Clustering-Based Multi-Touch Algorithm Framework for the Tracking Problem with a Large Number of Points

Shih-Lun Huang[†], Sheng-Yi Hung[†], and Chung-Ping Chen[†‡]
Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan[†]
Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan[‡]

ABSTRACT

Microcontrollers (MCUs) are extensively used in consumer devices for specific purposes because they are tiny, cheap, and low-power. Any time-consuming algorithm and any large-size program are not suited for MCUs. Recently, we found that the conventional multi-touch algorithm becomes computationally expensive to handle the applications of large-sized touch panels. Although a more high-end MCU can obtain an improvement on speed, it would increase manufacturing cost and operating power consumption as well. In the whole multi-touch algorithm flow, point tracking is the most computationally expensive part. Fortunately, touch point tracking is similar to the pin-assignment problem in EDA. To accelerate tracking, we employ EDA techniques, such as clustering, to speed up our multi-touch algorithm. Besides, we prove that the tracking problem would be solved in $O(n)$ time for practical cases and without losing its accuracy after clustering. Furthermore, we apply computational geometry techniques to develop an efficient clustering method. Experimental results show that clustering is efficient and effective. For the necessary requirement of large-area touch panels having 20 touch points, we can reduce the runtime by up to 70%. Besides, our multi-touch algorithm may support up to 80 touch points accompanied by a low-cost MCU.

1. INTRODUCTION

As the techniques for manufacturing touch panel continue to advance, the more large-sized touch panels are being produced. As shown in Figure 1, this experimental system includes a 23-inch multi-touch panel, which can support 15 touch points within 10-millisecond (ms) response time. These requirements are necessary for the certification of Windows 8 [3]. Currently, the touch tables are developed and strongly introduced [2], integrating the large-sized multi-touch display module of [1], which can allow for at least 40 touch points without considering response time and achieve fast 12-ms touch response with up to 20 touch points. Because large-sized touch panels can provide more intuitive and more convenient interaction interface, many complex applications such as multi-user collaboration and gaming on the large-sized touch panel require an efficient and effective multi-touch algorithm to work smoothly. In usual business strategies to simplify the system construction, a touch IC is only responsible for the tasks about touch signal sensing and sensing data processing. Also, in the keen commercial competition, the cost and the performance drive the success. According to previous two reasons, microcontrollers (MCUs) are the best choices for touch ICs. In this paper, we aim to propose an algorithm framework ideally suited to implement on an MCU, especially, for the even low-end MCU.

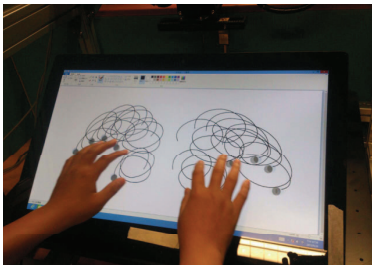

Figure 1: An experimental 23-inch multi-touch panel, where the tracks of 10 touch points are displayed.

Multi-touch algorithms are used to recognize all touch points and identify the moves of touch points. The performance of multi-touch algorithms can have a great impact on the fluency and the correctness of use. Multi-touch algorithms must give the accurate point positions and the correct tracking identifications. Recently, there are many methods proposed to calculate the accurate positions for many types of touch panels in [4, 5, 15]. However, as the touch points increases, the correct tracking identifications become more important and more difficult. In addition, as the touch panel size becomes larger, the touch points will be increasing. That is to say, more tasks need to be done in the same response time. Therefore, an efficient and effective algorithm plays an important role in all of the limited resources.

1.1 Previous Work

Point tracking is the most time-consuming part in the multi-touch algorithm, so there are a few methods proposed to speed up the calculation. Wang et al. in [14] proposed the Minimum Distance First (MDF) method to identify and track the corresponding touch points in two sequential frames. The MDF method considers the touch points having the shortest distance as the corresponding touch points. However, the correct solutions may not exist in the MDF method. Huang and Chen in [9] gave an example to show this case. For improving the accuracy, Simmons and Pickett in [13] first formulated the multi-touch tracking problem as a minimum weighted bipartite matching problem. Because the permutation method needs $O(n!)$ time to calculate the best solution, they proposed a greedy method to solve the prob-

978-3-9815370-4-8/DATE15/©2015 EDAA

lem for developing a suitable algorithm on an MCU. However, the method cannot guarantee finding an optimal solution. Huang and Chen in [9] used the Hungarian method to solve the problem, which guarantee an optimal solution. In addition, they also proposed a pre-processing method to accelerate the Hungarian method. However, this method cannot be adopted by large-sized multi-touch panels with supporting more than 20 touch points, since the complexity of the Hungarian method is $O(n^3)$. Although a more high-end MCU may obtain an improvement on speed, it would increase manufacturing cost and operating power consumption.

