use, it should not consume more than about 0.5mA.*

When powered with 3 volts, the accelerometer draws 0.4 mA of current (Harbaum).

10.6 Economic Analysis

- *The cost of adding a gesture recognition system to a modern PDA for mass production should not exceed $15.*

According to Analog Devices, an added accelerometer would be the third most expensive component in a PDA after the display and the microprocessor. Analog Devices can produce accelerometers at a price of $2 to $2.50 per unit in quantities on the order of 1 million units. The added capacitors and resistors plus other manufacturing costs bring to total cost to around $3. Assuming adding a gesture recognition system will justify a $15 increase in device price, adding the gesture recognition system would provide a profit of $12 per unit.

11 Budget

The project was completed using roughly a quarter of the 5,000 budget allowed. An itemized list of our expenses appears below:

Quantity	Item	Expense(dollars)
3	Handspring Visor Deluxe	747.94
1	Palm Pilot IIIxe	199.00
1	Microsoft Visual Studio	229.00
1	Code Warrior Palm Development Platform	120
~15	Color Overhead Projector Slides	12.00
-	Miscellaneous Expenses	31.15
Total		**1339.09**

12 Future Work

In order to fully take advantage of our gesture recognition system's capability, there is work to be done in the following areas.

12.1 Interaction Design

In order to make our device as user friendly as possible it would be useful to obtain feedback from a range of potential users. Now that we have a working prototype we can provide the users with a concrete example of what we are envisioning and hopefully obtain more meaningful data than we were able to get from our original survey. Some of the questions that need to be answered include where the optimal location for a button or buttons to supplement a gesture recognition system in enabling one-handed use is, what actions each gesture should cause, and is the currently implemented system sufficient to the users needs or will more gestures be needed. In addition, tests could be performed to determine what form of feedback about the device's position would be most useful for users.

12.2 Software Design

To facilitate software development the main gesture recognition routines should be encapsulated in a shared library. In the event that this system is fully adopted, the next step would be to fully incorporate it into the operating system in a similar manner to Graffiti.

To fully take advantage of the usefulness of this system, a suite of gesture recognition enabled programs would need to be created. This suite would include the standard Palm applications (Address Book, To Do List, Date Book, etc.), plus whatever other applications are particularly well suited for gestural control. Once this has been done, a launcher program would need to be written that would enable the user to navigate between the programs using gestures. Then the user would be able to access many of the device's features without ever removing the stylus.

12.3 Hardware Design

To eliminate the conflict with the infrared port, the accelerometer needs to be connected directly to the processor instead of to the Hotsync port. If this product is to be mass-produced, the accelerometer needs to be incorporated into the device and placed directly onto the main circuit board. Also, work needs to be done to investigate how to interface the accelerometer with other PDA types, like the Handspring Visor.

12.4 Prototype testing

Due to lack of time we never got the chance to have real users to try the prototype. The group members performed the prototype algorithm accuracy test only. In order to make the accuracy test more accurate and statistically sound, data from a larger number and wider range of users is necessary.

13 Conclusions

Several conclusions can be reached from our study of the feasibility of having gestures as an additional input method for PDAs and our implementation of a gesture recognition system for the Palm III.

Simpler is Better

Our initial algorithm research and implementations were of more complex, difficult algorithms. We found in the end that this complexity was unnecessary that a simple approach worked much better. Our final gesture recognition software prototype is simple state machine, requiring very little memory usage, and is easy to maintain due to its simplicity and small size. Despite this it gives us a high accuracy rate. Other approaches in which we attempted to recognize more complicated gestures had much lower recognition rates. Ironically, the surveys we conducted indicate that people do not want to perform the more complicated gestures anyways, preferring simpler gestures.

First get our hands dirty and then get analytical

At the start of our project we took a very high level approach to our problem, conducting surveys to verify a need for our system and researching broad algorithm categories. We would have perhaps been better served had we jumped right in and gotten a better idea of what aspects of the problem were difficult to solve. Once we decided to produce an implementation on the Palm as opposed to just a PC simulation of an algorithm, the building of the device and implementation of the software went very quickly. Had we attempted this earlier we would have been able to produce a much more finished product and not spent so much time investigating paths that were unnecessarily complex. Having a prototype to show would have allowed us to conduct much more meaningful surveys. An example of this is how after our project proposal one of the review board members said they would not use our system to scroll through their Address Book program since they had too many entries, but later at our final progress report after seeing the prototype mentioned how useful it would be for scrolling through their Address Book.

Better test data

Another problem we came across was that the test data we gathered was rather poor, and not conducive towards determining an algorithm's actual recognition rate. For example, even tilts, which are very simple to perform when feedback about hand position is given to the user, were misinterpreted due to the user tilting the device in more than one axis or not returning to their original hand position. Since we had a larger number of subjects performing each gesture a limited number of times, the data we gathered was highly varied and therefore harder to recognize than it should have been. A better approach would have been to have a smaller number of users perform each gesture a larger number of times while watching the waveforms on the screen to get feedback about the gestures they were performing. While better data would not have fixed all the problems with some of our algorithms, it would have allowed us to focus on what actually was not working as opposed to making them work for flawed data.

14 Acknowledgements

We would like to thank the following people for a invaluable support and encourage in our efforts of solving this problem.

Analog Devices, our sponsor, and especially:

Jim Doscher, Harvey Weinberg and, Christophe Lemaire.

Professor John Collier, our advisor at Dartmouth College, Thayer School of Engineering.

Software developer Whit Kelly at SignalQuest.

Software Engineer Michael Fromberger at Dartmouth College, Thayer School of Engineering.

Professors Eric Hansen, Ulf Osterberg, and Ted Cooley, from the Thayer School of Engineering.

The instrument room folks at Dartmouth College, Thayer School of Engineering.

Software developer Till Harbaum.

Our brave test subjects and survey respondents.

Many patient friends.

15 Works Consulted

Amazon.com, *Electronics > Categories > Handhelds & PDAs* <http://www.amazon.com/exec/obidos/tg/browse/-/172594/ref=e_hp_cs_2_1/103-0576145-3853446>

Aslam, Javed. “Computer Science 44, Artificial Intelligence course notes”, Dartmouth College, Winter 2001.

Bartlett, Joel F. *Rock ‘N’ Scroll is Here to Stay*. 2000/3. 10 March. 2001. <ftp://gatekeeper.dec.com/pub/DEC/WRL/research-reports/WRL-TR-2000.3.pdf>

Citizen, *Specifications and pictures on Citizen DataSlim2*. 16 Oct. 2000. <http://www.citizen.co.jp/english/release/00/dataslim2/e0427ds.htm>

Cooper, Allan. *The Inmates Are Running the Asylum.* Indianapolis. Macmillan Computer Publishing, 1999.

Ed. Dale Purves et al. *Neuroscience.* Sunderland. Sinauer Associates, 1997.

EDECAP Services, *White Paper on Palm vs. WidowsCE*. 1999. 12 Oct. 2000. <http://www.edecap.com/white.htm>

Harbaum, Till, *A tilt sensor for the palm pilot*. 10 March. 2001. <http://bodotill.suburbia.com.au/>

IBM, *Specifications and pictures on IBM Wrist Watch*. 16 Oct . 2000. <http://www.research.ibm.com/WearableComputing/factsheet.html>

Information on piezo-electric accelerometers:
13 Nov. 2000. http://www.endevco.com/piezopak.html
13 Nov.2000.<http://www.it.kth.se/edu/gru/Fingerinfo/telesys.finger/Mobile.VT98/ACH-04-05.html>
13 Nov. 2000. <http://www.pcb.com/products/svs/svs353b17.html>

Information on the advantages of gyroscopes.
13 Nov. 2000. <http://www.motusbioengineering.com/accelerometer-limitations.htm>

Info on pangrams.
Feb. 2000. <http://www.navvy.com/pdds/pangram.html>.

Kelly, Whit. Personal Communication. Dartmouth College. 2001.

Krause, W.L., J.C. Leiter, S.M. Tenney, and J.A. Daubenspeck. “Acute hypoxia activates human 8-12 Hz physiological tremor”. *Resp Physiol* 123 (2000): 131-141.

Leydon, Krispin. “Mobility, Ergonomics & Multitasking in Text Entry.” Hanover: Dartmouth College, 2000.

United States. Liberty Mutual. *Office Ergonomics* (pamphlet series). Krames Communications, 1996.

Lundstrom, Per. “Design Rules for Personal Digital Assistant Applications.” Stockholm: Stockholm University, February 1999.

NIOSH, *Carpel Tunnel Syndrome*. 10 March. 2001. <http://www.cdc.gov/niosh/ctsfs.html>

Rekimoto, Jun. “Tilting Operations for Small Screen (Tech Note)”. 10 March. 2001. <http://www.acm.org/pubs/articles/proceedings/uist/237091/p167-rekimoto/p167-rekimoto.pdf>

Rodgers, Suzanne ed. *Ergonomic Design for People at Work*. Eastman Kodak Company Ergonomics Group Health and Environmental Laboratory. Van Nostrand Reinhold. New York, 1986.

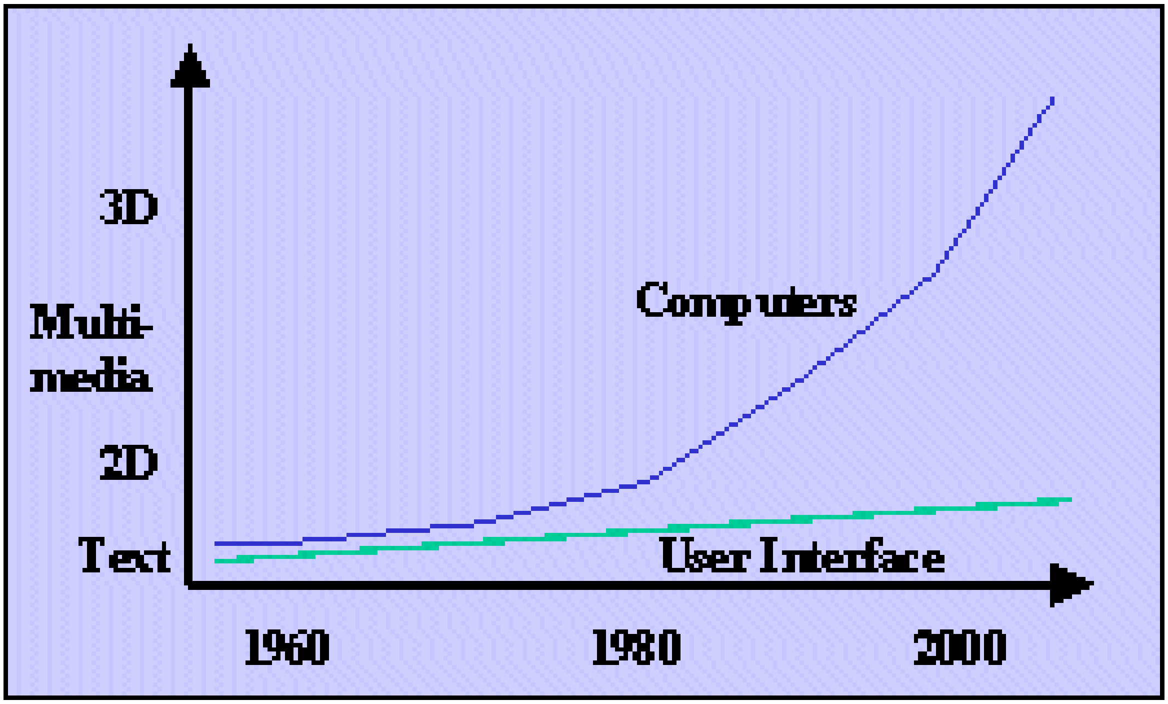
Rus, Daniela. “Computer Science 44, Artificial Intelligence course notes”, Dartmouth College. Winter 2001.

Sensable Technologies. *What is 3D Touch?*. 17 Oct. 2000. <http://www.sensABLE.COM/3D_Touch.htm>

Palm. *Specifications on Palm 1000*. 16 Oct, 2000. <http://www.dtek.chalmers.se/groups/pilot/doc/pptdg/ch01.htm>

Warakagoda, Narada. *Definition of Hidden Markov Model*. 17 Nov. 2000. <http://jedlik.phy.bme.hu/~gerjanos/HMM/node4.html>

Appendix A: Computational Power & User-Interface Capability Vs. Time

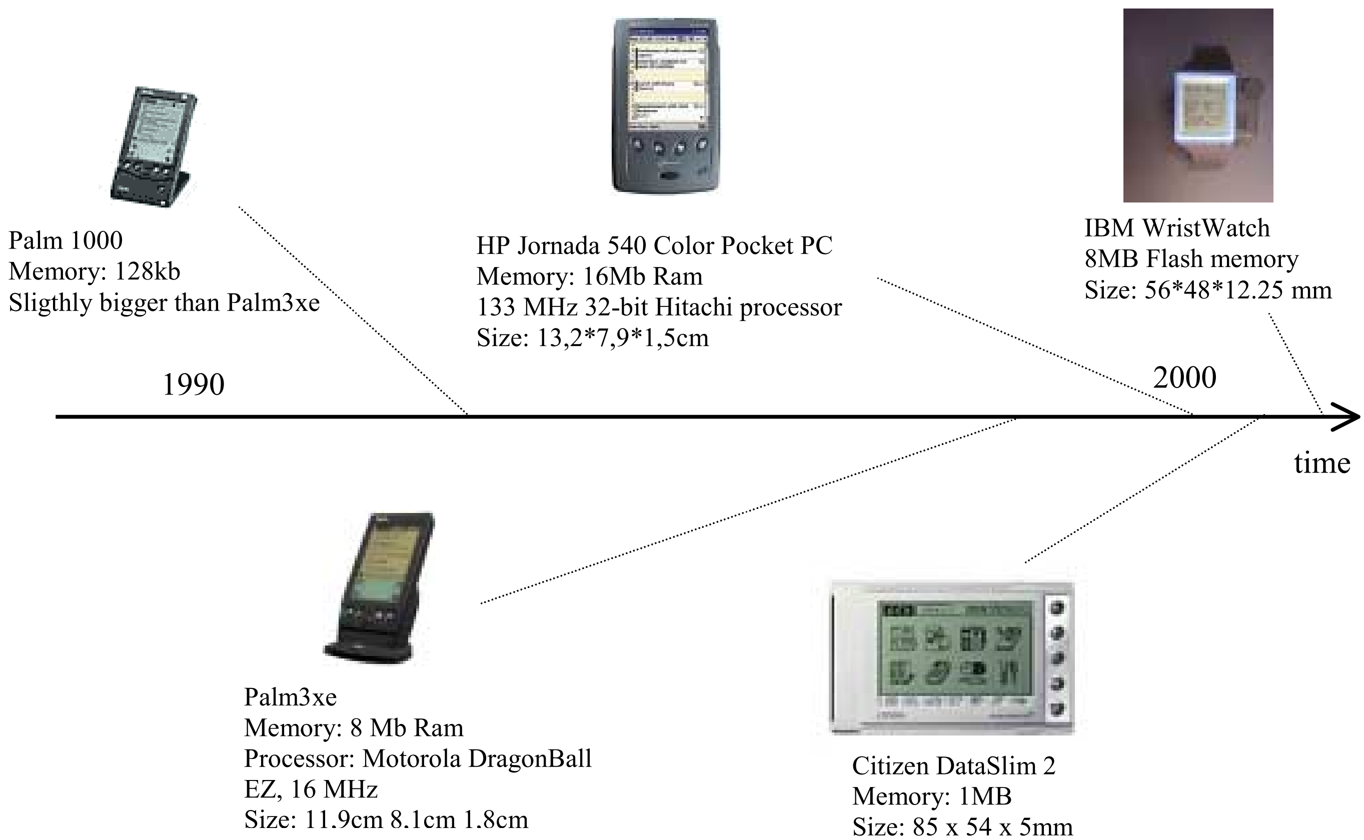


(Graph provided by Sensable Technologies)

The User Interface Bottleneck

This graph illustrates that in general, computational power increases exponentially while interface capability rises linearly. As the power of hardware and software grows, the need for quality user-interfaces that harness this power becomes increasingly obvious.

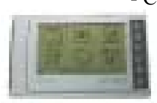
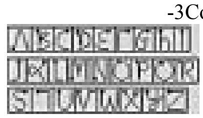
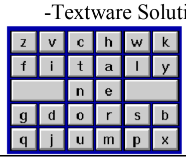
Appendix B: PDA Development Timeline



Comment

The timeline above shows that PDA's are getting smaller and more powerful.

Appendix C: Modes of Input for Mobile Informational Devices For Info Access (Control and Navigation Functions)

Mode	Example	(+) Advantages	(-) Disadvantages
Micro Keyboard *Like a computer keyboard, only smaller.*	Series5 - Psion 	• Wide range of options • Tactile feedback • Familiar • Somewhat mobile	• Stationary surface & two hands required • Operation obscures control surface • Many actuators required • Size limited
Thumb Keyboard *Keyboard, but designed to be operated by two thumbs*	Blackberry - RIM 	• Wide range of options • Tactile feedback • Familiar • Mobile	• Operation obscures control surface • Many actuators & both hands required • Size limited
Buttons	DataSlim - Citizen 	• Simple • Little space required • Tactile feedback • Mobile	• Limited option range • Operation obscures control surface
Chording *Buttons pressed in combinations ("chords").*	DataEgg -E2Solutions 	• Few actuators, wide range of options • Little space required • Tactile feedback • One-handed operation • Mobile	• Extremely poor learnability • Requires memorization
Writing Recognition *Drawn symbols interpreted as commands.*	Palm Pilot "Graffiti" -3Com 	• Familiar • Quick to learn • Mobile	• Stable surface required • Two hands required • No immediate feedback • Slow • Care and skill required • Visible screen required
"Soft" Buttons *Software buttons appearing on a touch screen*	Fitaly -Textware Solutions 	• Wide range of options • Adjustable button-command mapping • Quick to learn • Mobile	• No tactile feedback • Operation obscures control surface • Two hands required • Size limited • Visible screen required
Speech Recognition *Spoken words interpreted as commands*	***Currently no mass-market mobile solution***	• Familiar • No hands required • Size-independent • Mobile	• Requires training & a quiet environment • Expensive • Privacy an issue • Can't multitask vocally • Care and skill required
Gesture Recognition *Hand gestures interpreted as commands*	***Currently no mass-market mobile solution***	• Familiar • Size-independent • Mobile	• Limited option range • Care and skill required

Appendix D: Multiple Inputs, Parallel Paths

The best current PDAs make use of a wide variety of input modes *in combination.*

"Series5" (Psion)

User Input

- Keyboard
- Physical Buttons
- Stylus/Touch-Screen
  - Handwriting Recognition
  - "Soft" Buttons
  - "Soft Keyboards"
  - Windowing & Menus
- Voice Recording

Additional User Input

- From other devices via Infrared & Modem

Handspring "Visor"

"Best PDA" PC World, 7/2000
"Smart Choice", Smart Computing, 4/2000
"Top Rated", Family PC, 5/2000

User Input

- Physical Buttons
- Stylus/Touch-Screen
  - Handwriting Recognition
  - "Soft" Buttons
  - "Soft Keyboards"
  - Windows & Menus

Additional User Input

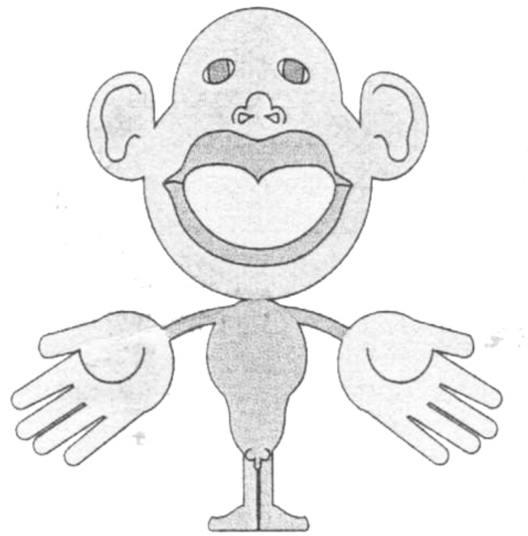
- Keyboards
- SpringBoard Modules
  - Voice Recording, Cell Phones, Cameras, etc.
- From other devices via Infrared, Modem

Appendix E: PDA Shapes and Sizes

No one knows what the form of future PDAs will be. Already there exist PDAs in the forms shown below. Some, such as iButton's "Decoder" Java Ring, Citizen's "DataSlim2" and IBM Research's Linux watch have frontal surface areas smaller than 3.5 cm^2 – the size below which (according to our survey data) a set of 5 buttons becomes unusable.

CW from top: "Decoder" Java Ring (iButton) , "DataSlim2" (Citizen), "Blackberry" (RIM), "DataEgg" (E2Solutions),"QuickLink" (WizCom Technologies), "Clio" (HP), "9000il" (Nokia), "Linux Wrist Watch" (IBM), "Visor" with Cell Phone module (Handspring).

Appendix F: The "Homunculus"



(Purves, 159)

This "homunculus" depicts graphically the space allocated in the brain for various regions of the body. Larger body features correspond to larger regions of the brain. Note that

Comparatively huge portions of the brain are dedicated to the hands and mouth.

This suggests a biological rational for the synergistic use of manual and vocal input in man/machine interfaces.

Appendix G: Graffiti Writing Samples

Recognition Rate Benchmark Testing

Each of the sentences below contains all the letters of the alphabet:

The quick brown fox jumps over the lazy dog
The five boxing wizards jump quickly
Pack my box with five dozen liquor jugs.
Waltz, dumb nymph, for quick jigs vex.
Sphinx of black quartz judge my vow.
Sympathizing would fix Quaker objectives.
Crazy Fredericka bought many very exquisite opal jewels.
Jaded zombies acted quaintly but kept driving their oxen forward.
Six big juicy steaks sizzled in a pan as five workmen left the quarry.
The sex life of the woodchuck is a provocative question for most vertebrate zoology majors.

