NTNU – Trondheim
Norwegian University of
Science and Technology

Detection and tracking of objects in a low resolution grayscale image

Sveinung Pettersen Røsok

Master of Science in Electronics
Submission date: June 2015
Supervisor: Lars Magne Lundheim, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Summary

Tracking is the technique of following an object in motion. We use tracking technology on an everyday basis through Navigation GPS in cars or smart phones. Many of these units also feature touch displays, which allow interaction using a finger or touch stylus. This master thesis focuses on with tracking fingers on touch displays, which appear as objects on a low-resolution image. Touch displays often use a low resolution sensor grid, which requires subpixel estimation prior to tracking. The coordinates produced by the tracking system is used by an application level not included in this project, which analyzes the input location and motion. With higher accuracy of the position and tracking, the smaller symbols can be utilized on the display, and the more sophisticated motions are possible to interpret for the application level. The tracker system is designed to work in *real time* for any touch display and tracks up to two fingers of any size.

High tracking accuracy was achieved using digital signal processing techniques. A signal processing model was created initially to define the system. The tracker system created consists of two modules: A *scanner* and a *tracker*. The scanner analyzes the data sets individually, and produces a high-resolution two-dimensional coordinate for each input. The tracker analyzes these observations collectively, validates that these are not caused by noise, and filters the positions through a (Kalman) tracking filter. Evaluation of the system was performed using data sets from a real touch display as well as simulated data sets.

Some assumptions and limitations had to be made to successfully handle all situations found in the data sets. Pressing hard on the touch displays creates large objects, and a size estimation algorithm was made. This was based on a sensor value threshold, but this makes the program non-ideal for displays with low signal to noise ratio. Two close objects might also be falsely identified as one large object, but this was solved in this project using previous knowledge and adjusting the sensor values to force the scanner to produce two outputs. Lastly, the tracking filter smoothed the trajectory of the objects, but it did not always provide better accuracy. A tracking filter should be considered based on the application intended.

All situations in the real data sets were handled, albeit with reduced accuracy for large objects and for two close objects. Further research involves handling three or more inputs, performing a running cost analysis of the algorithm and implementing this on a real tablet. Implementation will require an adjustment of the different thresholds and settings to match the touch display with regards to node resolution, sampling rate and sensor noise. This can be solved by the application level or further development of the tracker system.

[This page is intentionally left blank]

Sammendrag

Målfølgelse er et begrep som brukes for å beskrive følging av et mål i bevegelse. Alle som bruker GPS i bil eller smarttelefoner bruker målfølgingsteknologi. Mange av disse enhetene benytter også berøringsskjermer, som tillater interaksjon med å bruke en finger eller stylus. Denne masteroppgaven handler om målfølgelse av fingre på en berøringsskjerm, og disse fremstår som objekter på en lavoppløst bilde. Berøringsskjermer benytter ofte slike sensor-system, noe som medfører at man må estimere en underpiksel-posisjon, før man kan målfølge objektene. Koordinatene som produseres av målfølgingssystemet analyseres av et applikasjonsnivå, som er utenfor omfanget av dette prosjektet. Jo høyere nøyaktighet på posisjonene som produseres, jo mindre symboler kan benyttes på en berøringsskjerm, og jo mer avanserte bevegelser kan applikasjonsnivået tolke. Målfølgingssystemet fungerer i sanntid og for en generell, og ikke spesifikk, berøringsskjerm. Den er designet for å tolke inntil to objekt på samme tid.

Høy presisjon ble oppnådd med å bruke digital signalbehandling. En signalbehandlingsmodell ble laget initielt for å definere systemet. Målfølgingssystemet som ble laget består av to moduler: En *skanner* og en *målfølger*. Skanneren analyserer alle datasett fra sensoren til berøringsskjermen, og produserer en to-dimensjonalt og høy-oppløst posisjon for inntil to objekt. Målfølgeren analyserer alle observasjonene kollektivt, validerer at det ikke er støy, og sender posisjonene gjennom et (Kalman) målfølgingsfilter. Utvikling og vurdering av hele systemet ble gjort ved å benytte ekte datasett fra en berøringsskjerm i tillegg til simulerte datasett.

Noen begrensninger ble valgt for å få en vellykket håndtering av alle situasjonene som dukket opp i de ekte datasettene. Harde trykk på berøringsskjermer skaper store gjenstander i datasettene, og en algoritme for størrelsesestimering ble implementert. Denne algoritmen for størrelse baseres på en terskel for sensorverdi, men det medfører at målfølgingssystemet ikke er ideelt for skjermer med lavt signal til støyforhold. To nære objekter kan også feilaktig identifiseres som ett stort objekt, men dette ble løst ved hjelp av informasjon fra tidligere datasett og ved å justere sensorverdiene, slik at skanneren tvinges til å produsere to posisjoner. Målfølgingsfilteret glattet banen av objektene, men filteret gav ikke alltid en bedre nøyaktighet. Om et målfølgingsfilter skal benyttes må dette vurderes opp mot hvilken applikasjon som skal brukes.

Alle situasjoner i de virkelige datasettene ble håndtert, riktignok med noe redusert nøyaktighet for store objekter og to nære objekter. Videre forskning bør innebære håndtering av tre eller flere objekter, en løpende kostnadsanalyse av algoritmen samt å implementere dette på en ekte berøringsskjerm. Implementering krever justering av de forskjellige terskler og innstillinger for å matche den spesifikke berøringsskjermen med hensyn til node-oppløsning, samplingsfrekvens og målfølgingsfilteret, noe som kan løses av applikasjonsnivået eller en videreutvikling av målfølgingssystemet.

[This page is intentionally left blank]

Preface

This thesis is submitted to fulfill the formal requirements for the Master of Science in Electronics Engineering (MSc), at the Department of Electronics and Telecommunications, Norwegian University of Science and Technology (NTNU). The Master program is a continuation of my Bachelor in Military Leadership, with specialization in electronics engineering, completed at the Royal Norwegian Air Force Academy in 2008-2009 and the University of Agder in 2009-2011.

A working algorithm for touch displays already exists, but for educational development and professional reasons this task was chosen.

Acknowledgement

First I want to thank the Royal Norwegian Air Force, FLO staff and NTNU MIEL administration for creating the frame and giving me the opportunity to expand my knowledge in the technical world.

I would like to express my sincere gratitude to my supervisor, Professor Lars Lundheim, at NTNU, for all the advice and discussions throughout the semester. His commitment and professional input has contributed to make this thesis the highlight of my professional education.

I would also like to thank Einar Fredriksen from Atmel, for providing great data for testing the algorithms, and for good academic discussions.

Finally, a special thanks to my fellow students, friends and family for supporting me through this period.

[This page is intentionally left blank]

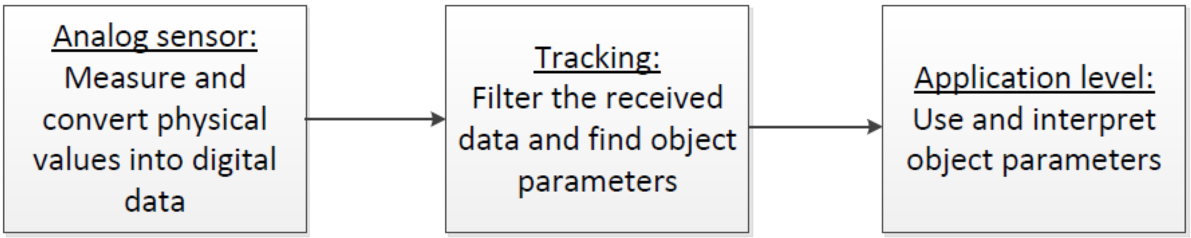
Table of contents

[This page is intentionally left blank]

1 Introduction

1.1 Background

The term tracking is used to describe the observation of an object in motion. A basic tracker overview is presented on figure 1.1. For human beings the act of tracking is performed unconsciously by the brain, by first sorting out relevant information collected from our sensory organs, such as the eyes or ears. The brain analyzes this information and calculates valuable information, like where an object is heading or how fast it is travelling. Tracking used by electrical systems like a Navigation GPS or a smartphone uses the same principles as our brain. The tracking begins with gathering physical measurements from a sensor like an antenna or lens. This analogue data is made digital and then filtered, using signal processing to calculate the object parameters. The information finally ends up as our personal location on your Navigation GPS or smartphone.



Analog sensor: Measure and convert physical values into digital data

Tracking: Filter the received data and find object parameters

Application level: Use and interpret object parameters

Figure 1.1: Simple tracking overview

Creating a good tracker is a challenge in the same way as designing an aircraft, car or smartphone; there is no ultimate tracker. It is dependent on both software and hardware and is a compromise of specification requirements. Hardware is limited by physical size, battery and processing power. Software is limited by development time and processing time. Development time is dependent on level of complexity, like how many sensors there are, and the quality of that information. Processing *time* is critical in cases where the program works in real time and where we want minimum delay, and it is limited by the processing power of the device. In addition, the selection of software and hardware comes with a cost. The engineering process therefore starts with defining minimum criteria of the performance parameters and the development of software and hardware starts from this point.

The tracking object in this thesis is a finger, and the sensor is a touch display known from tablets and smartphones. A higher tracking accuracy allows interaction with smaller objects on the screen, and also allows more sophisticated gestures for the application level to interpret.

