Detection and tracking of multiple objects in a low-resolution grey-scale image

SPECIALIZATION PROJECT – MSC ELECTRONICS, 2 YEARS

Sveinung Pettersen Røsok | TTT-4511 Specialization Project | 25. september 2014

Preface

This report was written as a mandatory part of the two-year Master Program in Electronics at NTNU. As part of the 2 year master educational program, the student has to specialize in a self-chosen topic or from a list of published tasks. The purpose of the Specialization Project, TTT-4511, is to give the student an opportunity to do independent work as a preparatory activity for the master project.

The project title is: “Detection and tracking of multiple objects in a low-resolution grey-scale image”. Due to lack of experience in tracking techniques, limited time and limited scope of the project (7,5scp), this was not completed. The primary focus shifted towards detection and peak estimation, which is the first part of a tracking algorithm.

The project was heavily focused around theory, but also involved practical Matlab algorithm development. The project group consisted of one person (author), with the help and guidance of Professor Lars Lundheim at NTNU and Einar Fredriksen from Atmel.

Many thanks to both my supervisors and the IME faculty for all the support throughout this period.

19.12.2014/NTNU Trondheim

Summary

In this project, a foundation for a tracking algorithm was created. The student looked at detection and peak estimation algorithms for multi touch displays. This is an important part of touch screen technology because it is the interface link between the user and machine. A good detection algorithm will make a touch display more intuitive and easy to use. Today, most multi touch displays use a mutual conductive touch technology. These touchscreens generate a uniform static field that, when affected or touched results in a change in this field that can be measured by *nodes*. For battery purposes, the number of nodes are kept to a minimum, and this results in having to use estimation to get sufficient touch display accuracy.

The algorithm in this project consists of two parts: One algorithm for detecting and separating the number of touch inputs, and one algorithm that estimates a sub-node position of each of these inputs. Through a literature study, five mathematical approaches were found to be fitting for peak estimation. Four of these techniques used a one-dimensional approach: Gaussian, linear, center of mass and parabolic. The last technique used a two-dimensional approach, called least squares.

All algorithms were recreated as peak estimation algorithms, and tested on simulated touch inputs with varying signal to noise ratio. This produced a lower bound, or the optimal performance in a noise free environment. This made it possible to compare algorithms with regards to performance and complexity.

Six batches of samples from a real touch tablet, with different touch input scenarios, was then subject of focus. These batches showed the different shapes that touch inputs may produce, and became crucial in setting boundaries for the detection algorithm. Through analyzing the noise, it also gave insight into the signal to noise ratios, or “working conditions” of the algorithms”, that would be present in a real situation. Finally, all batches were tested with all algorithms, and results were compared visually as the true input was not available.

All methods provided a sub-node accuracy of the peak position. However, tests showed that the Least Squares method would likely be the best choice. It was more complex than the simpler versions, but much more stable in regards to accuracy, especially at lower signal to noise ratios.

All programming was implemented using MATLAB 2014a, and the code is given in an appendix.

List of contents

1. Introduction

1.1 BACKGROUND

We have seen a dramatic increase in touch screen, or touch display, technology in the last 10 years, especially with the introduction of smart phones and tablets[1]. It is now almost expected that all new controls come as touch screens, and for a good reason: It is both intuitive and versatile. However, there are many challenges that lies beneath this technology. One of the most central challenges is finding the accurate location of the touch input. It is critical that the screen recognizes exactly where this is, and doing so with minimum latency, minimum battery usage and with a stable and high performance.

A touch input will now be defined as the physical touching of a display, which usually is either with a finger or a touch stylus. This is illustrated by figure 1.1.

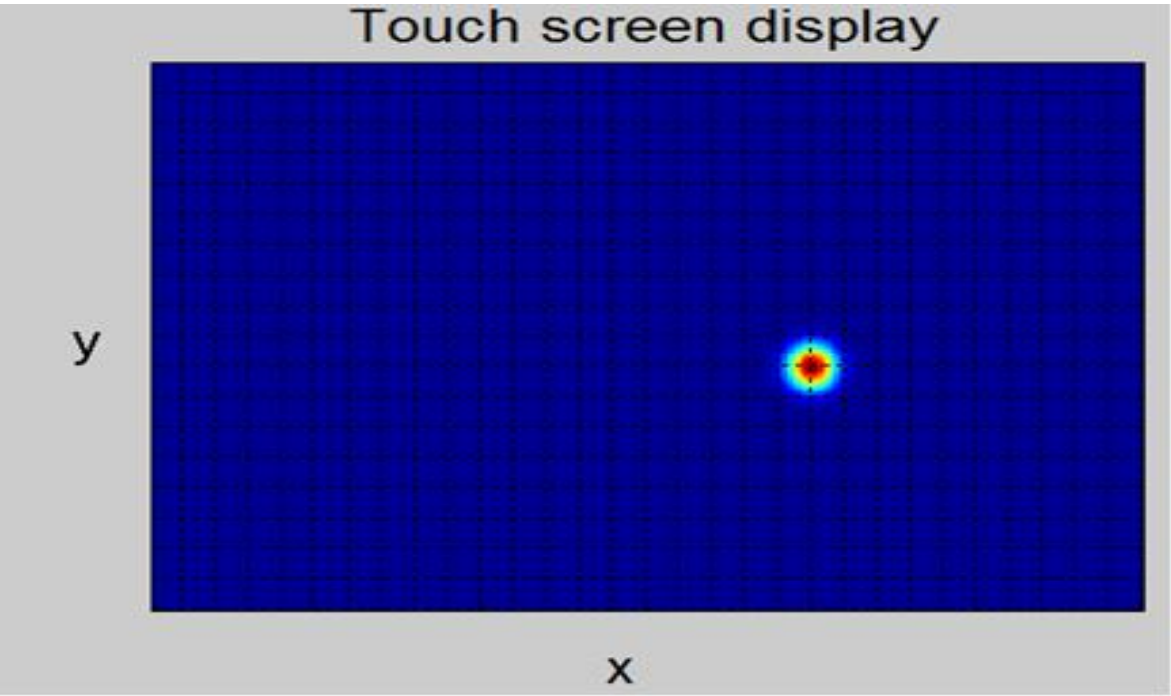


Figure 1.1: Illustration of touch display with a circular touch input

In this project, we will take a closer look at some of the challenges that it has to be overcome to give that strong and fluid interaction. More specific, how the position of the touch input is modeled and estimated, and the main elements that is involved in doing so.

[1] «Touch screen», retrieved 15.10.2014, from http://en.wikipedia.org/wiki/Touchscreen

1.2 PROBLEM DESCRIPTION

The touch display assumed in this project contains uniformly sized, uniformly distributed nodes that continuously samples intensity values, illustrated by the circles on figure 1.2. The number of nodes can vary based on the size of the screen, the space between the nodes and node layout pattern. This layout can be any form, but in this project a side-by-side pattern was assumed.

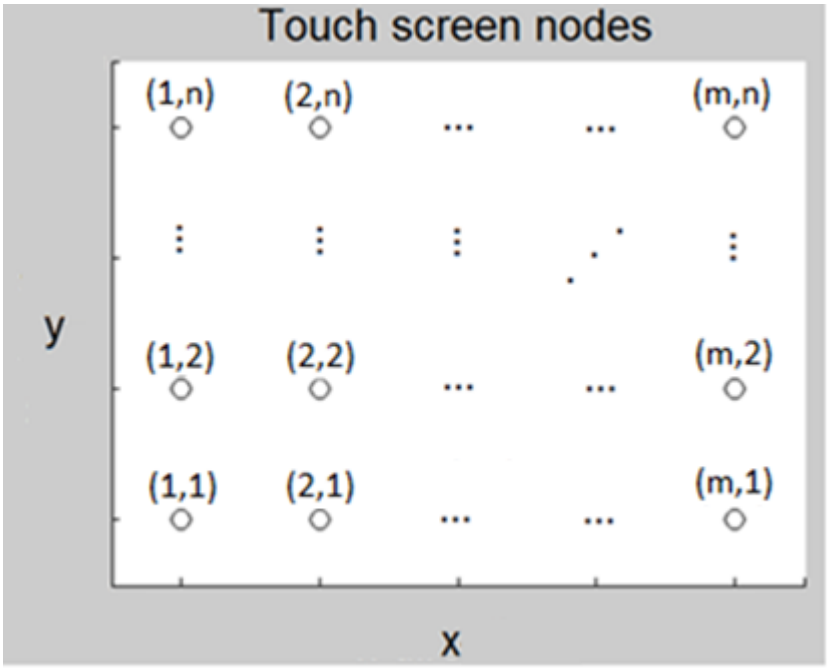


Figure 1.2: Touch screen node layout

When the display is touched, some of the nodes will measure increased intensity values, and the node with the highest value will most likely be the one closest to the center. However, the distance between the nodes is higher than our desired resolution, which means that finding the highest valued node does not provide sufficient accuracy. For a full HD display with 1920x1080 pixels this allows for very small visual images that one might want to interact with, and lack of accuracy will be particularly apparent in these situations. This makes it necessary to model and calculate a more accurate position based on these node values. If this is omitted it will result in touch display inaccuracy.

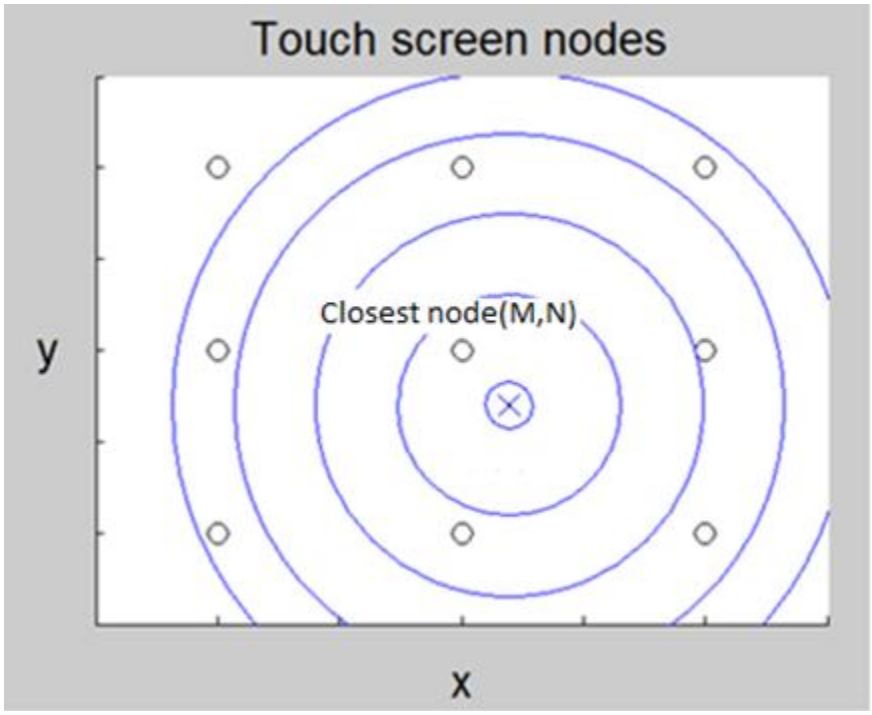


Figure 1.3: One circular touch input, marked by blue circles, centered at blue X

It is this distance we seek to minimize and that leads to the problem formulation:

Develop one or more detection and peak estimation algorithms that can be used on a general greyscale multitouch display.