Available at <http://www.navvy.com/pdds/pangram.html>

Appendix H: Graffiti Recognition Rate Benchmark Test Results

Individual Letter Trials

Letter	# iters	# errors	% error
A	105	2	1.90
B	105	27	25.71
C	105	20	19.05
D	105	16	15.24
E	105	18	17.14
F	105	4	3.81
G	105	15	14.29
H	105	9	8.57
I	105	0	0.00
J	105	20	19.05
K	105	12	11.43
L	105	9	8.57
M	105	35	33.33
N	105	3	2.86
O	105	9	8.57
P	105	30	28.57
Q	105	36	34.29
R	105	25	23.81
S	105	2	1.90
T	105	4	3.81
U	105	12	11.43
V	105	15	14.29
W	105	12	11.43
X	105	9	8.57
Y	105	24	22.86
Z	105	9	8.57
Total	**2730**	**377**	**13.81**

Essay portion summary

Letter	# iters	# errors	% error
A	75	2	2.67
B	33	1	3.03
C	39	9	23.08
D	42	2	4.76
E	120	19	15.83
F	39	4	10.26
G	30	5	16.67
H	39	2	5.13
I	99	0	0.00
J	30	2	6.67
K	33	4	12.12
L	39	4	10.26
M	36	1	2.78
N	42	3	7.14
O	93	7	7.53
P	30	3	10.00
Q	30	5	16.67
R	69	14	20.29
S	60	4	6.67
T	75	1	1.33
U	60	2	3.33
V	36	6	16.67
W	30	1	3.33
X	30	8	26.67
Y	39	9	23.08
Z	33	0	0.00
Total	**1281**	**118**	**9.21**

Total Letter Results

Letter	# iters	# errors	% error
A	180	4	2.22
B	138	28	20.29
C	144	29	20.14
D	147	18	12.24
E	225	37	16.44
F	144	8	5.56
G	135	20	14.81
H	144	11	7.64
I	204	0	0.00
J	135	22	16.30
K	138	16	11.59
L	144	13	9.03
M	141	36	25.53
N	147	6	4.08
O	198	16	8.08
P	135	33	24.44
Q	135	41	30.37
R	174	39	22.41
S	165	6	3.64
T	180	5	2.78
U	165	14	8.48
V	141	21	14.89
W	135	13	9.63
X	135	17	12.59
Y	144	33	22.92
Z	138	9	6.52
Total	**4011**	**495**	**12.34**

Appendix I: Mobile Gesture Based Interface Survey

Mobile Interface Design Project - Survey

As mobile devices become smaller and more feature-filled, developing intuitive ways to control them becomes increasingly difficult – and increasingly important.

We are a student design team committed to developing a more natural, intuitive way to control small digital devices, and we need your help. Your answers to some or all of the following questions would be a great assistance to our effort.

Thanks,

Andrew, Krispin & Jonas

1. Background

Name: [ ]

Occupation: [ ]

Age:

○ Under 20 ○ 20-30 ○ 31-40

○ 41-50 ○ 51-60 ○ Over 60

Sex: ○ **Female** ○ **Male**

Do you own a personal digital assistant (PDA)? ○ **Yes** ◉ **No**

Do you own any other portable digital devices (cell phones, pagers, etc.)?
If so, what device(s) do you own?

If you don't own a PDA, please skip to [Section 5](#)

---

2. PDA Usage

What PDA's do you own?

How long have you been using your PDA?

○ Less than 6 months ○ 6 months to 2 years ○ More than 2 years

How often do you use your PDA?

○ Less than once per day ○ 1 to 5 times per day ○ More than 5 times per day

What applications do you use your PDA for?

Check all that apply:

- ☐ Address Book
- ☐ Date Book
- ☐ To-Do Lists
- ☐ E-Mail
- ☐ Games
- ☐ Note Taking
- ☐ Other (please specify):

---

3. PDA Satisfaction

What features of your PDA do you like most?

What features of your PDA do you like least?

What features do you find most frustrating?

---

4. PDA Input Methods

What is your primary mode of input for entering *information* into your PDA?

- ○ I don't; I enter information via computer, then download it to my PDA.
- ○ Stylus, handwriting
- ○ Stylus, "screen" keyboard
- ○ Keyboard
- ○ Other (please specify):

What is your primary mode of input for *program navigation and control?*

- ○ Buttons
- ○ Stylus, handwriting
- ○ Stylus, "screen" keyboard
- ○ Keyboard
- ○ Other (please specify):

Please rate your primary mode of control/navigation input

- Extremely useful & intuitive
- Very useful & intuitive
- Useful & intuitive
- Not especially useful or intuitive
- Annoying & difficult to use

What would make navigating applications and controlling your PDA more intuitive?

Are there situations where you want to use your PDA, but can't, due to input restrictions?
If so, please elaborate on the situation(s) and restriction(s):

---

5. Motion-Based input

Suppose you could control a mobile digital device by moving it (up, down, left, right, tilting, shaking, gesturing, etc).

Would this capability be useful to you? Yes No

If so, in what context?

Would you like to have the option of controlling small devices, using motions or gestures as an input? Yes No

In what contexts do you think device-motion as a form of user-input would be most useful?

6. Further Assistance

We are looking for volunteers to help test any improved interface technology we develop. Are you interested? If you interested, and live near Hanover NH, please enter your e-mail address or phone number so that we can reach you:

Submit Results

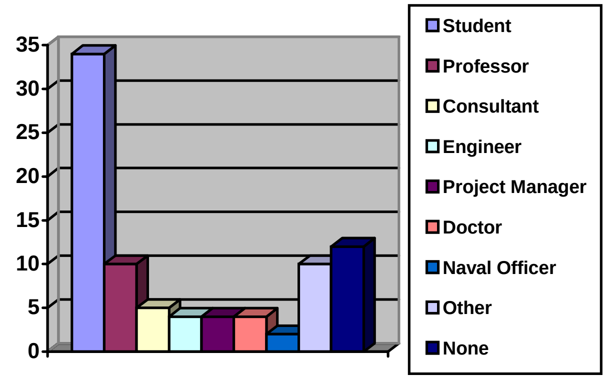
Appendix J: PDA Use Survey Results

There were 85 total responses to the survey.

Part 1: Background

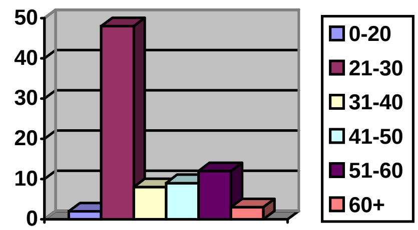
Occupation

Occupation	Number
Student	34
Professor	10
Consultant	5
Engineer	4
Project Manager	4
Doctor	4
Naval Officer	2
Other	10
None	12



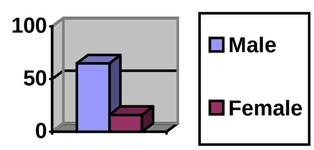
Age

Age bracket	Number	Percent
0-20	2	2.4%
21-30	48	58.5%
31-40	8	9.8%
41-50	9	11.0%
51-60	12	14.6%
60+	3	3.7%



Sex

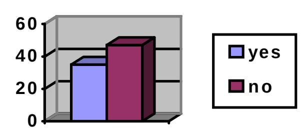
Sex	Number	Percent
Male	65	80.2%
Female	16	19.8%



Part 2: PDA Usage

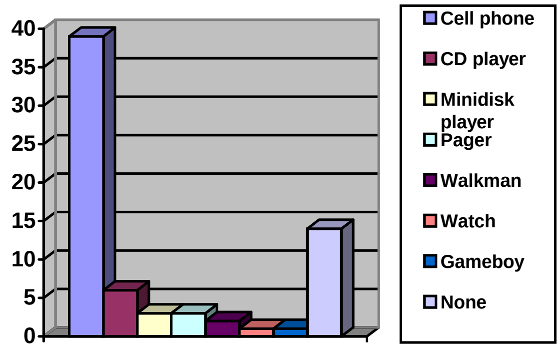
Do you own a personal digital assistant (PDA)?

Answer	Number	Percent
yes	35	42.7%
no	47	57.3%



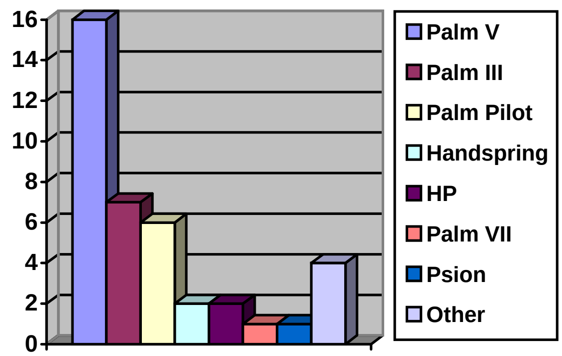
Do you own any other portable digital devices (cell phones, pagers, etc.)? If so, what device(s) do you own?

Device	Number
Cell phone	39
CD player	6
Minidisk player	3
Pager	3
Walkman	2
Watch	1
Gameboy	1
None	14



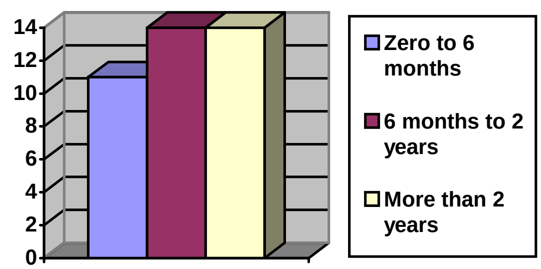
What PDA's do you own?

Brand	Number
Palm V	16
Palm III	7
Palm Pilot	6
Handspring	2
HP	2
Palm VII	1
Psion	1
Other	4



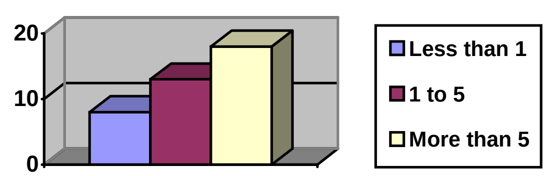
How long have you been using your PDA?

Length	Number	Percent
Zero to 6 months	11	28.2%
6 months to 2 years	14	35.9%
More than 2 years	14	35.9%



How often do you use your PDA?

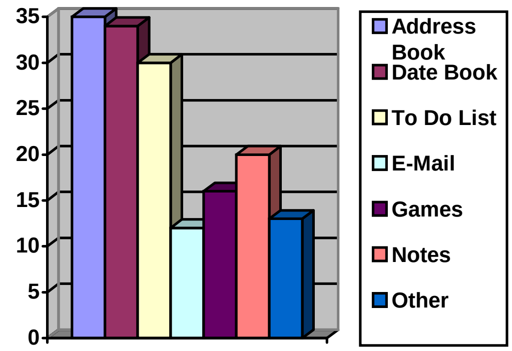
Uses per day	Number	Percent
Less than 1	8	20.5%
1 to 5	13	33.3%
More than 5	18	46.2%



What applications do you use your PDA for?

Total responses: 37

Application	Number	Percent
Address Book	35	94.6%
Date Book	34	91.9%
To Do List	30	81.1%
E-Mail	12	32.4%
Games	16	43.2%
Notes	20	54.1%
Other	13	35.1%



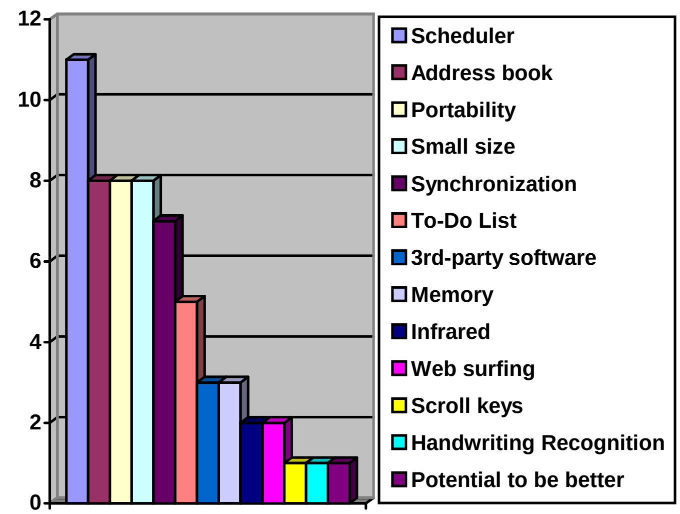
Popular other responses:

Calculator	4	10.8%
Web browsing	6	16.2%
Expenses	3	8.1%

Part 3: PDA Satisfaction

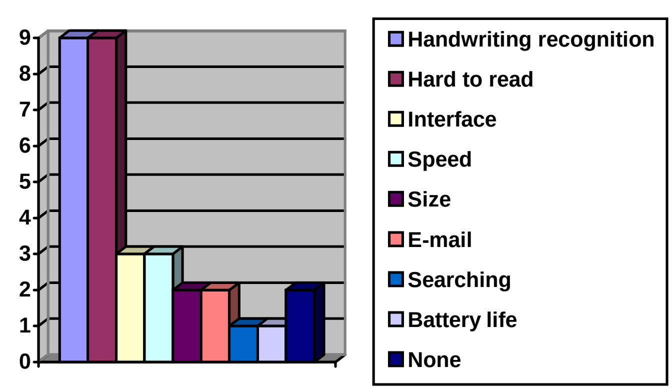
What features of your PDA do you like most?

Feature	Mentions
Scheduler	11
Address book	8
Portability	8
Small size	8
Synchronization	7
To-Do List	5
3rd-party software	3
Memory	3
Infrared	2
Web surfing	2
Scroll keys	1
Handwriting Recognition	1
Potential to be better	1



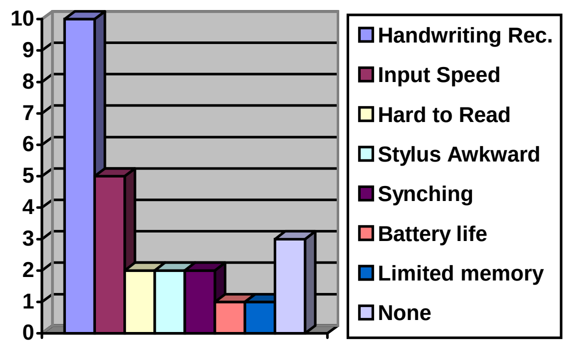
What features of your PDA do you like least?

Feature	Mentions
Handwriting recognition	9
Hard to read	9
Interface	3
Speed	3
Size	2
E-mail	2
Searching	1
Battery life	1
None	2



What features do you find most frustrating?

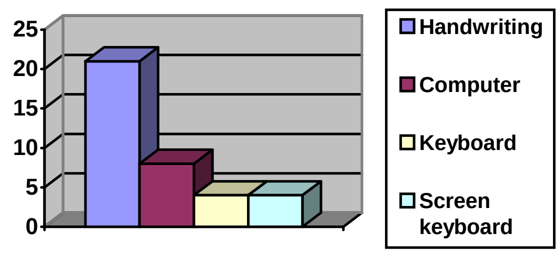
Feature	Mentions
Handwriting Rec.	10
Input Speed	5
Hard to Read	2
Stylus Awkward	2
Synching	2
Battery life	1
Limited memory	1
None	3



Part 4: PDA Input Methods

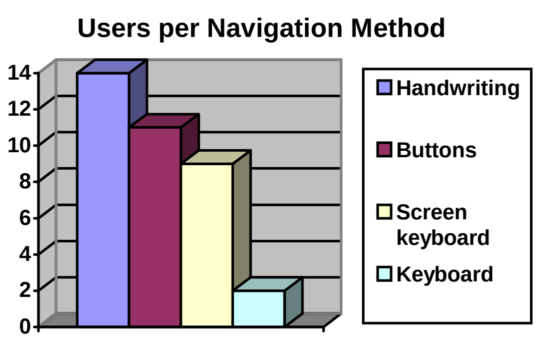
What is your primary mode of input for entering *information* into your PDA?

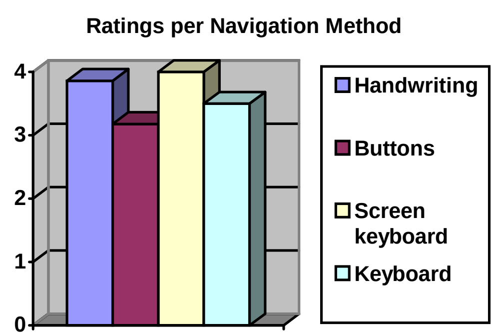
Method	Total	Percent
Stylus - Handwriting	21	56.8%
Computer	8	21.6%
Keyboard	4	10.8%
Stylus – "screen" keyboard	4	10.8%



What is your primary mode of input for *program navigation and control?*

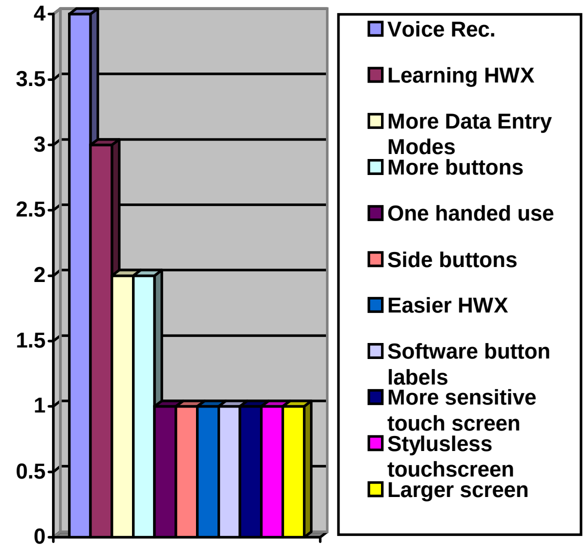
Method	Number	Percent	Average Rating
Stylus – handwriting	14	38.9%	3.86
Buttons	11	30.6%	3.18
Stylus – keyboard	9	25.0%	4.00
Keyboard	2	5.6%	3.50





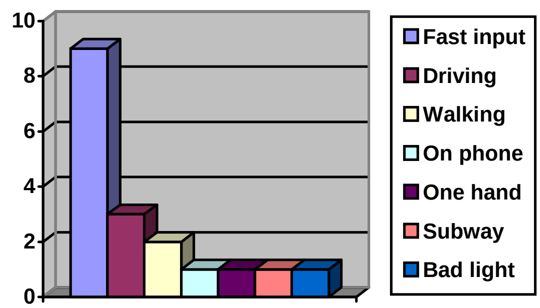
What would make navigating applications and controlling your PDA more intuitive?

Suggestion	Mentions
Voice recognition	4
Learning handwriting recognition	3
Additional modes of data entry	2
More buttons	2
One handed use	1
Buttons on the side	1
Easier Handwriting recognition	1
Software button labels	1
More sensitive touch screen	1
Ability to use touch screen without stylus	1
Larger screen	1



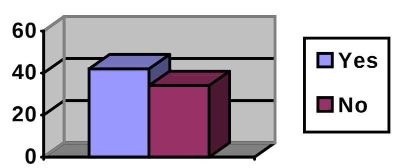
Are there situations where you want to use your PDA, but can't, due to input restrictions

Place	Mentions
When fast input is required	9
Driving	3
Walking	2
Talking on the phone	1
When one hand is available	1
Subway	1
In bright or dark light	1



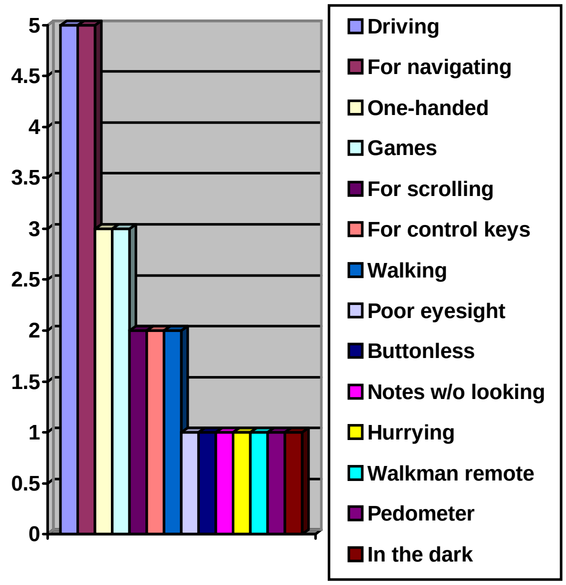
Would this capability be useful to you?

Answer	Number	Percent
Yes	42	55.3%
No	34	44.7%



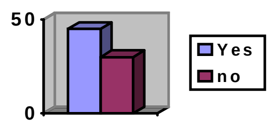
If so, in what context?

Context/use	Mentions
Driving	5
For navigating	5
One-handed operation	3
Games	3
For scrolling	2
For control keys	2
Walking	2
For people with bad sight	1
Avoids using buttons	1
Take notes without looking	1
When you're in a hurry	1
Remote control for walkman	1
As a pedometer	1
In the dark	1



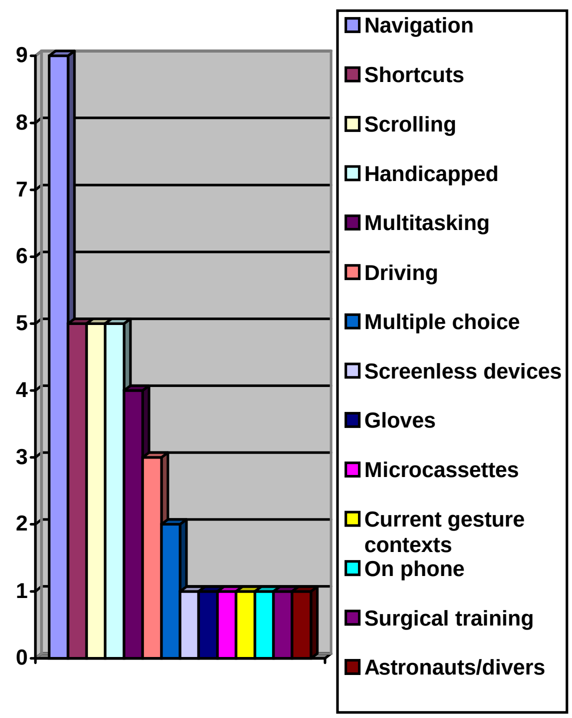
Would you like to have the option of controlling small devices, using motions or gestures as an input?

Answer	Number	Percent
Yes	45	60.0%
no	30	40.0%



In what contexts do you think device-motion as a form of user-input would be most useful

Context	Mentions
Navigation	9
For shortcuts to basic functions	5
Scrolling	5
For the handicapped	5
When multitasking	4
Driving	3
For multiple choice input	2
For devices without screens	1
When you have gloves on	1
Microcassette recorders (buttons too small)	1
Contexts where gestures are used already	1
When you're on the phone	1
For feedback during surgical training	1
For astronauts/divers	1



Appendix K: Button Feasibility Survey

Mobile Interface Design Project; Survey # 2

Name: [ ] Occupation: [ ] Age: [ ]

1) Digital Hand Tool with Screen

At what size does button use becoming frustrating?

○ A ○ B ○ C ○ D ○ E ○ F ○ G ○ H ○ I

2) Digital Hand Tool without Screen

At what size does button use becoming frustrating?