1.2 Touch display technology

A touch display is an interface which allows interaction with a computer, by either pushing the display surface at a specific position or by making a specific motion [1]. There are many different touch technologies, both related to software and hardware. Some technologies might require a special pen or stylus to be used, and some support only a single press at a time. More advanced touch displays support multiple inputs and advanced motions [2].

The first article describing a touch screen was presented in 1968, in the application of air traffic control [2]. In the 1990's, this technology became portable enough to be used on small electronic devices [2], but it was not until the release of the iphone in 2007 that this became the new standard [3]. The most common touch technology used today on smartphones and tablets is *projected capacitance* with *mutual capacitance*[4]. It is preferred because it is both optically clear and durable, due to no moving parts. It is also compatible with multiple inputs and can understand a wide array of fingers and styluses [2].

The mutual capacitance screen is made of several layers, as illustrated by figure 1.2. At the bottom we find the LCD display, followed by clear isolating and protective layer. Above this we find the driving lines and sensing lines, forming a hash pattern that is usually diamond or square. The lines are made of a conductive and nearly invisible metal alloy called indium tin oxide (ITO) [2].

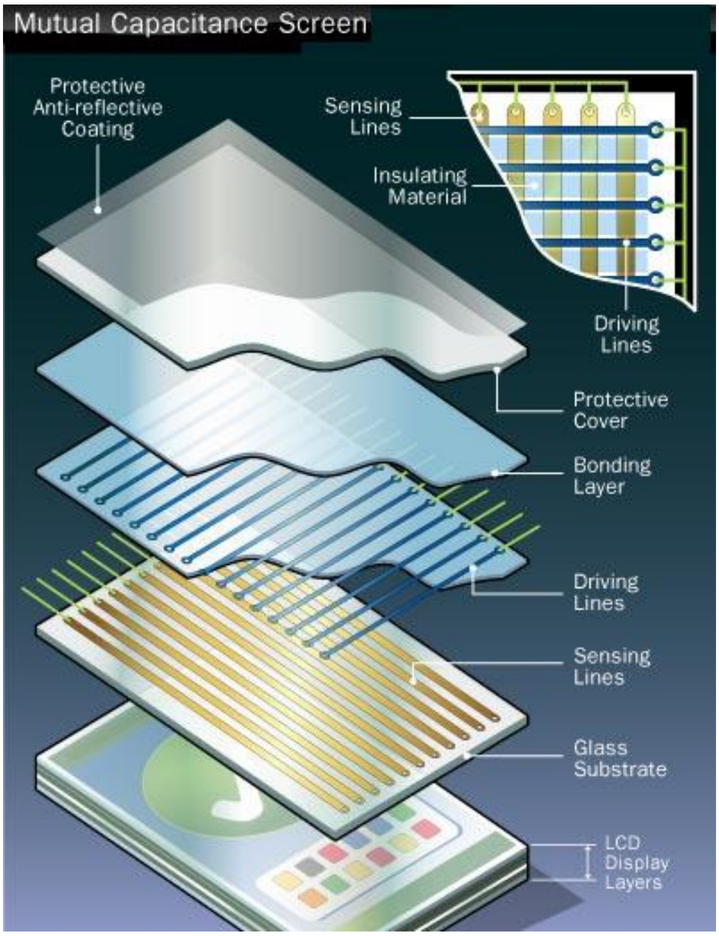


Figure 1.2: Different layers of a general mutual capacitance screen [5]

When a touch display is activated, the LCD display is powered on. In addition, a small voltage is applied to the driving lines, which creates a capacitive effect between each junction of the driving lines and sensing lines. This results in a unison electro-static field across the entire touch display surface. Every junction is termed a *node*, and these nodes constitutes the sensors of the touch display. The number of nodes can vary based on the size of the screen, the space between the nodes and the node layout pattern. In this thesis, a side-by-side pattern was assumed, illustrated by figure 1.3.

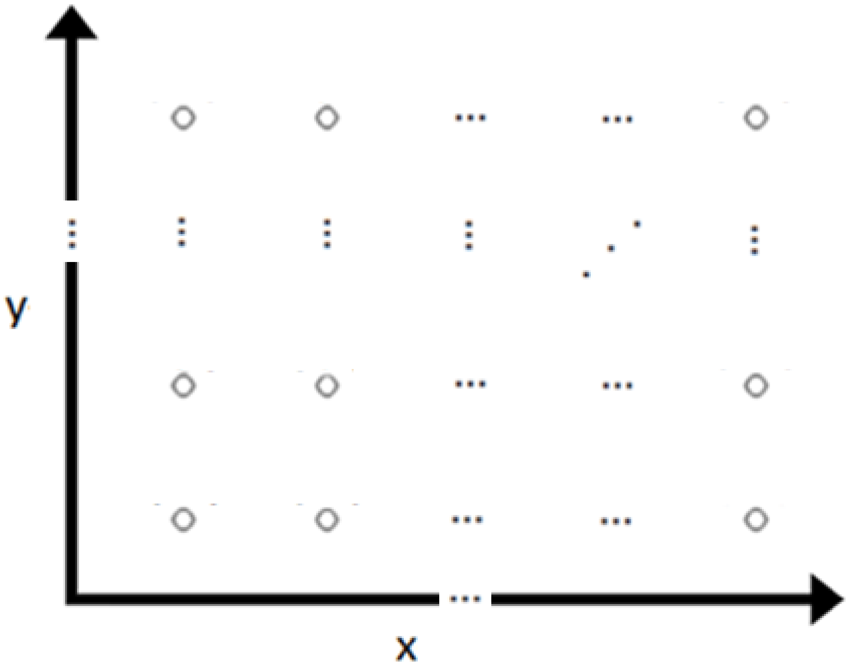


Figure 1.3: Touch screen node layout

The capacitive intensity value of each junction is sampled, at a fixed sampling rate, and stored in data sets. If a finger connects physically with the screen, it will reduce the capacitive effect of the closest nodes, because the finger works as natural ground and leads current away from the sensing lines, as illustrated by figure 1.4. These data sets are analyzed to determine the location of the finger on the screen.

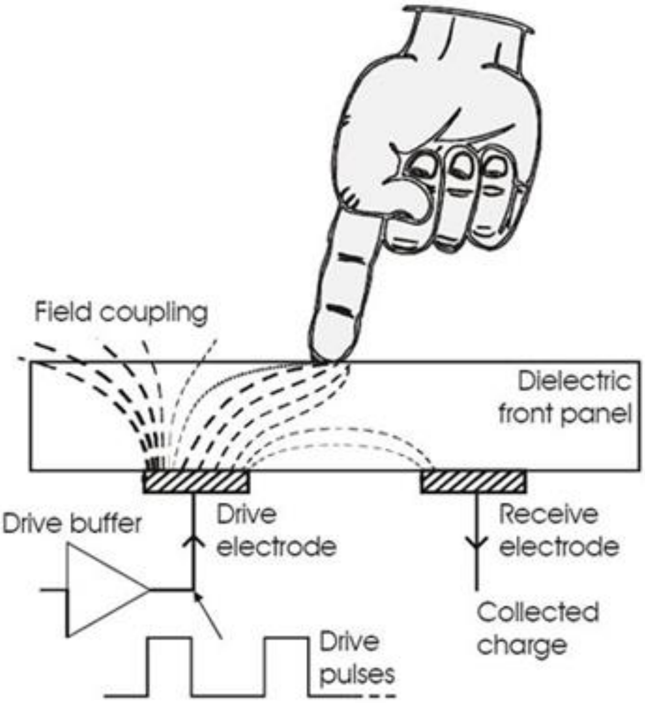


Figure 1.4: Capacitive screen behaviour [6]

In general, the more a node is covered, the higher the intensity value is measured and stored in the data sets. However, these are passive components and the intensity measurements do not scale linearly, which makes estimation harder. Figure 1.5 shows the touch display of four different smart phones with similar hardware, where a medium pressure is applied by a finger to draw straight lines. The results are varying from mildly sloped to curvy, which underlines the importance of signal processing.