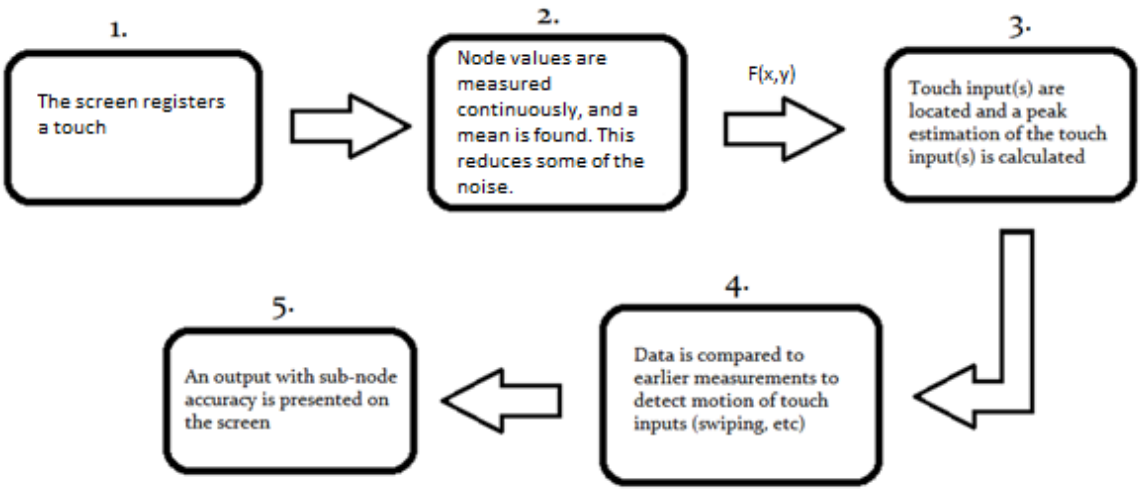
This is a challenge that is two-fold. First the number of inputs must be established. Then the accurate position of each input can be calculated.

The report start with a description of the hardware of a general touch display, and how the touch position data gathered and measured. Next, the touch display and touch inputs is defined to make it possible to concretize our problem and solution exactly. Real life samples and simulated samples are then presented, before comparison is made on accuracy and complexity. Finally, a discussion is had on the strengths and weaknesses of the solutions.

There are many formulas presented, but only the most important ones will be numbered. The author assumes that the reader has basic knowledge of estimation theory and signal processing.

1.3 FRAMEWORK

The basic method of touch displays consists of 5 steps:



This project was based on finding a solution to step 3. Values from step 2 were used, delivered as csv-files from Atmel. The other steps are software and hardware related outside of this project.

3. Touch display physical operation

Several schools of theory were explored in this project, both hardware related and software related. To create a good detection algorithm one needs to be proficient with signal processing, but to get the best possible solution it is also important to see how the data itself is measured and sampled. The touch display is a physical interface where an analog value is converted to a digital value. For the algorithm to be as good as possible, it is essential that one also considers the strength and weaknesses of the hardware as well as the mathematical formulas in the software.

The touch display is a basic screen that can be used or interacted by touching the surface directly with a stylus or finger. The difference touch and multi touch displays is that the latter can be used by multiple fingers, and this difference can be both software and hardware based. Today, several different basic techniques are used to measure the input or touch onto the display: Resistive, infrared, optical, capacitive and surface acoustic wave, etc.[2]. The most common touch technology, which is also applied in smart phones and tablets, is the capacitive technique.

The capacitive screen is a display with several thin layers and a protective film on top. By applying a small voltage to the surface, a uniform static field is created on the screen. When a finger, natural ground, touches the display, the static field changes and this causes capacitive changes which can be measured in many different ways to estimate the position of the touch input. This method is divided in two main categories: Surface capacitance and projected capacitance.

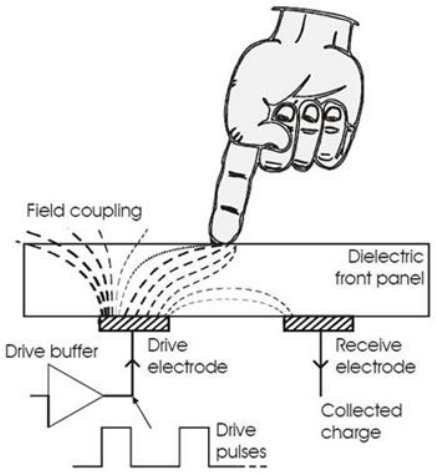


Figure 3.1: Capacitive screen behavior[3]

Surface capacitance is a technique where the screen has a thin internal air gap between the screen and the top layer, bending inwards when pushed. Doing so changes the capacitance in the four corners of the screen, which then can be used to estimate the finger-position. Due to the air gap and

[2] «Touch screen», retrieved 15.10.2014, from http://en.wikipedia.org/wiki/Touchscreen
[3] Image retrieved 1.11.2014, from http://m.eet.com/media/1046718/0909esd_Atmel01sm.gif

moving parts (bending screen) the screen regularly needs maintenance, as well as being inaccurate and with low screen clarity. It also does not support multi touch.

This was improved by developing a technique called projected capacitance, or projected capacitive touch. Common for these screens is that they are made up by a rows of driving lines and columns of sensing lines in a hash pattern, which together forms a matrix. This requires the use of a rare metal called indium tin oxide, ITO, which is nearly invisible to the naked eye yet able to lead a current. The pattern itself can be diamond, square or many other possibilities. The driving lines are then applied with a small voltage, which in turn creates a capacitive effect between the driving lines and sensing lines. A finger will act as a natural ground, and when it touches the grid it will cause a local change of the electrostatic field and these values can be measured. Since there are no moving parts this technology is both optically clear and much more durable. This technology is again divided into two types of technologies: Self-capacitance and mutual capacitance.

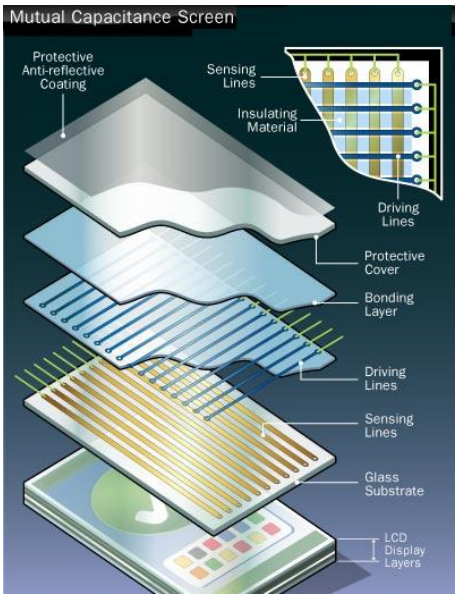


Figure 3.2: Different layers of a mutual capacitance screen[4]

The first is self-capacitance, where the driving and sensing lines operate independently. The change in the electro-static field is stronger compared to mutual capacitance, but this technology is unable to detect more than one finger without error an error called *ghosting*. The other is mutual capacitance, where the capacitance is measured at each junction of rows and columns, creating a matrix grid that then can be used to determine the location. This can be done because the driving lines and sensing lines are arranged in an orthogonal matrix.

[4] Image retrieved 1.11.2014, from http://s.hswstatic.com/gif/iphone-rev-2.jpg

The mutual capacitance technology is the one applied for this project but, while it may offer the best solution available for detection and tracking, it still has a challenge to overcome. The denser the grid of the driving and sensing lines, the more noise it will cause and the more power it needs. In addition the components are passive, and just like many other analog parts they have non-linear properties. This makes accurate estimation harder, and detection and tracking at the edges even more so.

Figure 3.3: Comparison of signal processing capabilities on mutual capacitance touch screens[5]

Figure 1.3 demonstrates the non-linear properties of the screen and the importance of signal processing. Four different smart phones with mutual capacitance touch technology are compared, where a medium-pressure is applied by a finger to draw straight lines. All these phones have relatively similar hardware, but the result is vastly different and this is mainly due to the signal processing. Even though this mainly demonstrates differences in tracking, it is certainly applicable to detection as well.

[5] Image retrieved 1.11, from http://techielobang.com/blog/wp-content/uploads/2010/01/line-drawing-test-2.jpg

4. Signal processing model

This chapter seeks to identify the parameters used for the detection and peak estimation. We will start by defining the model for a single touch approach, followed by defining the multi touch approach.

4.1 SINGLE TOUCH

First, we begin with defining the nodes as being whole numbered, illustrated by figure 4.1. Here, *m* represents the node sequence number in x-direction and *n* represents the sequence number in y-direction. The distance between two nodes in either x-direction or y-direction is defined to be 1. These nodes are placed on a display with of two axes, x and y, where:

$$x, y \in [0, \infty)$$

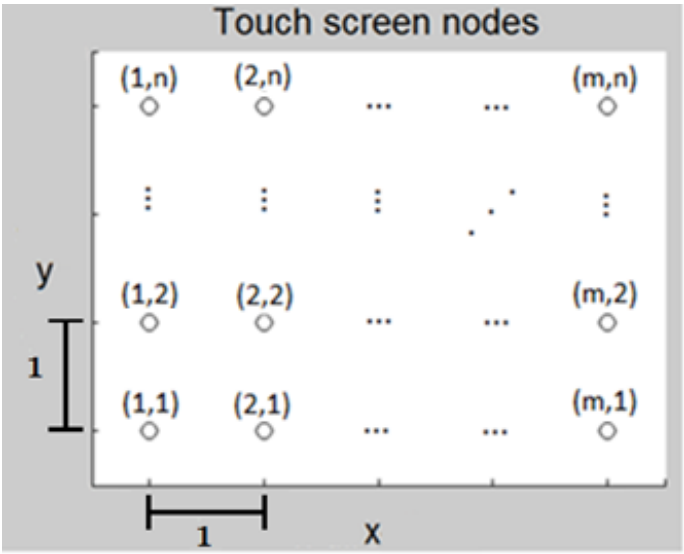


Figure 4.1: Node layout

The length of each axis is arbitrary, since the screen could be any size. Next, we assume the touch input, a finger or touch stylus, to be of any size or shape. It will be centered in origo, and with intensity values fading in all direction, modeled by:

$$f(x, y)$$

The true position, (x_0, y_0), will be the estimated maximum intensity of this function:

$$f(x - x_0, y - y_0)$$

An estimation of (x_0, y_0) will consist of two parts. The first is the position of the node with the highest measured intensity value. This is denoted (M, N), where M and N stands for a specific node and not to be confused with (m,n) which which is the total number of nodes. The second part is the displacement, (δ_x, δ_y), which is relative to the closest node. The estimated position becomes:

$$\hat{x}_0 = M + \hat{\delta}_x$$

$$\hat{y}_0 = N + \hat{\delta}_y$$

Like all estimations this will lead to an inaccuracy, or error distance, illustrated by figure 4.2. It is caused by two factors; algorithm inaccuracy and display noise, but for simplicity it is combined such that:

$$\varepsilon = \sqrt{(x_0 - \hat{x}_0)^2 + (y_0 - \hat{y}_0)^2}$$

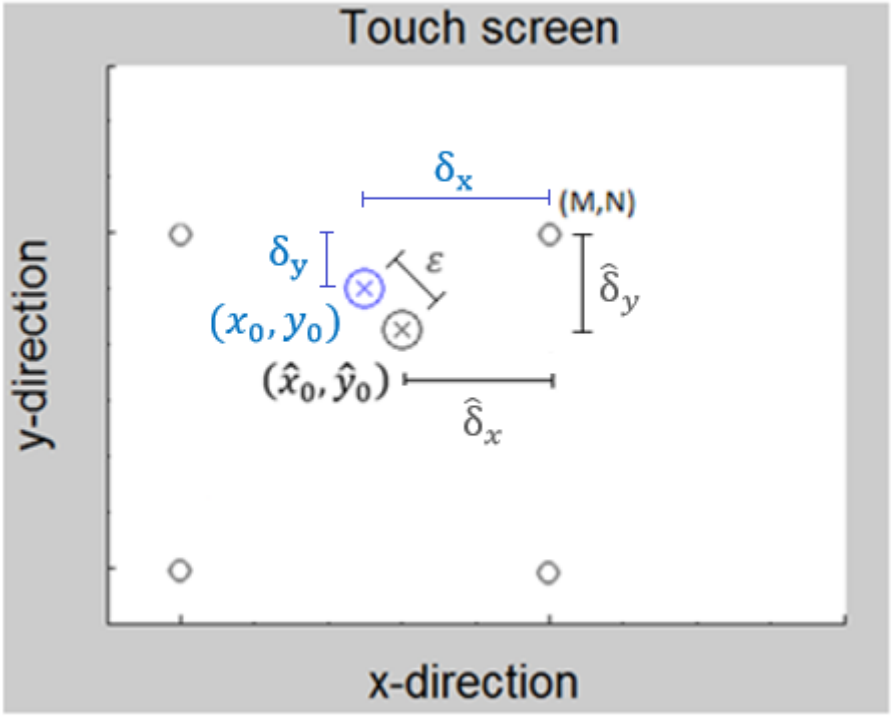


Figure 4.2: True position (blue cross) and estimated position (black cross) of touch input

Noise will be part of our model, and an analysis of a specific touch display is made in chapter 6.2. For this project it is assumed to be additive Gaussian white noise with characteristics:

$$w \sim N(\mu_{noise}, \sigma^2_{noise})$$

This results in the complete model for a single touch input to become:

$$F(x, y) = f(x - x_o, y - y_o) + w(x, y)$$

4.2 MULTI TOUCH

The transfer to multi touch is small, but still significant. For l touch inputs we get the sub-node shift for each:

$$\hat{x}_l = M_l + \delta_{xl}$$

$$\hat{y}_l = N_l + \delta_{yl}$$

This leads to the final model for our touch display inputs, with l number of inputs to be:

$$F(x, y) = \sum_l F_l(x, y) + w(x, y)$$

Where:

$$F_l(x, y) = f(x - x_l, y - y_l)$$

And (x_l, y_l) is the true position of touch input l.

5. Peak position estimation

To estimate the peak position, the use of mathematics is central. In the two upcoming subchapters, two different approaches are presented. The first is an algorithm with a two-dimensional approach, and the second is four algorithms using a one-dimensional approach. After this they will be tested on simulated targets, and compared with regards to complexity.

For reasons presented in chapter 6.1, the size of the touch input is modeled by a 3x3 matrix. This means that a sub-node peak position can be estimated by using between 2 and 9 nodes depending on the algorithm.

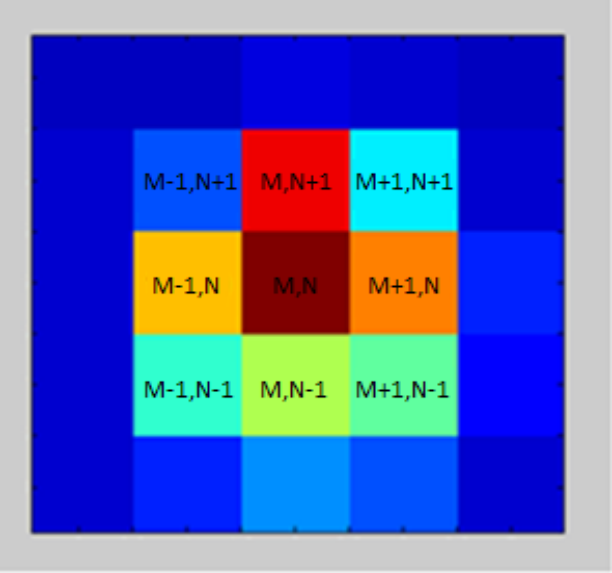


Figure 5.1: Available nodes for estimation

5.1 TWO-DIMENSIONAL ESTIMATION

Least squares (LS) is a very popular method from the classical approach of estimation theory. This method seeks to minimize the energy difference of the data and the assumed signal, and is applicable even for non-linear data[1][6]. The LS error model for a single touch input is defined here as:

$$\varepsilon = \sum_{M=-1}^{1} \sum_{N=-1}^{1} (F(x,y) - \hat{F}(x,y))^2$$

[6] S.M. Kay. *Statistical signal processing Volume 1*. Prentice Hall, 1993. p. 221

This method assumes a 3x3 matrix, which contain the node values of a touch input where the highest valued node is the center node (M,N), as illustrated by figure 5.1. The model creates a parabola and by using all nine values it seeks to create an accurate estimation. To fit this into the two-dimensional matrix, a two dimensional polynomial is used.

First, the approximation is defined:

$$\hat{F}(x,y) = ax^2 + bxy + cx + d + ey^2 + fy$$

Next, the sum of squared differences is defined within the 3x3 matrix:

$$\sum Sq.Diff = \sum_{x=M-1}^{M+1} \sum_{y=N-1}^{N+1} (ax^2 + bxy + cx + d + ey^2 + fy - F(x,y))^2$$

This sum is differentiated with regards to a-f:

$$\frac{\partial \sum Sq.Diff}{\partial a}, \frac{\partial \sum Sq.Diff}{\partial b}, \frac{\partial \sum Sq.Diff}{\partial c}, \frac{\partial \sum Sq.Diff}{\partial d}, \frac{\partial \sum Sq.Diff}{\partial e}, \frac{\partial \sum Sq.Diff}{\partial f}$$

This creates six equations with six unknown variables a-f, which then can be solved. The result is:

a = (M-1,N-1) + (M,N-1) + (M+1,N-1) - 2(M-1,N) - 2(M,N) - 2(M+1,N) + (M-1,N+1) + (M,N+1) + (M+1,N+1)

b = -(M-1,N-1) + (M+1,N-1) + (M-1,N+1) - (M+1,N+1)

c = -(M-1,N-1) + (M+1,N-1) - (M-1,N) + (M+1,N) - (M-1,N+1) + (M+1,N+1)

d = -(M-1,N-1) + 2(M,N-1) - (M+1,N-1) + 2(M-1,N) + 5(M,N) + 2(M+1,N) - (M-1,N+1) + 2(M,N+1) - (M+1,N+1)

e = (M-1,N-1) + (M,N-1) + (M+1,N-1) - 2(M-1,N) - 2(M,N) - 2(M+1,N) + (M-1,N+1) + (M,N+1) + (M+1,N+1)

f = (M-1,N-1) + (M,N-1) + (M+1,N-1) - (M-1,N+1) - (M,N+1) - (M+1,N+1)

Next, the approximation is differentiated twice, with regards to x and y respectively. When solved for zero the result is:

$$\frac{\partial \hat{F}(x,y)}{\partial x} = 2xa + by + c = 0$$

$$\frac{\partial \hat{F}(x,y)}{\partial y} = 2ey + bx + f = 0$$

Again we have two formulas with two unknowns. Solving for x and y respectively results in:

$$x = \frac{bf - 2ce}{4ae - b^2}$$

$$y = \frac{bc - 2af}{4ae - b^2}$$

Where x and y is the $\hat{\delta}$ of the max valued node (M,N):

$$\hat{\delta}_x = x$$

$$\hat{\delta}_y = y$$

5.2 ONE-DIMENSIONAL ESTIMATION

As will be seen in chapter 6, in many scenarios the touch input creates a cross-like signature where the corners of the 3x3 matrix are much less activated. This means that the corner nodes contain less valuable information, illustrated by the blue and light green values. In these cases the data can be simplified to two one-dimensional lines, in other words a single row and a single column, illustrated by figure 5.7. This simplifies the algorithm and requires less calculations than the two-dimensional least squares approach, while still achieving sub-node accuracy.

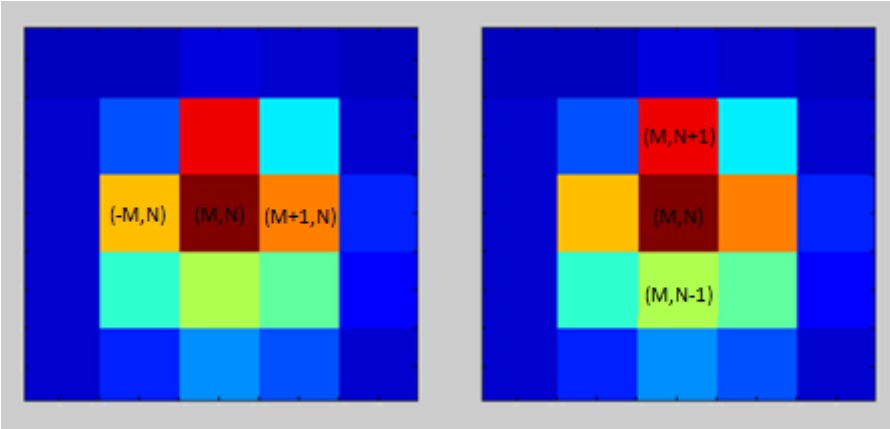


Figure 5.2: Close-up of figure 3.4 with designation of nodes

The three values, both in x-direction and y-direction, can form two separate bar charts that can be seen as samples in a continuous function, illustrated by figure 5.8.

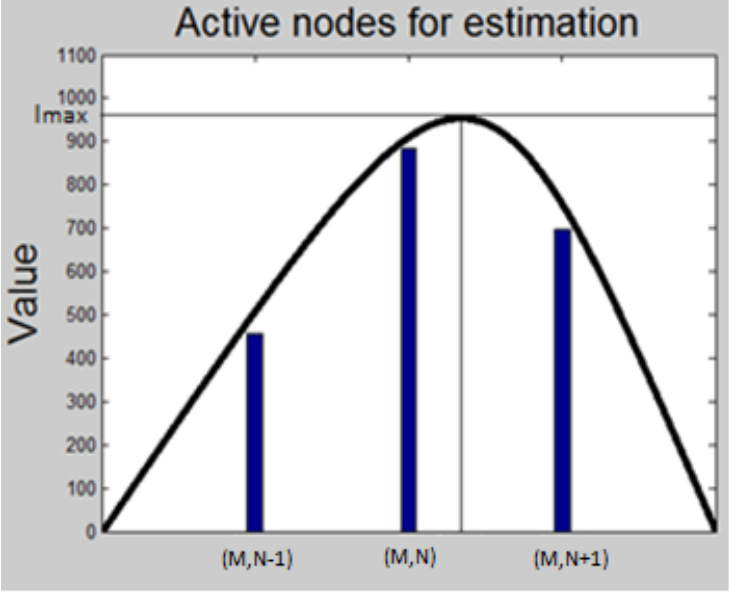


Figure 5.3: One dimensional peak estimation (thin grey line represents estimated max intensity value and displacement)

Estimating δ_x and δ_y is possible by estimating the highest value of that continuous function, and this can be done in many ways. Through the literature study, two articles[2][3][7][8] were found that focused on these techniques, and these highlighted the same four estimation methods due to their effectiveness and ease of application: Linear, Gaussian, parabolic and center of mass. The names indicate the shape of the curves used in each estimation, meaning that the parabolic method assumes the top position can be found by a parabolic shaped curve and so forth.