○ A ○ B ○ C ○ D ○ E ○ F ○ G ○ H ○ I

3) Gestural Control With and Without a "Start" Button

Assume you have a digital hand tool (PDA, calculator, radio, etc) that is controlled by hand gestures. For example, flicking your wrist left might control one action, while tilting your hand to the right might control another.

Now imagine that you have a choice:

> a) Operate the device's functions by pressing a button, doing the gesture, then releasing the button.
> b) Operate the device's functions by just doing the gesture, no button necessary.

Option "a" completely guarantees that the device's functions won't "go off" accidentally and provides you with tactile feedback that your action has been "read", but requires buttons and gestures.

Option "b" does not completely guarantee that device's functions won't go off accidentally but relies solely on gestures.

Which option would you choose? ○ A ○ B

4) Most Appropriate Use of Hand-Held Gestural Control

Digital hand tools are taking on a growing number of forms. In the world today there are digital devices that look like watches, rings, credit cards etc.

For what forms do you think a gestural control system would be the best option? (Keep in mind that buttons, knobs, voice commands and other types of input exist).

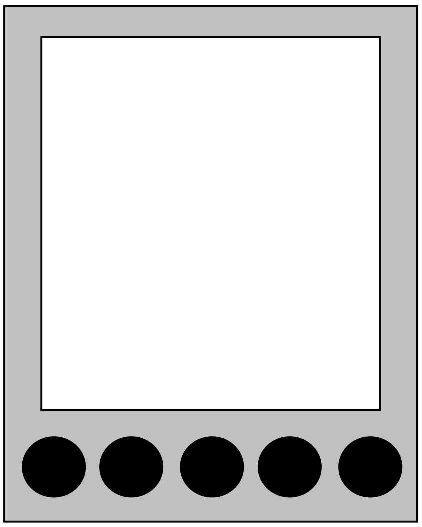
○ Ring Shape ○ Credit Card Shape ○ Watch Shape ○ Palm Pilot Shape
○ Cell Phone Shape
○ Other -- Please Elaborate: [ ]

Now imagine you have a gestural control system embedded into the shape you selected above. What on earth would you use it for?

PDA Size Templates

Users judged minimum button-operated PDA size based on the templates below:

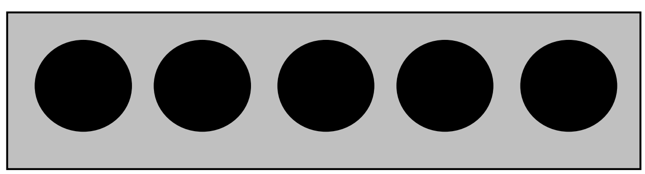
With Screen:



At the following sizes in cm X cm:

10 X 12.5
8 X 9.5
6 X 8
5 X 6.5
4.5 X 5.7
4 X 5
3.1 X 4
2.5 X 3.1
2 X 2.5

Without Screen:



At the following sizes in cm X cm:
10 X 2.5
7.5 X 2
6 X 1.5
5 X 1.2
4.6 X 1.2
4.2 X 1
.6 X 3

Appendix L: Mobile Gesture-Based Interface Survey #2

Name	Occupation	Age	Questions… Q1	Q2	Q3	Q4	Q5
LBS	student	30	D	D	No Button	Credit Card	Watch, Home Controller, "Ezpass" for ATMs
Doc Howe	retired	82	D	D	Button	Watch	Doctor Info, Access to Directions, Stock Market, Email
Sibby Howe	retired	81	D	D	Button	Credit Card	ToDoList, Reminder, Weather,Date,Directions
Eric Hansen	prof	47	E	F	Button	Credit Card	Calendar, Adresses Weather/Traffic Info
Leonard Parker	machinist	57	H	H	Button	Credit Card	House Controller
Tyler Smith	mad scientist	8	F	E	No Button	Cell Phone	Email Remote Controller
Ellen Kitchel	multipurpose	37	E	G	N/A	Credit Card	Remote Controller Computer Controller
Ayorkor	student	21	D	D	Button	Ring, Glove	Remote Controller
Jonathan	toddler	3	B	D	Button	Watch	Drawing, Toy Controller, Environment Controller
Shanna Davis	student	22	D	E	No Button	Ring, Watch	Remote control of house, Program switching
Average(a)	**student (p)**	**39 (a)**	**D (p)**	**D (p)**	**Button (p)**	**Credit Card (p)**	**Device or environment controller, mobile info access**
MostPopular(p)							

Appendix M: Use Contexts & User Personas

Gestural PDA input is likely to be the "best" mode of control in an extremely limited set of contexts—contexts where buttons, keyboards, styli and voice-activation are ineffective. These contexts are characterized by the following constraints:

- The user has only one free hand
- The PDA is on the order of 3.5 cm^2 (credit card sized)
- There is ambient noise, background conversation, or the need to speak while controlling PDA functions.

The following user personas illustrate several potential users and use contexts:

Dennis, IT Consultant
Dennis lives in Washington DC and rides the Metro to and from home every day. He enjoys surfing the web on his PDA during the trip. His train is usually packed, and he is annoyed because he can't hold onto the hand railing and read at the same time because he can't operate a stylus. Even when he gets a seat, the jarring of the train makes browsing through his downloaded articles a pain. Dennis reads in order to a) ameliorate the hassle of an uncomfortable and long commute, and b) keep up with the latest high-tech news.

Nathan, Accident Victim
Physical illness and neurological damage has temporarily (permanently?) made impossible Nathan's favorite activities; climbing hiking and traveling. To compensate for the loss, Nathan immerses himself in virtual worlds (computer games and the web), but is frustrated because he doesn't have the fine motor control in his hands necessary to navigate these virtual worlds with ease. Since he is exhausted most of the time, he would like to control his PC from bed.

Jaimie, Interface Designer for a Mobile Phone Company.
She is under pressure, as the internet content providers in negotiation with her employer want as much information as possible to be conveyed through mobile phone units, while the company's product design team insists consumers want a smaller phone package (with a tiny display). Jaimie is a perfectionist by nature, and the daunting challenges of a) channeling content initially meant for the web into a small display space and b) making that content intuitively accessible are driving her crazy.

Mason, Urban Landscaper/Hardscaper. To keep in contact with his clients, employees, and (expecting) wife, Mason needs to use a cell phone on the job. Mason wears heavy work gloves most of the time, and taking them off and putting them on each time he must use the phone interrupts his work flow. Bob wants to be able to make/take calls without changing gloves, in an environment filled with background construction noise.

Ms. Frost, Arctic Search & Rescue Team Member
Wants to operate a GPS system and access medical and geographic data while on the move & bundled up in scarf and gloves.

Appendix N: Generalized Navigation & Selection Commands

The navigation and selection commands for a large number of digital tools can be grouped into a set of five "generalized" functions: PREVIOUS, GO, NEXT, JUMP, and STOP. A control system that makes possible these five functions is capable of controlling any of the tools listed in the table below—tools that are (or could be) integrated into the functionality of a PDA:

TOOL	PREV	GO	NEXT	JUMP	STOP
Browser	Back	Refresh	Forward	Move between fields	Stop
Database	Previous Record	New Record	Next Record	Move between fields	Edit Record
Radio/TV Receiver	Channel Up		Channel Down	Scan Channels	Stop Scanning
Song Player	Previous Song	Play/Pause	Next Song		Stop
Movie Player	Previous Chapter	Play/Pause	Next Chapter	Show DVD Menu	Stop
Cell Phone	Go to Previous Number	Dial	Go to Next Number	Show Phone Number List	Hang Up
Application Switcher	Previous Application	Select Application	Next Application	Open Application Switcher	

Appendix O: Ergonomic Considerations

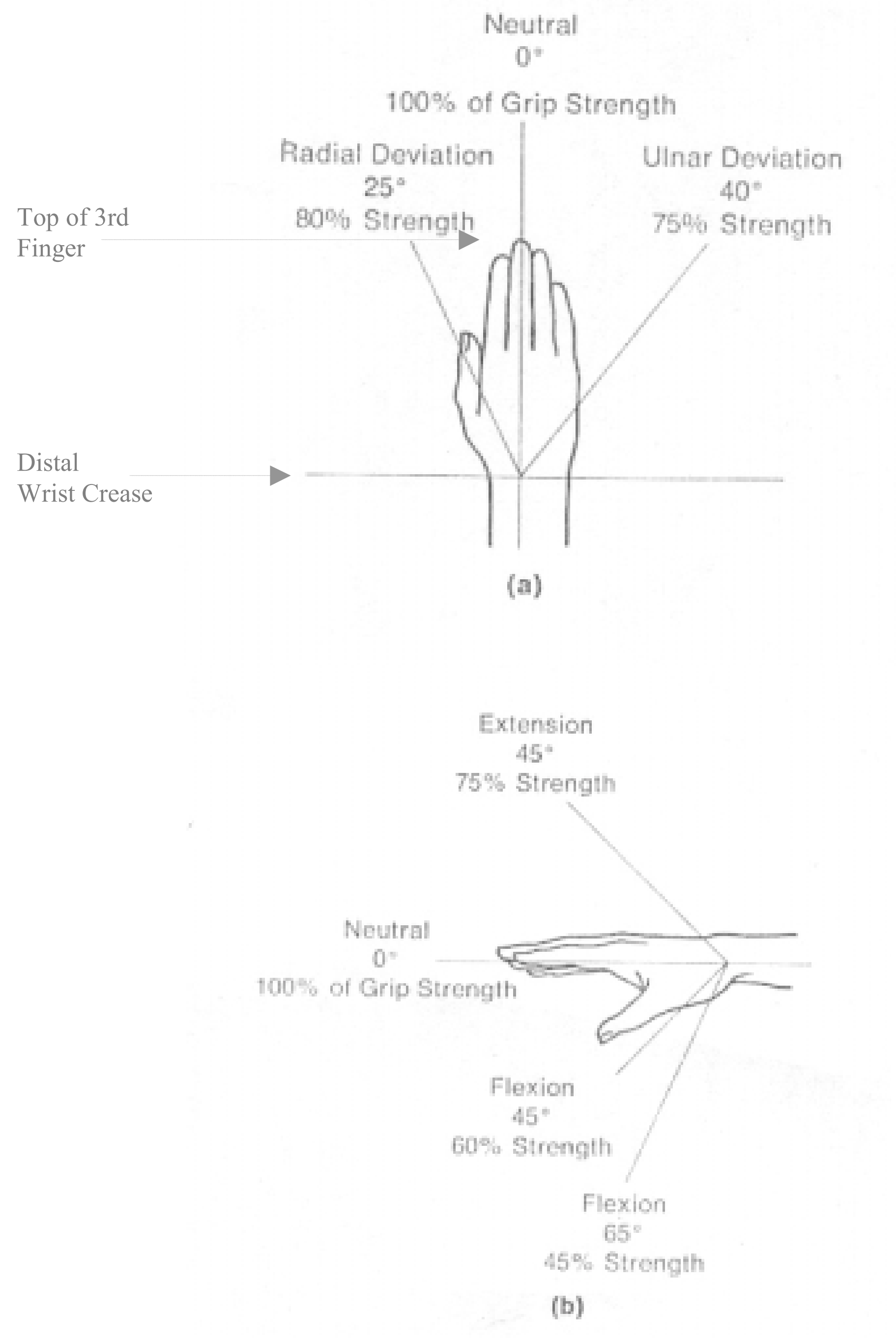
Adapted from Mobility, Ergonomics & Multitasking in Text Entry, (Appendix X03)

Ergonomics is an interdisciplinary field involving biomechanical, physiological, psychological and behavioral considerations. The field's complexity together with human individuality makes pronouncing a design "ergonomically correct" or "ergonomically incorrect" a somewhat pseudoscientific process. Nevertheless, there exist rules of thumb which can be used as approximate guidelines for "good" ergonomics.

The guidelines presented below are based on the recommendations of Liberty Mutual Group (America's largest workers' compensation insurer), The National Institute for Occupational Safety and Health (NIOSH), Lisa Tiraboschi (the associate director for Environmental Health and Safety at Dartmouth College), the Eastman Kodak Company Ergonomics Group Health and Environmental Laboratory, occupational therapist Denise Finch, and numerous individuals coping with keyboard related RSIs (Rodgers, Libery Mutual & NIOSH). The primary ergonomic considerations for keyboard design are:

- **Neutral wrist alignment** (See the following page for a visual definition of neutral alignment).
- **Use does not require static posture.** Static posture inhibits the circulation of blood.
- **Use does not constrain the upper body's range of motion.** The "best" posture is person-specific. The more a device constrains a person's range of motion, the less chance there is of a healthy posture being achieved.
- **Actuation does not require excessive or awkward motions and forces.**
- **Actuation does not cause pain.**

Neutral Wrist Alignment



Diagrams reproduced from
"Ergonomics for People at Work"(Rodgers, 470)

Appendix P: Decision Matrices

Our planned path to a solution involved a number of choices, and these choices appear in the following section in decision matrix form. The structure of the matrices is as follows: Alternatives appear on the X-axis of an XY table, while evaluation considerations appear on the Y-axis. Each alternative/consideration cell contains a number representing the alternative's "score" with respect to the associated consideration. (Scores represent the best guesses of the project team). The totals of each alternative score row are averaged - taking into consideration different numeric weights for different specifications - and the alternative with the highest total is deemed the "best" solution.

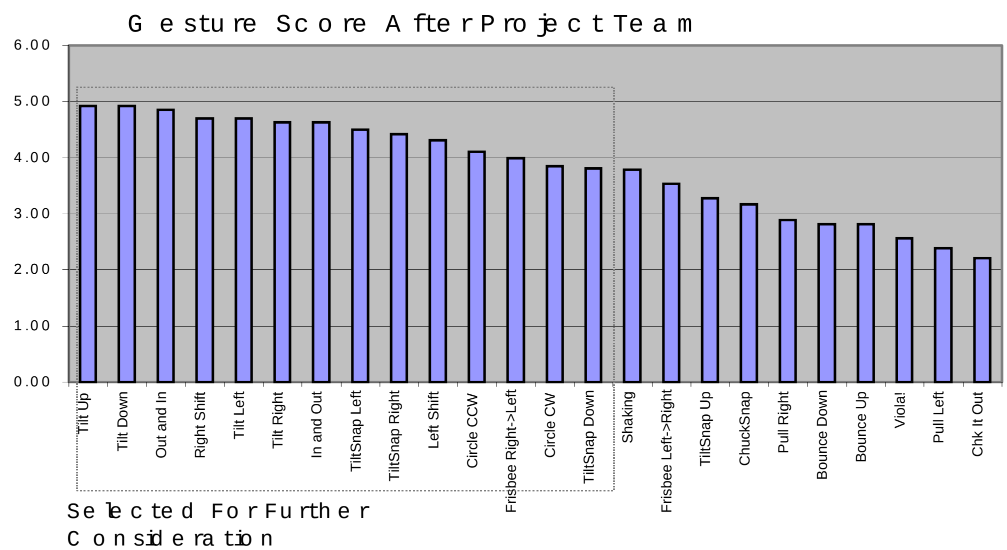
DECISION: Ranking of "Best" to "Worst" Gestures (Round 1)

Considerations:

Comfortability How comfortable were the gestures to perform?
Intuitiveness A combination of learnability and ease of envisioning use of the gesture as a control
Acceptability Taking into account social cultural and psychological considerations, were gestures
Recognizability Were the graphs for a given gesture visually distinguishable from the graphs of othe
Consistency Were the graphs for different recordings of the same gesture visually consistent?

1=Low (Worst), 5=High (Best)

Considerations	Comfortability	Intuitiveness	Acceptability	Recognizability	Consistency	Weighted Average
Weight [1-5]	**4**	**2**	**3**	**5**	**4**	
Alternatives:						
Tilt Left	5	4.33	5	5	4	**4.7**
Tilt Right	4.67	4.33	5	5	4	**4.63**
Tilt Up	4.67	5	5	5	5	**4.93**
Tilt Down	4.67	5	5	5	5	**4.93**
TiltSnap Left	5	5	5	4	4	**4.5**
TiltSnap Right	4.67	5	5	4	4	**4.43**
TiltSnap Up	5	5	5	2	1	**3.28**
TiltSnap Down	4	5	5	3	3.15	**3.81**
Frisbee Left->Right	4.67	4	5	2	3	**3.54**
Frisbee Right->Left	4	3.67	5	3.5	4	**3.99**
Bounce Down	4.33	4.67	5	1	1	**2.81**
Bounce Up	4.33	4.67	5	1	1	**2.81**
Viola!	3	2.33	4.33	2.5	1	**2.56**
Chk It Out	3	2.67	3.67	1.5	1	**2.21**
ChuckSnap	3	3	4	3	3	**3.17**
Shaking	3.33	3.33	4.33	4.25	3.5	**3.79**
Circle CW	3.67	4.33	4	4	3.5	**3.85**
Circle CCW	3.67	4.33	4	4.5	4	**4.1**
Out and In	4.33	5	5	5	5	**4.85**
In and Out	4.33	5	5	5	4	**4.63**
Right Shift	4.67	5	5	5	4	**4.7**
Left Shift	4.67	5	5	4	3.5	**4.31**
Pull Left	3	3	3.67	2	1	**2.39**
Pull Right	3	3	3.67	3	2	**2.89**



DECISION: Ranking of "Best" to "Worst" Gestures (Round 2)

Considerations:

Comfortability How comfortable were the gestures to perform?

Intuitiveness A combination of learnability and ease of envisioning use of the gesture as a control action.

Acceptability Taking into account social cultural and psychological considerations, were gestures acceptable?

Recognizability Were the graphs for a given gesture visually distinguishable from the graphs of other gestures?

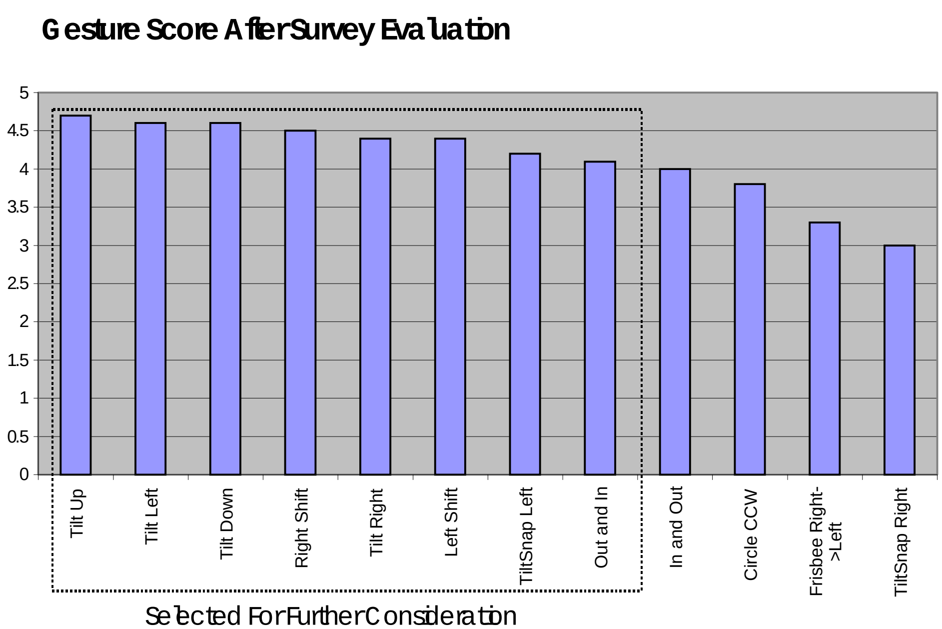
Consistency Were the graphs for different recordings of the same gesture visually consistent?

1=Low (Worst), 5=High (Best)

Considerations	Comfortability	Intuitiveness	Acceptability	Recognizability	Consistency	**Weighted Average**
Weight [1-5]	**4**	**2**	**3**	**5**	**4**	
Alternatives:						
Tilt Left	4.5	4.7	5	5	4	**4.63**
Tilt Right	2.6	4	5	5	5	**4.36**
Tilt Up	4	4.3	5	5	5	**4.70**
Tilt Down	3.7	4.3	5	5	5	**4.63**
TiltSnap Left	4.3	3.9	5	4.2	3.5	**4.17**
TiltSnap Right	2.7	3.4	5	3	2	**3.09**
Frisbee Right->Lef	4.2	3	5	2	3	**3.32**
Circle CCW	4.1	3.1	5	3	4	**3.81**
Out and In	4.1	3.1	5	4	4	**4.09**
In and Out	3.6	3.2	4.9	4.3	4	**4.06**
Right Shift	4.4	3.9	4.9	5	4	**4.51**
Left Shift	3.8	3.7	5	5	4	**4.37**

Sorted By Weighted Avera

Tilt Up	4.7
Tilt Left	4.6
Tilt Down	4.6
Right Shift	4.5
Tilt Right	4.4
Left Shift	4.4
TiltSnap Left	4.2
Out and In	4.1
In and Out	4
Circle CCW	3.8
Frisbee Right	3.3
TiltSnap Righ	3



DECISION: Gesture Control Scheme

Considerations:

Usability A combination of learnability, input rates, error & error recovery rates, comfort & acceptability.

Extendibility To what extent could the alternative be used in a wide variety of applications and scenarios?

Feasibility Given real-world constraints, is implementation of the alternative realistic?

1=Low (Worst), 5=High (Best)

Considerations	Usability	Extendibility	Feasibility	**Weighted Average**
Weight [1-5]	**5**	**3**	**5**	
Alternatives:				
Text Entry	1	2	1	1.23
Application-Specific Commands	3	2	4	3.15
User-Defined Control Mapping	5	5	3	4.23
Generalized Set of Navigation & Selection Commands	**5**	**4**	**4**	**4.33**

DECISION: Gestural Interaction Scheme

Considerations:

Feasibility Given real-world constraints, is implementation of the alternative realistic?

Usability A combination of learnability, input rates, error & error recovery rates, comfort & acceptability.

Extendibility To what extent could the alternative be used in a wide variety of applications and scenarios?

1=Low (Worst), 5=High (Best)

Considerations	Feasibility	Usability	Extendibility	**Weighted Average**
Weight [1-5]	**5**	**5**	**3**	
Alternatives:				
No buttons, Continuous Polling	2	4	5	3.46
Button Press Signals Start of One Gesture	5	4	4	4.38
Button Press Signals Gesture End of One Gesture	3	3	3	3.00
While Button is Pressed, system polls for single OR multiple gestures until Button Release	3	4	5	3.85
While button is pressed, system polls for a single gesture until Button Release	4	4	5	4.23
Button Press Activates/De-activates Gesture Recognition	**5**	**5**	**5**	**5**

DECISION: Gesture Recording System

Considerations:

Developer Support To what extent is there local and company support for the system?