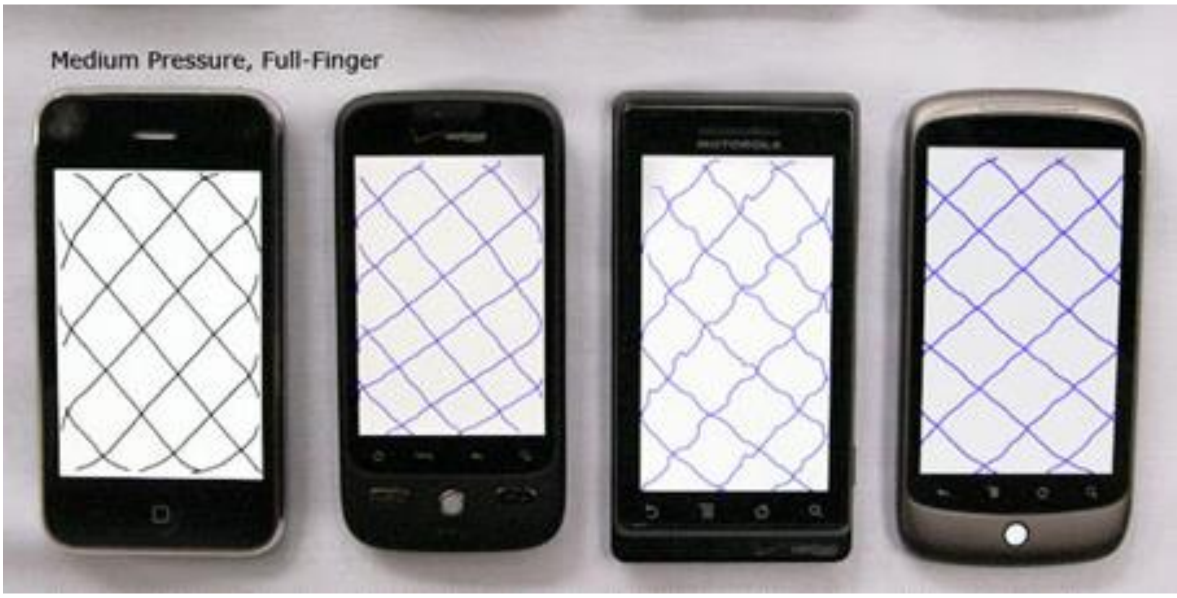


Figure 1.5: Comparison of signal processing capabilities on mutual capacitance touch screens [7]

1.3 Problem description

In this thesis, a tracking algorithm for a non-specific touch display is to be developed. This will be termed the *tracking system* from this point on. The tracking system will receive previously sampled and digitalized data sets from a mutual capacitance sensor, where the accuracy of tracking is limited by the node resolution and node value resolution from the sensor. Figure 1.6 shows a single data set, where the space between each node is 5 mm in both directions and the display has 22x37 nodes. The node value resolution in this data set goes from 0 to 1700. An index finger, or *touch input*, pressing on the display creates a small area with brighter colors, referred to as an *object*.

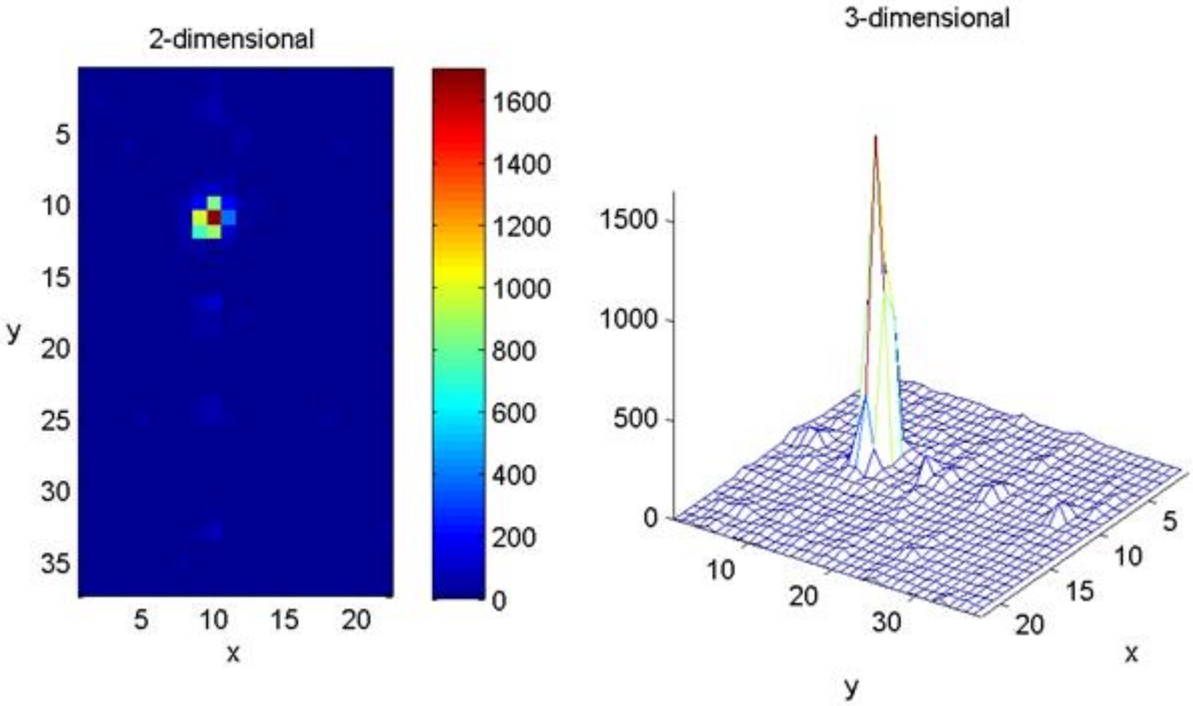


Figure 1.6: Visual representation of index finger from the touch display sensor point of view

A two-dimensional coordinate, or position, is to be produced for all present objects in each data set. The position of each object must be estimated with significantly greater accuracy than the current node resolution. This means that a high-resolution position must be estimated prior to tracking. This high-resolution position will be referred to as the *subpixel* position. The recipient of these coordinates is an application level, which is not in the scope of this thesis. The size of the node value resolution is important, but not discussed further in this thesis.

The algorithm should work for any screen size, node resolution and sampling rate. It should work for up to two objects, and for both fingers and styluses. In addition, the positions should have low jitter in the presence of noise. Jitter is here used to describe the natural variation of the estimated position, caused by hardware and software imperfections. The computational cost of the tracker system is not prioritized in this thesis.

A literature survey performed in the TTT-4511: Digital Signal Processing specialization project[8], showed that there were a scarce number of articles and little information available on the topic of object tracking in a low-resolution environment. It has not been possible to compare this thesis to other projects or systems and, for this reason, this master thesis is a continuation of the previous project mentioned, based on independent work. For testing and evaluation, both simulated data sets and real data sets are used. The real data sets are sampled from a real touch display, delivered by Atmel[9].

The author assumes that the reader has basic knowledge of estimation theory and signal processing.

A final note: While the title describes low-resolution on a *grey-scale* image, colors have been to better illustrate the intensity differences.

Thesis outline

Chapter 2. Signal processing model: Model concepts, definitions and terminology used in this thesis is presented.

Chapter 3. The tracker system: The Tracker system is introduced.

Chapter 4. The Scanner unit: The main algorithms of the scanner unit are presented, demonstrated and discussed.

Chapter 5. The Tracker unit: The main algorithms of the tracker unit are presented, demonstrated and discussed.

Chapter 6. Results: The complete tracker system is demonstrated and discussed.

Chapter 7. Topics for further research: This chapter presents a natural way forward, and the Tracker system is viewed in a big picture.

2 Signal processing model

In this chapter, the parameters and terminology used for detection and position estimation is explained. It will also investigate the challenge of defining the true position, how simulated objects are created and how simulated objects compare to real objects.

2.1 Single touch: 1 object

Each complete set of measurements for the nodes is defined to be

$$F_i(m, n) \tag{2.1}$$

where *i* is the sequence number of the data set. The nodes are an ordered pair (m, n), where *m* represents the node sequence number in x-direction and *n* represents the sequence number in y-direction. They are defined as natural numbers and the distance between two nodes in either x-direction or y-direction is defined to be one. The nodes are placed on a display with two axes, x and y, illustrated by figure 2.1.

$$m, n \in [1, \infty)$$

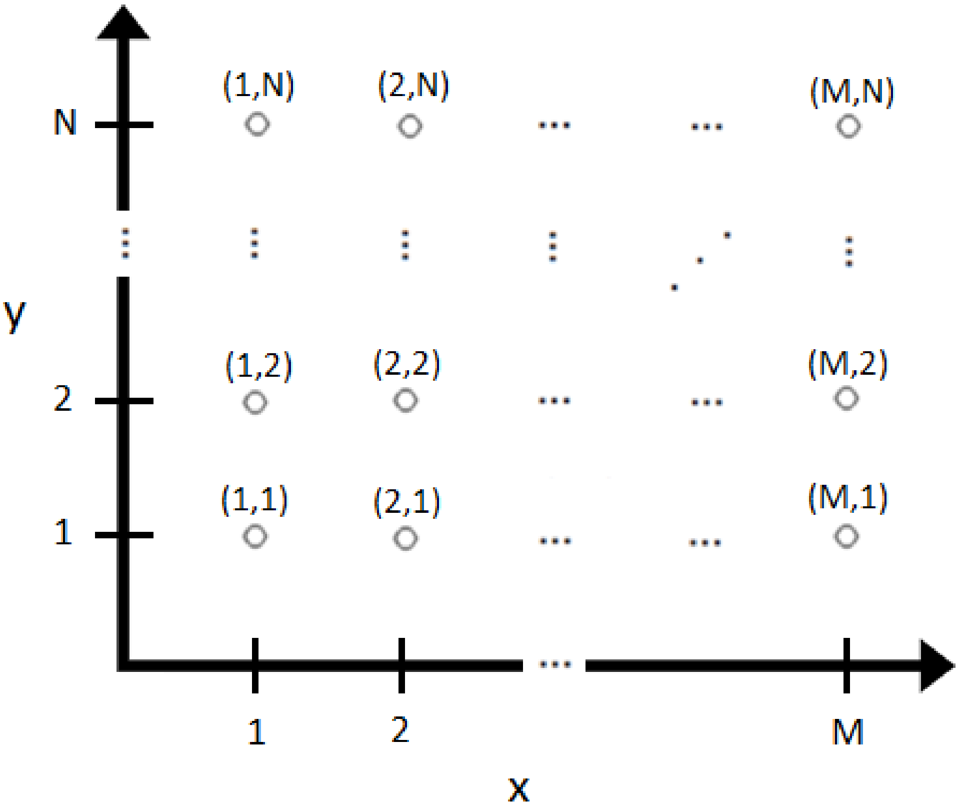


Figure 2.1: Node layout

The object has any size and is modeled with any shape and size, with a center in origo:

$$f(x, y). \tag{2.2}$$

The true position, (x_0, y_0), is modeled by equation (2.2).