Linear

$$\hat{\delta}_x = \left(\frac{1}{2}\right)\frac{(M+1,N)-(M-1,N)}{(M,N)-\min\{(M-1,N),(M+1,N)\}}$$

$$\hat{\delta}_y = \left(\frac{1}{2}\right)\frac{(M,N+1)-(M,N-1)}{(M,N)-\min\{(M,N-1),(M,N+1)\}}$$

Gaussian

$$\hat{\delta}_x = \left(\frac{1}{2}\right)\frac{\ln(M-1,N)-\ln(M+1,N)}{\ln(M+1,N)-2\ln(M,N)+\ln(M-1,N)}$$

$$\hat{\delta}_y = \left(\frac{1}{2}\right)\frac{\ln(M,N-1)-\ln f(M,N+1)}{\ln(M,N+1)-2\ln(M,N)+\ln(M,N-1)}$$

Parabolic

$$\hat{\delta}_x = \left(\frac{1}{2}\right)\frac{(M-1,N)-(M+1,N)}{(M+1,N)-2(M,N)+(M-1,N)}$$

$$\hat{\delta}_y = \left(\frac{1}{2}\right)\frac{(M,y-1)-(M,y+1)}{(M,N+1)-2(M,N)+(M,N-1)}$$

Center of mass

$$\hat{\delta}_x = \frac{(M+1,N)-(M-1,N)}{(M+1,N)+(M,N)+(M,N-1)}$$

$$\hat{\delta}_y = \frac{(M,N+1)-(M,N-1)}{(M,N+1)+(M,N)+f(M,N-1)}$$

[7] Fisher, R. B. and Naidu, D. K., (1996), '*A comparison of algorithms for subpixel peak detection*', p. 1-24, retrieved 20.08.2014

[8] Baharav, Z. and Kakarala, R., (2013) '*Capacitive touch sensing: Signal and image processing algorithms', p. 1-12, retrieved 20.08.2014*

5.3 SIMULATED TOUCH INPUTS

The practical testing of the algorithms starts with *simulating* a touch input model. Since we control this environment, we can see if there is a collapsing point of accuracy as well as measuring the maximum accuracy in a noise free ideal environment.

Realistic touch inputs are hard to simulate, but steps can be taken to make it as close to reality as possible. A simple Gaussian distributed circular touch input is created, with an intensity *I*, which leads to the intensity peak and true position on the display:

$$f(x,y) = Ie^{-\frac{(x-x_0)^2+(y-y_0)^2}{\sigma^2}}$$

After the touch input is created, additive white Gaussian noise was added to each node to resemble noise. Following we get a ratio between a signal to noise-ratio, here defined by:

$$SNR = \frac{P_{signal}^2}{\sigma_{noise}^2}$$

Where P_{signal}^2 is the power of the generated touch input, and σ_{noise}^2 is the power of the noise. Here, values were chosen such that it resembled a realistic touch input. The specific intensity value is unimportant, as it is the value distribution and the ratio between the noise and the signal that matters. The real samples are presenter in 6.1, and a discussion regarding noise is explained further in chapter 6.2.

Several simulations were made using Monte Carlo Method, and two cases will now be presented to compare the accuracy of the algorithms. The first is simulation near the center of a node, which results in small values for δ_x and δ_y. An intensity peak was created with $\delta_x = \delta_y = 0.1$, illustrated by a black cross on figure 5.4a and 5.4b. The error distance, ε, was then measured and averaged over 100 simulations, while the signal-to-noise ratio was changed at 2dB intervals between 20-60dB, to see how this affected the accuracy of each algorithm.

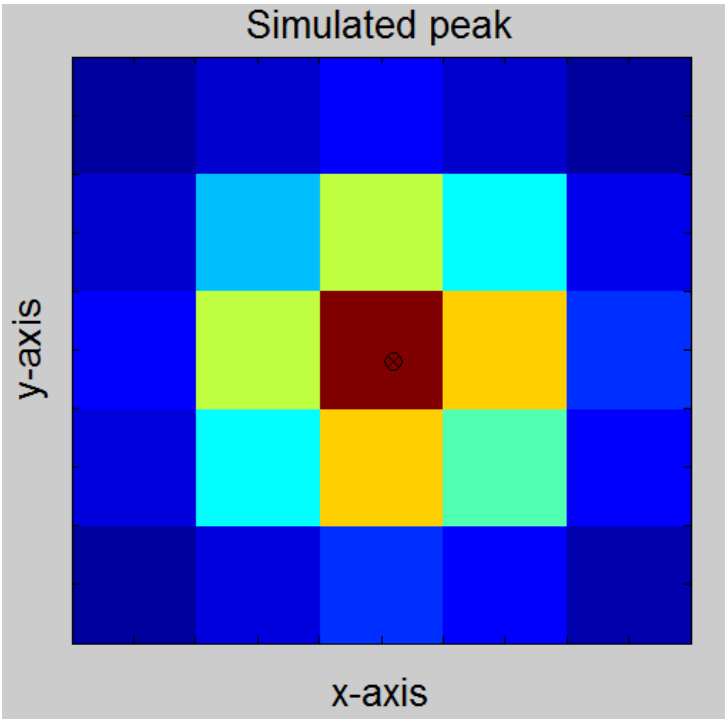


Figure 5.4a: Simulated touch input near center of node

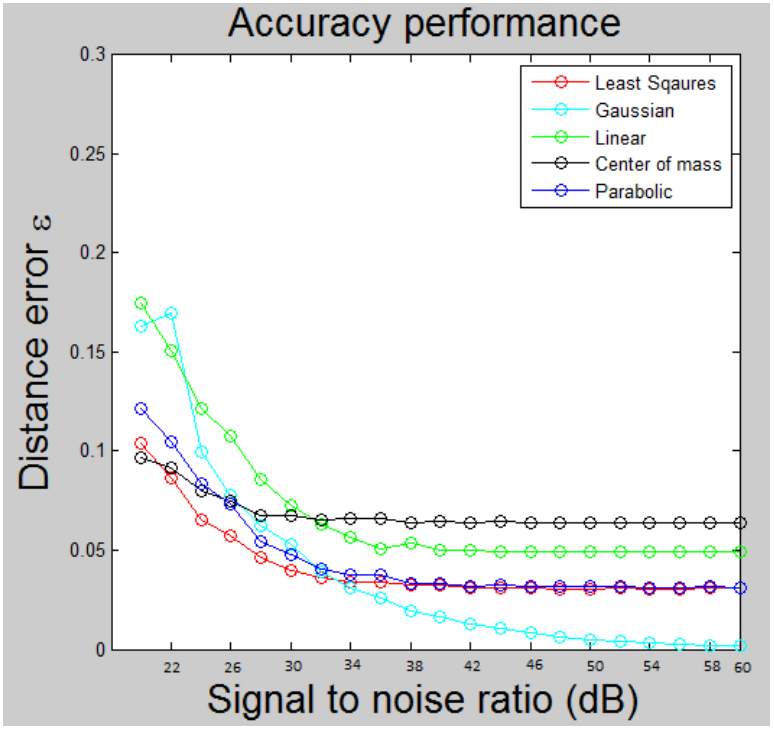


Figure 5.4b: Error distance related to signal-to-noise ratio

We start by observing that the center of mass approach is the best performing algorithm at the lowest SNR, but as the ratio is increased it quickly becomes underperforming when compared to the rest. We also observe that the Least Squares approach is outperforming the others at 22-32dB, but is surpassed by the Gaussian at this point. This is probably because it uses nine nodes in its estimation, which consequently mitigates some of the noise. With one exception, all algorithms flatten at around 32dB, and the performance is now virtually unaffected by noise. The only exception is the Gaussian approach that approaches zero as 60dB.

The second simulated case is on the opposite side of the scale, where the simulated intensity peak is close to the border between many nodes. This consequently makes δ_x and δ_y large, and simulations were made with an intensity peak centered at $\delta_x = \delta_y = 0.4$. The result is shown on figure 5.5a and 5.5b.

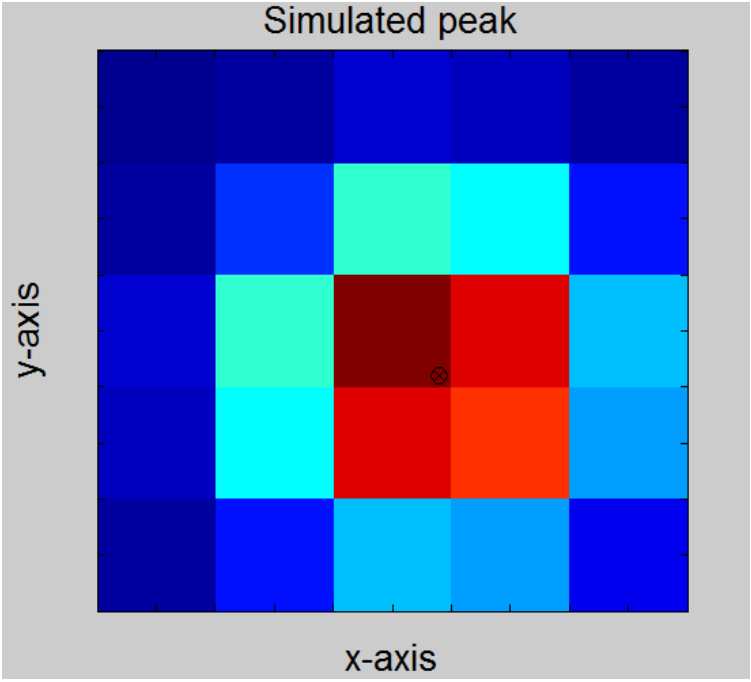
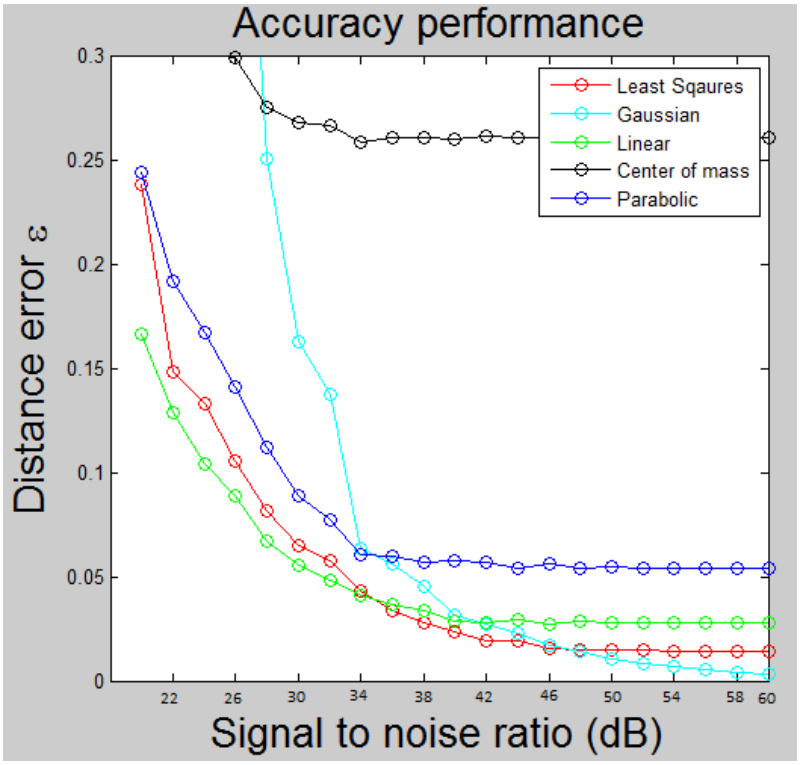