Simplicity of Operation How easy is it to use the system and its resulting data files?

1=Low (Worst), 5=High (Best)

Considerations	Developer Support	Simplicity of Operation	Weighted Average
Weight [1-5]	**4**	**3**	
Alternatives:			
Develop a New Data Acquisition System	1	3	1.86
Use a Signal Quest Data Aq. System	4	3	3.57
Use a Crossbow Data Aq. System	**5**	**4**	**4.57**

DECISION: PDA Platform

Considerations:

Developer Support To what extent is there local and company support for the system

Feasibility Given real-world constraints, is implementation of the alternative realistic?

Extendibility To what extent could the alternative be extended to suite different applications and scenarios?

1=Low (Worst), 5=High (Best)

Considerations	Development Support	Feasibility	Extendibility	Weighted Average
Weight [1-5]	**5**	**4**	**3**	
Alternatives:				
Windows CE, Phillips Niño	2	2	3	2.25
PalmOS, PalmPilot	**5**	**5**	**4**	**4.75**
PalmOS, Handspring Visor	4	4	5	4.25

DECISION: Signal Processing Location

Considerations:

Feasibility Given real-world constraints, is implementation of the alternative realistic?

Extendibility To what extent could the alternative be extended to suite different applications and scenarios?

Low Cost (Ideally, adds no more than 4$ to the production cost of a PDA, as per sponsor wishes.)

1=Low (Worst), 5=High (Best)

Considerations	Feasibility	Extendibility	Low Cost	Weighted Average
Weight [1-5]	**5**	**3**	**4**	
Alternatives:				
Hardware in SpringBoard module; using Filtering ICs & Combinational Logic	3	1	2	2.17
Firmware in SpringBoard Module, Using Separate Microprocessor.	5	4	3	4.08
Software Running on the Visor's Processor	**4**	**5**	**5**	**4.58**

DECISION: Graffiti algorithm vs. Own algorithm

Considerations:

Implementation Flexibility How much flexibility will we have in changing the code?

Robustness How accurate will the algorithm be given the variability involved?

Tweakability How easy will it be to make algorithm changes?

Memory Usage How much memory will be required?

Knowledge How much will we know about each algorithm?

Development Time Which algorithm will require the most development time?

Platform Independence Will it be easy to implement the algorithm on a variety of platforms?

1=Low (Worst), 5=High (Best)

Considerations	Implementation Flexibility	Robustness	Tweak-ability	Memory Usage	Learning Value	Development Time	Platform Independence	Weighted Average
Weight [1-5]	**4**	**5**	**4**	**2**	**4**	**3**	**5**	
Alternatives:								
Graffiti	3	4	2	5	2	4	1	**2.78**
Own Algorithm	**5**	**4**	**5**	**3**	**5**	**3**	**5**	**4.44**

DECISION: Algorithm Selection (Round 1)

Considerations:

Robustness How accurate will the algorithm be given the variability involved?

Extendibility To what extent could the alternative be extended to suite different applications and scenarios?

Programming Time How much programming time will the algorithm require?

Feasibility How feasible is a real time implementation of this algorithm on a PDA?

1=Low (Worst), 5=High (Best)

Considerations	Robustness	Extendibility	Programming Time	Feasibility	Weighted Average
Weight [1-5]	**5**	**4**	**3**	**5**	
Alternatives:					
State Machine - Simple Peak Detection	**1**	**1**	**5**	**5**	**2.88**
State Machine - Threshold Range	**4**	**4**	**4**	**4**	**4.00**
Hidden Markov Models	4	4	3	3	3.53
Neural Networks	4	5	2	3	3.59

DECISION: Algorithm Selection (Round 2)

Considerations:

Development Time How much programming time will the algorithm require?

Recognition Accuracy How well will this approach meet the goal of recognizing gestures?

Extendability To what extent could this approach be extended to suite different applications and include additiona

Processor Memory Is this approach feasible with respect to a PDA's program & storage memory constraints?

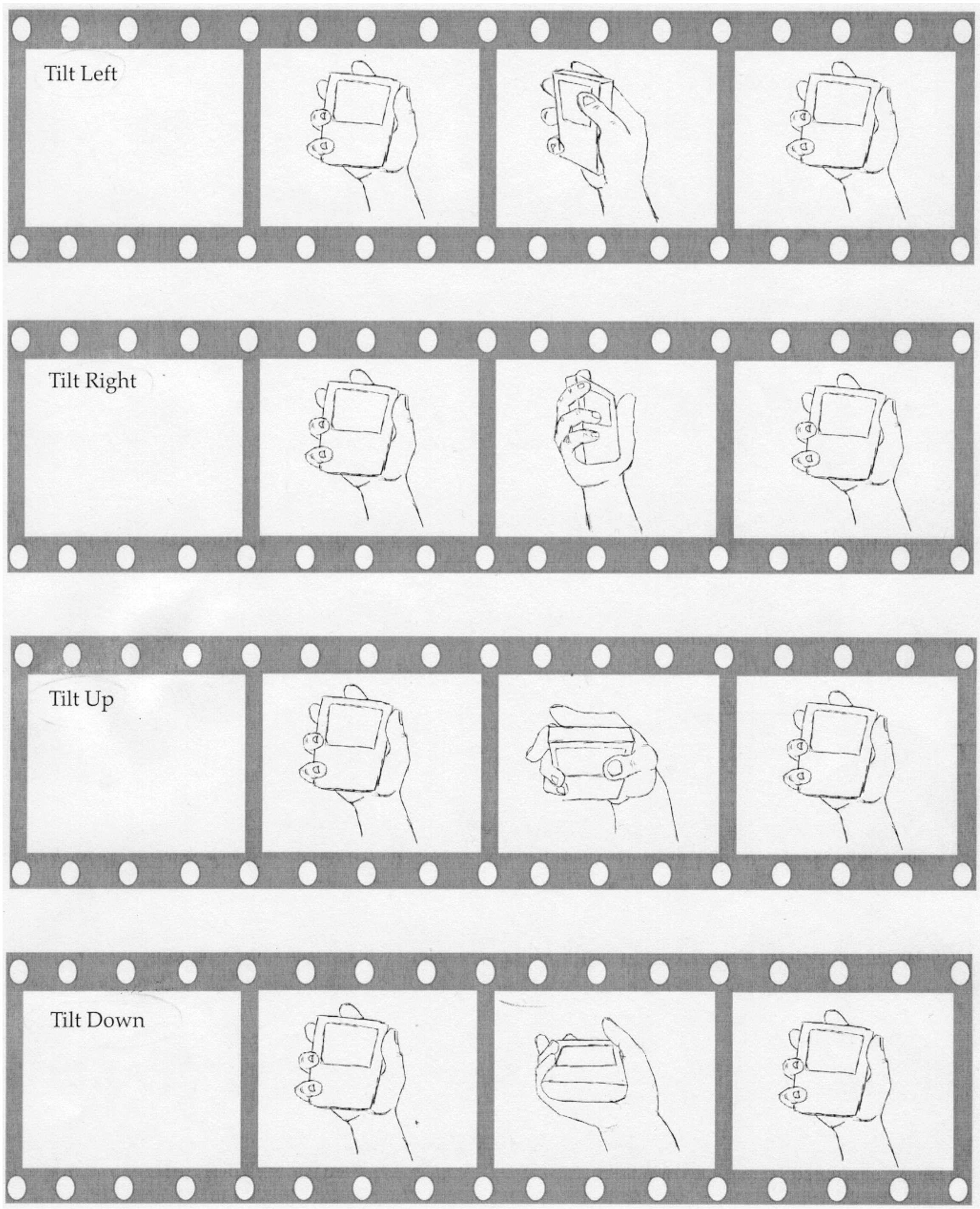
Processor Time Is a real time implementation of this algorithm on a PDA feasible with respect to time?

Learning Value Will this approach increase our intuition and facilitate the trial of additional approaches?

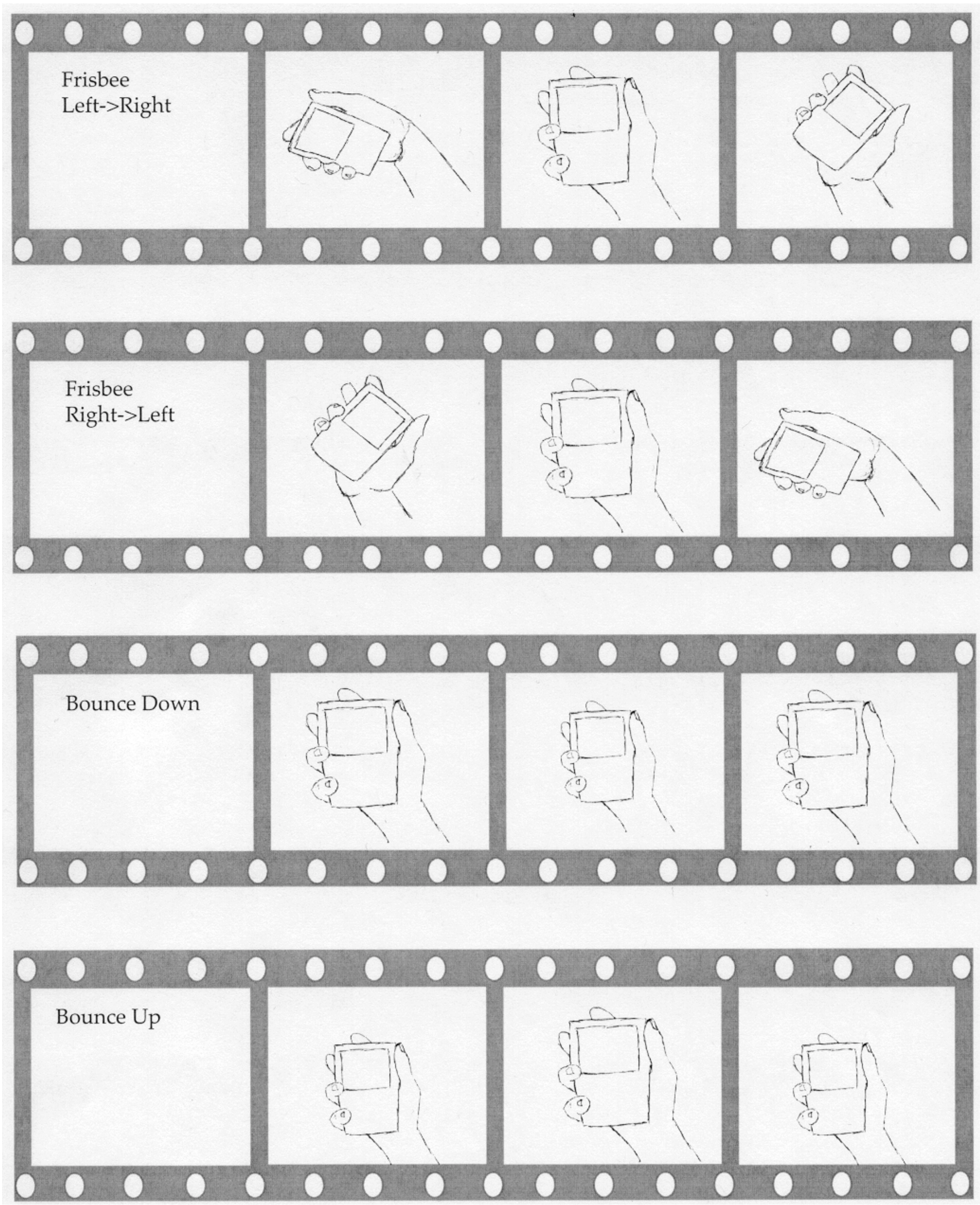
1=Low (Worst), 5=High (Best)

Considerations	Development Time	on Accuracy	Extendability	Processor Memory	Processor Time	Learning Value	Weighted Average
Weight [1-5]	**5**	**5**	**3**	**3**	**3**	**4**	
Alternatives:							
Statistical Comparison	**5**	**4**	**1**	**5**	**4**	**5**	**4.13**
State Machine - Magnitude Thresholds	**5**	**4**	**3**	**5**	**5**	**4**	**4.33**
State Machine - Time & Magnitude	**4**	**5**	**5**	**4**	**4**	**5**	**4.52**
Genetic Algorithms	1	3	5	3	1	2	2.39
Neural Networks	2	2	3	3	2	1	2.09
Hausdorf Metric	3	4	4	2	2	1	2.33
Hidden Markov Model	2	5	5	2	2	3	3.16

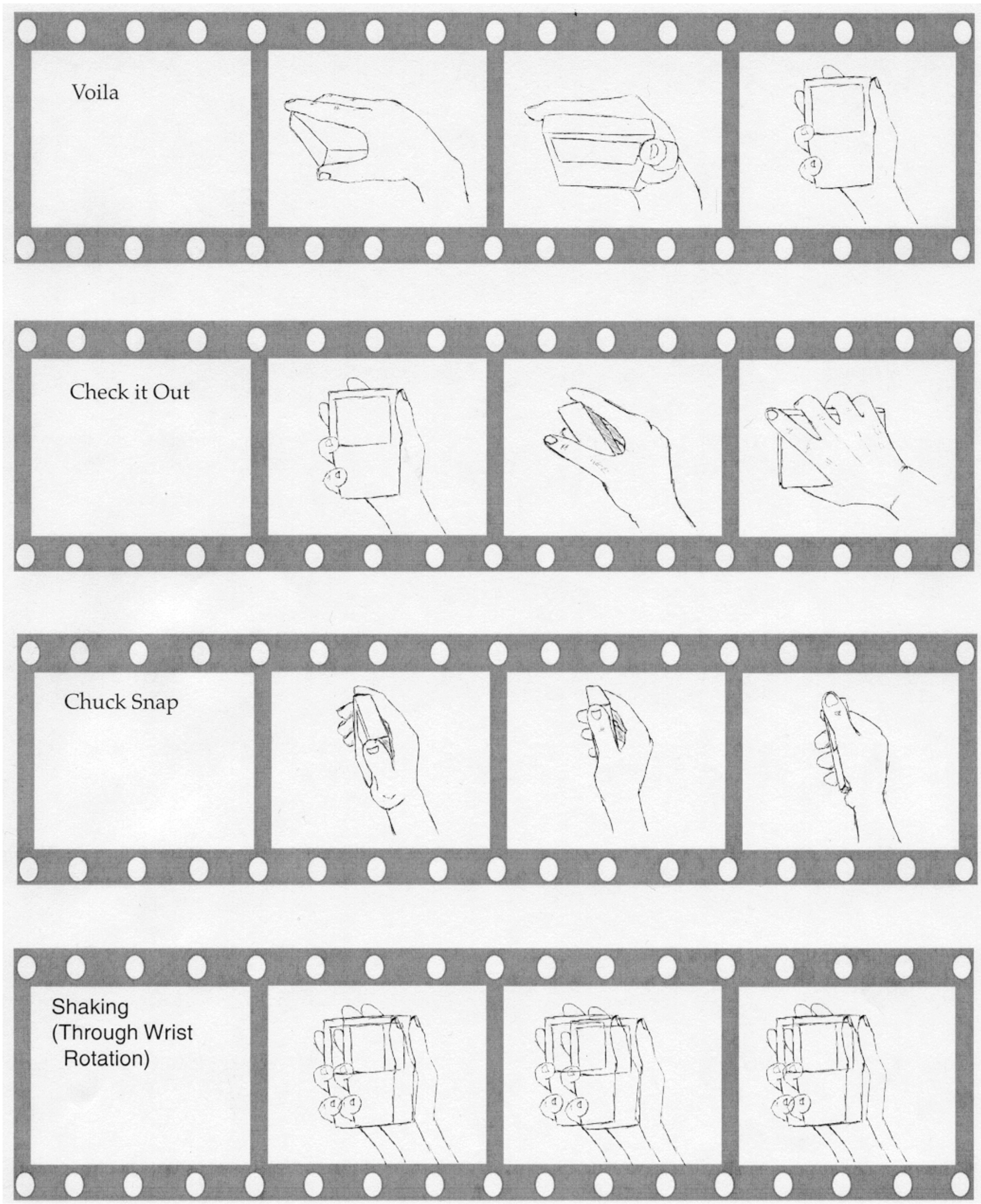
Appendix Q: Gesture Direction Sheets



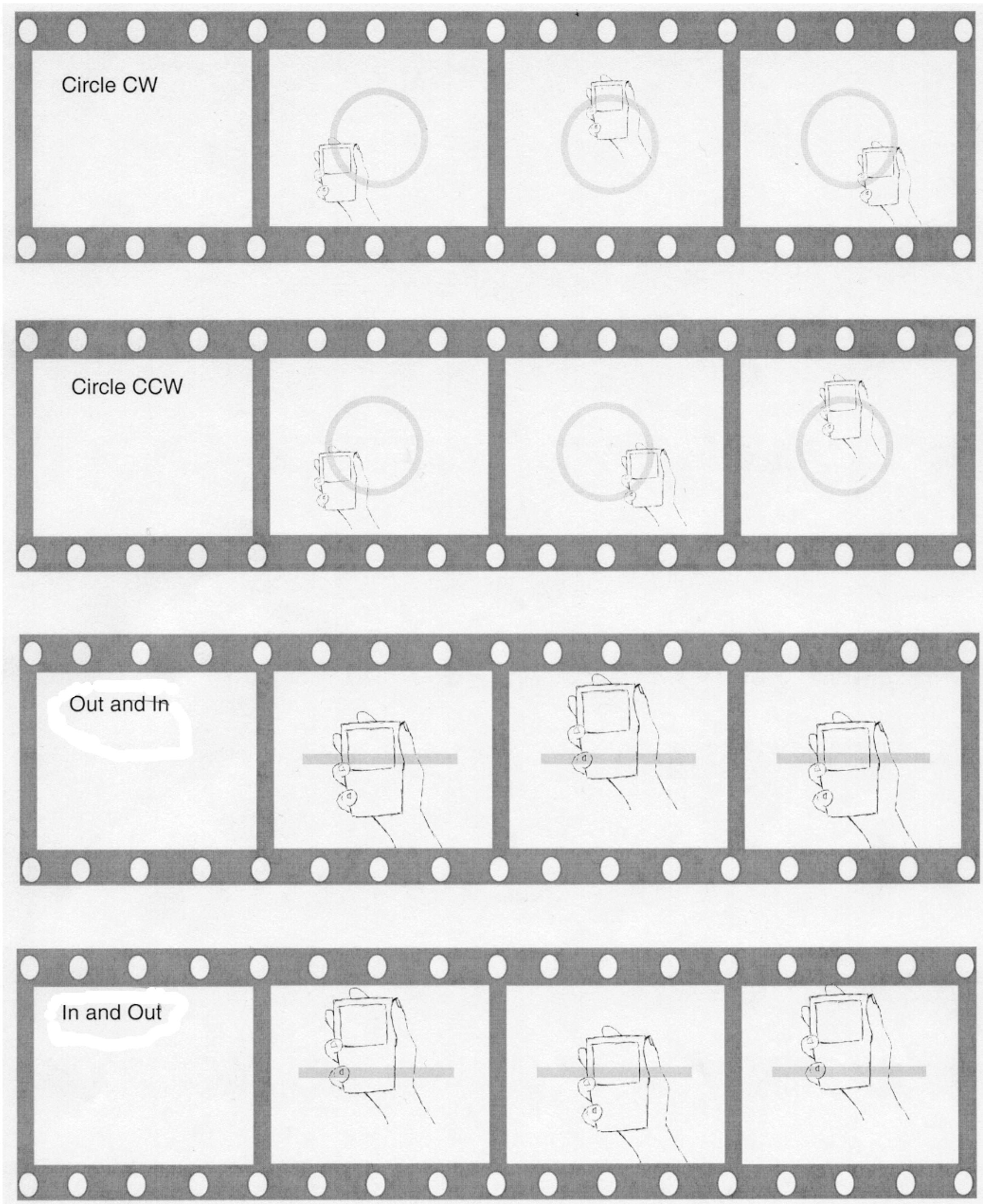
Gesture Direction Sheets



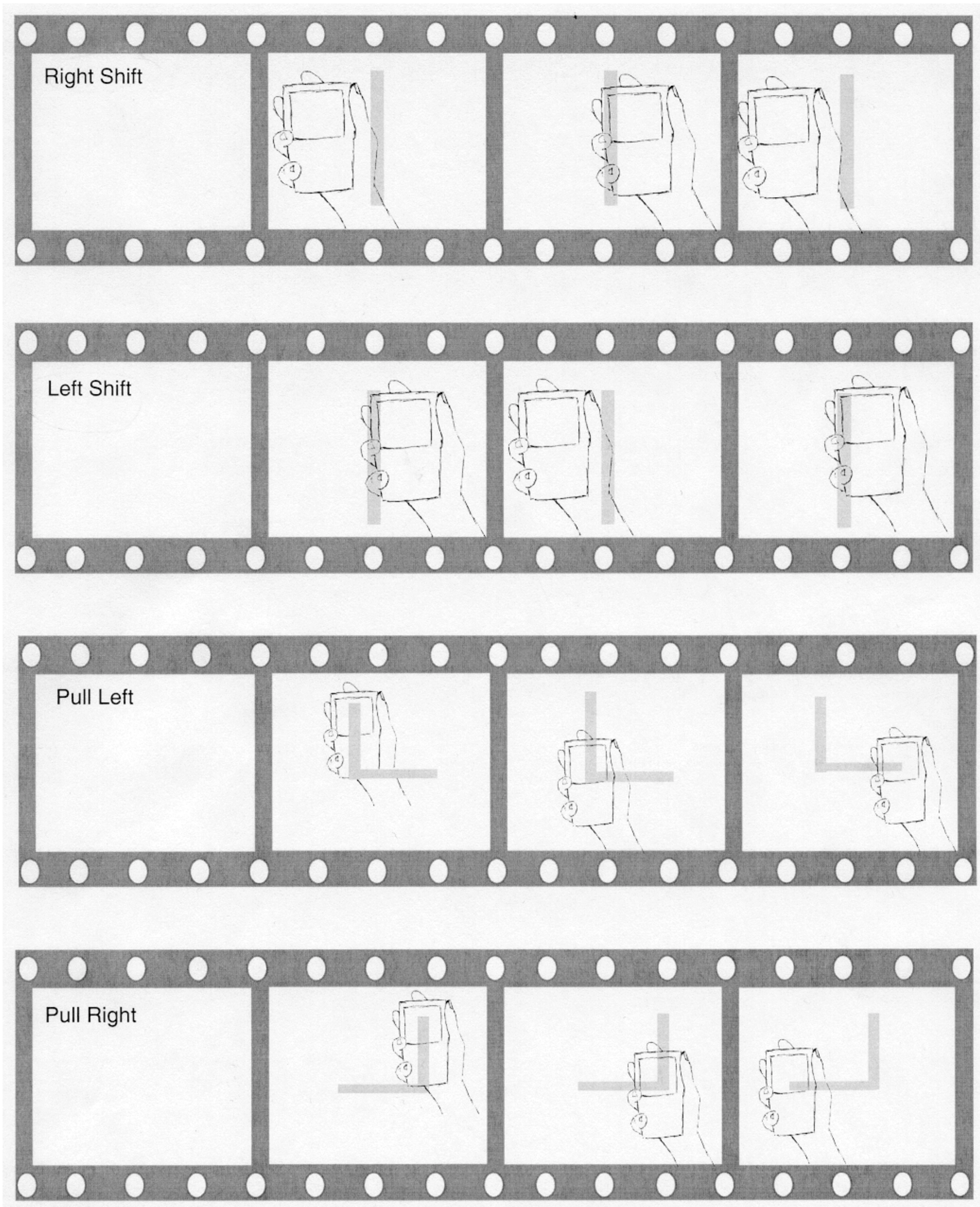
Gesture Directions (Continued)



