Adaptive Touch Sampling for Energy-Efficient Mobile Platforms

Alexander W. Min, Kyungtae Han, Dongho Hong and Yong-joon Park
Intel Corporation
2111 N.E. 25th Avenue, Hillsboro, OR 97124
Email: {alexander.w.min, kyungtae.han, dongho.hong, yong-joon.park}@intel.com

***Abstract*—In today's mobile computing environments, touch display and interface is becoming a primary means to enable interactive and perceptual applications, e.g., 3D mobile gaming. To realize highly interactive and responsive applications, touch display constantly scans touch input signals at high frequency, thus wasting energy. However, it is challenging to optimize the touch scan frequency due the highly unpredictable nature of human-touch interactions. In this work, we propose an adaptive touch sampling frequency scaling algorithm based on users' touch behavior, to improve touch devices' energy efficiency while improving user experience. We implemented the proposed algorithm to demonstrate its efficacy for mobile tablet platforms, and our evaluation results show that our proposed adaptive touch scan algorithm reduces touch power consumption by up to 44 % compared to the conventional fixed scan rate algorithm, while enhancing user experience when needed.**

***Index Terms*—Energy efficiency, touch interface, adaptive sampling, mobile platforms.**

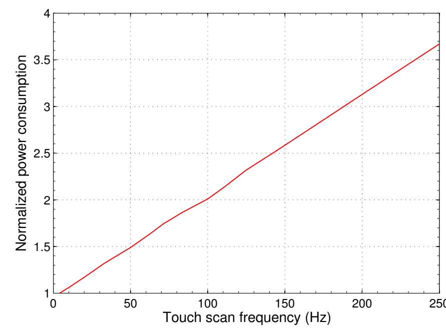


Fig. 1. Normalized power consumption for different touch scan frequencies.

I. Introduction

A. Motivation

Small form-factor mobile platforms, such as smartphones, tablets, and Ultrabooks, are rapidly becoming the main vehicle for computing, networking, and entertaining. Today's mobile devices are equipped with touch interfaces/sensors [1], which are widely used as a primary means to enable intuitive, prompt, and accurate user-device interactions, making their energy-efficient design even more important.

However, despite the rapidly growing demand and popularity of touch interfaces [2], [10], [13], the problem of optimizing touch interface for low-power consumption has received little attention. As the touch interfaces are widely used as a primary means to enable user-device interaction, and the display size of mobile platforms continue to increase, an efficient power management of touch interface becomes even more important. We observed that the current touch interface design is energy inefficient [9] mainly due to its simple design, but exploiting the power-responsiveness tradeoff to improve energy efficiency remains a challenging problem. Touch interface is managed by a touch controller, which in general scans the display at a fixed, pre-defined frequency to detect user input, i.e., (x,y)-coordinate of the touch on the display. Although a frequent scan promises better responsiveness, a too frequent scan wastes power, not only for the touch device itself, but also for the entire platform, because each scan event/interrupt may prevent the CPU/platform from entering lower-power sleep states [8], [11], [12]. In fact, our measurement results reveal that touch power consumption increases almost linearly with touch scan frequency, as shown in Fig. 1. This indicates that lowering scan frequency may reduce the power consumption, at the risk of degraded touch responsiveness and touch detection accuracy, which can ultimately hurt user experience. Therefore, touch scan algorithms must be designed carefully to optimize the power consumption, while delivering good user experience. In this paper, we present a new touch power management framework that intelligently adapts the touch scan rate "on-the-fly" based on user behavior.