1.2 Our Contributions

In this paper, we propose a clustering method that partition the tracking problem of large-sized touch panels efficiently, and integrate the clustering method into the existing multi-touch algorithm. Clustering is a well-known way to organize objects into groups whose members are similar. In EDA, many examples employ clustering to reduce the sizes of their problems, such as circuit partitioning [12, 6, 7]. Hence, utilizing the concept of clustering, we can partition a large-scale tracking problem first, and then use the existing methods to solve each smaller problem. We apply the *plane sweep paradigm* [8] to develop the clustering method, which is fast and its time complexity is merely $O(n\log n)$. After our clustering, the tracking problem would be solved in $O(n)$ time for practical cases.

Experimental results show that clustering can provide a fast and effective partitioning and keep the large-scale tracking to run under 10 ms, which is the acceptable latency of the user experience mentioned in [11]. Compared with [9], our first and second method can further reduce the execution time averagely by 85.6% and 91.9% respectively on an ARM-M0 processor, which is a common low-power and low-cost MCU in the market.

The rest of this paper is organized as follows. Section 2 gives an introduction to the architecture of a large-scale multi-touch panel system, describes the point tracking problem, and formulates the clustering problem. Section 3 presents a cluster-based multi-touch algorithm framework, analyzes the effect of clustering on tracking, and employs computational geometry techniques to develop a clustering algorithm. Experimental results are reported in Section 4, and conclusions are given in Section 5.

2. PRELIMINARIES

We first give an introduction to the architecture of a large-sized touch panel system. Then, we describe the point tracking problem, and formulates the clustering problem.

2.1 Architecture of Touch Panel System

A large-sized touch panel system consists of a large-sized touch panel, a cascaded touch controller, and a low-power MCU, as shown in Figure 2. The touch panel has multiple driving channels and sensing channels. The cascaded touch controller (CTC) is made of several touch controllers (TCs), which is used for a large-sized touch panel system, since single TC cannot meet the needs of driving and sensing. In the beginning, the MCU gives the driving and sensing commands to the CTC. Then, the CTC controls the driving signals to the driving channels, and captures the capacitance changes of the touch panel from the sensing channels. The

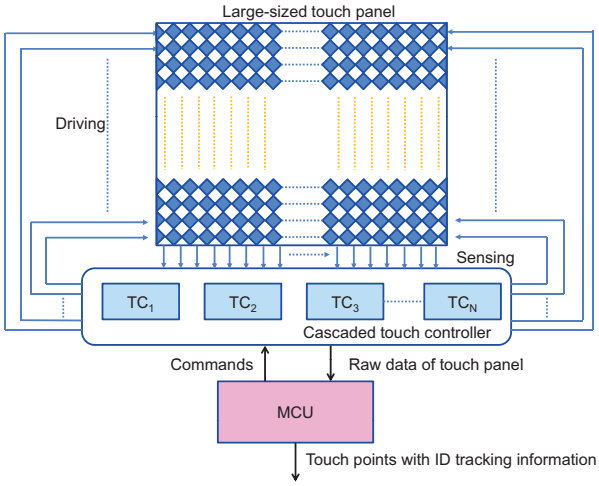

Figure 2: The architecture of a large-sized touch panel system.

CTC would transmit the collected raw data of the whole panel to the MCU. The MCU would process the raw data and output the touch points with ID tracking information by the multi-touch algorithm. After that, the MCU will give the next driving and sensing commands to the CTC again as the above steps.