$$f(x - x_0, y - y_0) \tag{2.3}$$

This position will consist of two parts, for both the x-dimension and y-dimension. The first part is the position of the closest node, denoted (m, n). The second part of the position is the displacement, (δ_x, δ_y), defined by equation (2.4). This is the displacement of the true position relative to (m, n), and it serves to increase the resolution of the nodes.

$$\delta_x, \delta_y \in \mathbb{R} \tag{2.4}$$

The true position is now described by equation (2.5) and (2.6).

$$x_0 = m + \delta_x \tag{2.5}$$

$$y_0 = n + \delta_y \tag{2.6}$$

The estimated position of the object is consequently defined by equation (2.7) and (2.8).

$$\hat{x}_0 = \hat{m} + \hat{\delta}_x \tag{2.7}$$

$$\hat{y}_0 = \hat{n} + \hat{\delta}_y \tag{2.8}$$

The estimation will lead to an inaccuracy, or error distance ε, which is caused by algorithm inaccuracy and hardware imperfections. For simplicity this error distance is combined, shown by formula (2.9) and illustrated by figure 2.2.

$$\varepsilon = \sqrt{(x_0 - \hat{x}_0)^2 + (y_0 - \hat{y}_0)^2} \tag{2.9}$$

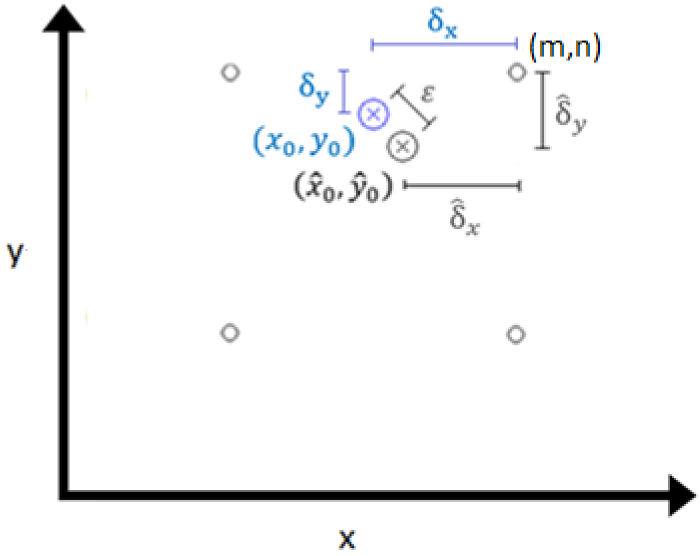


Figure 2.2: True position (blue circle with cross) and estimated position (black circle with cross) of object

Noise caused by hardware imperfections is part of model and, in this thesis, it is assumed that it is additive white Gaussian noise with general characteristics, given by equation (2.10).

$$w \sim \mathcal{N}(\mu_w, \sigma_w^2) \tag{2.10}$$

The complete model for a single touch input will therefore be

$$F(m,n) = f(m - x_o, n - y_o) + w(m,n).$$

2.2 Multi touch: 2+ objects

Multi touch displays are compatible with analyzing more than for each data set $F_i(m,n)$. For L number of touch inputs, the position for each object is:

$$x_l = m_l + \delta_{xl}$$

$$y_l = n_l + \delta_{yl}.$$

This leads to the final model for our touch display inputs:

$$F(m,n) = \sum_{l=1}^{L} f_l(m - x_l, n - y_l) + w(m,n)$$

In this thesis, the number of objects to be detected is up to two, or $L = 2$.

2.3 True position of object

The true position of an object was denoted (x_o, y_o) in formula (2.5) and (2.6), but it is not obvious what this position is, even if an object has a symmetric form. Figure 2.2 illustrates this challenge using a non-symmetric one-dimensional figure, with the node intensity values from figure 1.6, from y-row 11. A fictional function, $f(x)$, is added, and the nodes $m = 8$ to $m = 12$ are used for this function.

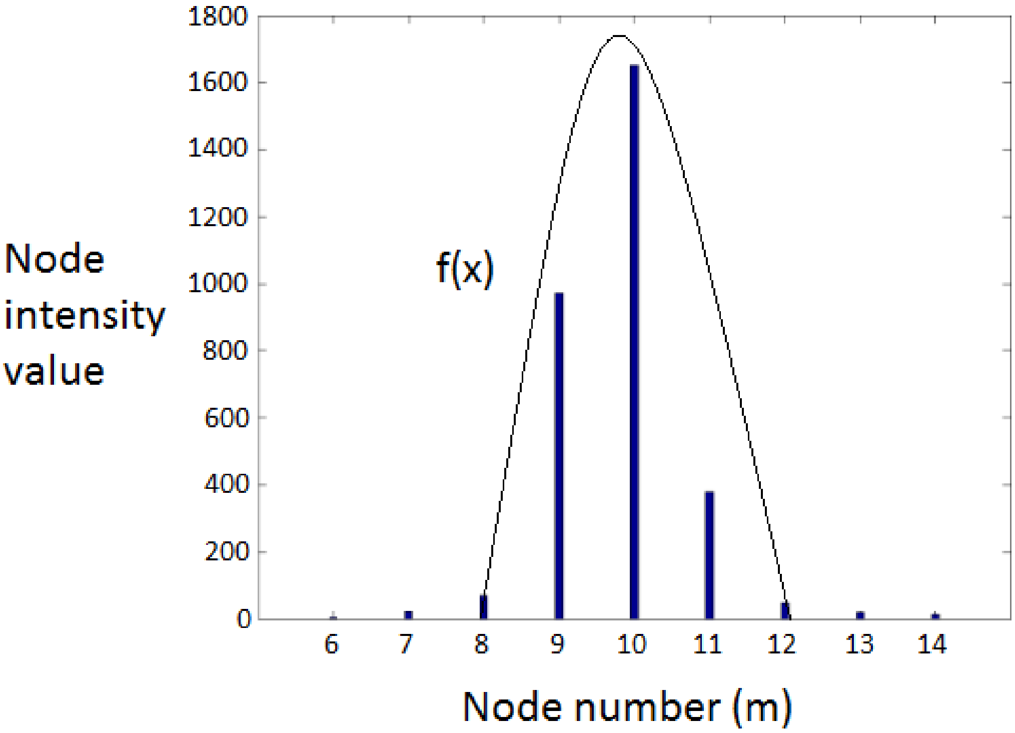


Figure 2.3: The challenge of the true position

Two definitions for the true position are used in this thesis. The first is the *peak value*, which views the discrete measured values as a continuous function $f(x)$, and is mathematically defined by formula (2.11).

$$x_o = arg \max_x f(x) \tag{2.11}$$

The second definition is *center of mass*, defined mathematically by equation 2.12.

$$x_o = \int_{-\infty}^{\infty} xf(x)dx \tag{2.12}$$

The practical consequence of these two definitions is shown on the next page, beginning with the peak value.

In the specialization project, peak estimation techniques were compared [8]. All techniques assumed that nine nodes, shaped in a quadratic area of 3x3, contained all information needed to get a good subpixel estimate of an object. This is illustrated by figure 2.4 (page 14), and objects of this approximate size will be referred to as *small objects*. The best peak estimation technique overall was *Least Squares.* This technique seeks to find the peak, by minimizing the intensity difference, or Least Squares Error (LSE), using equation (2.13). The complete derivation can be found in the appendix [B].

$$LSE = \sum_{m=-1}^{1} \sum_{n=-1}^{1} (\, F(m,n) - \hat{F}(m,n) \,)^2 \qquad (2.13)$$

The second method is *center of mass*. This method was chosen because it complements peak value estimation, by being applicable for any size. The estimated true position was found using equation (2.14) and (2.15), where the size of the rectangular area is:

$$M_1 < m < M_2$$
$$N_1 < n < N_2.$$

$$\hat{x}_0 = \frac{\sum_{m=M_1}^{M_2} m \sum_{n=N_1}^{N_2} F_i(m,n)}{\sum_{m=M_1}^{M_2} \sum_{n=N_1}^{N_2} F_i(m,n)} \qquad (2.14)$$

$$\hat{y}_0 = \frac{\sum_{m=N_1}^{N_2} n \sum_{n=M_1}^{M_2} F_i(m,n)}{\sum_{m=M_1}^{M_2} \sum_{n=N_1}^{N_2} F_i(m,n)} \qquad (2.15)$$

2.4 Real and simulated objects

Real objects are hard to simulate because they come in different sizes and shapes, and are often non-symmetrical. Simulating objects cannot replace real data sets, but allows for testing if different algorithms work as intended in a controlled environment. It also allows quantifying algorithm accuracy because the true position is known. For this reason the practical testing of the algorithms includes *simulating* objects on a simulated touch display model. A display was simulated initially by creating a 22x38 matrix, equal to the size of the data sets from Atmel. A Gaussian distributed object was then created on the display, with an intensity *I*, created by formula (2.16).