B. Limitation of Current Approach

Most of the current touch sampling algorithms are not optimized in terms of power and performance, because they use fixed scan frequency, regardless of users' touch behavior. For example, Microsoft defined the minimum scan frequency of 100 Hz (i.e., 10 ms scan interval) to guarantee good touch responsiveness [3]. On today's touch-enabled mobile devices, the scan frequency is also fixed at 111 Hz when touch events occur. However, we observed that 10 ms scan interval may not be sufficient to accurately detect the trace of fast touch motion. Fig. 2 illustrates two circles drawn with fast touch motion under different scan frequencies. The circle under 10 ms scan interval (see Fig. 2(a)) appears as a polygon rather than a circle. From the power perspective, if the touch scan interval is reduced from 10 ms (i.e., 100 Hz) to 4 ms (i.e., 250 Hz) for better user experience (see Fig. 2(b)), the (normalized) touch power consumption will be significantly increased by 1.8 times from 2.01 to 3.67, as shown in Fig. 1. On the other hand, 10 ms scan interval might be too frequent (i.e., over-sampling) for slow touch motion, resulting in a waste of energy, as shown in Figs. 2(c), 2(d). Our measurement study indicates that, for slow touch motion, 10 ms scan interval oversamples the touch input (see Fig. 2(c)), hence wasting power, whereas 30 ms scan interval still captures enough touch events to draw a smooth circle (see Fig. 2(d)). Thus, a touch scan algorithm with fixed scan rate (e.g., once every 10 ms) may not be sufficient to provide best touch experience with high energy efficiency.

To address this problem, we propose an intelligent touch interface control system that can dynamically adjust touch scan frequency "on-the-fly" based on the speed of the touch motion, thus saving power and improving touch responsiveness. The key idea is to adjust the touch interval (or sampling frequency) so that the distance between two consecutive touches can be maintained at a predefined target distance. By doing this, we can avoid over- or under-sampling touch inputs, making it energy efficient while providing a high-quality, consistent touch experience. Specifically, we employ the proportional-integral-derivative (PID) controller [6] to accurately track the touch input coordinates and speed, and implemented it in Intel platforms.

II. The Proposed Dynamic Touch Scan Rate Optimization

In this section, we introduce the proposed adaptive touch sample algorithm that uses the PID controller to adjust touch sample rate based on user inputs.

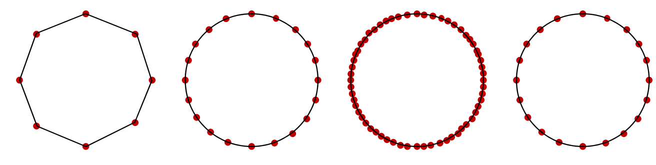


Fig. 2. Circles drawn with: (a) fast motion, 10 ms scan interval; (b) fast motion, 4 ms scan interval; (c) slow motion, 10 ms scan interval; and (d) slow motion, 30 ms scan interval.

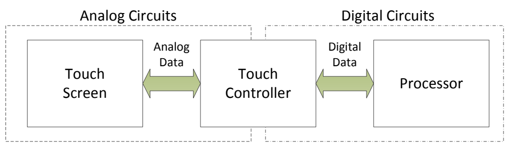


Fig. 3. Block diagram of touch screen system.

A. Background

A touch screen is an electronic visual display that users can control through touch gestures, by touching the screen with one or more figures or other objects (e.g., touch pen). There are a variety of touch screen technologies with different methods of sensing touch, e.g., resistive or capacitive touch screens. In general, a touch screen system is composed of three components: (i) a touch screen, (ii) a touch controller, and (iii) a processor, as shown in Figure 3. The touch screen senses the touch input in the screen surface as analog data. Then the touch controller converts the analog sensing data into digital value using an A/D converter, and calculates the location on the touch using signal processing algorithms, including noise filtering. The processor receives the touch location through serial interface (e.g., I2C or USB), and sends it to user applications, such as drawing.

The touch screen samples touch events at a pre-defined scan rate, which affects touch responsiveness and power consumption. The scan rate determines the frequency of the touch samples being produced by the touch controller, i.e., a higher scan rate results in more sampled data. Higher scan rate, in general, provides higher touch responsiveness at the cost of higher power consumption, due to the additional energy overhead to the analog and digital circuits. Some touch controllers can dynamically change the scan rate by programming in the processor. We use this programming feature in the touch controllers. The next section presents the proposed dynamic touch scan rate algorithm to optimize power and responsiveness.