2.2 Point Tracking

Let $P = \{p_1, p_2, p_3, ..., p_m\}$ be a set of touch points of Frame A and $C = \{c_1, c_2, c_3, ..., c_n\}$ be a set of touch points of Frame B. Frame A and Frame B are two sequential frames from time to time. The information between P and C we have is the Euclidean distance between every point in P and every point in C. In addition, a distance threshold is given to eliminate those impossible moves and identify the new touch event due to the reaction speed limitation of human fingers. From the cases, we induce all possibilities in Figure 3. A real connection means that the current one is moved from the previous one whose Euclidean distance is smaller than the threshold, as shown in Figure 3(a). A fake connection means that they are two individual touch points because the Euclidean distance is larger than the threshold, as shown in Figure 3(b). No connection means that the current one is a new touch event because of no previous one to match up, and vice versa, as shown in Figures 3(c) and (d). Here, our goal is to differentiate the moves of the touch points in Frame A and Frame B from one of the four cases and that is called point tracking. For the accuracy, Simmons and Pickett in [13] first formulated the tracking problem as the minimum weighted bipartite matching problem. For the efficiency and effectiveness, Huang and Chen in [9] proposed the pre-processing method for the problem to reduce the computation time and keep the accuracy.

2.3 Clustering for Point Tracking

Point tracking is computationally expensive. We plan to employ the clustering technique to reduce the problem size and keep the accuracy. Clustering takes the touch points of two sequential frames as its input and produces groups of touch points based on the distance between each point

Figure 3: Four cases of touch are caused by matching. (a) A real connection. (b) A fake connection. (c) No connection. (d) No connection.

in two sequential frames. Figure 4 shows an example. After clustering, we only need to deal with the points group by group independently for point tracking. The clustering problem is formulated as follows:

- **Clustering Problem**: Given two sets P and C of touch points and a distance threshold T, partition P and C into sub-groups (P_1, C_1), (P_2, C_2), ..., and (P_k, C_k), $1 \leq k < MAX(m, n)$, such that the number of sub-groups is maximized and the individual sub-grouping tracking is finally equivalent to the final tracking without doing clustering.

Note that P_o, $1 \leq o \leq k$, and C_q, $1 \leq q \leq k$, could be an empty subset, and clustering would not cause the optimal solution to lose because of the equivalent results compared to the tracking without clustering.

3. MULTI-TOUCH ALGORITHM FRAMEWORK

In this section, we propose our multi-touch algorithm framework integrating the clustering concept. Then, we analyze the advantage that the clustering stage brings. Finally, we apply the computational geometry technique to propose an efficient and effective clustering algorithm.

3.1 Multi-Touch Algorithm Flow

Figure 5 shows our multi-touch algorithm framework. If a touch panel has X driving channels and Y sensing channels, a raw data map has $X \times Y$ values. Due to the parasitic capacitors and resistors, the charger noise, display noise, etc., the raw data map need to apply some de-noising algorithms to eliminate those noises. If the noises are not severe, we basically subtract the baseline noise from the received raw data map. The method only spends $O(n)$ time. If the data is noisy, we will give up the data and adjust the driving and sensing settings to get a new raw data map again. Note that the noise issue is considered as a complicated problem and in most of time the hardware solution is the best choice, so we do not discuss it in more details here.

After getting a de-noised data map, some algorithms are applied to obtain touch points and to skim touch palms. We find the touch palm by calculating the touch area of a touch point. If the touch area is large enough, we will consider it as a palm and ignore it. To find a touch point, we seek the peak signal first and then use the method of center-of-gravity to calculate its accurate coordinate. The peak locations are searched sequentially and every node is traversed 1 to 2 times. The time complexity is also $O(n)$ time.

After getting the accurate touch points, the moves of touch points need also to be confirmed. The correct tracking of points can reduce the disconnected lines, which may cause irritation when users are painting or to be confused when

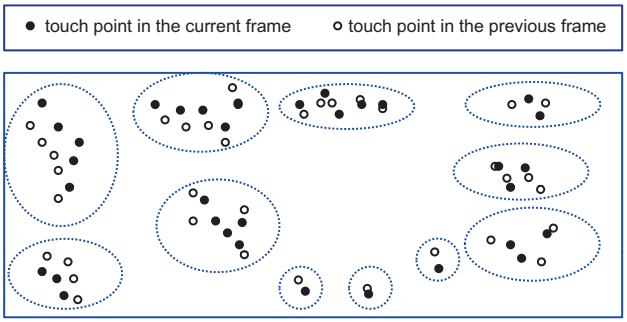

Figure 4: A clustering example.