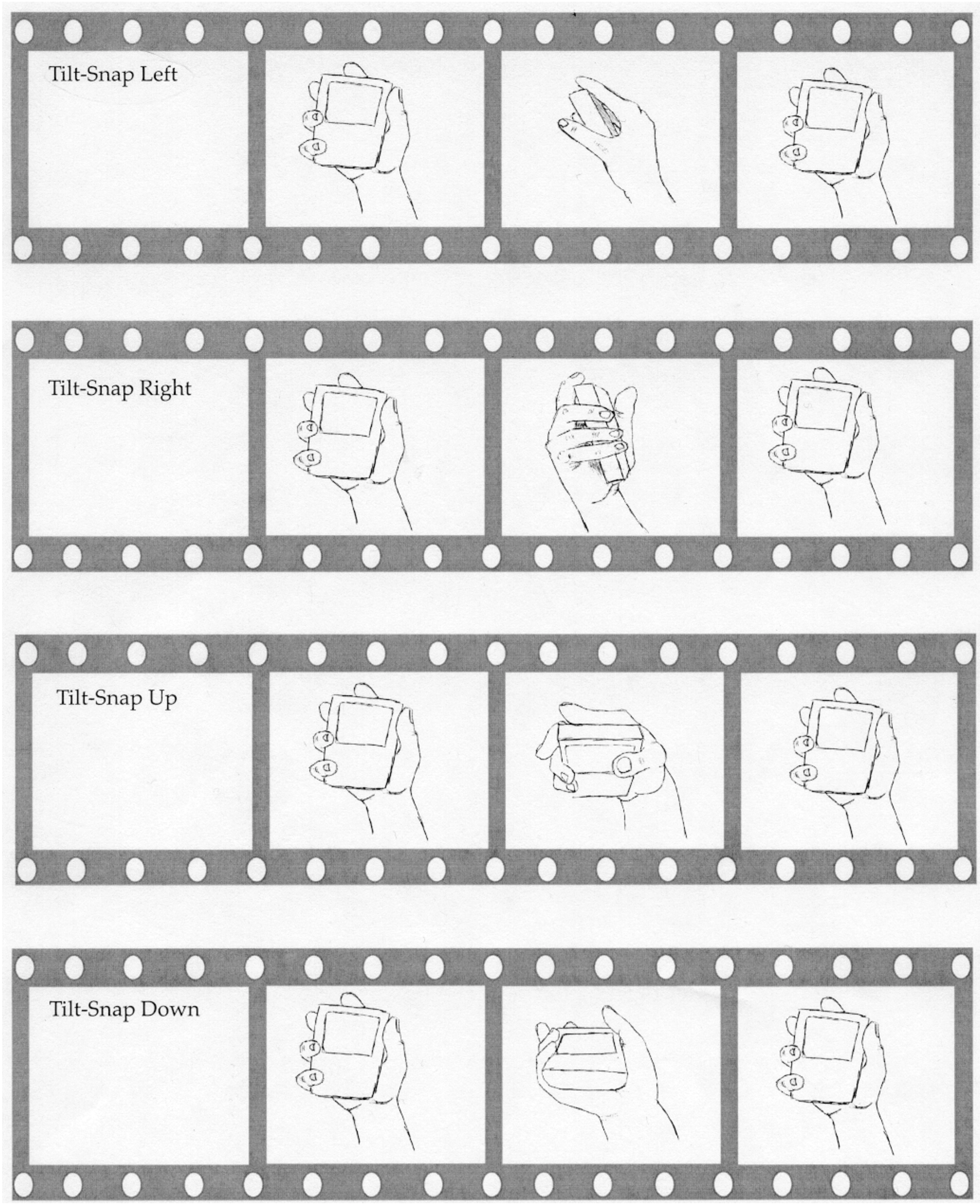
Gesture Directions (Continued)



Gesture Directions (Continued)



Gesture Directions (Continued)



Appendix R: Gesture Selection Test Form

Gesture Test Form

Name: ***Gender: M / F*** ***Age:*** ***Height:***

Email: ***PDA User: Y / N*** ***Profession:***

Grade gestures from 1 to 5 where 1 is worst and 5 best.

(File #)	Gesture	Comfortable?	Intuitive?	Acceptable?
	Tilt Left			
	Tilt Right			
	Tilt Up			
	Tilt Down			
	TiltSnap Left			
	TiltSnap Right			
	Frisbee Right->Left			
	Circle CCW			
	Out and In			
	In and Out			
	Right Shift			
	Left Shift			

Notes:

(Our Notes:)

Appendix S: Gesture Data Acquisition Apparatus

A Handspring "Visor" PDA (back view), together with a Crossbow data acquisition board. This set-up was used to record x and y acceleration components for each gesture.

Appendix T: TilePlot & TFPlot

In order to view and manipulate accelerometer data, two MATLAB applications were created; "TilePlot" and "TFPlot". TilePlot made it possible to view numerous trials of a given gesture side by side, while TFPlot made it possible to look closely at one gesture in both the time and the frequency domains. Screenshots of these two applications appear below:

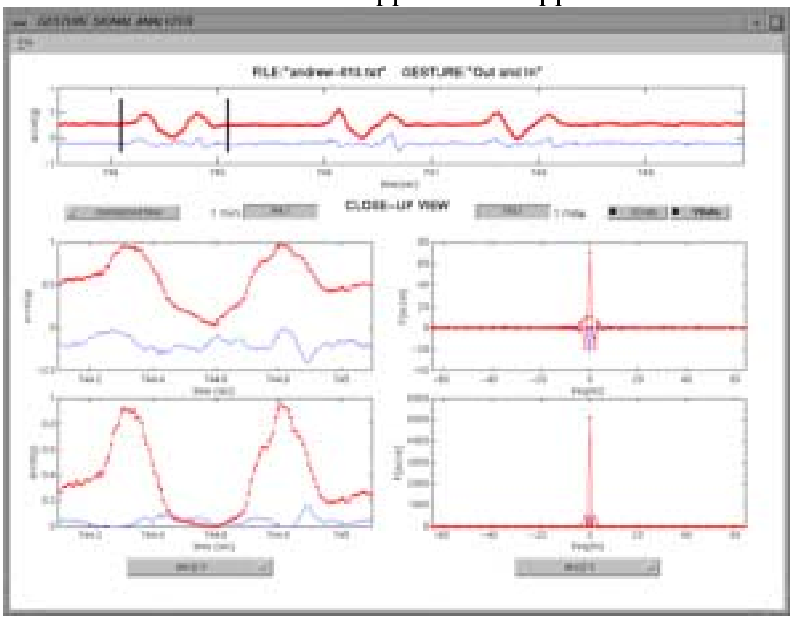
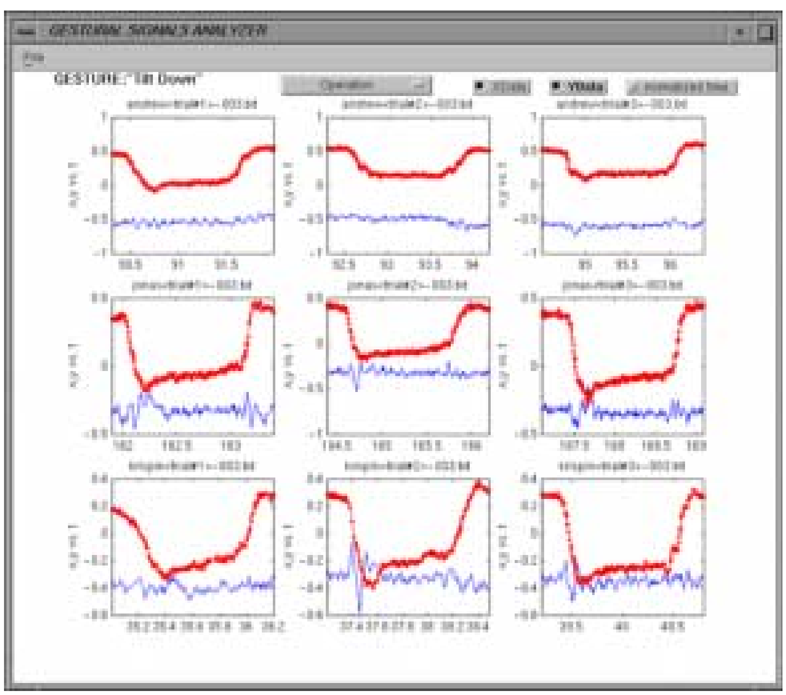


Figure 28: Tfplot – Facilitated cropping gesture data and viewing signals in detail



TilePlot – Facilitated comparison between instances of a gesture.

Appendix U: Piece-Wise Gesture Recognition Code Listing

This code runs on SGI Unix (Irix).

The main program driver:

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++
//GestRec.cpp

//This program recognizes gestures based on acceleration
//data coming from an ADXL202 test board plugged into
//an SGI serial port.


#include "AccelData.h"
#include "Recorder.h"
#include "Analyzer.h"
#include <stdio.h>
#include <unistd.h>


int main()
{
  AccelData AD;
  Recorder R(&AD);
  Analyzer A(&AD);

  while(1)
    {
     if (R.record())
                        {
                        A.check_for_features();
                        A.print_feature_name();
                        //AD.print_out();
                        }
    }