B. Proposed Adaptive Touch Architecture

The ultimate goal of our adaptive touch sample algorithm is to minimize the number of touch sample instances, thus minimizing the touch-incurred energy consumption, while enhancing (or at least preserving) touch quality, e.g., responsiveness. To achieve this goal, we propose to adjust touch scan frequency based on touch speed—e.g., lower the touch sample rate when a user's touch motion is slow, and vice versa. This is quite different from the traditional touch scan mechanism, where the scan rate is fixed regardless of the speed of the touch motion.

One of our key observations is that, with the traditional fixed scan rate algorithm, the touch samples are often over-sampled or under-sampled depending on the touch input speed, resulting in either (i) a waste of energy (when over-sampled) or (ii) a degradation of touch quality (when under-sampled). For example, when touch motion is fast, the inter-touch-sample distance becomes too long due to under-sampling, as shown in Fig. 2(a). On the other hand, when touch motion is slow, the inter-touch-sample distance becomes unnecessarily close, wasting power without improving touch experience.

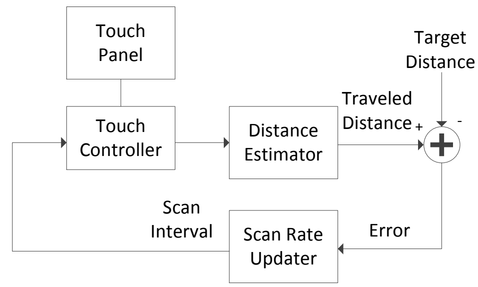


Fig. 4. Block diagram of the proposed dynamic touch scan rate architecture.

The proposed algorithm aims to maintain the sampled touch distance to a desired target distance by dynamically adjusting the scan rate on-the-fly, based on previous touch history. By doing this, our approach can not only reduce the power consumption significantly by avoiding an over-sampling of touch inputs, but also provide a smoother touch experience. After each touch input, a new scan rate is calculated by keeping track of the distance that a touch finger/stylus travels per scan, thus providing consistent user experience regardless of the touch motion speed. In addition, for slow touch motion, it provides significant power savings due to its ability to dynamically reduce the touch scan rate.

In order to maintain the sampled touch distance at a desired target distance, we employed a PID control mechanism. The PID controller suits our need because it has generic control loop feedback mechanism widely used in control systems, and it has historically been considered to be the best controller in the absence of knowledge of the underlying process [7]. Basically, the controller attempts to minimize the error by adjusting the control inputs, where the error values are calculated as the difference between a measured distance and a target distance. Fig. 4 illustrates the proposed scan rate adaptation algorithm using the PID controller.

There are three new blocks for the dynamic scan rate adaptation: (i) distance estimator, (ii) comparator, and (iii) scan rate updater.

- The distance estimator calculates the traveled distance between two consecutive touch samples. In the implementation, we considered a pixel as a unit of traveled distance. We employed a simple distance estimation to minimize computation, as:

$$\Delta d_i = |x_i - x_{i-1}| + |y_i - y_{i-1}|, \tag{1}$$

  where x_i and y_i are the pixel coordinates touch occurred at the i_{th} touch event.
- The comparator calculates the actual error, i.e., the difference between actual traveled distance and the pre-defined target distance.
- The scan rate updater updates the touch scan rate based on the three error components with their associated weights (see Eq. (2)).

These three components closely interact with each other, and update the touch scan interval while touch inputs are generated.