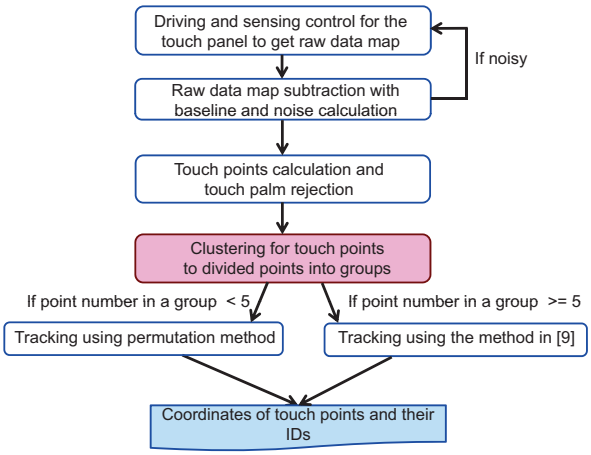

Figure 5: Our proposed algorithm framework.

they try to use some gestures. Therefore, some algorithms are proposed to track the moves and to give same IDs for those touch points as the previous frame to identify where they are from. However, the tracking becomes more and more difficult as the number of touch points increase. Therefore, before applying the tracking algorithm, we do the clustering first. After clustering, the touch points are divided into a few groups. If the point number of a group is smaller than 5, we directly use the permutation method in [13] to find the best matching. Otherwise, we use the method [9]. After tracking, every point would be assigned a proper ID which sufficiently represents its tracking result.

3.2 The Effect of Clustering on Tracking

If we do not have the clustering, the time complexity of the multi-touch algorithm guaranteeing an optimal solution [9] is $O(n^3)$ time, which is very time-consuming when the size becomes larger. After clustering, all the sub-graphs can be solved individually in the tracking stage. In practical cases, the number of touch points in each sub-graph is usually small (<=5) due to the fingers of user and the space. Therefore, the performance of tracking with the clustering stage would be much better than the one without the clustering stage. Even for nowadays most popular 20-point touch panels, the clustering reduces the tracking time which is benefit to developing more advanced gesture identifications.

Algorithm: Clustering(P, C, T, N, S)
Input: P /* the set of points of the previous frame */
C /* the set of points of the current frame */
T /* threshold */
Output: N /* the number of sub-groups */
S /* the set of sub-groups */
1 $S = \emptyset$
2 $N = 0$
3 $Y = \emptyset$ /* the set of point candidates */
4 Put all points of P and C into U, and
sort points of U in the non-decreasing x-coordinate order.
5 Put all points into G''
6 Perform line sweeping in U from left to right to add edges for G''
7 **if** the line meets the point n_1 in P
8 Check whether any points in Y and in C
has the Euclidean distance $\leq T$ with n_1
9 **if** the points exist
10 add edges between n_1 and those points
11 **else if** the line meets the point n_1 in C
12 Check whether any points in Y and in P
has the Euclidean distance $\leq T$ with n_1
13 **if** the points exist
14 add edges between n_1 and those points
15 Put n_1 into Y
16 Delete the points in Y whose x-distances with n_1
are larger than T
17 **do**
18 Pick one un-traversed point as the root r
19 Traverse G'' from r and record the traversed points as a
sub-Group L
20 Put L into S
21 $N = N + 1$
22 **until** all points in G'' are traversed
23 **return** N and S

Figure 6: The geometric clustering algorithm.