Figure 5.5a: Touch input near edge of node



Figur 5.7: Error distance related to signal-to-noise ratio

This time, accuracy is slightly worse, and it flattens at around 38dB for most algorithms. We also observe the linear approach being the best up to around 34dB, before the least squares approach performs better. At this point, the Gaussian approach also gains a massive increase in accuracy. The last observation is the poor performance of the center-of-mass algorithm, which stabilizes at approximately $\varepsilon = 0.26$, which is an odd result considering the good performance of the previous simulation. Overall the simulations show that the algorithms work as intended, though which one that is better heavily depend on the signal-to-noise ratio.

A theoretical resolution increase, *R*, by the peak estimation can now be calculated. Assuming a conservative accuracy of $\varepsilon = 0.04$, we get:

$$R = \frac{Distance\ between\ nodes}{\varepsilon} = \frac{1}{0{,}04} = 25$$

Since this resolution applies for both x-direction and y-direction we get:

$$R^2 = 25^2 = 625$$

In other words, the results seem to indicate that one could only detect the highest valued node and not use peak estimation, one would need 625 times more nodes to get the same accuracy. However, this is a theoretical approach and it assumes a noise free environment and a perfect circular signal. The improvement on a real signal will likely be a lot smaller, but now a lower bound on performance has been established. In chapter 6 we will try to analyze the practical performance on real data.

5.4 COMPLEXITY OF ALGORITHMS

No information was given on hardware of Hawkeye, so a study was necessary to determine the complexity of each algorithm. Since most touch displays have vastly different processing power, this becomes an important aspect. Algorithm can be described in number of calculations regardless of the hardware. As noted earlier, these algorithms are divided in two groups, one-dimensional and two-dimensional and this affects the complexity.

Results of complexity are here given in two different tables; One for single touch input, and one for two touch inputs. The tables show a comparison between the numbers of calculations in each algorithm. For simplicity, addition and subtraction has been linked together. The same applies to divison and multiplication. Assuming a 50 Hz refresh rate, this means that the table must be multiplied by 50 to get the total amount of calculations required per second.

Table 5.2: # of calculations for one touch input

Algorithm	# Addition & Subtractions	#Multiplications	# Divisions
Linear	8	2	2
Gaussian	10	12	2
Parabolic	10	2	2
Center of mass	10	0	2
Least Squares	35	23	2

Table 5.3: # of calculations for two touch inputs

Algorithm	# Addition & Subtractions	#Multiplications	# Divisions
Linear	16	4	4
Gaussian	20	24	4
Parabolic	20	4	4
Center of mass	20	0	4
Least Squares	70	46	4

Here the same results, but this time two fingers are detected. This is the first table multiplied by two, to illustrate the increasing demand of processing. The number of multiplications and divisions are here separated as divisions, in general, require more calculation power than multiplications.

One algorithm that separates itself from the rest is again the Least Squares method, but this time negatively. This is a bit more complex than the others with 3-4 times as many additions and subtractions, and approximately 2-10 times as many multiplications. With a 50 Hz refresh rate of the touch display and for a single touch input, this amounts to 1250 extra additions and subtractions per second, and about 1000 extra multiplications per second. Depending on the hardware of the touch display this results in reduced battery life and in the worst case, latency on peak estimation output. How much this affects the individual system totally is dependent on the processing power of the central processing unit, but for all practical purposes this could be negligible. An iPhone 4S is capable of 271 million Integer math operations per second[9], but this processing unit also does many other calculations as well.

With this in mind, the algorithms' processing requirements, and consequently battery life implications, will not be taken into account when evaluating the algorithms. Overall, the Least Squares method provided the best accuracy, especially for low signal-to-noise ratio. Even though it is more complex, it is more than compensated for with the added accuracy. Using nine nodes in the estimation proved seems to reduce the negative effect of the noise.

[9] «iPhone 5 review», retrieved 06.11.2014, from http://www.macintouch.com/reviews/iphone5/

6. Multi touch tablet

In this chapter we will look at some real measured node values from a touch display, and the justification for estimating with a 3x3 matrix will be presented. We also analyze the noise present, and run the detection and peak estimation algorithm on a specific scenario.

6.1 REAL MEASUREMENTS

The author was given several different batches a multi touch tablet. The files contained six scenarios of touch inputs, presented on a touch display with 18x30 nodes, as illustrated by figure 6.1. These are named no touch, single touch, large touch, pinch, double touch and close double touch. All of them are real scenarios that must be expected to occur with usage. Note that row 12-18 are uniformly blue or green for all batches, due to an error in sampling. It was excluded in the project and did not affect the rest of the project.

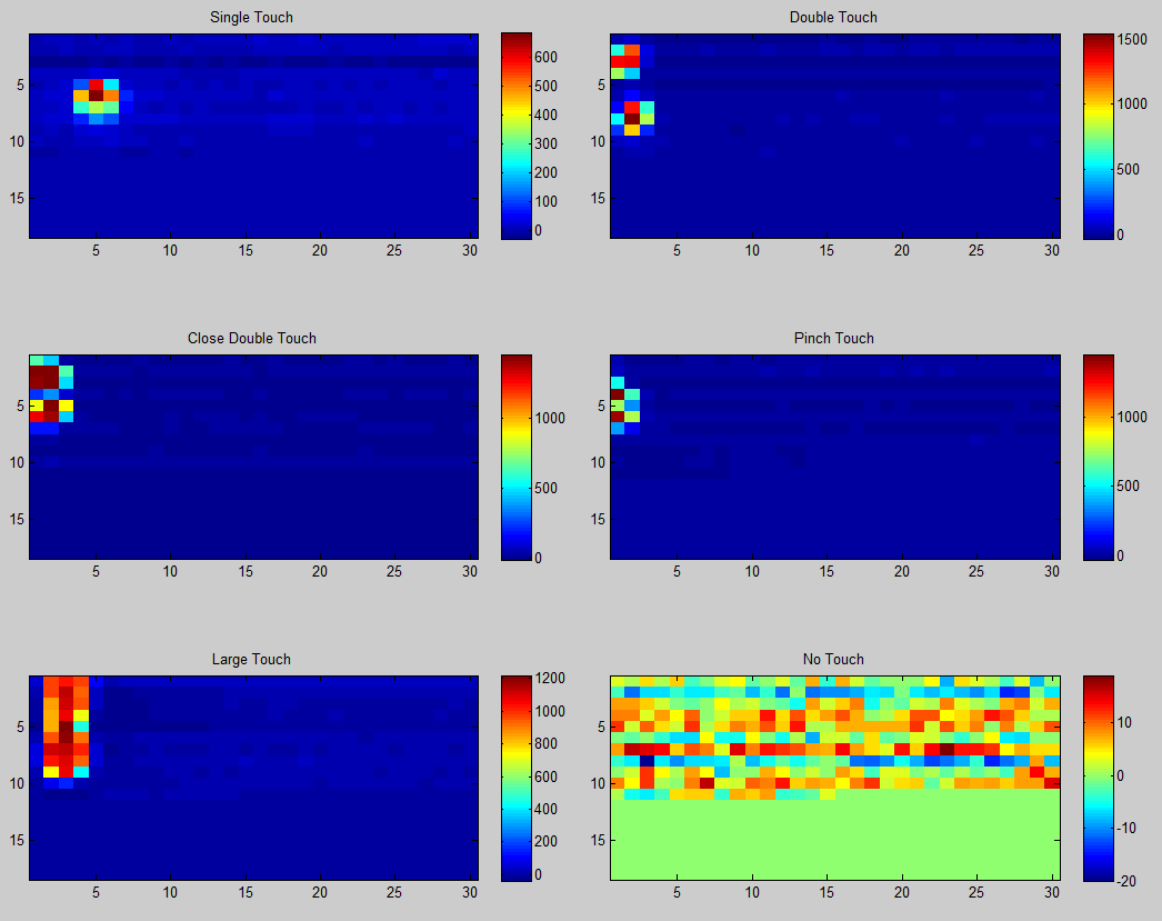


Figure 6.1: Batches of different scenarios, provided by Atmel

When looking at the size and shape of the touch inputs, illustrated by figure 6.1, there are many similar features. The first is that many of the inputs cover an area of 3x3 nodes, and the intensity is often faded to noise outside of this. Another observation is that the input often has a cross-like shape. In addition, the *Close double touch* scenario shows that two close fingers will unlikely get closer than two nodes.

From these observations it was concluded that for estimating the accurate position, a 3x3 matrix would suffice and would actually the biggest usable too. Nodes outside of this provide little to no information to help estimate the peak and could, in the worst case, result in degraded performance if two touch inputs were mistaken to be one touch input. This also makes sense in node design as well because the more nodes on a touch display, the more power is required, resulting in a reduced battery life. This decision means that the algorithm will not be able to decipher scenarios like *large touch* properly, since it will register this as two individual touch inputs. A solution to this was not found during this project.

There was also a recording with 500 sequential batches provided by Atmel, with a sampling rate of 3-4 Hz, illustrated by figure 6.2 and figure 6.3. This was given late in the project and not subject to a lot of focus.