$$f(x,y) = Ie^{-\frac{(x-x_0)^2+(y-y_0)^2}{\sigma^2}} \tag{2.16}$$

This leads to a true position located at $(x - x_0, y - y_0)$. Since the object is symmetric and unimodal the peak and center of mass are theoretically equal. Additive white Gaussian noise is added to each node after creating the object. The added noise was created and scaled with formula (2.17).

$$SNR = \frac{I}{\sigma_w^2} \tag{2.17}$$

I is the magnitude of the generated touch input, found in formula (2.12), and σ_w^2 is the magnitude of the hardware noise, defined in formula (2.9). The chapter is ended with a comparison of a real data set and a simulated data set, shown by figure 2.4. The real data set contains a data set where two index fingers are pressing near the middle of the screen

$$(x_1, y_1) = (12{,}6)$$

$$(x_2, y_2) = (17{,}12)$$

The simulated data set aims to imitate the real data set in a 30dB SNR environment. The simulated objects are arbitrarily placed at:

$$(x_1, y_1) = (14{,}10)$$

$$(x_2, y_2) = (19{,}16)$$

Values were chosen to resemble a real object, both in size and intensity. The parameters used for testing in this thesis, in formula 2.12 and 2.13, are:

$$I = 900$$

$$\sigma^2 = 1.28$$

$$\sigma_w^2 = 0.9.$$

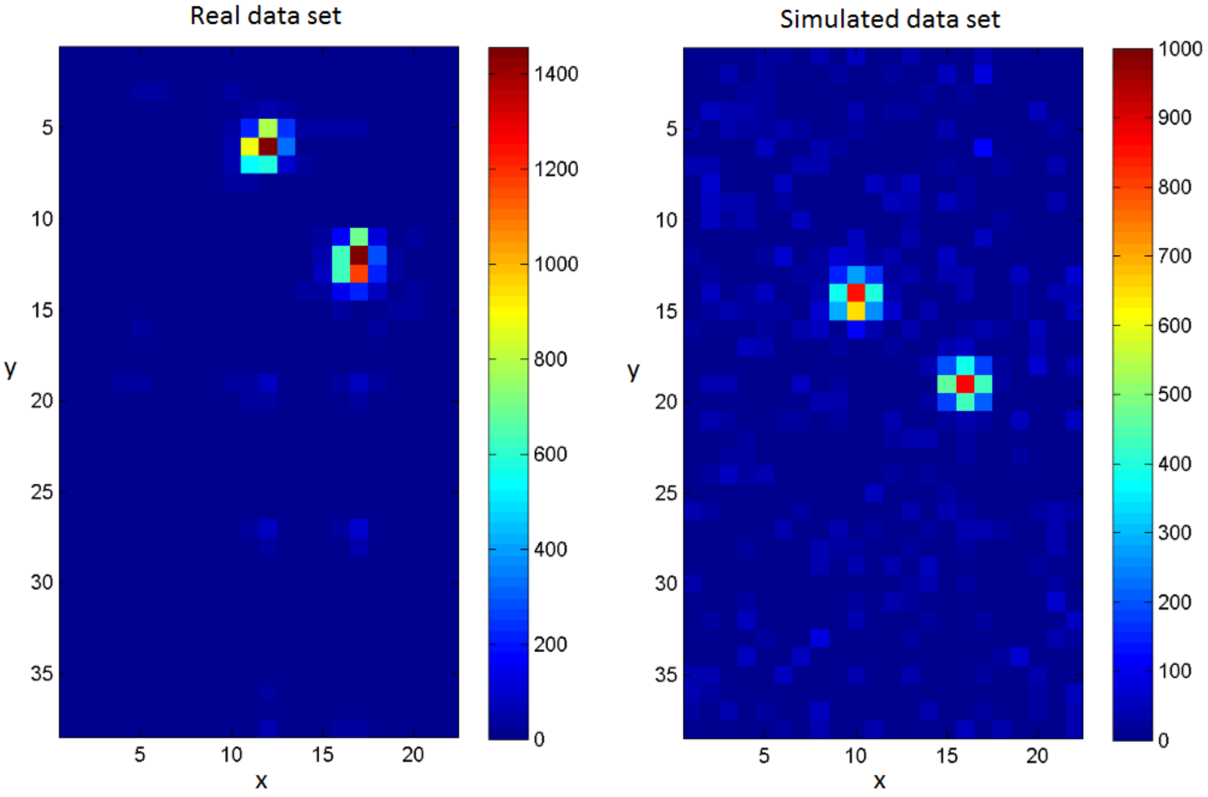


Figure 2.4: Comparison of real data set and simulated data set

The size and shape of the objects in the data sets are similar, while the intensity is slightly lower for the simulated data set. This might affect the accuracy results shown later, both positively and negatively, but the limit was chosen based on the real data sets from the specialization project [8]such that SNR levels could be calculated. Both data sets have two nodes that stands out, with higher intensity value, shown with dark brown or red color. Both objects, for both data sets, are small objects, defined in chapter 2.3.

We observe that the real data set and the simulated data set appear slightly different with regard to noise. The real data set has increased noise in the same columns as the present objects, while the simulated data has noise on the entire touch display. This because values are sampled column-wise for the real data sets, and noise is added to all nodes for the simulated display.

Overall, the simulated conditions are considered satisfactory for testing and evaluation of the algorithms in the tracker system.

3 Tracker system overview

A tracker system has to filter and analyze measurements from all nodes in each data set. Even if processing time is not prioritized, this should be performed in an effective manner, to ensure that the time consumed is minimized.

A two-module system was chosen, illustrated by figure 3.1. This system consists of a *scanner* unit and a t*racker* unit, which performs different tasks and communicates with each other in the process.

The scanner unit scans and analyzes each dataset, $F_i(m, n)$, individually and independently, and estimates the subpixel position, $(\hat{x}_0, \hat{y}_0)$, of the objects. The tracker unit does not receive the data set, but instead receives these estimated positions. The task of this unit is to validate the objects, label them, and filter the positions through a tracking filter. The result is the updated and confirmed position of each object, $(\hat{\hat{x}}_0, \hat{\hat{y}}_0)$, which is sent to the application level. The tracker unit can also send parameters to the scanner, which is described further in chapter 4.4.

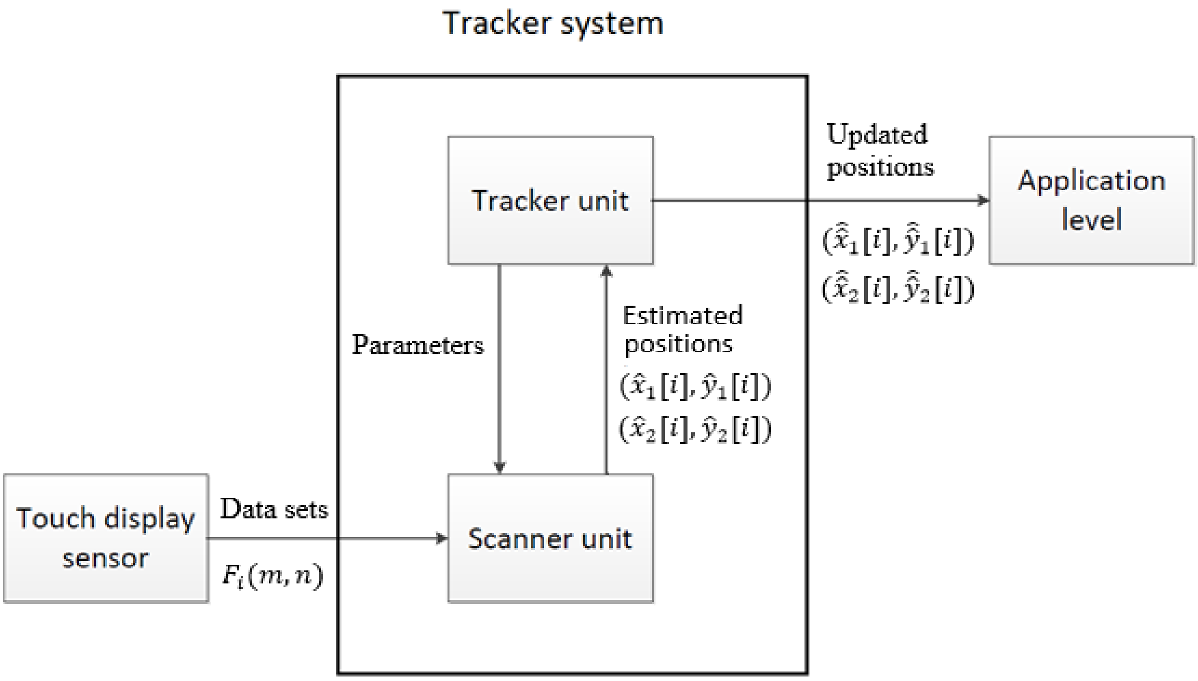
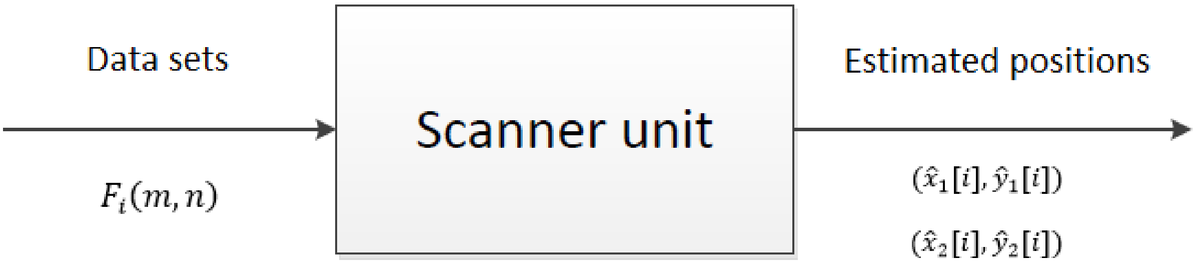