  return(1);
}
-----------------------------------------------------------------------------------------
```

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++
//AccelData.h

//This is a class for storing and accessing accelerometer data.

//NOTES:
// 1) Accelerometer data is stored in arrays.
//Arrays are circular; they are always indexed "mod" ARRAY_LENGTH.
//By doing this, we avoid falling off the end of arrays.

// 2) The current_i value is an index into the first FILTERED
// acceleration data samples.  New samples are placed in front of the
// filter--AHEAD of the current index.


#ifndef ACCELDATA_H
#define ACCELDATA_H


//INCLUDES
#include <iostream.h>
#include <fstream.h>

//TYPEDEFS
typedef unsigned long  timestamp;

//DEFINE GLOBAL CONSTANTS
```

```
const int SAMPLING_RATE = 30; //Hz
const double MAX_GESTURE_TIME = 1.2; //Seconds
const int FILTER_LENGTH = 5; //#of coefficients in filter window; hardcoded.
const int ARRAY_LENGTH = (int)(SAMPLING_RATE*MAX_GESTURE_TIME +
                                                        1.5*FILTER_LENGTH);
//Note: ARRAY_LENGTH accounts for the time lag and sample loss due to the
//filter.

// Set to 1 to read from the file, 0 to read from accelerometer...
const bool FILEIN = 0;

class AccelData
{
 public:

  // CONSTRUCTOR, DESTRUCTOR
  AccelData();
  ~AccelData();


  //FUNCTIONS
  double get_Ax(const int& i);          //Returns Ax[i]
  double get_Ay(const int& i);          //Returns Ay[i]
  double get_dAx(const int& i);             //Returns dAx[i]
  double get_dAy(const int& i);         //Returns dAy[i]
  double get_t(const int& i);               //Returns t[i] (sample times)

  int get_current_i();                  //Returns index of x&y samples corresponding
                 //to current index.

  int get_start_i();                    //Returns index of x&y samples corresponding
                 //to start of gesture.

  int get_end_i();                      //Returns index of x&y samples corresponding
                 //to end of gesture.

  void set_start_i(const int& i); //Sets the index of x&y samples corresponding
                  //to gesture start to i.

  void set_end_i(const int& i);          //Sets the index of x&y samples corresponding
                  //to gesture end to i.

  void take_sample();    //Gets accelerometer data, translates to "g"s,
               //calculates dAx & dAy based on previous sample
               //points.

  bool take_sample_from_file();
               //***FOR TESTING PURPOSES***
               //Gets data from a file, rather than from
               //accelerometer subroutine.

  void reset();            //Resets the object to its starting state.


  void set_offsets();    //Sets the values that ensure an incoming signal
               //starts with a magnitude close to zero.

  void set_offsets_from_file();
               //***FOR TESTING PURPOSES***
               //Sets offsets from file data, rather than from
               //an accelerometer

  bool from_file(); // Returns true if data is coming from a file.

  void print_out(); //Print contents of each array from start to finish

 private:

  //FUNCTIONS
  double deriv(const double& A1, const double& A2,
```

```
                                   const timestamp& t1, const timestamp& t2); //Calculates dAx[current-1]
                                               //and dAy[current-1] based on:
                                               //Ax[current] & Ax[current-1],
                                             //and Ay[current] & Ay[current-1].

  void apply_filter();           //Applies the filter to FILTER_LENGTH spots in
                              //Ax and Ay ahead of the current spot.

  //ARRAY DATA
  double Ax[ARRAY_LENGTH];        //x-axis acceleration
  double Ay[ARRAY_LENGTH];        //y-axis acceleration
  double dAx[ARRAY_LENGTH];       //running derivative of x-axis acceleration
  double dAy[ARRAY_LENGTH];       //running derivative of y-axis acceleration
  timestamp t[ARRAY_LENGTH];      //sample times
  double filter[FILTER_LENGTH];   //An array holding low-pass filter coefficients.


  //INDEX DATA
  int start_i;                    //Sample index corresponding to gesture start.
  int end_i;                      //Sample index corresponding to gesture end.
  int current_i;                  //Current sample index. Corresponds to first
                              //sample that has passed completely through
                              //the filter.

  //INPUT VARIABLES
  int term;
  double Ax_in;
  double Ay_in;
  timestamp time_in;

  //Zeroing values
  double Ax_offset;
  double Ay_offset;

  //For reading in data files:
  ifstream data;
  ofstream log;
};
#endif //ACCELDATA_H



//AccelData.cpp

//Functions for a class that stores and provides access to accelerometer data.

//INCLUDE DIRECTIVES
#include "AccelData.h"
#include "Adxl.h"     //for accelerometer stuff
#include <unistd.h>   //for "close()"
#include <stdio.h>    //for "fprintf" and stderr
#include <iostream.h> //for cout cin, etc
#include <fstream.h>  //for filehandles...

// CONSTRUCTOR
AccelData::AccelData()
{
  //Initialize data array indices
  start_i = 2*ARRAY_LENGTH; //Dummy value indicates start hasn't been set.
  end_i = 2*ARRAY_LENGTH;     //Dummy value indicates end hasn't been set.
  current_i = 0;

  //Initialize data arrays with zeros
  for(int i=0; i<ARRAY_LENGTH; i++)
    Ax[i] = 0.0;

  for(int i=0; i<ARRAY_LENGTH; i++)
    Ay[i] = 0.0;

  for(int i=0; i<ARRAY_LENGTH; i++)
```

```
    dAx[i] = 0.0;

  for(int i=0; i<ARRAY_LENGTH; i++)
    dAy[i] = 0.0;

  //Set filter coefficients
  filter[0] = 0.0169;
  filter[1] = 0.2279;
  filter[2] = 0.5104;
  filter[3] = 0.2279;
  filter[4] = 0.0169;


  if (FILEIN) //File is input source
    {
     log.open("log.txt");

     //Prompting for file to open
     cout << "What file would you like to read from? " << endl;
     char name[30];
     cin >> name;
     data.open(name);

     //Opening file.
     if ( data.fail() )
                      {
                       cout << "Couldn't open file " << name << "\n";
                       exit(0);
                      }

     //Removing first 13 lines of file (header info).
     char junk[80];
     for(int i=0; i<13; i++)
                      data.getline(junk, 80);

     //Setting the initial "Zeroing" values for Ax, Ay.
     set_offsets_from_file();
    }

  else //Accelerometer is input source
    {
     //Set up serial port for getting accelerometer data
     if((term = get_serial_port(2)) < 0)
                      {
                       fprintf(stderr, "Problem accessing serial port\n");
                       exit(1);
                      }

    set_offsets();
    }
}

//DESTRUCTOR
AccelData::~AccelData()
{
  //Close the serial port connection
  if (FILEIN)
    {
     data.close();
    }
  else
    {
     close(term);
    }
}

//GET_AX
//Returns Ax[i]
double AccelData::get_Ax(const int& i)
{
```

```
  return (Ax[i % ARRAY_LENGTH]);
}


//GET_AY
//Returns Ay[i]
double AccelData::get_Ay(const int& i)
{
  return (Ay[i % ARRAY_LENGTH]);
}

//GET_DAX
//Returns dAx[i]
double AccelData::get_dAx(const int& i)
{
  return (dAx[i % ARRAY_LENGTH]);
}

//GET_DAY
//Returns dAy[i]
double AccelData::get_dAy(const int& i)
{
  return (dAy[i % ARRAY_LENGTH]);
}

//GET_T
//Returns t[i] (sample times)
double AccelData::get_t(const int& i)
{
  return (t[i % ARRAY_LENGTH]);
}

//GET_CURRENT_I
//Returns index of x&y samples corresponding
//to current sample.
int AccelData::get_current_i()
{
  return (current_i);
}

//GET_START_I
//Returns index of x&y samples corresponding
//to start of gesture.
int AccelData::get_start_i()
{
  return (start_i);
}

//GET_END_I
//Returns index of x&y samples corresponding
//to end of gesture.
int AccelData::get_end_i()
{
  return (end_i);
}

//SET_START_I
//Sets the index of x&y samples corresponding
//to gesture start to i.
void AccelData::set_start_i(const int& i)
{
  start_i = i % ARRAY_LENGTH;
}

//SET_END_I
//Sets the index of x&y samples corresponding
//to gesture end to i.
void AccelData::set_end_i(const int& i)
{
  end_i = i % ARRAY_LENGTH;
```

```
}

//TAKE_SAMPLE
//Gets accelerometer data, translates to "g"s,
//calculates dAx & dAy based on previous sample points.
void AccelData::take_sample()
{
  //Get Data
  if(get_adxl_packet(term, &Ax_in, &Ay_in, &time_in))
    {}
  else
    {perror("get_adxl_packet"); }

  //Convert to "g"s
  Ax_in = (Ax_in-50)/12.5 - Ax_offset;
  Ay_in = (Ay_in-50)/12.5 - Ay_offset;

  //Create a previous index and increment the current index
  int prev_i = current_i;
  current_i = (current_i+1) % ARRAY_LENGTH;

  //Place new Ax, Ay & t values right BEFORE the filter's leading edge.
  Ax[(current_i+FILTER_LENGTH)%ARRAY_LENGTH] = Ax_in;
  Ay[(current_i+FILTER_LENGTH)%ARRAY_LENGTH] = Ay_in;
  t[(current_i+FILTER_LENGTH)%ARRAY_LENGTH] = time_in;

  //Pass the filter one step forward
  apply_filter();

  //Calculate dAx, dAy values for sample position
  //right AFTER the filter's trailing edge.
  dAx[current_i] = deriv(Ax[prev_i], Ax[current_i], t[prev_i], t[current_i]);
  dAy[current_i] = deriv(Ay[prev_i], Ay[current_i], t[prev_i], t[current_i]);

  //**FOR TESTING**
  //   fprintf(stderr,"\nt:%g Ax:%g Ay:%g dAx%g dAy%g\n", double(t[current_i])/1000000,
  //   Ax[current_i],Ay[current_i],
  //  dAx[current_i], dAy[current_i]);
}

//TAKE_SAMPLE_FROM_FILE
//***FOR TESTING PURPOSES***
bool AccelData::take_sample_from_file()
{
  if ( data.eof() )
    return (0);
  double scrap1, scrap2, temp;
  data >> temp >> scrap1 >> scrap2 >> Ax_in >> Ay_in;

  Ax_in = Ax_in - Ax_offset;
  Ay_in = Ay_in - Ay_offset;
  time_in = (timestamp)(temp*1000000);

  //Create a previous index and increment the current index
  int prev_i = current_i;
  current_i = (current_i+1) % ARRAY_LENGTH;

  //Place new Ax, Ay & t values right BEFORE the filter's leading edge.
  Ax[(current_i+FILTER_LENGTH)%ARRAY_LENGTH] = Ax_in;
  Ay[(current_i+FILTER_LENGTH)%ARRAY_LENGTH] = Ay_in;
  t[(current_i+FILTER_LENGTH)%ARRAY_LENGTH] = time_in;

  //Pass the filter one step forward
  apply_filter();

  //Calculate dAx, dAy values for sample position
  //right AFTER the filter's trailing edge.
  dAx[current_i] = deriv(Ax[prev_i], Ax[current_i], t[prev_i], t[current_i]);
  dAy[current_i] = deriv(Ay[prev_i], Ay[current_i], t[prev_i], t[current_i]);
```

```
  return (1);
}


//RESET
//Resets the object to its starting state.
void AccelData::reset()
{
  //Close the serial port connection
  close(term);

  //Invoke the constructor
  AccelData();
}

//SET_OFFSETS
//Set the offset values that effectively "zero" the data.
void AccelData::set_offsets()
{
  if(get_adxl_packet(term, &Ax_in, &Ay_in, &time_in))
    {}
  else
    {perror("get_adxl_packet");}
  Ax_offset = (Ax_in-50)/12.5;  //Converting to gs.
  Ay_offset = (Ay_in-50)/12.5;
}

//SET_OFFSETS_FROM_FILE
//***FOR TESTING PURPOSES***
//Sets offsets with parameters.
void AccelData::set_offsets_from_file()
{
  if ( data.eof() )
    exit(0);

  double scrap1, scrap2, temp;

  data >> temp >> scrap1 >> scrap2 >> Ax_offset >> Ay_offset;

  cout << "OFFSET VALS:" << Ax_offset << "  " << Ay_offset << "\n";
}

//DERIV
//Calculates the derivative dA[current-1]/dt[current-1]:
//
double AccelData::deriv(const double& A1, const double& A2,
                                                          const timestamp& t1, const timestamp& t2)
{
  return (1000000*(A2-A1)/(double)(t2-t1));
}

//APPLY_FILTER
//Applies the filter to FILTER_LENGTH spots in
//Ax and Ay ahead of the current spot.
void AccelData::apply_filter()
{
  //Create a temp variable to store the filtered sample
  //as it is being "built".
  double run_sum;

  //Apply to Ax
  run_sum = 0;
  for(int i=0; i<FILTER_LENGTH; i++)
      run_sum += Ax[(current_i+i)%ARRAY_LENGTH]*filter[i];
  Ax[current_i] = run_sum;

  //Apply to Ay
  run_sum = 0;
  for(int i=0; i<FILTER_LENGTH; i++)
      run_sum += Ay[(current_i+i)%ARRAY_LENGTH]*filter[i];
```

```
  Ay[current_i] = run_sum;
}

bool AccelData::from_file() {
  return FILEIN;
}


//Print contents of each array from start to finish
void AccelData::print_out()
{
  //Print time:
  fprintf(stderr, "\nt = [");
  for(int i=start_i; i<=end_i; i=(i+1)%ARRAY_LENGTH)
    {
      fprintf(stderr, " %lu", t[i]);
      if (i == end_i)
                    fprintf(stderr, " ] \n");
    }


  //Print Ax:
  fprintf(stderr, "\nAx = [");
  for(int i=start_i; i<=end_i; i=(i+1)%ARRAY_LENGTH)
    {
      fprintf(stderr, " %g", Ax[i]);
      if (i == end_i)
                    fprintf(stderr, " ] \n");
    }

  //Print dAx:
  fprintf(stderr, "\ndAx = [");
  for(int i=start_i; i<=end_i;  i=(i+1)%ARRAY_LENGTH)
    {
      fprintf(stderr, " %g", dAx[i]);
      if (i == end_i)
                    fprintf(stderr, " ] \n");
    }

  //Print Ay:
  fprintf(stderr, "\nAy = [");
  for(int i=start_i; i<=end_i;  i=(i+1)%ARRAY_LENGTH)
    {
      fprintf(stderr, " %g", Ay[i]);
      if (i == end_i)
                    fprintf(stderr, " ] \n");
    }

  //Print dAy:
  fprintf(stderr, "\ndAy = [ ");
  for(int i=start_i; i<=end_i;  i=(i+1)%ARRAY_LENGTH)
    {
      fprintf(stderr, " %g", dAy[i]);
      if (i == end_i)
                    fprintf(stderr, " ] \n");
    }

  fprintf(stderr, "\nplot(t, Ax,'-b' , t,Ay,'.-r')\n");
}
------------------------------------------------------------------------------------------



++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Recorder.h

//This class holds the necessary data for recording one gesture.


#ifndef RECORDER_H
```

```
#define RECORDER_H


//INCLUDE DIRECTIVES
#include "AccelData.h"
#include <sys/time.h> //for type: "timestamp"


//CONSTANTS
//Constants representing gesture start and end point criteria.
const double VE_START = .3;
const double DE_START = .02 * SAMPLING_RATE;
const double VE_END = .5;
const double DE_END = .04*SAMPLING_RATE;
const int ENDING_LENGTH = 4;


class Recorder
{
 public:

  //CUNSTRUCTOR
  Recorder(AccelData * ADparam);

  //FUNCTIONS
  bool record();        //A function that records one potential gesture or the
                    //maximum number of sample points.  Returns TRUE if a
                    //potential gesture has been recorded (i.e. a motion
                    //with a start and a finish), otherwise returns FALSE.

  void reset();                 //Resets the recorder object to its initial state.


 private:

  //FUNCTIONS
  bool start_detected();                       //Returns TRUE if the start of a potential
                          //gesture has been detected.  Otherwise returns
                          //false.
  bool end_detected();                                              //Returns TRUE if the end of a potential gesture
                            //has been detected.  Otherwise returns false.

  bool timed_out();      //Returns TRUE if gesture started but didn't end.

  bool dont_sample_yet();//Returns TRUE if less than 1/SAMPLING_RATE has
                          //elapsed since the last sample was taken.

  timestamp get_current_clk_time();  //Returns the current time,
//in microseconds.



  //VARIABLES
  AccelData * AD;         //A pointer to object holding accelerometer data.

  struct timeval time_val;     //Used to get time values from the system.

  time_t beginning_of_time; //Machine time in (secs) when program starts.
                         //Used as a baseline, so that time starts
                         //at t=0.

  timestamp most_recent_sample_time;  //In microseconds.
  timestamp current_time;           //In microseconds.
  timestamp dummy;
};

#endif //RECORDER_H


//Recorder.cpp
```

```
//Functions for a recorder that records accelerometer
//data into an AccelData object.  The recorder has the
//ability to detect the start and end of a gesture.



//INCLUDE DIRECTIVES
#include "Recorder.h"
#include "AccelData.h"
#include <sys/time.h> //For time stuff
#include <stdio.h>    //for "fprintf" and stderr
#include <unistd.h>


//CONSTRUCTOR
Recorder::Recorder(AccelData * ADparam)
{
  //Set the acceleration data pointer
  AD = ADparam;

  //Zero the initial values for Ax and Ay
  if (AD->from_file())
    AD->set_offsets_from_file();
  else
    AD->set_offsets();

  //Initialize the program's clock
  gettimeofday(&time_val);
  beginning_of_time = time_val.tv_sec;

  //Initialize current & most recent sampling times
  current_time = get_current_clk_time();
  most_recent_sample_time = current_time;
}



//RECORD
//A function that records one potential gesture or the maximum
//number of sample points.  Returns TRUE if a potential
//gesture has been recorded (i.e. a motion with a start
//and a finish), otherwise returns FALSE.
bool Recorder::record()
{

  //Reset the recorder to initial values.
  reset();

  //Let the first samples pass through the filter
  for (int i = 0; i<=FILTER_LENGTH+1; i++)
    {
      //Wait until its time to take the next sample.
      while(dont_sample_yet())
                           {}; //do nothing

      //Update most recent sampling time
      most_recent_sample_time = get_current_clk_time();

      //Take a sample.
      if (AD->from_file())
                           AD->take_sample_from_file();
      else
                           AD->take_sample();
    }


  fprintf(stderr, "\nREADY\n");
  //WIERD HERE, BECAUSE WHILE WAITING FOR GESTURE TO START,
  //CANT MOVE TO A NEW OFFSET POSITION...
```

```
  //Loop until start of a gesture is recognized.
  while(!start_detected())
    {
      //Wait until its time to take the next sample.
      while(dont_sample_yet())
                    {}; //do nothing

      //Update most recent sampling time
      most_recent_sample_time = get_current_clk_time();

      //Take a sample.
      if (AD->from_file())
                    AD->take_sample_from_file();
      else
                    AD->take_sample();
    }


  //  fprintf(stderr, "\n**********START FOUND***********\n");


  //Loop until end of a gesture is recognized.
  while(!end_detected())
    {
      //Wait until its time to take the next sample.
      while(dont_sample_yet())
                    {} //do nothing

      //Update most recent sampling time
      most_recent_sample_time = get_current_clk_time();

      //Take a sample.
      if (AD->from_file())
                    {
                     if (AD->take_sample_from_file()==0)
                       {
                       return(0);
                       }
                    }
      else
                    {
                    AD->take_sample();
                    }

      //If a potential gesture has started, and
      //recording has taken longer than the maximum
      //length of a gesture, stop recording, return FALSE.
      if (timed_out())
                    {
                     //fprintf(stderr, "\n**********TIMED OUT***********\n");
                     return (0);
                    }
    }

  //fprintf(stderr, "\n**********END FOUND***********\n");

  //Start and end have been detected; we got us a potential
  //gesture to analyze!
  return (1);
}



//RESET
//Resets the recorder object to its initial state.
void Recorder::reset()
{
  //  printf("IN RECORDER RESET\n");
```

```
  //Zero the initial values for Ax and Ay
  if (AD->from_file())
    AD->set_offsets_from_file();
  else
    AD->set_offsets();

  //Initialize the program's clock
  gettimeofday(&time_val);
  beginning_of_time = time_val.tv_sec;

  //Initialize current & most recent sampling times
  current_time = get_current_clk_time();
  most_recent_sample_time = current_time;
}



//START_DETECTED
//Returns TRUE if the start of a potential
//gesture has been detected.  Otherwise returns false.
bool Recorder::start_detected()
{
  //Get index of current sample
  int i = AD->get_current_i();

  //If current sample has broken out of the zeroed flat range,
  //the start of the gesture has occured.  Set the start_i index
  //and return TRUE.
  if (  (((AD->get_Ax(i) > VE_START) || (AD->get_Ax(i) < -VE_START)) &&
                              ((AD->get_dAx(i) > DE_START) || (AD->get_dAx(i) < -DE_START))) ||
                             (((AD->get_Ay(i) > VE_START) || (AD->get_Ay(i) < -VE_START)) &&
                              ((AD->get_dAy(i) > DE_START) || (AD->get_dAy(i) < -DE_START)))   )
    {
      int back_up_a_bit_i = i - 5;
      if (back_up_a_bit_i < 0)
                             back_up_a_bit_i += ARRAY_LENGTH;

      AD->set_start_i(back_up_a_bit_i);

      return(1);
    }

  //Start has not occured.  Return FALSE.
  return(0);
}



//END_DETECTED
//Returns TRUE if the end of a potential gesture
//has been detected.  Otherwise returns false.
bool Recorder::end_detected()
{
  //Get index of current sample
  int current_i = AD->get_current_i();

  //Get index of sample that is ENDING_LENGTH samples before the
  //current sample.  Account for array wrapping.
  int earlier_i = current_i - ENDING_LENGTH;
  if (earlier_i < 0)
    earlier_i += ARRAY_LENGTH;

  //Loop through the last ENDING_LENGTH samples. If all of them
  //fall within the magnitude & deriv constraints for a gesture's
  //end, end of a gesture has occured. Set the end_i index.
  //and return TRUE.
  int end_count = 0; //running count of samples that meet end
                   //conditions.
  for (int i = earlier_i; i<current_i; i++)
      if ( ((AD->get_Ax(i) < VE_END) && (AD->get_Ax(i) > -VE_END)) &&
```

```
                                   ((AD->get_dAx(i) < DE_END) && (AD->get_dAx(i) > -DE_END)) &&
                                   ((AD->get_Ay(i) < VE_END) && (AD->get_Ay(i) > -VE_END)) &&
                                   ((AD->get_dAy(i) < DE_END) && (AD->get_dAy(i) > -DE_END)) )
                                 end_count++;

  if (end_count == ENDING_LENGTH)
    {
      AD->set_end_i(current_i);
      return(1);
    }

  //End has not occured. Return FALSE.
  return(0);
}



//TIMED_OUT
//Returns TRUE if gesture started but didn't end.
//NOTE: This functions assumes that a potential gesture has started.
//(The start_i index within the AccelData object called AD has been
//set to a valid value.)  If this is not the case, function won't work.
bool Recorder::timed_out()
{
  //If the filter in front of the AccelData object's current index bumps
  //into the start index, the recorder has gone one full cycle without
  //the gesture ending; the gesture has timed out.  Return TRUE.
  //Otherwise return false.

 return((AD->get_current_i() + FILTER_LENGTH) % ARRAY_LENGTH ==
                                 AD->get_start_i());
}



//DONT_SAMPLE_YET
//Returns TRUE if less than one sample interval has elapsed
//since the last sample was taken.
bool Recorder::dont_sample_yet()
{
  if ( AD->from_file() )
    return 0;
  else
    {
      //Get the current time in microseconds
      current_time = get_current_clk_time();

      //Return TRUE if less than one sample interval has elapsed
      //since last sample was taken, else false.

      double interval = (1.0/(double)SAMPLING_RATE) * 1000000; //-2000???
      double diff = (double)current_time - (double)most_recent_sample_time;

      return ( diff < interval);
    }
}



//GET_CURRENT_CLK_TIME
//Returns the current time, in microseconds.
timestamp Recorder::get_current_clk_time()
{
```

```
  //Get the current time in microseconds
  gettimeofday(&time_val);
  return(((time_val.tv_sec - beginning_of_time) * 1000000) + time_val.tv_usec);
}
-----------------------------------------------------------------------------------------
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++
//Analyzer.h

//This object analyzes the data analyzes accelerometer data to determine
//if one of several specified gestures has been performed.


#ifndef ANALYZER_H
#define ANALYZER_H


//INCLUDE DIRECTIVES
#include "AccelData.h"


//CONSTANTS
const int NUM_FEATURES = 12;


//A struct to hold the time/magnitude/derivative information that characterizes
//a particular feature.
struct FeatureWindow
{
  double HM;  //Horizontal middle
  double HE;  //Horizontal extent (window's horizontal range is: HM +/- HE)
  double VM;  //Vertical middle
  double VE;  //Vertical extent (window's vertical range is: VM +/- VE)
  double DE;  //Derivative extent (deriv is: +/- DE)
};


//A struct to hold which features were found and which weren't
//as an array of bools
struct Result
{
  bool array[NUM_FEATURES];
};


class Analyzer
{

 public:

  //CONSTRUCTOR
  Analyzer(AccelData * ADparam);

  //FUNCTIONS
  Result check_for_features();    //One by one, checks if certain features are
                          //present within the accelerometer data.
                          //Sets and returns a "result" value that
                          //is a binary sum of all the features found.
                          //For example, if only features 1 & 9 are found,
                          //result is set and returned to be 2^1 + 2^9.

  void print_feature_name();    //Prints the name of the feature to screen,
                          //based on the result number.

 private:

  //FUNCTIONS
```

```
  int in_window(const FeatureWindow& W, const bool& x_axis);
                    //Returns the number of samples
                    //in the potenial gesture that
                    //"fit" the feature window. The
                    //second parameter dictates whether
                    //we are examining x or y axis accel data.

  int index(const double& nt); //Given a normalized time value, returns the
                //closest index into the AccelData arrays
                //between potential gesture's start and end
                //indices.

 int two_to_the(const int& x);  //Returns 2^x.

  //DATA
  AccelData * AD;        //A pointer to object holding accelerometer data.

  FeatureWindow W[NUM_FEATURES];  //An array of windows representing the
                 //time/magnitude/derivative criteria
                 //for each looked-for feature in
                 //a potential gesture


  Result result;     //Holds a binary value expressing presence or absence
           //of each feature.
};
#endif //ANALYZER_H



//Analyzer.cpp

//Functions for an object that analyzes accelerometer
//data to determine if one of several specified gestures
//has been performed.


//INCLUDE DIRECTIVES
#include "Analyzer.h"
#include "AccelData.h"
#include <stdio.h>    //for "fprintf" and stderr

//CONSTRUCTOR
Analyzer::Analyzer(AccelData * ADparam)
{
  //Set the acceleration data pointer
  AD = ADparam;

  //Set up feature window data

  //FEATURE 0: A STRONG PEAK IN Ax IN THE MIDDLE OF THE GESTURE.
  W[0].VM = 1.075;
  W[0].VE = .725;
  W[0].HM = .5;
  W[0].HE = .12;
  W[0].DE = .4*(SAMPLING_RATE);

  //FEATURE 1: A STRONG VALLEY IN Ax IN THE MIDDLE OF THE GESTURE.
  W[1].VM = -.975;
  W[1].VE = .625;
  W[1].HM = .44;
  W[1].HE = .16;
  W[1].DE = .4*(SAMPLING_RATE);

  //FEATURE 2:  A WEAK PEAK IN Ax IN THE FIRST HALF OF THE GESTURE.
  W[2].VM = .586;
  W[2].VE = .32;
  W[2].HM = .25;
  W[2].HE = .14;
  W[2].DE = .5*(SAMPLING_RATE);
```

```
 //FEATURE 3:  A WEAK PEAK IN Ax IN THE SECOND HALF OF THE
 W[3].VM = .485;
 W[3].VE = .32;
 W[3].HM = .75;
 W[3].HE = .25;
 W[3].DE = .5*(SAMPLING_RATE);

 //FEATURE 4:  A WEAK VALLEY IN Ax IN THE FIRST HALF OF THE GESTURE.
 W[4].VM = -.585;
 W[4].VE = .325;
 W[4].HM = .20;
 W[4].HE = .12;
 W[4].DE = .5*(SAMPLING_RATE);

 //FEATURE 5:  A WEAK VALLEY IN Ax IN THE SECOND HALF OF THE GESTURE.
 W[5].VM = -.585;
 W[5].VE = .325;
 W[5].HM = .75;
 W[5].HE = .25;
 W[5].DE = .5*(SAMPLING_RATE);

 //FEATURE 6:  A STRONG PEAK IN Ay IN THE MIDDLE OF THE GESTURE.
 W[6].VM = .975;
 W[6].VE = .625;
 W[6].HM = .44;
 W[6].HE = .16;
 W[6].DE = .5*(SAMPLING_RATE);

 //FEATURE 7:  A STRONG VALLEY IN Ay IN THE MIDDLE OF THE GESTURE.
 W[7].VM = -.95;
 W[7].VE = .65;
 W[7].HM = .44;
 W[7].HE = .16;
 W[7].DE = .1*(SAMPLING_RATE);

 //FEATURE 8:  A WEAK PEAK IN Ay IN THE FIRST HALF OF THE GESTURE.
 W[8].VM = .74;
 W[8].VE = .51;
 W[8].HM = .25;
 W[8].HE = .14;
 W[8].DE = .2*(SAMPLING_RATE);

 //FEATURE 9:  A WEAK PEAK IN Ay IN THE SECOND HALF OF THE GESTURE.
 W[9].VM = .715;
 W[9].VE = .535;
 W[9].HM = .75;
 W[9].HE = .14;
 W[9].DE = .2*(SAMPLING_RATE);

 //FEATURE 10:  A WEAK VALLEY IN Ay IN THE FIRST HALF OF THE GESTURE.
 W[10].VM = -.74;
 W[10].VE = .51;
 W[10].HM = .25;
 W[10].HE = .14;
 W[10].DE = .2*(SAMPLING_RATE);

 //FEATURE 11:  A WEAK VALLEY IN Ay IN THE SECOND HALF OF THE GESTURE.
 W[11].VM = -.75;
 W[11].VE = .54;
 W[11].HM = .75;
 W[11].HE = .14;
 W[11].DE = .2*(SAMPLING_RATE);
}


//CHECK_FOR_FEATURES
//One by one, checks if certain features are present within the
//accelerometer data.  Sets and returns a "result" value that
```

```
//is a binary sum of all the features found. For example, if only
//features 1 & 9 are found, result is set and returned to be 2^1 + 2^9.
Result Analyzer::check_for_features()
{

  //Check for X Acceleration Features
  for (int i=0; i<6; i++)
    result.array[i] = in_window(W[i],1);

  //Check for Y Acceleration Features
  for (int i=6; i<12; i++)
    result.array[i] = in_window(W[i],0);

  return(result);
}



//PRINT_FEATURE_NAME
//Prints the name of the feature to screen,
//based on the result number.
void Analyzer::print_feature_name()
{

  /*
  //Print the presence/absence of the 12 features as a binary string.
  fprintf(stderr, "\nFEATURES RECOGNIZED: ");
  for (int i=0; i<NUM_FEATURES; i++)
    fprintf(stderr, "%d", result.array[i]);
  fprintf(stderr, "\n");
  */

  //Print the name of the gesture
  if (!result.array[0] && result.array[1] && !result.array[2] &&
      !result.array[3])
    {
      fprintf(stderr, "TILT-SNAP LEFT\n");
      return;
    }

  else if (result.array[1] && result.array[2] && !result.array[0])
    {
      fprintf(stderr, "LEFT SHIFT\n");
      return;
    }

  else if (result.array[0] && result.array[4])
    {
      fprintf(stderr, "RIGHT SHIFT\n");
      return;
    }

  else if (result.array[7] && result.array[8])
    {
      fprintf(stderr, "OUT & IN\n");
      return;
    }

  else if (result.array[10]) //&& [6] ?
    {
      fprintf(stderr, "IN & OUT\n");
      return;
    }

  else
    {
      fprintf(stderr, "NOT RECOGNIZED\n");
      return;
    }
}
```

```
//IN_WINDOW
//Returns the number of samples that "fit" within the
//magnitude, normalized-time & derivative constraints for
//a given feature. When parameter axis=0, we are examining x-axis
//acceleration data.  When axis=1, we are examining y-axis
//acceleration data.
int Analyzer::in_window(const FeatureWindow& W, const bool& x_axis)
{
  //Variables to hold acceleration and deriv of acceleration samples.
  double A, dA;

  // Look through A and dA from index corresponding to
  //gesture window's normalized start time to index corresponding to
  //window's normalized end time.  If a sample is found that
  //is within the specified accel magnitude and accel derivative ranges,
  //return 1.  If the whole window is searched without a "fit", return 0.

  if (x_axis) //Examine x-axis acceleration data.
    {
      for(int i=index(W.HM-W.HE); i<index(W.HM+W.HE); i++)
                    {
                     A = AD->get_Ax(i);
                     dA = AD->get_dAx(i);

                     if ( (W.VM-W.VE < A) && (A < W.VM+W.VE) &&
                          (-W.DE < dA) && (dA < W.DE)  )
                       return(1);
                    }
      return(0);
    }

  else //Examine y-axis acceleration data.
    {
      for(int i=index(W.HM-W.HE); i<index(W.HM+W.HE); i++)
                    {
                     A = AD->get_Ay(i);
                     dA = AD->get_dAy(i);

                     if ( (W.VM-W.VE < A) && (A < W.VM+W.VE) &&
                          (-W.DE < dA) && (dA < W.DE)  )
                       return(1);
                    }
      return(0);
    }
}



//INDEX
//Given a normalized time value, returns an index into AccelData arrays.
//This index is between the potential gesture's start and end indices.
//(The start index corresponds to a normalized time of 0, and the end
//index corresponds to a normalized time of 1).
int Analyzer::index(const double& nt)
{
  //Get the start and end indices from the AccelData object.
  int start_i = AD->get_start_i();
  int end_i = AD->get_end_i();

  //If, due to wrapping, end_i < start_i, add ARRAY_LENGTH to
  //end_i.
  if (end_i<start_i)
    end_i += ARRAY_LENGTH;

  //Calculate and return the index corresponding to the normalized time:
  //nt.
  return (((int)(((double)(end_i - start_i))*nt) + start_i) % ARRAY_LENGTH);
```

```
}



//TWO_TO_THE
//Returns 2^x.
int Analyzer::two_to_the(const int& x)
{
  int result = 1;

  for(int i=0; i<x; i++)
    result = result * 2;

  return result;
}
------------------------------------------------------------------------------------------




++++++++++++++++++++++++++++++++++++++++++++++++++++++
/*
  Adxl.h

  Subroutines for communicating with the ADXL202 RS-232 evaluation
  board.
   by Michael J. Fromberger <sting@linguist.dartmouth.edu>
  Copyright (C) 2000 The Trustees of Dartmouth College

  $Id$
 */

#ifndef ADXL_H_
#define ADXL_H_

typedef unsigned long  timestamp;

/* Acquire a serial port and return it, or -1 in case of error */
int get_serial_port(int which);

/* Send a request to the port, and read back a packet */
int get_adxl_packet(int fd, double *x_val, double *y_val, timestamp *clk);

#endif /* end ADXL_H_ */

/*
  Adxl.c

  Subroutines for communicating with the ADXL202 RS-232 evaluation
  board.
  by Michael J. Fromberger <sting@linguist.dartmouth.edu>
  Copyright (C) 2000 The Trustees of Dartmouth College

  $Id$
 */
#include "Adxl.h"
#include <stdio.h>
#include <limits.h>
#include <assert.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/time.h>

#include <termios.h>

#define  PACKET_LEN   4 /* size of packets returned by serial device */

/* This is subtracted from seconds, after first initialization */
```

```
static time_t baseline = 0;

/* Open a serial port for reading and writing, return a descriptor */
static int s_open_serial(int which);

/* Set the relevant control values for the descriptor */
static void s_set_params(int fd);

/* {{{ get_serial_port(which) */

/* Acquire a serial port and return it, or NULL in case of error */
int get_serial_port(int which)
{
  int           fd;
  struct timeval tv;

  if(which < 1 || which > 2)
    return NULL;

  if((fd = s_open_serial(which)) < 0)
    return -1;

  s_set_params(fd);

  /* Initialize timestamp offset */
  gettimeofday(&tv);
  baseline = tv.tv_sec;

  return fd;

} /* end get_serial_port() */

/* }}} */

/* {{{ get_adxl_packet(fd, x_val, y_val, clk) */

/* Send a request to the port, and read back a packet */
int get_adxl_packet(int fd, double *x_val, double *y_val, timestamp *clk)
{
  unsigned char  buf[PACKET_LEN];
  unsigned int   val;
  int            rtn;
  struct timeval tv;

  assert(x_val != NULL && y_val != NULL && fd >= 0 && clk != NULL);

  buf[0] = 'G';

  if(write(fd, buf, 1) != 1)
    return 0;

  tcdrain(fd);

  memset(buf, 0, sizeof(buf));
  if((rtn = read(fd, buf, PACKET_LEN)) != PACKET_LEN) {
    return 0;
  }

  /* Get timestamp, subtract baseline offset, and convert to usec */
  gettimeofday(&tv);
  *clk = ((tv.tv_sec - baseline) * 1000000) + tv.tv_usec;

  *x_val = *y_val = 0.0;

  val = (buf[0] * 256) + buf[1];
  *x_val = (double)val / 100.0;

  val = (buf[2] * 256) + buf[3];
  *y_val = (double)val / 100.0;
```

```
  return 1;

} /* end get_packet() */

/* }}} */

/*----------------------------------------------------------------------*/

/* {{{ s_set_params(fd) */

static void s_set_params(int fd)
{
  struct termios    info;

  assert(fd >= 0);

  /* Obtain current settings for this descriptor */
  if(tcgetattr(fd, &info) < 0) {
    perror("s_set_params: tcgetattr");
    return;
  }

  /* Set port speed to 38400bps */
  cfsetispeed(&info, B38400);
  cfsetospeed(&info, B38400);


  info.c_cflag &= ~CSIZE;    /* Clear size bits  */
  info.c_cflag |= CS8;       /* 8 data bits      */

  info.c_cflag &= ~CSTOPB;   /* 1 stop bit       */

  info.c_cflag &= ~PARENB;   /* No parity        */
  info.c_iflag &= ~INPCK;    /* ... on input too */

  info.c_cflag |= (CREAD|HUPCL); /* Read enabled */

  info.c_lflag = 0;          /* no local options */

  info.c_iflag = 0;  /* disable input processing */

  /* Disable flow control */
  info.c_cflag &= ~(IXON|IXOFF);
  info.c_cflag &= ~CNEW_RTSCTS;

  info.c_cc[VMIN] = PACKET_LEN; /* read each character */
  info.c_cc[VTIME] = 5;         /* block until ready   */

  /* Write the changes back */
  if(tcsetattr(fd, TCSAFLUSH, &info) < 0) {
    perror("s_set_params: tcsetattr");
  }

}
/* }}} */
/* {{{ s_open_serial(which) */
static int s_open_serial(int which)
{
  char buf[PATH_MAX];
  assert(which == 1 || which == 2);
  sprintf(buf, "/dev/ttyd%d", which);
  /*fprintf(stderr, "opening %s for read and write\n", buf);*/
  return open(buf, O_RDWR);
}
/* }}} */

/*----------------------------------------------------------------------*/
/* HERE THERE BE DRAGONS                                                */
------------------------------------------------------------------------------------------
```