C. Algorithm Description

Here we elaborate on the proposed touch scan rate algorithm. In the PID controller, a control output, i.e., touch scan rate, can be calculated with summation of the proportional, integral, and derivative terms, as shown below:

$$u(t) = K_p e(t) + K_i \int_0^t e(r)dr + K_d \frac{d}{dt} e(t), \tag{2}$$

where K_p, K_i, K_d, e and t denote the proportional (P) gain, integral (I) gain, derivative (D) gain, error, and time, respectively. In the

Algorithm 1 ADAPTIVE TOUCH SCAN RATE ALGORITHM

set K_p, K_i, K_d, initial integral, initial scan interval, max scan interval, min scan interval and target distance
while (true)
  scan interval = initial scan interval
  integral = initial integral
  previous error = 0
  while (consecutive touch events)
    calculate Δd
    error = target distance - Δd
    integral = integral + error $\times$ scan interval
    derivative = (error - previous error) / scan interval
    scan interval = K_p $\times$ error + K_i $\times$ integral + K_d $\times$ derivative
    if scan interval $>$ max scan interval **then**
      scan interval = max scan interval
    if scan interval $<$ min scan interval **then**
      scan interval = min scan interval
    previous error = error
    update touch controller with scan interval
  end while
end while

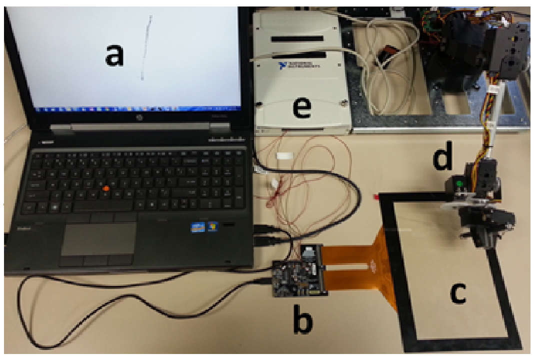


Fig. 5. **Evaluation setup:** (a) host, (b) touch controller, (c) touch panel, (d) robot arm, and (e) power measurement unit.

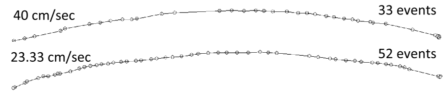
implementation, we empirically set the PID coefficients P, I, and D, as 0.1, 2000, and 0.005, respectively, throughout the measurement tests. The coefficient can be further optimized to improve the efficiency or accuracy of the controller.

Algorithm 1 describes the PID algorithm employed in the proposed touch scan rate adaptation. The initial touch scan interval is set properly to provide a good responsiveness for the first touch event. In the algorithm, we set the initial scan interval to 10 ms. We limit the upper bound of the touch scan interval as 35 ms to guarantee minimum responsiveness and react effectively against sudden motion speed changes. Also, we limit the lower bound of the touch scan interval at 4 ms due to hardware limitation of the touch sensor device.

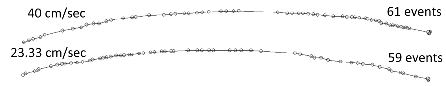
The most important parameter is the target distance, i.e., the desired distance between touch samples. The target distance should be set carefully, so that it can provide a smooth touch experience while avoiding over-sampling of the touch events. The optimal target distance also depends on the size and resolution of the panel. In the implementation, we empirically set the target distance fixed at 15.

III. Evaluation

In this section, we demonstrate the efficacy of the proposed adaptive touch scan algorithm via in-depth measurement results on tablet mobile platform. We first present the testing setup and methodology, and then evaluate the performance.

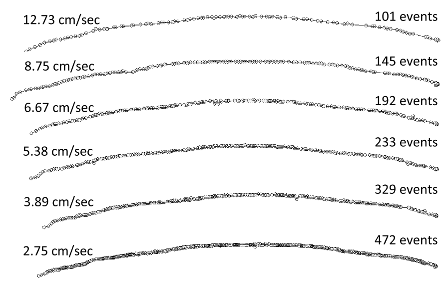


(a) Conventional

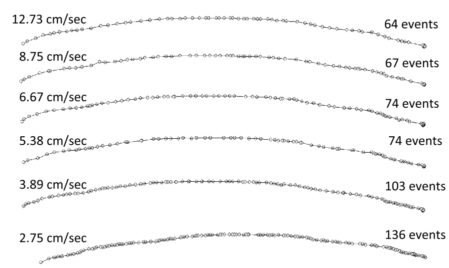


(b) Proposed

Fig. 6. **Curve drawings using robot arm for relatively fast touch motion speeds:** (a) Traditional fixed scan interval (10ms) and (b) Proposed dynamic scan interval. The proposed algorithm generates more touch events than (a), improving touch experience at the cost of increased touch power consumption.