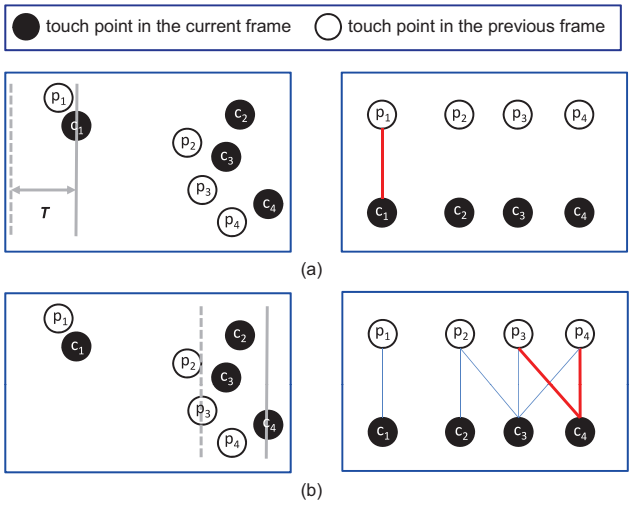

Figure 7: An example of the geometric algorithm. (a) The sweeping line moves to c_1, and an edge is added between p_1 and c_1. (b) When the sweeping line moves to c_4, the list Y only contains $\{p_3, p_4, c_2, c_3\}$ due to the threshold T. Then, two edges are added between c_4 and $\{p_3$ and $p_4\}$.

THEOREM 1. *The time complexity of the tracking algorithm after the clustering stage is O(n) time for practical cases.*

PROOF 1. *In practical cases, the number of points and edges in each sub-graph is constant, O(1). Therefore, it only requires O(1) for any tracking algorithm. Because the number of sub-groups is O(n), the time complexity for the tracking stage is O(n) totally.*

Although the time complexity of the tracking stage is reduced to O(n) for practical cases, the complexity of the total multi-touch algorithm would be dominated by the clustering algorithm. Therefore, we employ computational techniques to propose a more efficient algorithm.

3.3 Geometric Algorithm

Our geometric clustering algorithm first employs *plane sweep paradigm* [8] to construct a graph like G', which has no edges larger than the threshold T, and then applies a traversal algorithm on the graph to parse out all the connected sub-graphs.

The geometric clustering algorithm is summarized in Figure 6. First, (Line 4) the points of P and C are put into a list U and sorted in the non-decreasing x-coordinate order. Then, the line sweeping for U is performed from left to right. During the line sweeping, those points on the left side of the line would be checked whether any points have the Euclidean distances not larger than the threshold T with the point n_1 which is met by the sweeping line. For efficiency, those points are stored in the leaves of the balanced binary search tree Y and sorted in the y-coordinate order. When the line meets one point n_1 in U and P, the points in Y are checked whether any points belong to C and their Euclidean distances are not larger than T (Lines 7-10). If points meet the conditions, edges are added between those points and n_1. Similarly, when the line meets one point n_1 in U and C, the points in Y are checked whether any points belong to P and their Euclidean distances are not larger than T (Lines 11-14). If points meet the conditions, edges are added between those points and n_1, as shown in Figure 7(a). Then, n_1 is put into the search tree Y. Some points are deleted from Y if their x-distances between them and n_1 are larger than T (Lines 15-16). As shown in Figure 7(b), when the sweeping line moves to c_4, the Y only contains $\{p_3, p_4, c_1, c_2\}$, whose x-distances are not larger than T. Finally, the sweeping line moves to the next points in U, and the above steps are repeated until all points in U are swept. Next, a traversal algorithm is applied on the graph to parse out all the connected sub-graphs (Lines 17-22).

THEOREM 2. *The geometric clustering algorithm requires O(nlogn) time for practical cases.*

PROOF 2. *The complexity of the sorting of U is O(nlogn) because the number of points is O(n). The line sweeping for U is performed from left to right, so it meets the points O(n) times. In each meeting, the searching in Y is O(logn) because it is stored in the balanced binary search tree. However, the searching is reduced to O(1) for practical cases since the number of points in Y is always O(1). Due to the practical cases, the number of points whose costs are $\leq T$ is also O(1). Therefore, the total complexity of the line sweeping is O(n), which is derived from O(n) × O(1) × O(1). Finally, the traversal time is O(n) because the edge number $|E'|$ for practical cases is O(n). Therefore, the overall time complexity is O(nlogn), which is derived from O(nlogn) + O(n) + O(n).*

Table 1: Experimental results that compare the efficiency between two different multi-touch algorithm frameworks.