Appendix V: Final Testing of the Piecewise Algorithm

TESTING 5 GESTURES INDIVIDUALLY

Subject1:

Gesture	Trials	Misinterpret	Noninterpret	Error rate
LEFT SHIFT	35	2	13	0.43
RIGHT SHIFT	35	0	7	0.2
OUT AND IN	35	0	16	0.46
IN AND OUT	35	0	6	0.17
TILT SNAP LEFT	35	4	12	0.46
TOTAL	175	6	54	0.34

Subject2:

Gesture	Trials	Misinterpret	Noninterpret	Error rate
LEFT SHIFT	35	6	2	0.23
RIGHT SHIFT	35	6	7	0.37
OUT AND IN	35	9	0	0.26
IN AND OUT	35	4	1	0.14
TILT SNAP LEFT	35	1	6	0.20
TOTAL	175	26	16	0.24

Subject3:

Gesture	Trials	Misinterpret	Noninterpret	Error rate
LEFT SHIFT	35	4	13	0.49
RIGHT SHIFT	35	2	11	0.37
OUT AND IN	35	2	17	0.54
IN AND OUT	35	2	8	0.29
TILT SNAP LEFT	35	3	4	0.20
TOTAL	175	13	53	0.38

TESTING 5 GESTURES TOGETHER

Subject1:

Gesture	Trials	Misinterpre	Noninterpr	Error rate
LEFT SHIFT	10	1		0.1
RIGHT SHIFT	10		5	0.5
OUT AND IN	10		5	0.5
IN AND OUT	10		2	0.2
TILT SNAP LEFT	10		3	0.3
TOTAL	50	1	15	0.32

Subject2:

Gesture	Trials	Misinterpre	Noninterpr	Error rate
LEFT SHIFT	10		3	0.30
RIGHT SHIFT	10		6	0.60
OUT AND IN	10		2	0.20
IN AND OUT	10	1	1	0.20
TILT SNAP LEFT	10		1	0.10
TOTAL	50	1	13	0.28

Subject3:

Gesture	Trials	Misinterpre	Noninterpr	Error rate
LEFT SHIFT	10	1	3	0.40
RIGHT SHIFT	10		3	0.30
OUT AND IN	10		4	0.40
IN AND OUT	10		3	0.30
TILT SNAP LEFT	10		1	0.10
TOTAL	50	1	14	0.30

COMBINED TEST SUBJECT RESULTS

Subject	Trials	Misinterpret	Noninterpret	Error rate
1	175	6	54	0.34
2	175	26	16	0.24
3	175	13	53	0.38
Total/Average	525	45	123	0.32

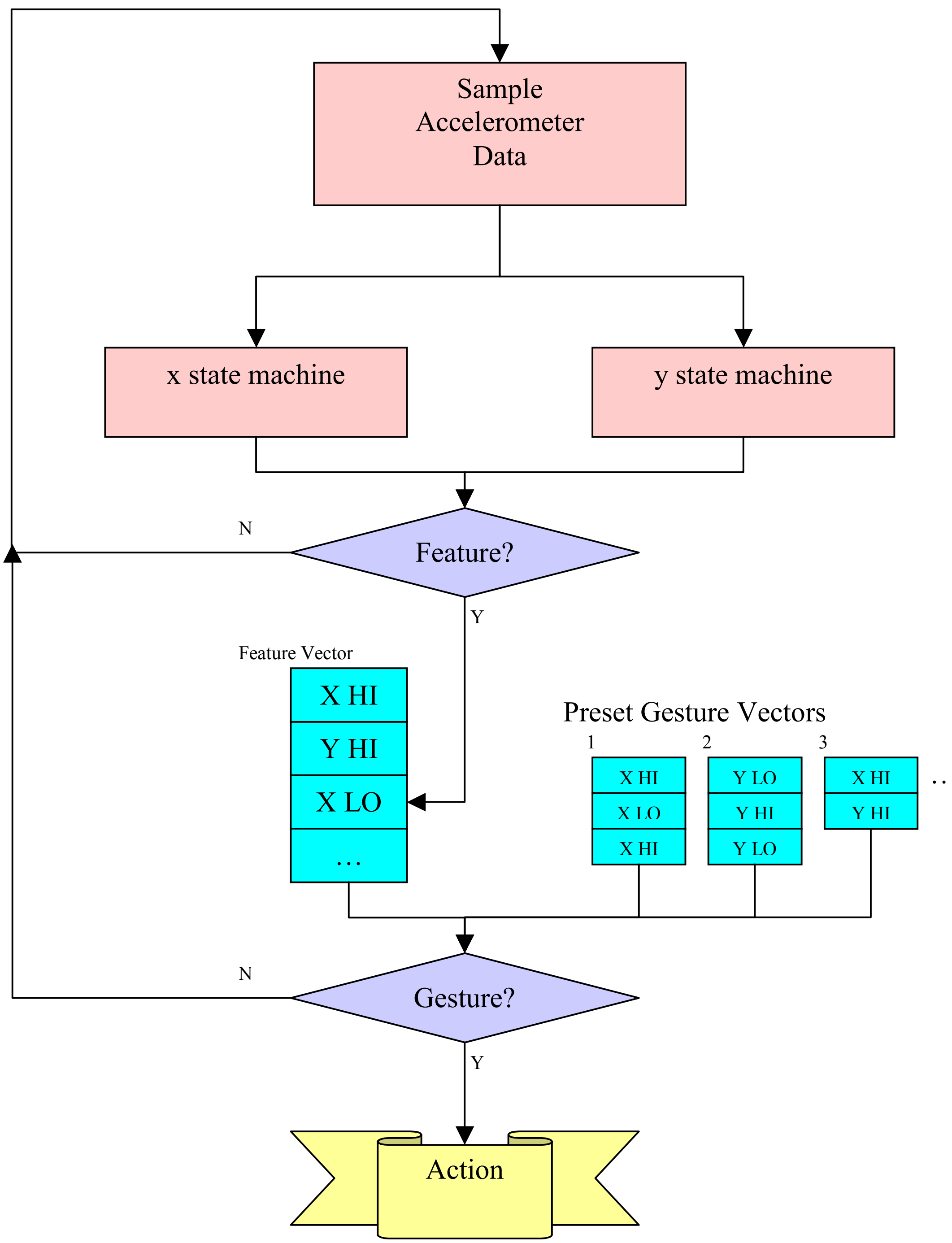
Subject	Trials	Misinterpre	Noninterpr	Error rate
1	50	1	15	0.32
2	50	1	13	0.28
3	50	1	14	0.30
Total/Average	150	3	42	0.30

COMBINED GESTURES DONE INDIVIDUALLY/GESTURES DONE TOGETHER RESULTS

Individ/Together	Trials	Misinterpret	Noninterpret	Error rate
Individual	525	45	123	0.32
Together	150	3	42	0.30
Total/Average	**675**	**0.071**	**0.24**	**0.31**

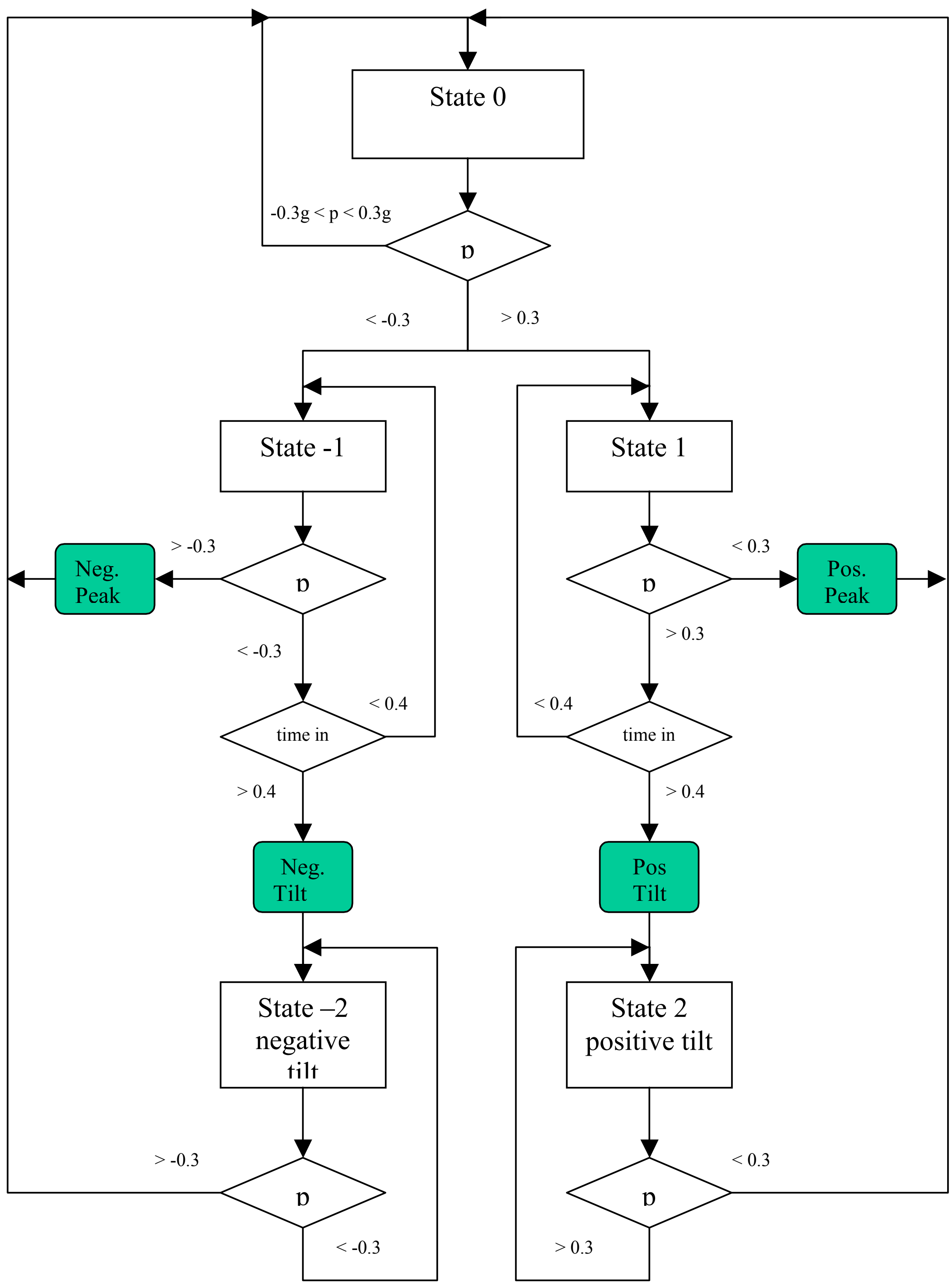
Appendix W: Threshold-Based State Machine #1

Algorithm



Threshold-Based State Machine #1 (Continued)

Corresponding State Machine



Appendix X: Threshold Based State Machine Test Results

Considerations:	Accuracy	Error Rate	False Positives	Num. of Gestures Recognized
Alternatives:				
Threshold Based First Try	75.9%	1.1%	20.7%	6
Threshold Based Second Try	52.4%	28.6%	68.8%	7

Between the first attempt and the second attempt the only change was that the gesture Tilt-snap Left was added to the system. This caused conflict with other gestures and dramatically lowered the recognition rate.

Appendix Y: Modified Hot Sync Cradle

Appendix Z: Accelerometer – Palm III Connection Instructions

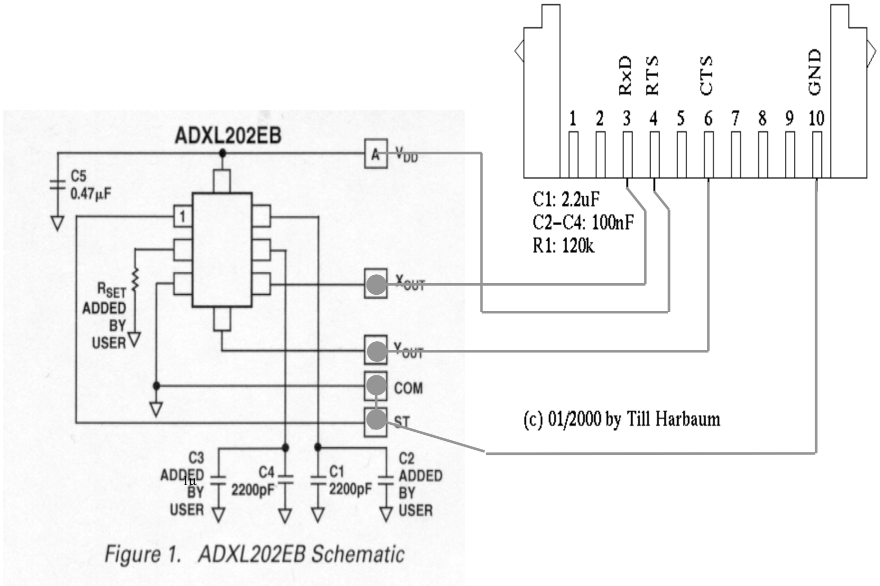
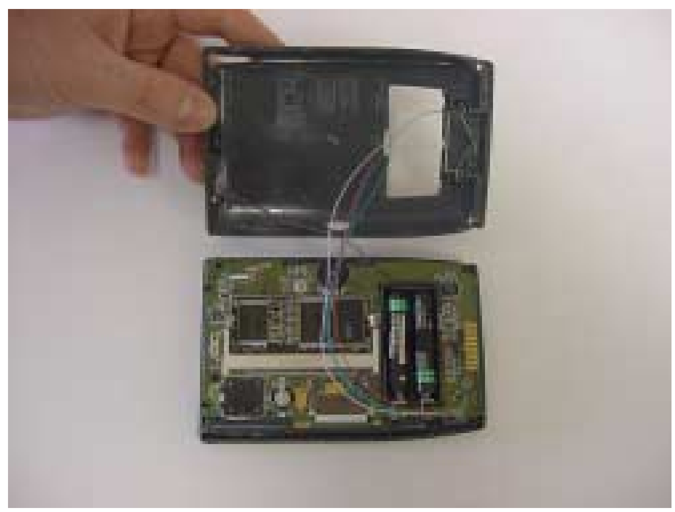