(a) Conventional



(b) Proposed

Fig. 7. **Curve drawings using robot arm for relatively slow touch motion speeds:** (a) Traditional fixed scan interval (10ms), and (b) Proposed dynamic scan interval. The proposed algorithm generates less touch events than (a), significantly reducing power without hurting touch experience.

A. Evaluation Setup and Methodology

In our experiments, we compare the following two testing schemes: (i) the conventional fixed touch scan interval scheme (scan interval fixed at 10 ms), and (ii) the proposed dynamic touch scan interval scheme. For repeatable experiments in a controlled environment, we used a robot arm (Fig. 5(d)) to draw a constant testing pattern with a configurable drawing speed on the touch panel (Fig. 5(c)). The touch controller (Fig. 5(b)) processes a touch event when a touch is detected by the touch panel and sends the event packet to the host (Fig. 5(a)). The event packet contains coordinates of the touch event on the screen. The host calculates Δd and feeds the PID algorithm with Δd and the current touch scan interval. PID returns the new touch scan interval and the host updates the touch controller's registry with

TABLE I
PERFORMANCE/POWER COMPARISON: TRADITIONAL VS. PROPOSED TOUCH SCAN ALGORITHMS

		conventional fixed scan interval				proposed dynamic scan interval			
speed (cm/sec)	time (ms)	event count	frequency (Hz)	avg. scan interval (ms)	power (normalized)	event count	frequency (Hz)	avg. scan interval (ms)	power (normalized)
40.00	350	33	94.29	10.6	1.00	61	174.29	5.7	1.52
23.33	600	52	86.67	11.5	0.94	59	98.33	10.2	1.02
12.73	1100	101	91.82	10.9	0.98	64	58.18	17.2	0.76
8.75	1600	145	90.63	11.0	0.98	67	41.88	23.9	0.66
6.67	2100	192	91.43	10.9	0.98	74	35.24	28.4	0.61
5.38	2600	233	89.62	11.2	0.97	74	28.46	35.1	0.57
4.52	3100	281	90.65	11.0	0.97	91	29.35	34.1	0.57
3.89	3600	329	91.39	10.9	0.97	103	28.61	35.0	0.57
3.41	4100	379	92.44	10.8	0.99	113	27.56	36.3	0.56
3.04	4600	425	92.39	10.8	0.99	124	26.96	37.1	0.56
2.75	5100	472	92.55	10.8	0.99	136	26.67	37.5	0.56

the new touch scan interval for the next scan. The touch controller is wired into the power measurement unit (Fig. 5(e)).

The hardware specification of the test environment is listed below:

- Host computer: Intel Core i5 CPU @2.6 GHz; 4 GB RAM; Windows 7
- Touch panel: 10.1”
- Robot arm: RB-Lyn-644 [4]
- Power measurement unit: NI USB-6289 [5]

B. Evaluation Results

Touch sampling frequency has a direct impact on the smoothness of touch experience, especially for applications such as drawing, scrolling, dragging; e.g., a too infrequent touch event sampling may cause either (i) an angulated curve for drawing applications, or (ii) a jerky animation for scrolling and dragging applications. On the other hand, a too frequent touch sampling beyond a certain frequency level may result in a waste of energy, without improving smoothness. Our evaluation results indicate that the proposed adaptive touch sampling algorithm minimizes energy consumption while providing smooth touch experience.