Case	With Clustering						Without Clustering				
	Data Map Processing (ms)	Point Finding (ms)	Clustering (ms)	Tracking (ms)	Total (ms)	Meet Spec.	Data Map Processing (ms)	Point Finding (ms)	Tracking[9] (ms)	Total (ms)	Meet Spec.
G01	1.231	1.815	1.152	1.513	5.711	Yes	1.231	1.815	7.246	10.292	No
G02	1.218	1.617	1.121	1.397	5.353	Yes	1.218	1.617	6.734	9.569	Yes
G03	1.233	1.786	1.137	1.461	5.617	Yes	1.233	1.786	7.632	10.651	No
G04	1.217	1.638	1.157	1.353	5.365	Yes	1.217	1.638	7.643	10.498	No
G05	1.229	1.754	1.132	1.526	5.641	Yes	1.229	1.754	6.275	9.258	Yes
G06	1.212	1.895	1.187	1.463	5.757	Yes	1.212	1.895	7.565	10.672	No
G07	1.238	1.777	1.168	1.583	5.766	Yes	1.238	1.777	7.257	10.272	No
G08	1.211	1.851	1.173	1.315	5.55	Yes	1.211	1.851	6.569	9.631	Yes
G09	1.212	1.832	1.127	1.497	5.668	Yes	1.212	1.832	6.254	9.298	Yes
G10	1.223	1.731	1.189	1.438	5.581	Yes	1.223	1.731	7.847	10.801	No
Avg.	1.222	1.770	1.154	1.455	5.601	Yes	1.222	1.770	7.102	10.0942	No

DEFINITION 1. *The graph G' is the graph that removes those edges whose cost are larger than the threshold T from the complete bipartite graph G.*

DEFINITION 2. *The graph G'' is the graph that is constructed by the geometric clustering algorithm.*

THEOREM 3. *The graph G'' is equivalent to the graph G'.*

PROOF 3. *G' contains edges, whose the Euclidean distances are not larger than the threshold T. The geometric algorithm performs line sweeping to search those points in Y, where the x-distances of those points are not larger than T, and then see whether their Euclidean distances are not larger than T. If yes, the geometric algorithm adds edges for them. While the x-distance is larger than T, the Euclidean distance is also larger than T. That is to say, the line sweeping does not need to consider those edges larger than T. Therefore, the graph G'' is equivalent to the graph G'.*

THEOREM 4. *For all independent sub-graphs in G, the individual sub-grouping tracking is equivalent to the tracking on the graph G'.*

PROOF 4. *An augment path is a path with more matching points and the same total weight in the bipartite graph, as defined in the Hungarian algorithm [10]. Because the independent sub-graph does not connect with another sub-graph, no augment path exists between any two sub-graphs. Therefore the optimal tracking in G' is equivalent to the individual sub-group tracking in G.*

4. EXPERIMENTAL RESULTS

We implemented our multi-touch algorithm in the C/C++ language on a platform, including a 50 MHz ARM-M0 with 64 KB SRAM, a cascaded touch controller and a 23-inch multi-touch panel, as shown in Figure 1. In our first set of experiment, there are totally 10 gesture cases, and each case has 20 touch points simultaneously. The threshold to judge the final connection is set to a given specific value based on the speed of 5 millimeters per 10 ms. The frame rate for the industry specification is larger than 100 Hz [11, 3]. Therefore, to meet the specification, the reasonable runtime of a multi-touch algorithm must be smaller than 10 ms. We compared with another multi-touch algorithm without clustering, whose tracking method employs the state-of-the art method in [9]. Table 1 shows the runtime consumed by each stage, the total runtime, and the average runtime. The total runtimes of our algorithm is much better than the one of the algorithm without clustering. Besides, all cases have no disconnected line and no wrong tracking. This shows our multi-touch algorithm is very suitable for use in multi-touch systems.

To show that clustering is useful for tracking, we conducted another experiment on a 50 MHz ARM-M0 simulator with 64 KB SRAM. There are totally 11 constrained random test cases generated to simulate the real finger-moving cases, and the maximum size of touch points is up to 80. The experiment emphasizes on the time of clustering and tracking and the accuracy. In the experiment, we applied our clustering method to the tracking [9] and compared our method to the one without the clustering method. Besides, to demonstrate our geometric algorithm is very efficient, we designed another method using another obvious algorithm, whose time complexity is $O(n^2)$. The obvious algorithm first constructs a complete bipartite graph based on the distance of each point between two sequential frames. Then, some edges are removed, when their costs are larger than the threshold. Next, the disconnected graph is partitioned to obtain all the connected sub-graphs by a graph traversal algorithm, depth-first search(DFS).