Tracker system
Tracker unit
Updated positions
Application level
$(\hat{\hat{x}}_1[i], \hat{\hat{y}}_1[i])$
$(\hat{\hat{x}}_2[i], \hat{\hat{y}}_2[i])$
Parameters
Estimated positions
$(\hat{x}_1[i], \hat{y}_1[i])$
$(\hat{x}_2[i], \hat{y}_2[i])$
Touch display sensor
Data sets
$F_i(m, n)$
Scanner unit

Figure 3.1: 2-module tracker system

[This page was intentionally left blank]

4 Scanner unit

This unit has one objective; to analyze the incoming data sets and produce a two-dimensional coordinate, $(\hat{x}_0, \hat{y}_0)$, for two objects. Figure 4.1 illustrates a simplified model of the scanner unit.



Data sets
$F_i(m, n)$
Scanner unit
Estimated positions
$(\hat{x}_1[i], \hat{y}_1[i])$
$(\hat{x}_2[i], \hat{y}_2[i])$

Figure 4.1: Scanner unit box

The scanner unit has no memory. For each data set, $F_i(m, n)$, it performs a scan based on preset parameters, and three different situations can occur. Observe that the scanner reports the position (0,0) for non-existing objects.

Situation 1: No objects found

$$(\hat{x}_1[i], \hat{y}_1[i]) = (0{,}0)$$

$$(\hat{x}_2[i], \hat{y}_2[i]) = (0{,}0)$$

Situation 2: One object found:

$$(\hat{x}_1[i], \hat{y}_1[i]) = \left(\hat{m}_1 + \hat{\delta}_{x_1}, \hat{n}_1 + \hat{\delta}_{y_1}\right)$$

$$(\hat{x}_2[i], \hat{y}_2[i]) = (0{,}0)$$

Situation 3: Two objects found

$$(\hat{x}_1[i], \hat{y}_1[i]) = \left(\hat{m}_1 + \delta_{x_1}, \hat{n}_1 + \delta_{y_1}\right)$$

$$(\hat{x}_2[i], \hat{y}_2[i]) = \left(\hat{m}_2 + \delta_{x_2}, \hat{n}_2 + \delta_{y_2}\right)$$

A simple flow chart for the scanner unit is shown by figure 4.2.

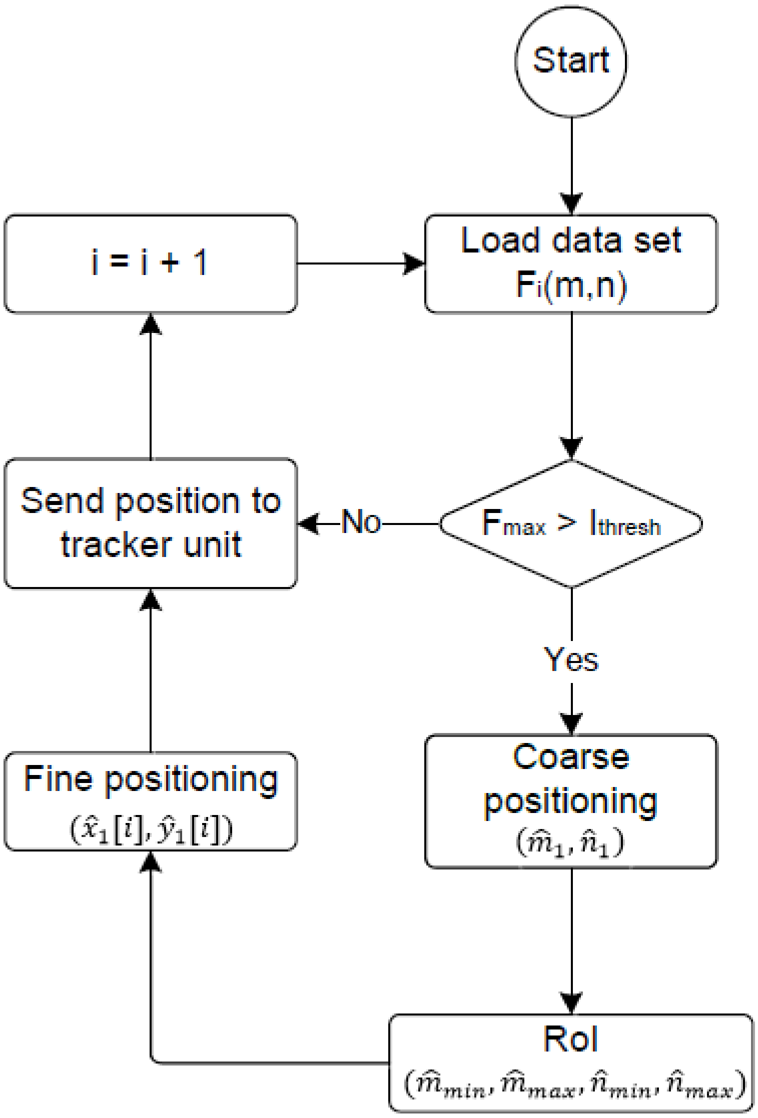

Start
i = i + 1
Load data set
$F_i(m,n)$
Send position to tracker unit
No
$F_{max} > I_{thresh}$
Yes
Fine positioning
$(\hat{x}_1[i], \hat{y}_1[i])$
Coarse positioning
$(\widehat{m}_1, \hat{n}_1)$
RoI
$(\widehat{m}_{min}, \widehat{m}_{max}, \hat{n}_{min}, \hat{n}_{max})$

Figure 4.2: Find 1-algorithm flow chart

The rest of this chapter is dedicated to provide insight and understanding to the processes that are running in the scanner, shown on the flow chart above.

4.1 No objects

When there is conductive pressure on the touch display, the measured node intensity values increase in the local area where it is pressed, as shown in chapter 1.2. The quality of the sensor hardware, combined with the physical environment where the touch display is used, creates a SNR, defined in chapter 2.4. Identifying the presence of a single object can be the simplest task of the touch display algorithm, if the SNR is sufficient. In this project it is assumed that it is not, and a test was developed to stop the process of fine-positioning if this happens.

A minimum threshold, I_{thresh}, is introduced, to represent the limit for activating the fine positioning algorithm. We compare this to the highest intensity value measured in a data set, denoted F_{max}, defined by equation 4.1.

$$F_{max} = \max_{m,n} F_i(m,n) \tag{4.1}$$

If F_{max} exceeds I_{thresh} the algorithm will continue to calculate a subpixel position. If the limit is not exceeded, the data set $F_i(m,n)$ is discarded, and $F_{i+1}(m,n)$ is loaded. The process is explained using the pseudo-code below, and this is also shown on the flow chart on figure 4.2.

Pseudo code: Activate subpixel estimation

```
$if\ F_{max} > I_{thresh}$
    $activate\_subpixel\_estimation;$
$else$
    $(\hat{x}_1, \hat{y}_1) = (0,0);$
    $(\hat{x}_2, \hat{y}_2) = (0,0);$
$end$
```

It is assumed that even if no object is present, the intensity of the noise might exceed I_{thresh}, and this results in the subpixel estimation of *false objects*. The challenge of identifying false targets is solved by the tracker unit, presented in chapter 5. For now, it is assumed that all identified objects are real.

4.2 Find one object

The process of finding one object is demonstrated in this section. Assuming $F_{max} > I_{thresh}$, the scanner will continue to find the estimated position of object 1, $(\hat{x}_1, \hat{y}_1)$. In the specialization project [8] this was done in a two-step process: *Coarse positioning* and *fine positioning*. Situation 1 is shown on figure 4.3 and it illustrates a real data set, where an index finger touches bottom half of the screen.