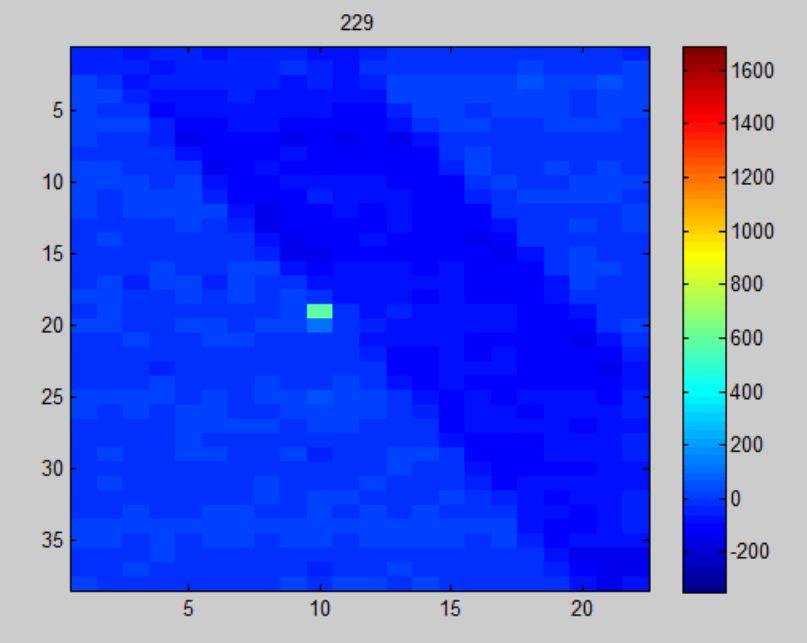


Figure 6.2: Snapshot of a touch input drawn along a ruler

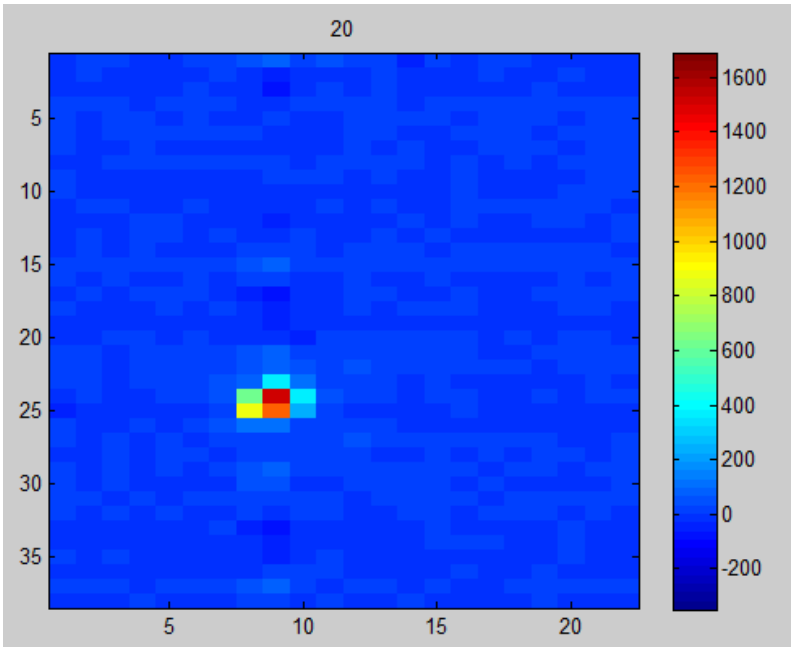


Figure 6.3: Snapshot of a touch input in motion

The observation of the sequence still proved useful. Looking at figure 6.3, it demonstrates the small values of a touch stylus. This is useful for setting a threshold such that the touch display can decide if the signal is noise or a deliberate interaction from a user.

6.2 NOISE ANALYSIS

Noise continues to be one of the biggest challenge of smooth interaction with a capacitive touch screen. The significance of this is easily verified if trying to google *touch display noise*. The most common sources of noise is the screen and are from chargers and inadequate shielding of components or user. Through technical design and effective signal processing many solutions have been made to reduce or mitigate noise on capacitive touch screens: Shielding, avoiding the noise frequencies, digital filters, touch screen sensor design, synchronization and more.

The way it affects the display is by disturbing the static field lines so they are is no longer uniform. To an extreme this can render the touch display complete useless, but this is in the extreme cases. The effect is illustrated below by figure 6.4. Noise on the Hawkeye touch display is illustrated by the non-uniform color of the node intensity values, for example on figure 6.3.

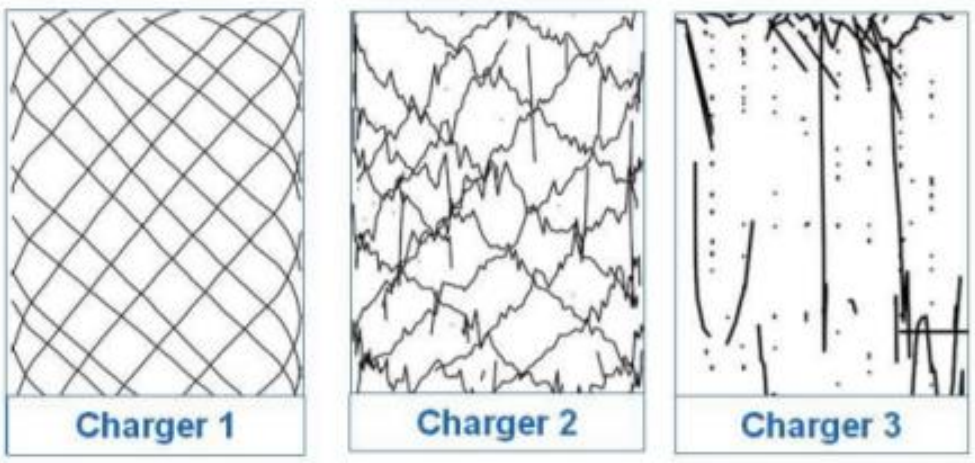


Figure 6.4: Criss-cross patterns for different chargers[10]

[10] Image retrieved 1.1.2014, from
http://larrylisky.files.wordpress.com/2014/03/chargenosiescreentest.png

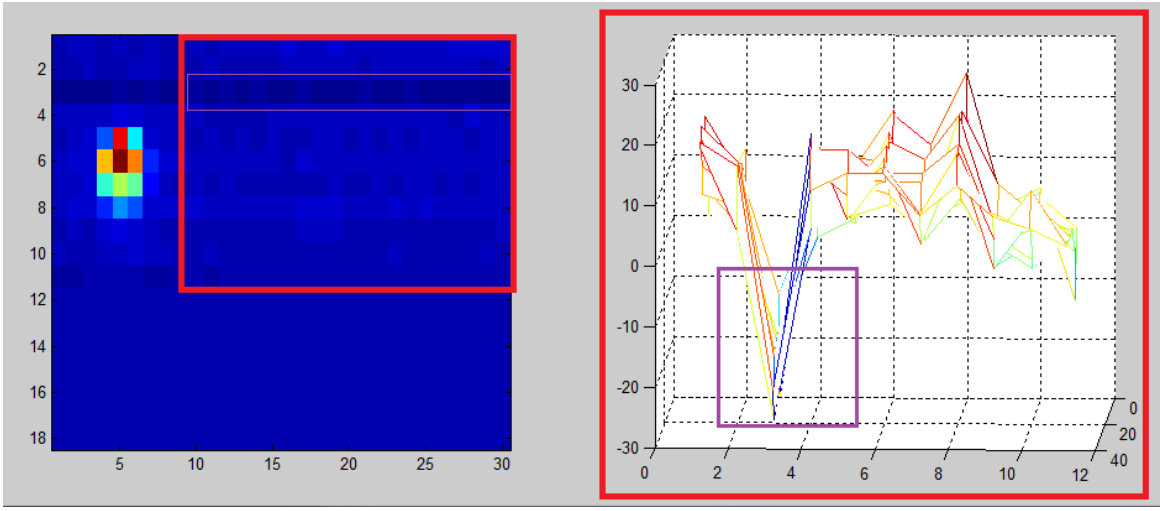


Figure 6.5: Noise on the touch display, illustrated by uneven intensity spikes

The magnitude and characteristics of the noise was found by copying the intensity value of all nodes outside of the touch inputs, and it was assumed to be Gaussian distributed for simplicity. Analyzing the noise from all batches resulted in this characterization:

$$w \sim N(\mu_{noise}, \sigma^2_{noise})$$

$$w \sim N(2.3, 72)$$

The lowest registered intensity value for a touch input, in the batches provided by Atmel was 684. This was also assumed when the signal-to-noise ratio was calculated, to design with a conservative approach. This leads to:

$$P_{signal} \approx 684$$

$$\sigma^2_{noise} \approx 72$$

$$SNR = \frac{P^2_{signal}}{\sigma^2_{noise}} = \frac{684^2}{72} \approx 6475 \approx 38{,}2dB$$

This is a rough estimate with a conservative design. From chapter 5.3, the performance of the algorithm in a simulated environment was flattening at around 32dB. At 38.2dB, the algorithms will perform virtually unaffected by noise, with a margin of around 6.2dB which is an encouraging result.

6.3 DETECTION AND PEAK ESTIMATION OF MULTI TOUCH INPUT

In this chapter the detection and different peak estimation algorithms will be demonstrated on a real set of data. The most important result will be presented using the *Double touch* scenario. This demonstrates the ability of the detection algorithm to separate between two inputs, as well as the estimation of the peak, using the 5 different techniques described earlier. The true positions of the touch inputs are not given, but the results are still useful. The white and blue cross and circle marks the estimated intensity peaks.