The tracking solutions of these three methods are all the same among these three methods, which is consistent with our proved Theorem 4 and Theorem 3 and shows that our clustering method would not worse the accuracy. As shown in Table 2, the clustering with the obvious algorithm and the clustering with the geometric algorithm can reduce the average runtime by 75.6% and 91.9% over [9], respectively. Besides, the two methods can support up to 50 and 80 touch points, respectively. On the contrary, [9] only supports to about 20 touch points. In addition, since the 20-point case is the popular requirement for current large-area touch panels, the two methods result in time reductions of 66.0% and 70.5% over [9], respectively. As shown in Figure 8, by the least squares fitting of power trendline, the empirical time complexities of the tracking time plus the clustering time of the [9], the obvious, and the geometric methods are $O(n^{2.36})$ and $O(n^{1.46})$, and $O(n^{1.04})$, respectively. It shows that the pre-processing method in [9] helps reduce the time complexity of the Hungarian algorithm to $O(n^{2.36})$, but it is still an extreme complexity order for large-scale cases. The other two time complexities show that the clustering is very help-

Table 2: Experimental results that compare the efficiency among different tracking algorithms.

Case	Points	[9]		Obvious Clustering + [9]				Geometric Clustering + [9]			
		Tracking (ms)	Meet Spec.	Clustering (ms)	Tracking (ms)	Total (ms)	Meet Spec.	Clustering (ms)	Tracking (ms)	Total (ms)	Meet Spec.
T01	10	1.244	Yes	0.297	0.700	0.997	Yes	0.361	0.700	1.061	Yes
T02	20	7.603	Yes	1.028	1.559	2.587	Yes	0.686	1.559	2.245	Yes
T03	30	18.562	No	2.202	2.349	4.551	Yes	1.050	2.349	3.399	Yes
T04	40	36.221	No	3.793	3.096	6.889	Yes	1.372	3.096	4.468	Yes
T05	50	61.937	No	5.840	3.787	9.627	Yes	1.847	3.787	5.634	Yes
T06	55	82.218	No	7.015	4.178	11.193	No	2.067	4.178	6.245	Yes
T07	60	96.952	No	8.297	4.560	12.857	No	2.328	4.560	6.888	Yes
T08	65	112.914	No	9.690	4.916	14.606	No	2.583	4.916	7.499	Yes
T09	70	131.267	No	11.213	5.286	16.499	No	2.942	5.286	8.228	Yes
T10	75	149.717	No	12.822	5.668	18.490	No	3.188	5.668	8.856	Yes
T11	80	169.392	No	14.550	6.002	20.552	No	3.437	6.002	9.439	Yes
Avg.		1				0.144				0.081	

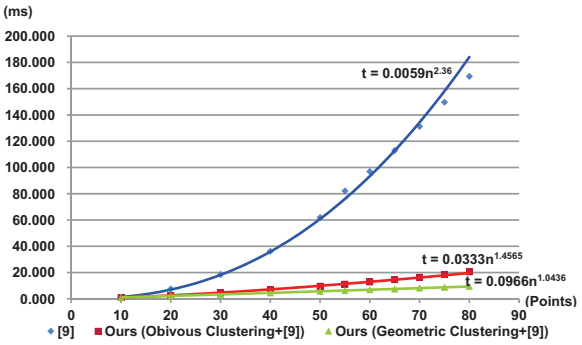

Figure 8: The least squares fitting of power trendline of runtime for the total tracking time of the three methods.

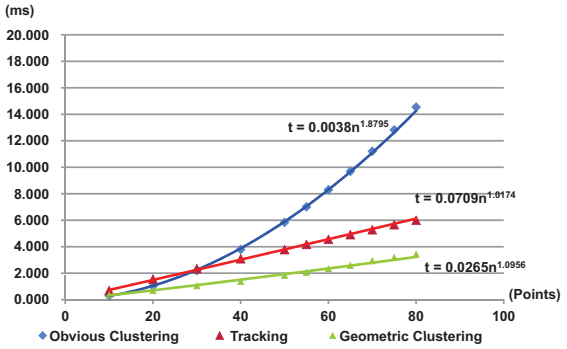

Figure 9: The least squares fitting of power trendline of runtime for the obvious clustering, the geometric clustering, and the tracking stage after the clustering.

ful to the tracking stage. Something that is worth to be mentioned, our geometric method can reduce the time complexity close to $O(n)$, and this means that the time complexity of the multi-touch algorithm flow would be reduced to $O(n)$. The results show that clustering is efficient and effective for tracking.