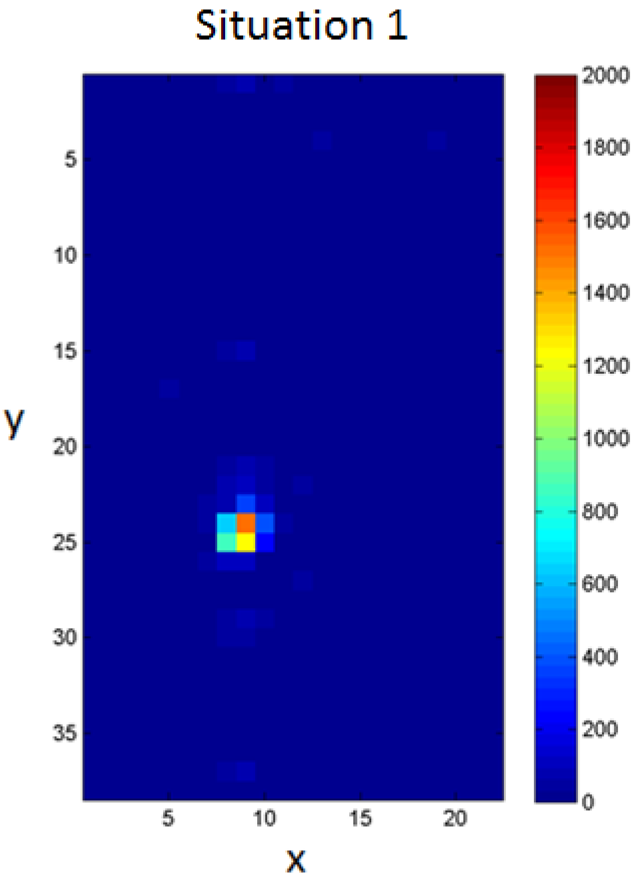


Figure 4.3: Index finger

The coarse positioning is performed initially by finding the most activated node, which is the same node that has the registered F_{max}. The result produced is $(\hat{m}_1, \hat{n}_1) = (9\,,24)$, and is illustrated by figure 4.4.

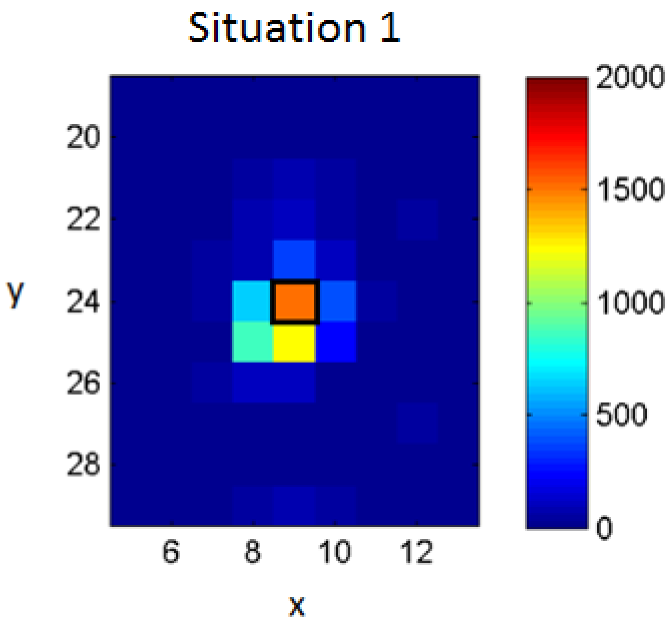


Figure 4.4: Coarse positioning completed

The fine positioning-algorithm estimates the subpixel position, using the Least Squares method shown in chapter 2.3. The final result for Situation 1 is shown by figure 4.5, with the positions calculated below.

$$\hat{x}_1 = \hat{m}_1 + \hat{\delta}_{x_1} = 9 + (-0.17) = 8.83$$

$$\hat{y}_1 = \hat{n}_1 + \hat{\delta}_{y_1} = 24 + 0.40 = \ 24.40$$

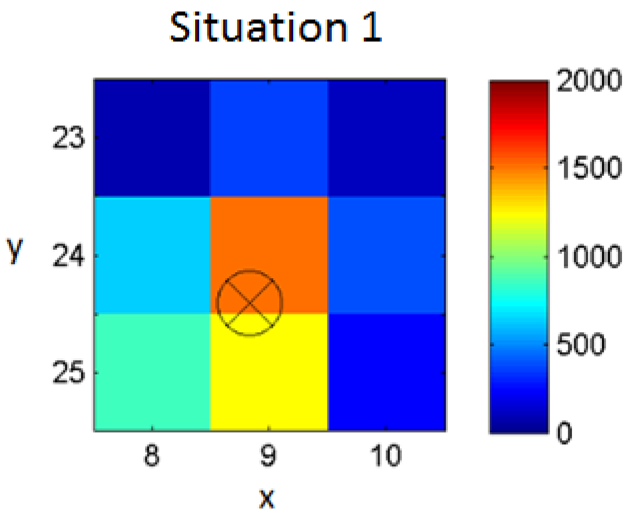
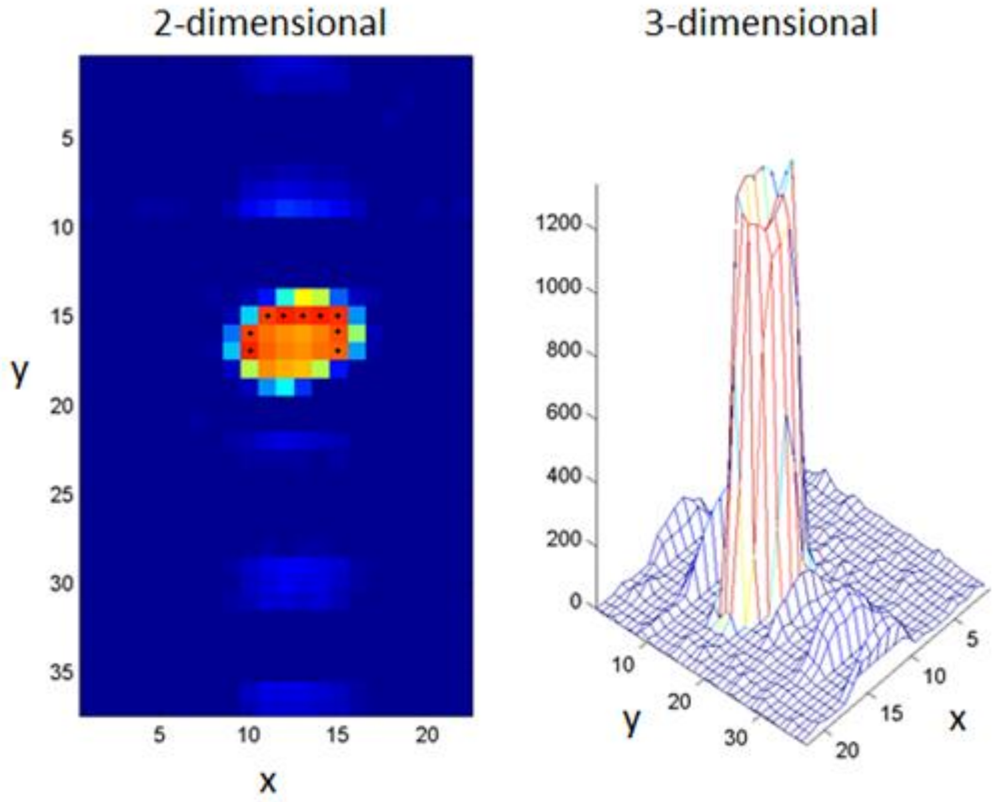


Figure 4.5: Fine positioning completed

The specialization project [8] was limited in scope by only dealing with small objects. The term *large objects* is introduced, defined as all objects larger than the small object. A large object from a real data set does not necessarily have F_{max}, at or around its center, and the next section will show how the small object approach is insufficient.

In general, the harder a display is pressed, the larger an area will be covered by the finger and the more nodes get increased intensity values. Figure 4.6 shows a real data set with a stationary thumb, which results in a large object. The node values are the average measurements of 100 data sets.



Figur 4.6: Average node values caused by a thumb

A visual inspection might suggest that the nodes covered by the thumb are:

$$9 < \hat{m}_1 < 16$$

$$14 < \hat{n}_1 < 19.$$

In the data set sequence this was taken from, all red and orange pixels marked with a black dot on figure 4.6 contain F_{max} in different data sets. It is desirable to be able to interpret large objects, and consequently introduce two additional algorithms: A data association algorithm and a fine positioning algorithm for large objects.

Region of interest

The task of the data association algorithm is to register all nodes that contain information about a specific object. These nodes constitute an area called the *region of interest* (RoI). Since there is no ideal way of separating nodes that contain object information from the nodes that do not contain object information a design choice has to be made; potentially missing valuable nodes, or potentially assigning nodes with no valuable information.

The RoI algorithm is designed such that the latter happens more often than the former. It assumes that the entire object is mapped by drawing straight and diagonal lines from the node with F_{max}. It sets the object lower area-limit to be 3x3, and finds the upper limit by expanding outwards in all directions, as illustrated on figure 4.7. The RoI algorithm compares the next node outwards to I_{thresh} from chapter 4.1.

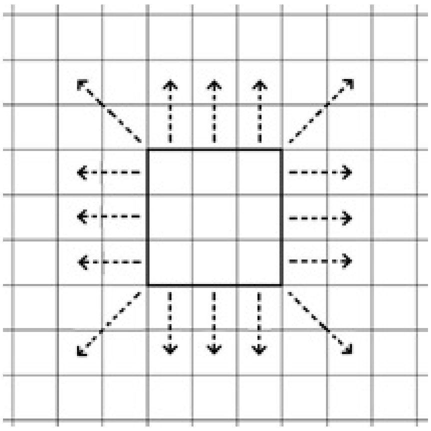


Figure 4.7: Touch input size estimation

When the algorithm is finished, it will report quadratic or rectangular area, with a minimum and maximum x- and y-node, which is the region of interest:

$$(\widehat{m}_{min}, \widehat{m}_{max}, \hat{n}_{min}, \hat{n}_{max})$$

However, the algorithm is not without disadvantages. It is assumed that all nodes above a certain threshold contains valuable information about the object in question. For low SNR displays this will lead to poor performance because the algorithm includes too many nodes. One solution is limiting the maximum size of the object, or create a noise-detector which adaptively adjusts the I_{thresh}, or a combination. Due to the limited time span of this project, this was not prioritized or followed further.