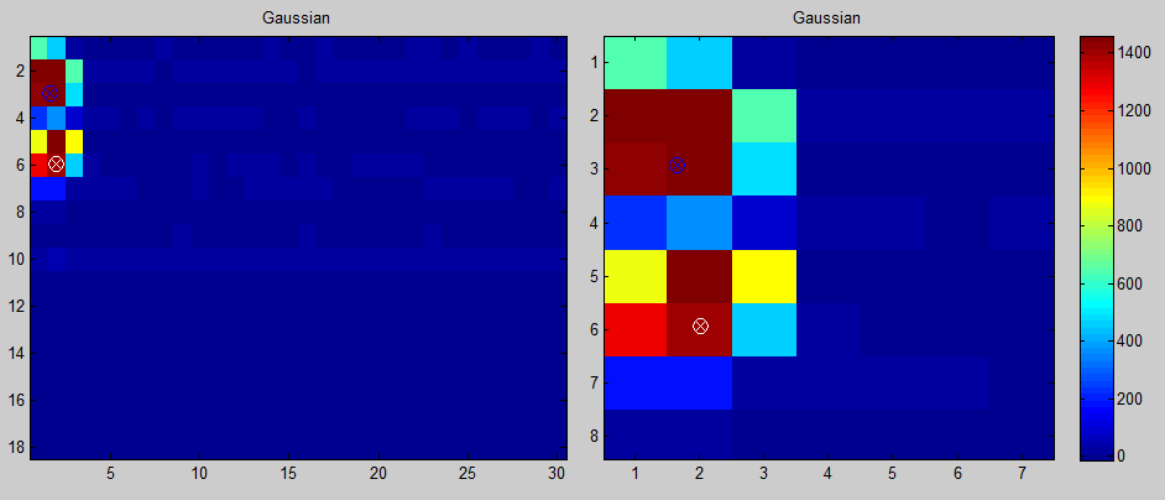


Figure 6.6: One-dimensional peak estimation using Gaussian estimation

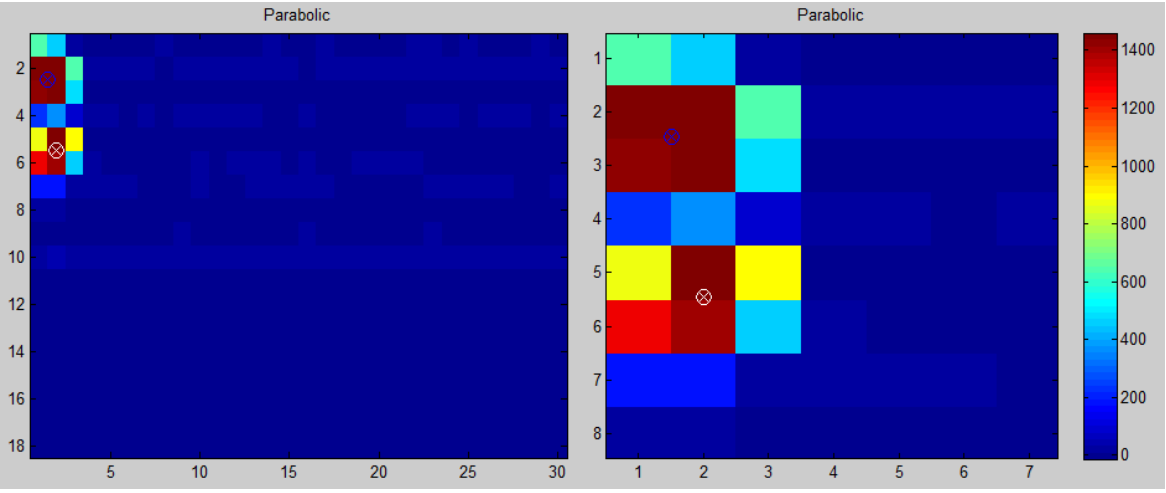


Figure 6.7: One-dimensional peak estimation using Parabolic estimation

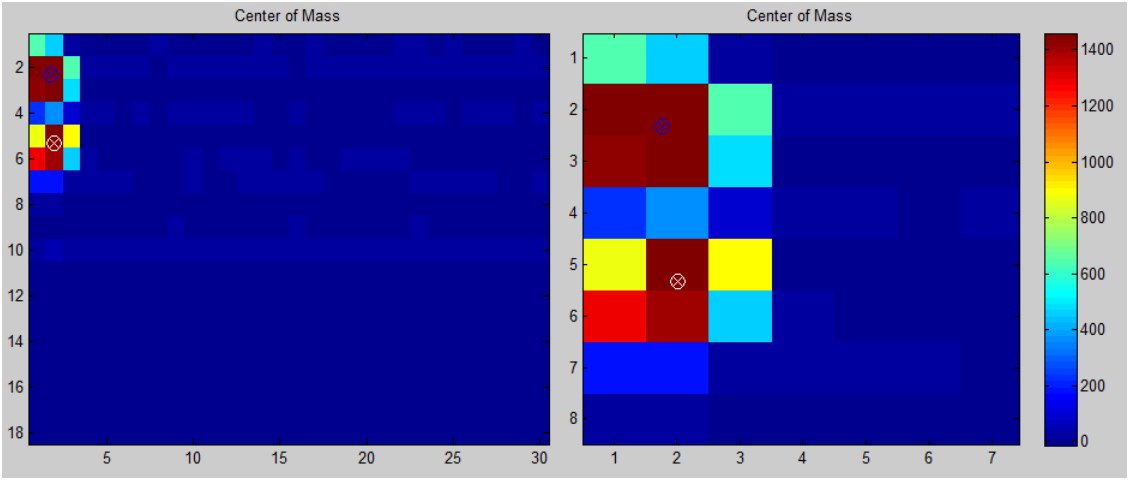


Figure 6.8: One-dimensional peak estimation using Center-of-Mass estimation

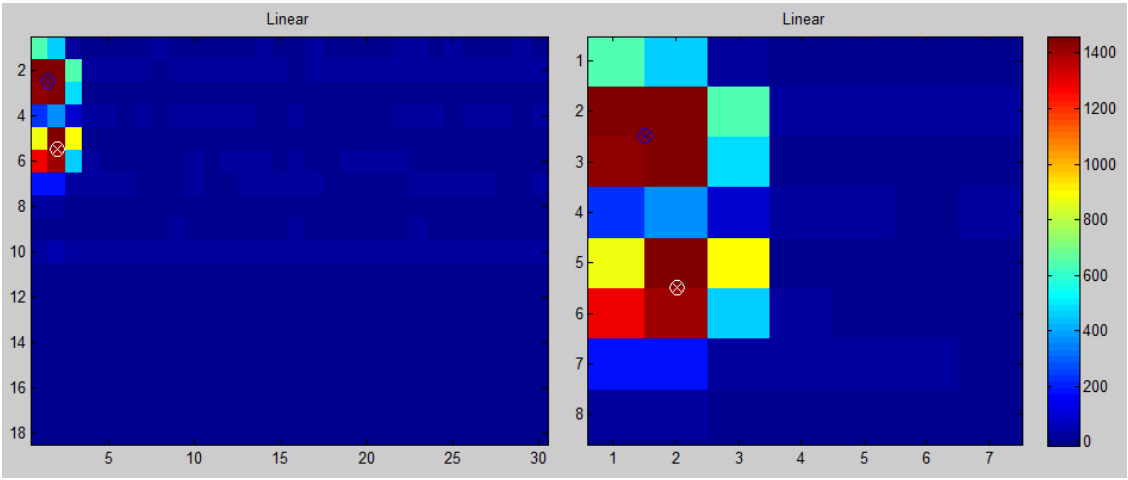


Figure 6.9: One-dimensional peak estimation using Linear estimation

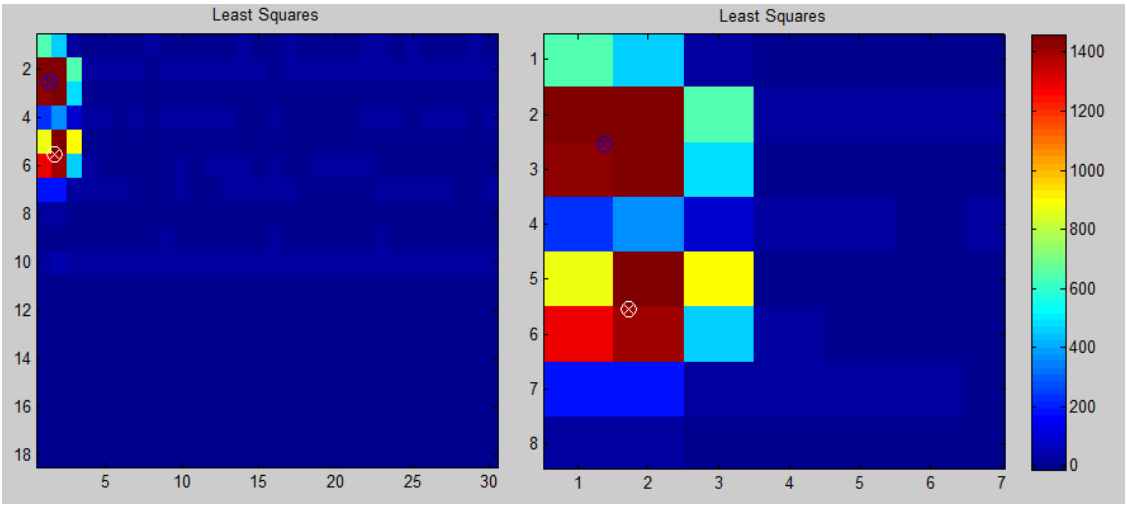


Figure 6.10: Two-dimensional peak estimation using Least Squares estimation

At first glance all the detection algorithms work fairly well. Looking at the upper touch input of the *Close double touch* plots one can see that all algorithms are virtually at the same spot. Though the nodes themselves are big at 5x5mm the icons used on a touch display are rarely smaller than this. When typing text messages with the IPhone 4S, the letters are of size 4x6mm or 5x6mm depending on the orientation of the screen. For larger phones, the letters are even bigger. This makes the margin of error relatively large, and it is likely to assume that all five algorithms developed in this project would suffice these kind of simple tasks. In other words, they should not cause any perceptual errors for the user. Human factors could also contribute as a positive buffer, as a person will naturally adapt to the shortcomings of a screen. The last part was not assumed in this project.

A bigger difference in accuracy is shown when looking at the lower touch input. Here we can see that none of the four one-dimensional algorithms are displaced in the x-direction. This happens because the simple algorithms do not use the corner nodes in the peak estimation. When high values are measured in any of the corner nodes of (M,N) the displacement caused by these will be ignored totally. This is also a situation that one must assume happens frequently, and the one-dimensional approach will therefore be insufficient.

Overall, when combining the two simulated peaks, the least squares method is the best method. This is in particular for touch inputs that are more complex than the 3x3 square. It is reason to believe that real measurements will result in more complex shapes than squares of size 3x3, with evenly distributed data that is suitable for one-dimensional estimation.

Another advantage of the least squares method is that it is less susceptible to noise since it is using nine points to calculate a peak position. An estimation using three nodes would be much more susceptible for this because it is provides the position based on fewer nodes.

7. Conclusion

In this specialization project one detection algorithm and five different peak detection algorithms were created for a multi touch display. This is a small but essential part of a multitouch tracking algorithm. The central challenge was creating algorithms that used a low-resolution input to create a high-resolution output. In addition, doing this in a noisy environment that uses non-linear components. This was done in two steps: Determining the number of touch inputs present, and then performing a peak estimation for each of these inputs.

The detection algorithm was based on studying real data sampled from a touch tablet. This showed that an average touch input covers approximately 3x3 nodes, which suggests that it is reasonable to isolate each input as a 3x3 matrix, with the highest valued node at the center, and then perform the peak estimation on the isolated cases. In addition, the real data provided insight into which signal to noise ratios that can be expected in practical usage.

Peak estimation is based on mathematical formulas, and suitable methods were found through a literature study. Five algorithms were developed based on these. Four of these were one-dimensional estimation methods that used the closest nodes values adjacent in x-direction and y-direction to the node value with the highest intensity (M,N). In addition, one estimation method used a two-dimensional approach that used all eight nodes surrounding (M,N).

The first comparison of algorithms was in regards to complexity. Due to the powerful hardware of todays touch displays, this was not given much attention. However, the two-dimensional approach naturally required more computational power to produce the peak estimation.

All algorithms were then tested quantitatively by using simulated targets. These were Gaussian distributed circular touch inputs, where noise was added and adjusted between 20dB and 60dB. The use of Monte Carlo Method produced the lower bound for zero noise environments for all algorithms, and showed how accuracy varied as a function of signal to noise ratios. For simulated targets, all five algorithms delivered a high performance, which all managed to improve accuracy by a high degree.

With real sampled values, it is important to consider that the touch input might have more complex and non-square shapes, which makes one-dimensional estimation insufficient. This was hard to measure quantitatively as the true position was not provided. However, visual inspection showed that high values in corner nodes was ignored for the one-dimensional approaches, which resulted in a visual inaccuracy.

For simple tasks, like pushing normal sized buttons on a touch display, all algorithms would likely perform decently. In a tracking application, indication have shown that the Least Squares approach is superior. However, the one-dimensional approach would also surely work. The question is at which level, but this was not investigated in this project.