Figs. 6, 7 show the drawings captured by the robot with the speed of the touch motion and number of touch events generated during the test. We made two key observations. First, when the touch speed is relatively fast (i.e., 23.33 cm/sec and 40 cm/sec), as shown in Fig. 6, the proposed scheme samples touch events more frequently to provide a better touch experience or smoothness, at the expense of increased touch controller power consumption. Second, when the touch speed is relatively slow (i.e., from 2.75 cm/sec to 12.73 cm/sec), as shown in Fig. 7, the proposed scheme samples less number of touch events to avoid over-sampling without degrading touch experience, thus significantly reducing touch power consumption. Note that the typical touch input speed belongs to this touch speed range, thus our algorithm can save power in most of the practical touch usage scenarios. Therefore, we can conclude that the proposed touch sampling algorithm can balance the performance-power tradeoff more efficiently, thanks to its ability to dynamically adjust touch frequency on-the-fly.

Table I compares the touch power consumption, among other metrics, between the conventional fixed scan rate algorithm, and the proposed adaptive scan rate algorithm. The measurement results show that the fixed scan rate algorithm consumes a similar amount of power regardless of the touch motion speed, being highly inefficient. In contrast, the touch power consumption of the proposed algorithm gradually decreases as the touch motion speed decreases from 40 cm/sec to 2.75 cm/sec, and it reduces the power consumption by up to 44 % at a very slow speed, demonstrating its efficacy in balancing the performance-power tradeoff. Note that the power overhead from switching the scan rates is negligible, and power savings from our algorithm outweighs them.

IV. CONCLUSION

The current touch sampling algorithm is highly energy inefficient, mainly because it uses fixed touch sampling rate. To address this problem, we proposed a new touch scan frequency control system that can adapt the scan frequency “on-the-fly” based on the speed of touch motion—it dynamically increases the scan frequency for fast touch motion and decrease scan frequency for slow touch motion—thus achieving better touch responsiveness and higher energy efficiency for mobile platforms. For future work, we are considering having a universal target distance using DPI of display screen by converting number of pixels to inch to make the proposed technique be independent from the platform screen size and resolution.

REFERENCES

[1] Android Sensor Box, `https://play.google.com/store/apps/details?id=imoblife.androidsensorbox&hl=en`.

[2] The Leap Motion Controller, `https://www.leapmotion.com`.

[3] Microsoft device requirements for touch sampling rate, `http://msdn.microsoft.com/en-us/library/windows/hardware/jj134351.aspx/`.

[4] Lynxmotion AL5D 4 Degrees of Freedom Robotic Arm, `http://www.robotshop.com/lynxmotion-al5d-4-degrees-robotic-arm-flowstone.html`.

[5] National Instrument USB-6289 Multifunction DAQ, `http://sine.ni.com/nips/cds/view/p/lang/en/nid/209154`.

[6] K. Åström and T. Hägglund. *Advanced Pid Control*. ISA-The Instrumentation, Systems, and Automation Society, 2006.

[7] S. Bennett. *A History of Control Engineering, 1930-1955*. IEE control engineering series. P. Peregrinus, 1993.

[8] L. S. Brakmo, D. A. Wallach, and M. A. Viredaz. μSleep: A Technique for Reducing Energy Consumption in Handheld Devices. In *ACM MobiSys*, June 2004.

[9] A. Carroll and G. Heiser. An Analysis of Power Consumption in a Smartphone. In *USENIX ATC*, June 2010.

[10] Juha Häikiö and Minna Isomursu and Tapio Matinmikko and Arto Wallin and Heikki Ailisto and Tua Huomo. Touch-Based User Interface for Elderly Users. In *Mobile HCI*, September 2007.

[11] A. W. Min, R. Wang, J. Tsai, M. A. Ergin, and T.-Y. C. Tai. Improving Energy Efficiency for Mobile Platforms by Exploiting Low-power Sleep States. In *ACM Computing Frontiers*, May 2012.

[12] A. W. Min, R. Wang, J. Tsai, and T.-Y. C. Tai. Joint Optimization of DVFS and Low-power Sleep-state Selection for Mobile Platforms. In *IEEE ICC*, June 2014.

[13] L. Zhong and N. K. Jha. Energy Efficiency of Handheld Computer Interfaces: Limits, Characterization and Practice. In *ACM MobiSys*, June 2005.