Next, we compared the runtime among the obvious clustering, the geometric clustering, and the tracking stage after clustering. As shown in Figure 9, by the least squares fitting of power trendline, the empirical time complexities of the obvious clustering stage, the geometric clustering stage, and the tracking stage are $O(n^{1.88})$, $O(n^{1.10})$, and $O(n^{1.02})$, respectively, which correlate with Theorem 2 , and with Theorem 1. As shown in Figure 9, we can see that the total tracking time is dominated by the clustering method using the obvious algorithm beyond 30 points. Therefore, it shows that the clustering may be more time-consuming than the tracking, if the clustering algorithm is not efficient. However, the total tracking time is always dominated by the tracking stage using our geometric algorithm with up to 80 points. These results show that our geometric clustering algorithm is very efficient.

5. CONCLUSIONS

In this paper, we have proposed a suitable multi-touch algorithm framework on an MCU. In the multi-touch algorithm, point tracking is the most computationally expensive part. To accelerate tracking, we have integrated clustering into the algorithm. Besides, we have proven that the tracking problem would be solved in $O(n)$ time for practical cases with clustering. Furthermore, we have applied computational geometry techniques to develop an efficient clustering algorithm. Experimental results have shown that clustering is efficient and effective for large-sized multi-touch panel systems.

6. REFERENCES

[1] 3M Company, http://www.3Mtouch.com.

[2] Ideum, http://ideum.com/products/multitouch.

[3] Window 8 Device Requirement, http://msdn.microsoft.com/en-us/library/windows/hardware/jj134351.aspx.

[4] Z. Baharav and R. Kakarala, "Glass Impact on Capacitive Touchsensing Algorithms: Thinner and Shaped Cover Glass," in *Proc. SID*, pp. 1856–1859, 2011.

[5] Z. Baharav and R. Kakarala, "Capacitive Touch Sensing: Signal and Image Processing Algorithms," in *Proc. SPIE*, vol. 7873, pp. 78730H-1–78730H-12, 2011.

[6] J. Cong and H. Li and C. Wu, "Simultaneous Circuit Partitioning/Clustering with Retiming for Performance Optimization," in *Proc. DAC*, pp. 460–465, 1999.

[7] J. Cong and S. K. Lim, "Edge Separability-Based Circuit Clustering with Application to Multilevel Circuit Partitioning," *TCAD*, vol. 23, no. 3, pp. 346–357, 2004.

[8] K. Hinrichs, J. Nievergelt and P. Schorn, "Plane-Sweep Solves the Closest Pair Problem Elegantly," *Information Processing Letters*, vol. 26, no. 5, pp. 255–261, 1988.

[9] S.-L. Huang and C. C.-P. Chen, "A Significant Multi-Touch Algorithm for the Tracking Problem Based on the Hungarian Algorithm," in *Proc. SID*, pp. 1505–1508, 2013.

[10] J. Munkres, "Algorithms for the Assignment and Transportation Problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[11] A. Ng and P. H. Dietz, "The Need for Speed in Touch Systems," in *Proc. SID*, pp. 547–550, 2013.

[12] P. Pan, A. K. Karandikar, and C. L. Liu, "Optimal Clock Period Clustering for Sequential Circuits with Retiming," *TCAD*, vol. 17, no. 6, pp. 489–498, 1998.

[13] M. Simmon and D. Pickett, "Multi-Touch Tracking," *US Patent No. 2010/0097342 A1*, 2010.

[14] F. Wang, X. Ren and Zhen Liu, "A Robust Blob Recognition and Tracking Method in Vision-Based Multi-touch Technique," in *Proc. ISPA*, pp. 971–974, 2008.

[15] X. Wu, B.-W. Lee, C. Joung and S. Jang, "Touchware: A Software Based Implementation for High Resolution Multi-touch Applications," in *Proc. CIT*, pp. 1703–1710, 2010.