Figure 1. ADXL202EB Schematic

http://www.analogdevice http://bodotill.suburbia.com.au/adxl202/adxl2



Appendix AA: Palm Gesture Recognition Code

```
/**********************************************************************
 *
 *   Internal Functions
 *
 **********************************************************************/

/**********************************************************************
 *
 * FUNCTION:    GetAccelValues
 * DESCRIPTION: Function to retrieve new values from the accelerometer.
 *
 * PARAMETERS:  none
 *
 * RESULTS:     Increments the index variable, puts the x and y values
 *                      from the accelerometer in the appropriate element in the
 *              Ax and Ay arrays. Sets x and y to the average of the
 *              values in the x and y arrays.
 *
 * REVISION HISTORY:
 *
 *
 **********************************************************************/
void GetAccelValues()
{
        index++;
        TiltLibGet(TiltRef, (Word *)&Ax[index%5], (Word *)&Ay[index%5]);
        time = TimGetTicks();

        // finding averaged value
        x = (Ax[0] + Ax[1] + Ax[2] + Ax[3] + Ax[4])/5;
        y = (Ay[0] + Ay[1] + Ay[2] + Ay[3] + Ay[4])/5;
}

/**********************************************************************
 *
 * FUNCTION:    ShiftXZero
 * DESCRIPTION: Implements the floating zero on the x-axis. Should be
 *                      called after GetAccelValues() and before Recognize().
 *
 * PARAMETERS:  none
 *
 * RESULTS:    Updates the zero offset variable x_zero and sets x to be
 *                    the new zeroed value.
 *
 * REVISION HISTORY:
 *
 *
 **********************************************************************/
void ShiftXZero()
{
        if ( x_state == 0 )
                x_zero += x - x_zero/zero_sensitivity;

        // remove if non-zeroed data is not needed
        if (display)
                x_nonzeroed = x;

        x = x - x_zero/zero_sensitivity;
}


/**********************************************************************
 *
 * FUNCTION:    ShiftYZero
 * DESCRIPTION: Implements the floating zero on the y-axis. Should be
 *                      called after GetAccelValues() and before Recognize().
 *
```

```
 * PARAMETERS:  none
 *
 * RESULTS:    Updates the zero offset variable y_zero and sets y to be
 *                the new zeroed value.
 *
 * REVISION HISTORY:
 *
 *
 ********************************************************************/
void ShiftYZero()
{
	if ( y_state == 0 )
		y_zero += y - y_zero/zero_sensitivity;

	// remove if non-zeroed data is not needed
	if (display)
		y_nonzeroed = y;

	y = y - y_zero/zero_sensitivity;
}

/********************************************************************
 *
 * FUNCTION:    Recognize
 * DESCRIPTION: The routine that tries to recognize gestures.
 *
 * PARAMETERS:  none
 *
 * RESULTS:    Returns the index number of the gesture recognized.
 *                      Recognizes all the Tilts and Tilt Snaps
 *
 * REVISION HISTORY:
 *
 *
 ********************************************************************/
int Recognize()
{
	// Recognition algorithm
	if ( x_state == 0 ) {
		if ( x < -x_threshold ) {
			x_state = -1;
			x_state_time = time;
		}
		else if ( x > x_threshold ) {
			x_state = 1;
			x_state_time = time;
		}
	}
	else if ( x_state < 0 ) {
		if ( x > -x_threshold ) {
			x_state_time = time;
			if ( x_state == -1 ) {
				x_state = 0;
				return 1;
			}
			else {
				x_state = 0;
			}
		}
		else if ( time - x_state_time > ticsPerSec*4/10 ) {
			x_state = -2;
			return 5;
		}
	}
	else { // if ( x_state > 0 )
		if ( x < x_threshold ) {
			x_state_time = time;
			if ( x_state == 1 ) {
				x_state = 0;
				return 2;
```

```
                        }
                        else {
                                x_state = 0;
                        }
                }
                else if ( time - x_state_time > ticsPerSec*4/10 ) {
                        x_state = 2;
                        return 6;
                }
        }

        if ( y_state == 0 ) {
                if ( y < -y_threshold ) {
                        y_state = -1;
                        y_state_time = time;
                }
                else if ( y > y_threshold ) {
                        y_state = 1;
                        y_state_time = time;
                }
        }
        else if ( y_state < 0 ) {
                if ( y > -y_threshold ) {
                        y_state_time = time;
                        if ( y_state == -1 ) {
                                y_state = 0;
                                return 3;
                        }
                        else {
                                y_state = 0;
                        }
                }
                else if ( time - y_state_time > ticsPerSec*4/10 ) {
                        y_state = -2;
                        return 7;
                }
        }
        else { // if ( y_state > 0 )
                if ( y < y_threshold ) {
                        y_state_time = time;
                        if ( y_state == 1 ) {
                                y_state = 0;
                                return 4;
                        }
                        else {
                                y_state = 0;
                        }
                }
                else if ( time - y_state_time > ticsPerSec*4/10 ) {
                        y_state = 2;
                        return 8;
                }
        }

        return 0;
}


/***********************************************************************
 *
 * FUNCTION:    Recognize_TiltSnaps
 * DESCRIPTION: The routine that tries to recognize gestures.
 *
 * PARAMETERS:  none
 *
 * RESULTS:     Returns the index number of the gesture recognized.
 *                      Only recognizes Tilt Snaps
 *
 * REVISION HISTORY:
 *
```

```
 *
 *********************************************************************/
int Recognize_TiltSnaps()
{
	// Recognition algorithm
	if ( x_state == 0 ) {
		if ( x < -x_threshold ) {
			x_state = -1;
			x_state_time = time;
		}
		else if ( x > x_threshold ) {
			x_state = 1;
			x_state_time = time;
		}
	}
	else if ( x_state < 0 ) {
		if ( x > -x_threshold ) {
			x_state_time = time;
			x_state = 0;
			return 1;
		}
	}
	else { // if ( x_state > 0 )
		if ( x < x_threshold ) {
			x_state_time = time;
			x_state = 0;
			return 2;
		}
	}

	if ( y_state == 0 ) {
		if ( y < -y_threshold ) {
			y_state = -1;
			y_state_time = time;
		}
		else if ( y > y_threshold ) {
			y_state = 1;
			y_state_time = time;
		}
	}
	else if ( y_state < 0 ) {
		if ( y > -y_threshold ) {
			y_state_time = time;
			y_state = 0;
			return 3;
		}
	}
	else { // if ( y_state > 0 )
		if ( y < y_threshold ) {
			y_state_time = time;
			y_state = 0;
			return 4;
		}
	}

	return 0;
}

/*********************************************************************
 *
 * FUNCTION:    GestureName
 *
 * DESCRIPTION: Returns the name associated with the gesture index.
 *
 * PARAMETERS:  gesture         - number based on which gesture was
 *                                recognized (see function Recognize).
 *
 * RETURNED:    A string with the gesture name.
 *
 * REVISION HISTORY:
```

```
 *
 *
 *********************************************************************/
char* GestureName(int i)
{
   switch (i)
   {
       case 0:
              return "Nothing";
       case 1:
              return "TiltSnap Left";
       case 2:
              return "TiltSnap Right";
       case 3:
              return "TiltSnap Up";
       case 4:
              return "TiltSnap Down";
       case 5:
                     return "Tilt Left";
              case 6:
                     return "Tilt Right";
              case 7:
                     return "Tilt Up";
              case 8:
                     return "Tilt Down";
              default:
                     return "Unknown";
                     break;
       }
}


/*********************************************************************
 *
 * FUNCTION:    Init----Rectangle()
 *
 * DESCRIPTION: This routine sets the values of the rectangles.
 *
 * PARAMETERS:  none
 *
 * RETURNED:    nothing
 *
 * REVISION HISTORY:
 *
 *
 *********************************************************************/
void InitGraphRectangle()
{
       // Rectangle for waveforms
       GraphRect.topLeft.x = 1;
       GraphRect.extent.x = 159;
       GraphRect.topLeft.y = 15;
       GraphRect.extent.y = 85;
}

void InitReportingRectangle()
{
       // Rectangle for reporting gesture
       GestRect.topLeft.x = 40;
       GestRect.extent.x = 58;
       GestRect.topLeft.y = 147;
       GestRect.extent.y = 12;
}

void InitFeedbackRectangle()
{
       DispRect.topLeft.x = 0;
       DispRect.extent.x = display_size*2 + 2;
       DispRect.topLeft.y = 160 - display_size*2 - 2;
       DispRect.extent.y = display_size*2 + 2;
```

```
	WinDrawRectangle(&DispRect, 0);

	DispRect.extent.x -= 2;
	DispRect.extent.y -= 2;
	DispRect.topLeft.x += 1;
	DispRect.topLeft.y += 1;


	origin_x = DispRect.topLeft.x + DispRect.extent.x / 2;
	origin_y = DispRect.topLeft.y + DispRect.extent.y / 2;
}


/***********************************************************************
 *
 * FUNCTION:    DisplayGesturePreferences
 *
 * DESCRIPTION: Pops up the gesture preferences dialog and sets the
 *              appropriate variables
 *
 * PARAMETERS:  none
 *
 * RETURNED:    nothing
 *
 * REVISION HISTORY:
 *
 *
 ***********************************************************************/
void DisplayGesturePreferences()
{
	ListPtr listPX, listPY, listPZ;
	ControlPtr ctlDisplay, ctlZero, ctlSens;
	FormPtr frmP;

	MenuEraseStatus(0);
	frmP = FrmInitForm(AccelPrefsForm);

	listPX = (ListPtr) FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP, AccelPrefsXSensitivityList));
	listPY = (ListPtr) FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP, AccelPrefsYSensitivityList));
	listPZ = (ListPtr) FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP, AccelPrefsZeroSensitivityList));
	ctlSens = (ControlPtr) FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP, AccelPrefsZeroSensitivityPopTrigger));
	ctlZero = (ControlPtr) FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP, AccelPrefsZeroCheckbox));
	ctlDisplay = (ControlPtr) FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP, AccelPrefsDisplayCheckbox));

	LstSetSelection(listPX, x_threshold - 1);
	LstSetSelection(listPY, y_threshold - 1);
	LstMakeItemVisible(listPX, x_threshold - 1);
	LstMakeItemVisible(listPY, y_threshold - 1);

	LstSetSelection(listPZ, zero_sensitivity/10 - 1);
	StrPrintF(tmp, "%d", zero_sensitivity);
	CtlSetLabel( ctlSens, tmp );

	CtlSetValue( ctlZero, zero );
	CtlSetValue( ctlDisplay, display );

	FrmDoDialog(frmP);

	x_threshold = LstGetSelection(listPX) + 1;
	y_threshold = LstGetSelection(listPY) + 1;
	zero_sensitivity = ( LstGetSelection(listPZ) + 1 ) * 10;
	zero = CtlGetValue( ctlZero );
	display = CtlGetValue( ctlDisplay );

	FrmDeleteForm(frmP);
}

/***********************************************************************
 *
```

```
 * FUNCTION:    ReportRecognition
 *
 * DESCRIPTION: Displays the name of the gesture at the bottom of the
 *                  form.  To be run always after the Recognition function.
 *
 * PARAMETERS:  nothing
 *
 * RETURNED:    nothing
 *
 * REVISION HISTORY:
 *
 *
 ***********************************************************************/
void ReportRecognition(UInt16 gest)
{
	if ( gest ) {
		WinEraseRectangle(&GestRect, 0);
		StrPrintF(tmp, "%s", GestureName(gest));
	    WinDrawChars(tmp, StrLen(tmp), 40, 147);
	    displayed = gest;
	    display_time = time;
	}
	else {
		if ( displayed > 0 && displayed <= 4 &&
			 time - display_time > 4 * ticsPerSec/10 ) {
			WinEraseRectangle(&GestRect, 0);
			displayed = 0;
		}
		else if ( displayed > 4 ) {
			WinEraseRectangle(&GestRect, 0);
			displayed = 0;
		}
	}
}

/***********************************************************************
 *
 * FUNCTION:    ReportX_Y
 *
 * DESCRIPTION: Displays the most recent x and y values sampled in the
 *                  bottom left of the display.
 *
 * PARAMETERS:  nothing
 *
 * RETURNED:    nothing
 *
 * REVISION HISTORY:
 *
 *
 ***********************************************************************/
void ReportX_Y(UInt16 gest)
{
	StrPrintF(tmp, "%d    ", x);
	WinDrawChars(tmp, StrLen(tmp), 0, 147);

	StrPrintF(tmp, "%d   ", y);
	WinDrawChars(tmp, StrLen(tmp), 20, 147);
}

/***********************************************************************
 *
 * FUNCTION:    UpdateFeedbackBox
 *
 * DESCRIPTION: Updates the feedback box.
 *
 * PARAMETERS:  none
 *
 * RETURNED:    nothing
 *
 * REVISION HISTORY:
```

```
 *
 *
 *******************************************************************/
void UpdateFeedbackBox()
{
        WinEraseRectangle(&DispRect, 0);
        if (x > x_threshold)
                display_x = display_size;
        else if ( x < -x_threshold )
                display_x = -display_size;
        else
                display_x = x*display_size/x_threshold;
        if ( display_x > display_size )
                display_x = display_size;

        if (y > y_threshold)
                display_y = display_size;
        else if ( y < -display_size )
                display_y = -display_size;
        else
                display_y = y*display_size/y_threshold;
        if ( display_y < -display_size )
                display_y = -display_size;

        WinDrawLine(origin_x, origin_y, origin_x + display_x, origin_y + display_y );
}

/*******************************************************************
 *
 * FUNCTION:    UpdateGraph
 *
 * DESCRIPTION: Updates the graph.
 *
 * PARAMETERS:  nothing
 *
 * RETURNED:    nothing
 *
 * REVISION HISTORY:
 *
 *
 *******************************************************************/
void UpdateGraph()
{
        WinCopyRectangle(NULL, NULL, &GraphRect, 0, 15, scrCopy);

        // erasing the last line
        WinEraseLine(159, 15, 159, 149);


        // draw graphic bars
        if(y < -30) y = -30;
        if(y >  30) y =  30;

        if(x < -30) x = -30;
        if(x >  30) x =  30;

        // draw new part of bar
        WinDrawLine(158,  50+last_x, 159,  50+x);
        WinDrawLine(158, 110+last_y, 159, 110+y);

        last_x = x;
        last_y = y;

        if (display) {
                if(y_nonzeroed < -30) y_nonzeroed = -30;
                if(y_nonzeroed >  30) y_nonzeroed =  30;

                if(x_nonzeroed < -30) x_nonzeroed = -30;
                if(x_nonzeroed >  30) x_nonzeroed =  30;
```

```
            // draw new part of bar
            WinDrawLine(158,  50+last_x_nonzeroed, 159,  50+x_nonzeroed);
            WinDrawLine(158, 110+last_y_nonzeroed, 159, 110+y_nonzeroed);

            last_x_nonzeroed = x_nonzeroed;
            last_y_nonzeroed = y_nonzeroed;
      }
}


/***********************************************************************
 *
 * FUNCTION:     AppStart
 *
 * DESCRIPTION:  Get the current application's preferences.
 *
 * PARAMETERS:   nothing
 *
 * RETURNED:     Err value 0 if nothing went wrong
 *
 * REVISION HISTORY:
 *
 *
 ***********************************************************************/
/*static Err AppStart(void)
{
   GestRecPreference prefs;
   UInt16 prefsSize;

      // Read the saved preferences / saved-state information.
      prefsSize = sizeof(GestRecPreference);
      if (PrefGetAppPreferences(appFileCreator, appPrefID, &prefs, &prefsSize, true) ==
            noPreferenceFound)
      {
            prefs.x_threshold = 10;
            prefs.y_threshold = 10;
            prefs.recognizing = true;
            prefs.zero = true;
            prefs.display = true;
      }
      else {
            x_threshold = prefs.x_threshold;
            y_threshold = prefs.y_threshold;
            recognizing = prefs.recognizing;
            zero        = prefs.zero;
            display     = prefs.display;
      }

      zero_sensitivity = 40;

}
*/

/***********************************************************************
 *
 * FUNCTION:     InitializeGestureRecognition
 *
 * DESCRIPTION:  Get the current application's preferences.
 *
 * PARAMETERS:   none
 *
 * RESULTS:      Returns Err value 0 if nothing went wrong.  Initializes
 *                     tilt sensor library and sets all gesture variables.
 *
 * REVISION HISTORY:
 *
 *
 ***********************************************************************/
Err InitializeGestureRecognition()
```

```
{
	Err err;

   // try to open tilt sensor lib
   err = SysLibFind("Tilt Sensor Lib", &TiltRef);
   if (err)
	err = SysLibLoad('libr', 'Tilt', &TiltRef);

	if(!err) {
	// try to initialize sensor
	err = TiltLibOpen(TiltRef);
	if(err != 0) {
		if(err == TILT_WRONG_OS)
				FrmCustomAlert(AccelErrorAlert, "Wrong tilt sensor support installed.", "", "");

		else if(err == TILT_NO_SENSOR)
				FrmCustomAlert(AccelErrorAlert, "Tilt sensor hardware not found.", "", "");

		else if(err == TILT_HW_BUSY)
				FrmCustomAlert(AccelErrorAlert, "Tilt sensor interface is busy.", "", "");

		else
				FrmCustomAlert(AccelErrorAlert, "Unknown error during tilt sensor detection.", "", "");

		return true;
	}
	}
	else {
	FrmCustomAlert(AccelErrorAlert, "Unable to open tilt sensor driver.", "", "");
	return true;
	}

	TiltLibZero(TiltRef);

	// initializing time keeping variables
	ticsPerSec = SysTicksPerSecond();
	time = TimGetTicks();

	// reseting state variables
	x_state = y_state = 0;
	x_state_time = y_state_time = time;

	// resetting x and y values to 0
	for (index = 0; index < 5; index++) {
		Ax[index] = Ay[index] = 0;
	}

	index = 0;

	return errNone;
}


/***********************************************************************
 *
 * FUNCTION:    GestureRecognitionStop
 *
 * DESCRIPTION: Closes the tilt sensor library.
 *
 * PARAMETERS:  nothing
 *
 * RETURNED:    nothing
 *
 * REVISION HISTORY:
 *
 *
 ***********************************************************************/
void GestureRecognitionStop(void)
{
	UInt16 numapps;
```

```
	if(TiltRef != 0) {
	// close sensor lib
	TiltLibClose(TiltRef, &numapps);

	// Check for errors in the Close() routine
	if (numapps == 0) {
		SysLibRemove(TiltRef);
	}
	}
}
```

Appendix BB: Testing of the PDA Prototype

TESTING 8 GESTURES INDIVIDUALLY

Subject1:

Gesture	Trials	Misinterpret	Noninterpret	Error rate
Tilt Snap Left	35	6	1	20.00
Tilt Snap Right	35	2	0	5.71
Tilt Snap Up	35	0	0	0.00
Tilt Snap Down	35	1	1	5.71
Tilt Left	35	1	1	5.71
Tilt Right	35	0	0	0.00
Tilt Up	35	1	0	2.86
Tilt Down	35	0	0	0.00
Total/Average	280	11	3	5.00

Subject2:

Gesture	Trials	Misinterpret	Noninterpret	Error rate
Tilt Snap Left	35	1	1	5.71
Tilt Snap Right	35	0	0	0.00
Tilt Snap Up	35	1	1	5.71
Tilt Snap Down	35	0	0	0.00
Tilt Left	35	0	0	0.00
Tilt Right	35	0	0	0.00
Tilt Up	35	0	0	0.00
Tilt Down	35	0	0	0.00
Total/Average	280	0.71	0.71	1.429

Subject3:

Gesture	Trials	Misinterpret	Noninterpret	Error rate
Tilt Snap Left	35	2	0	5.71
Tilt Snap Right	35	1	0	2.86
Tilt Snap Up	35	1	0	2.86
Tilt Snap Down	35	1	0	2.86
Tilt Left	35	0	0	0.00
Tilt Right	35	0	0	0.00
Tilt Up	35	1	0	2.86
Tilt Down	35	0	0	0.00
Total/Average	280	2.14	0	2.143

TESTING 8 GESTURES TOGETHER

Subject1:

Gesture	Trials	Misinterpret	Noninterpret	Error rate
Tilt Snap Left	10	2	0	20.00
Tilt Snap Right	10	0	0	0.00
Tilt Snap Up	10	0	0	0.00
Tilt Snap Down	10	1	0	10.00
Tilt Left	10	0	0	0.00
Tilt Right	10	0	0	0.00
Tilt Up	10	0	0	0.00
Tilt Down	10	1	0	10.00
Total/Average	80	4	0	5.00

Subject2:

Gesture	Trials	Misinterpret	Noninterpret	Error rate
Tilt Snap Left	10	0	0	0.00
Tilt Snap Right	10	0	0	0.00
Tilt Snap Up	10	0	0	0.00
Tilt Snap Down	10	1	0	10.00
Tilt Left	10	0	0	0.00
Tilt Right	10	0	0	0.00
Tilt Up	10	0	0	0.00
Tilt Down	10	0	0	0.00
Total/Average	80	0.36	0	1.250

Subject3:

Gesture	Trials	Misinterpret	Noninterpret	Error rate
Tilt Snap Left	10	0	0	0.00
Tilt Snap Right	10	0	0	0.00
Tilt Snap Up	10	0	0	0.00
Tilt Snap Down	10	0	0	0.00
Tilt Left	10	0	0	0.00
Tilt Right	10	0	0	0.00
Tilt Up	10	0	0	0.00
Tilt Down	10	1	0	10.00
Total/Average	80	0.36	0	1.250

COMBINED TEST SUBJECT RESULTS

Gesture	Trials	Misinterpret	Noninterpret	Error rate
Tilt Snap Left	105	9	2	10.48
Tilt Snap Right	105	3	0	2.86
Tilt Snap Up	105	2	1	2.86
Tilt Snap Down	105	2	1	2.86
Tilt Left	105	1	1	1.90
Tilt Right	105	0	0	0.00
Tilt Up	105	2	0	1.90
Tilt Down	105	0	0	0.00
Total/Average	840	2.26	0.595238095	2.857

Subject	Trials	Misinterpret	Noninterpret	Error rate
1	280	11	3	5.000
2	280	2	2	1.429
3	280	6	0	2.143
Total/Average	840	2.26	0.60	2.857

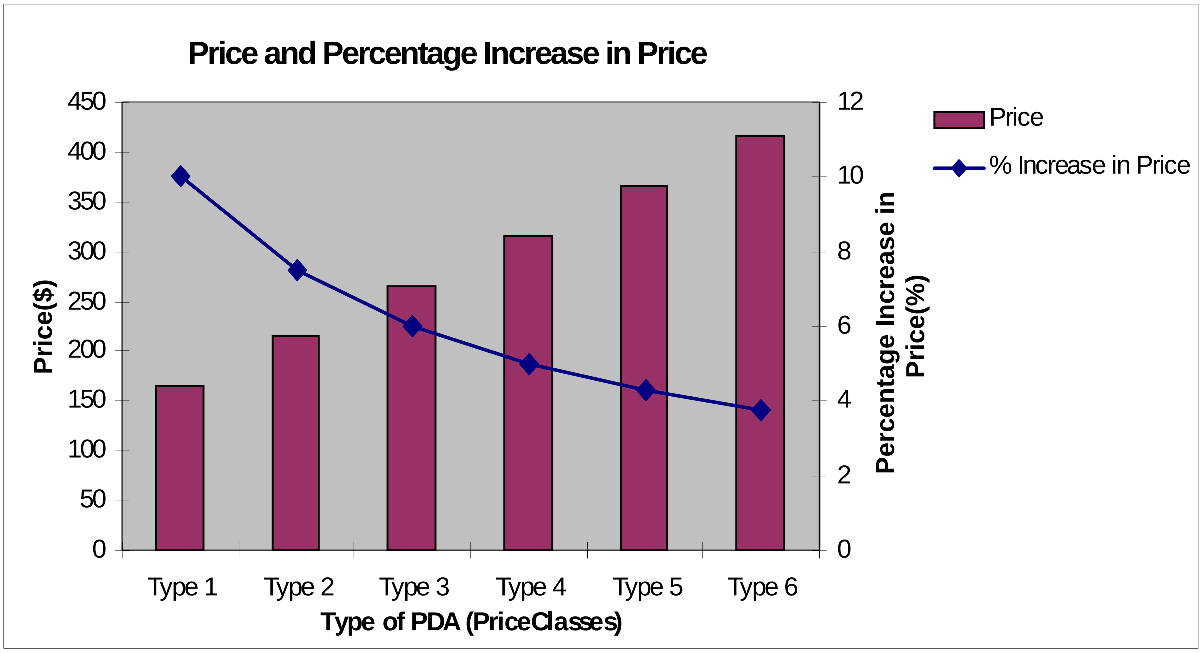
Gesture	Trials	Misinterpret	Noninterpret	Error rate
Tilt Snap Left	30	2	0	6.67
Tilt Snap Right	30	0	0	0.00
Tilt Snap Up	30	0	0	0.00
Tilt Snap Down	30	2	0	6.67
Tilt Left	30	0	0	0.00
Tilt Right	30	0	0	0.00
Tilt Up	30	0	0	0.00
Tilt Down	30	2	0	6.67
Total/Average	240	0.71	0	2.500

Subject	Trials	Misinterpret	Noninterpret	Error rate
1	80	4	0	5.000
2	80	1	0	1.250
3	80	1	0	1.250
Total/Average	240	2.5	0	2.500

COMBINED GESTURES DONE INDIVIDUALLY/GESTURES DONE TOGETHER RESULTS

Individ/Together	Trials	Misinterpret	Noninterpret	Error rate
Individual	840	19	5	2.857
Together	240	6	0	2.5
Total/Average	**1080**	2.31	0.46	**2.679**

Appendix CC: Price and Percentage Increase in PDA Price



Graph showing that as the price goes up, the addition to the price due to the accelerometer goes down. Different price classes correspond to different price classes. Type 1 and 6 correspond to a $150 and $400 PDA respectively.