Overall, the least squares method was more complex than the others, as it required a larger amount of calculations for each peak estimation. However, this is a small part of many calculations that has to be made, and with the modern processor it is unlikely that this would reduce the performance by causing delays in the output.

8. Further Work

For further work there are a few options that can be chosen, both theoretically and practically.

Theoretically the first natural approach would be developing new algorithms with even better accuracy or better versatility.

Practically, since no screen was available for practical testing, it would be interesting to see just how well the algorithms would perform in a live setting. At some level of application complexity, such as the size of the different buttons, the algorithms would go from usable to unusable. Even though these results can be deduced theoretically, being able to test this on a human perceptual level would be interesting. One could also increase the functionality of the detection algorithm such that it could handle the *large touch input*.

The author will create a tracking algorithm as a continuation of this project. This will require that some design choices are being made early on:

It is likely that the estimated position will fluctuate at the speed of the screen refresh rate, creating an undesirable shaking of the estimated position. How can this be avoided?

Tracking will offer an opportunity to use earlier data to better estimate the true position, such as Kalman filters. How thorough should this be explored?

Is batch processing a reasonable approach? How does it fare compared to Kalman filtering?

How many fingers should the algorithm be able to understand?

Is it important to rank inputs sequentially, or just the relative distance and motion between them?

9. Literature list

[1] S.M. Kay. *Statistical signal processing Volume 1*. Prentice Hall, 1993. p. 221

[2] Fisher, R. B. and Naidu, D. K., (1996), '*A comparison of algorithms for subpixel peak detection*', p. 1-24, retrieved 20.08.2014

[3] Baharav, Z. and Kakarala, R., (2013) '*Capacitive touch sensing: Signal and image processing algorithms', p. 1-12, retrieved 20.08.2014*

10. Appendix

A. MATLAB CODE

```
%% All algorithms with Monte Carlo simulations
% Least squares, Gaussian, linear, parabolic, center of mass

clear all;

% SNR dB scale ranges from +20 to + 60
snr_db_scale = 20:2:60;
snr_scalar_scale = 10.^(snr_db_scale/10);

% Imax is calculated highest intensity of node if pushed directly
I_max = 900;

for k=1:21
    noise_values(k) = ( I_max / sqrt(10.^(snr_db_scale(k)/10)));
end


for s = 1:21
    noise = noise_values(s);

for q = 1:200
%Syntethic target input
Amp = 30;
sigma = 1;

mu_x = 3.1;
mu_y = 3.1;

x_center = 17-1+mu_x;
```

```
y_center = 8-1+mu_y;


% Create synthetic target
M = zeros(18,30);

% x-direction
M(8,17:21)  = Amp*exp(-((1:5) - mu_x).^2/(2*sigma^2));
M(9,17:21)  = Amp*exp(-((1:5) - mu_x).^2/(2*sigma^2));
M(10,17:21) = Amp*exp(-((1:5) - mu_x).^2/(2*sigma^2));
M(11,17:21) = Amp*exp(-((1:5) - mu_x).^2/(2*sigma^2));
M(12,17:21) = Amp*exp(-((1:5) - mu_x).^2/(2*sigma^2));

% y-direction
for i = 1:5
    M((7+i),17) = M((7+i),17) .* Amp*exp(-(i - mu_y).^2/(2*sigma^2));
    M((7+i),18) = M((7+i),18) .* Amp*exp(-(i - mu_y).^2/(2*sigma^2));
    M((7+i),19) = M((7+i),19) .* Amp*exp(-(i - mu_y).^2/(2*sigma^2));
    M((7+i),20) = M((7+i),20) .* Amp*exp(-(i - mu_y).^2/(2*sigma^2));
    M((7+i),21) = M((7+i),21) .* Amp*exp(-(i - mu_y).^2/(2*sigma^2));
end

for i = 1:18
    for j = 1:30
        M(i,j)=M(i,j)+noise*randn;
    end
end

% Adding noise to whole matrix
track_img = M;

% Check matrix size
matrix_size = size(track_img);


%% find highest value in matrix, and the surrounding nodes
[v1,p1] = max(track_img(:));
[max_ny,max_nx]=ind2sub(matrix_size,p1);

% PAN = Primary Active nodes, with algorithm for edges
PAN = zeros(3,3);

% Algorithm for edges, else error because attempting to access node
% outside of grid
if max_nx == 1
    PAN(1:3,2:3) = track_img(max_ny-1:max_ny+1,max_nx:max_nx+1);
elseif max_ny == 1
    PAN(2:3,1:3) = track_img(max_ny:max_ny+1,max_nx-1:max_nx+1);
elseif max_nx == matrix_size(2)
    PAN(1:2,1:3) = track_img(max_ny-1:max_ny,max_nx-1:max_nx+1);
elseif max_ny == matrix_size(1)
    PAN(1:3,1:2) = track_img(max_ny-1:max_ny+1,max_nx-1:max_nx);
else
```

```
    PAN = track_img(max_ny-1:max_ny+1,max_nx-1:max_nx+1);
end


% No touch --> no track, threshold at 300
if v1<300
    disp('No touch present');
    v1 = [0 0];
    return;
end


%% Least squares

a = (PAN(2,1)+PAN(1,1)-2*PAN(1,2)+PAN(1,3)-2*PAN(3,2)-
2*PAN(2,2)+PAN(2,3)+PAN(3,1)+PAN(3,3));
b = (PAN(3,3)+PAN(1,1)-PAN(1,3)-PAN(3,1));
c = (-PAN(1,1)+PAN(1,3)-PAN(2,1)+PAN(2,3)-PAN(3,1)+PAN(3,3));
d = (-2*PAN(2,1)+PAN(1,1)+PAN(1,2)+PAN(1,3)+PAN(3,2)-2*PAN(2,2)-
2*PAN(2,3)+PAN(3,1)+PAN(3,3));
e = (-PAN(1,1)-PAN(1,2)-PAN(1,3)+PAN(3,1)+PAN(3,2)+PAN(3,3));

% Relative shift of peak node
y_shift = (b*c-2*a*e)/(4*d*a-b^2);
x_shift = (b*e-2*d*c)/(4*d*a-b^2);

%y_shift = (6*b*c-8*a*e)/(16*d*a-9*b^2);
%x_shift = (6*b*e-8*d*c)/(16*d*a-9*b^2);


center_of_finger_LS = [y_shift+max_ny, x_shift+max_nx];



%% Gaussian
row_d_gauss = (log(PAN(2,1)) - log(PAN(2,3))) / 2*(log(PAN(2,1)) -
2*log(PAN(2,2)) + log(PAN(2,3)));
col_d_gauss =(log(PAN(1,2)) - log(PAN(3,2))) / 2*(log(PAN(1,2)) -
2*log(PAN(2,2)) + log(PAN(3,2)));

center_of_finger_gauss =[sqrt(col_d_gauss.^2)+max_ny,
sqrt(row_d_gauss.^2)+max_nx];

%% Linear

row_d_lin = (PAN(2,3) - PAN(2,1)) / (2*(PAN(2,2) -
min(PAN(2,1),PAN(2,3))));
col_d_lin =(PAN(3,2) - PAN(1,2)) / (2*(PAN(2,2) -
min(PAN(1,2),PAN(3,2))));
center_of_finger_lin = [col_d_lin+max_ny, row_d_lin+max_nx];

%% Center-of-Mass
```

```
row_d_com = (PAN(2,3) - PAN(2,1)) / (PAN(2,1) + PAN(2,2) + PAN(2,3));
col_d_com =(PAN(3,2) - PAN(1,2)) / (PAN(1,2) + PAN(2,2) + PAN(3,2));
center_of_finger_com = [col_d_com+max_ny, row_d_com+max_nx];


%% Parabolic
row_d_parab = (PAN(2,1) - PAN(2,3)) / (2*(PAN(2,1) - 2*PAN(2,2) +
PAN(2,3)));
col_d_parab =(PAN(1,2) - PAN(3,2)) / (2*(PAN(1,2) - 2*PAN(2,2) +
PAN(3,2)));
center_of_finger_parab = [col_d_parab+max_ny, row_d_parab+max_nx];


%% Calculate error distance

monte_carlo_sim_LS(q)   = sqrt( (center_of_finger_LS(1)-y_center)^2 +
(center_of_finger_LS(2)-x_center)^2);
monte_carlo_sim_gauss(q)= sqrt( (center_of_finger_gauss(1)-y_center)^2 +
(center_of_finger_gauss(2)-x_center)^2);
monte_carlo_sim_lin(q)  = sqrt( (center_of_finger_lin(1)-y_center)^2 +
(center_of_finger_lin(2)-x_center)^2);
monte_carlo_sim_com(q)  = sqrt( (center_of_finger_com(1)-y_center)^2 +
(center_of_finger_com(2)-x_center)^2);
monte_carlo_sim_parab(q)= sqrt( (center_of_finger_parab(1)-y_center)^2 +
(center_of_finger_parab(2)-x_center)^2);

end

%% Find average error distance
monte_carlo_mean_LS(s) = mean(monte_carlo_sim_LS);
monte_carlo_mean_gauss(s) = mean(monte_carlo_sim_gauss);
monte_carlo_mean_lin(s) = mean(monte_carlo_sim_lin);
monte_carlo_mean_com(s) = mean(monte_carlo_sim_com);
monte_carlo_mean_parab(s) = mean(monte_carlo_sim_parab);



end

%% Average accuracy
monte_carlo_mean_LS_mean     = mean( monte_carlo_mean_LS(s)     );
monte_carlo_mean_gauss_mean  = mean( monte_carlo_mean_gauss(s)  );
monte_carlo_mean_lin_mean    = mean( monte_carlo_mean_lin(s)    );
monte_carlo_mean_com_mean    = mean( monte_carlo_mean_com(s)    );
monte_carlo_mean_parab_mean  = mean( monte_carlo_mean_parab(s)  );
mean_bars = [monte_carlo_mean_LS_mean monte_carlo_mean_gauss_mean
monte_carlo_mean_lin_mean monte_carlo_mean_com_mean
monte_carlo_mean_parab_mean];
```

```
%% Plot Graph
% Simulated peak with true position
subplot(1,2,1);
imagesc(track_img);axis([16.5 21.5 7.5 12.5]);
title('Simulated peak','FontSize',20);
xlabel('x-axis','FontSize',20);
ylabel('y-axis','FontSize',20);
hold on;
scatter(x_center,y_center,100,'k','x')
scatter(x_center,y_center,100,'k','o')
hold off;

% Accuracy related to S/N
subplot(1,2,2);
plot(monte_carlo_mean_LS,'-ro');
hold on;
plot(monte_carlo_mean_gauss,'-co');
plot(monte_carlo_mean_lin,'-go');
plot(monte_carlo_mean_com,'-ko');
plot(monte_carlo_mean_parab,'-bo');
hold off;
legend('Least Sqaures','Gaussian','Linear','Center of mass','Parabolic')
title('Accuracy performance','FontSize',20);
axis([0 21 0 0.3]);
xlabel('Signal to noise ratio (dB)','FontSize',20);
ylabel('Distance error \epsilon','FontSize',20);

% Average accuracy depending on S/N
%figure;bar(mean_bars);title('Average accuracy
performance','FontSize',20);
%ylabel('Distance error \epsilon','FontSize',20);
```