Additional fine-positioning algorithm

The last part of the *find 1 object* algorithm can now be performed. The center of mass technique, shown in chapter 2.4, is used as the second fine positioning algorithm. If the RoI algorithm reports a large object, a center of mass calculation is performed to estimate the center. If the RoI algorithm reports a small object, the least squares peak estimation is performed, as shown before.

Figure 4.8 illustrates a single data set from the data set sequence shown on figure 4.6, to show the result. To the left is the unfiltered and unaltered data set, while the right data set shows the parameters extracted by the scanner unit. The RoI nodes are copied and the node with F_{max}, $(\hat{m}, \hat{n}) = (15\,, 15)$ is marked with a black square. The estimated position, based on center of mass, is shown by the circle with cross. All nodes marked with a black dot is found by the RoI algorithm.

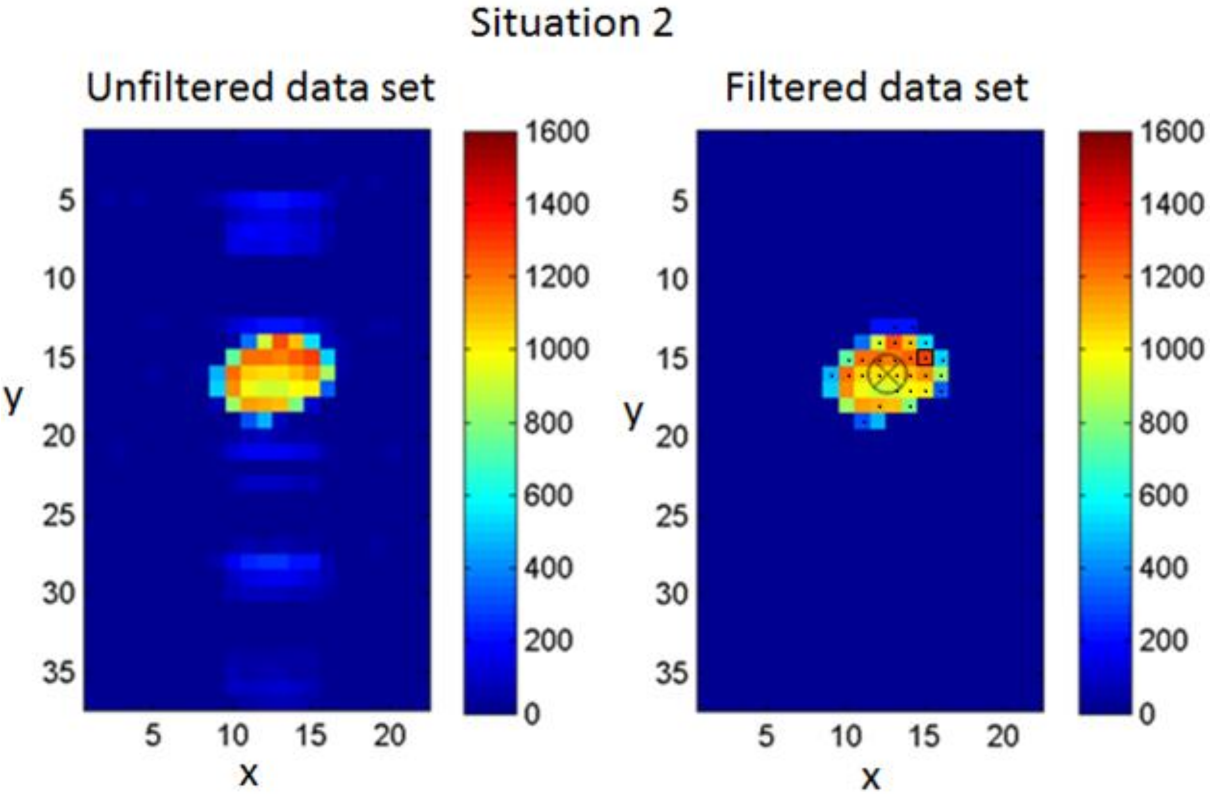


Figure 4.8: Algorithm demonstrated on a large object

Even though thumbs are commonly used on touch displays, it is assumed that they are not used in applications where accuracy is critical. For this reason, the rest of the thesis revolves around achieving maximum accuracy using small objects.

4.3 Find two objects

Finding two objects requires a slightly different approach than finding one object. This requires separation of between object one, noise and a potential second object. This is illustrated by using a new data set from a real display, shown on figure 4.9.

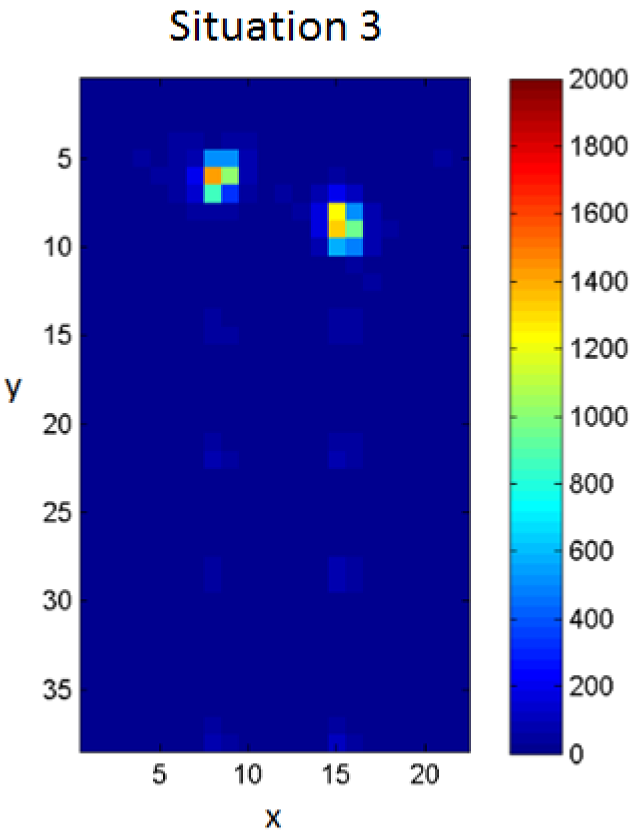


Figure 4.9: Successful separation and position estimation of objects

For many cases the second object can be found by using the algorithms already introduced. The process of finding two objects starts in the identical way as finding one object. When the RoI algorithm is completed, all these nodes are set to zero and a new threshold search is executed. If I_{thresh} is reached again, another execution of the coarse positioning- and RoI algorithm is performed. Finally, both inputs will run the fine-positioning algorithm, based on whether it is a large object or small object, as described in chapter 4.2. The result from the scanner for the data set in this section is shown by figure 4.10.

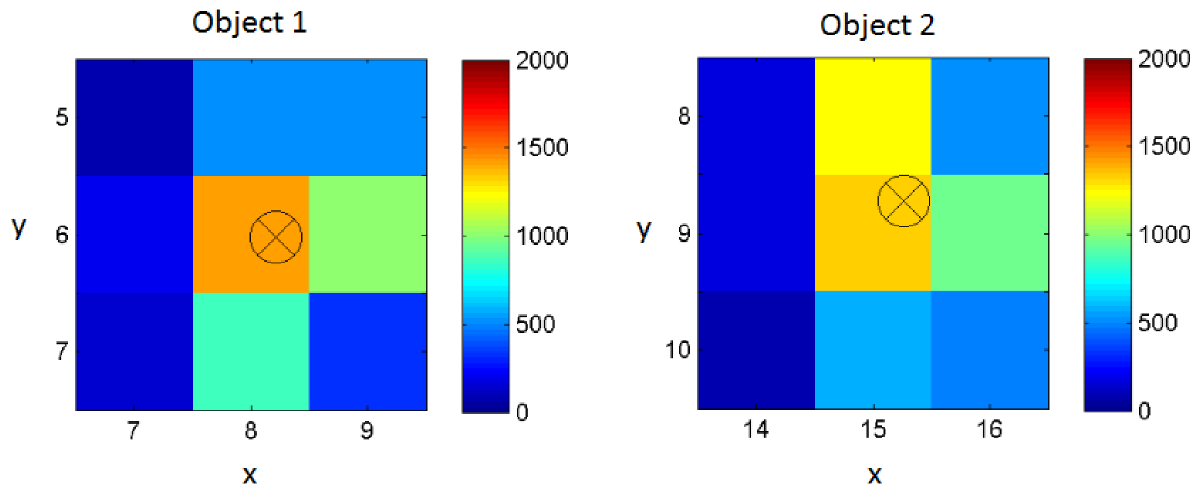


Figure 4.10: Find two objects result(fiks colorbar)

The coarse positioning and fine positioning result for this data set is:

$$(\hat{x}_1, \hat{y}_1) = (8.21\,, 6.02)$$

$$(\hat{x}_2, \hat{y}_2) = (15.266\,, 8.731).$$

The flow chart for the *Find 2* algorithm is presented on figure 4.11.

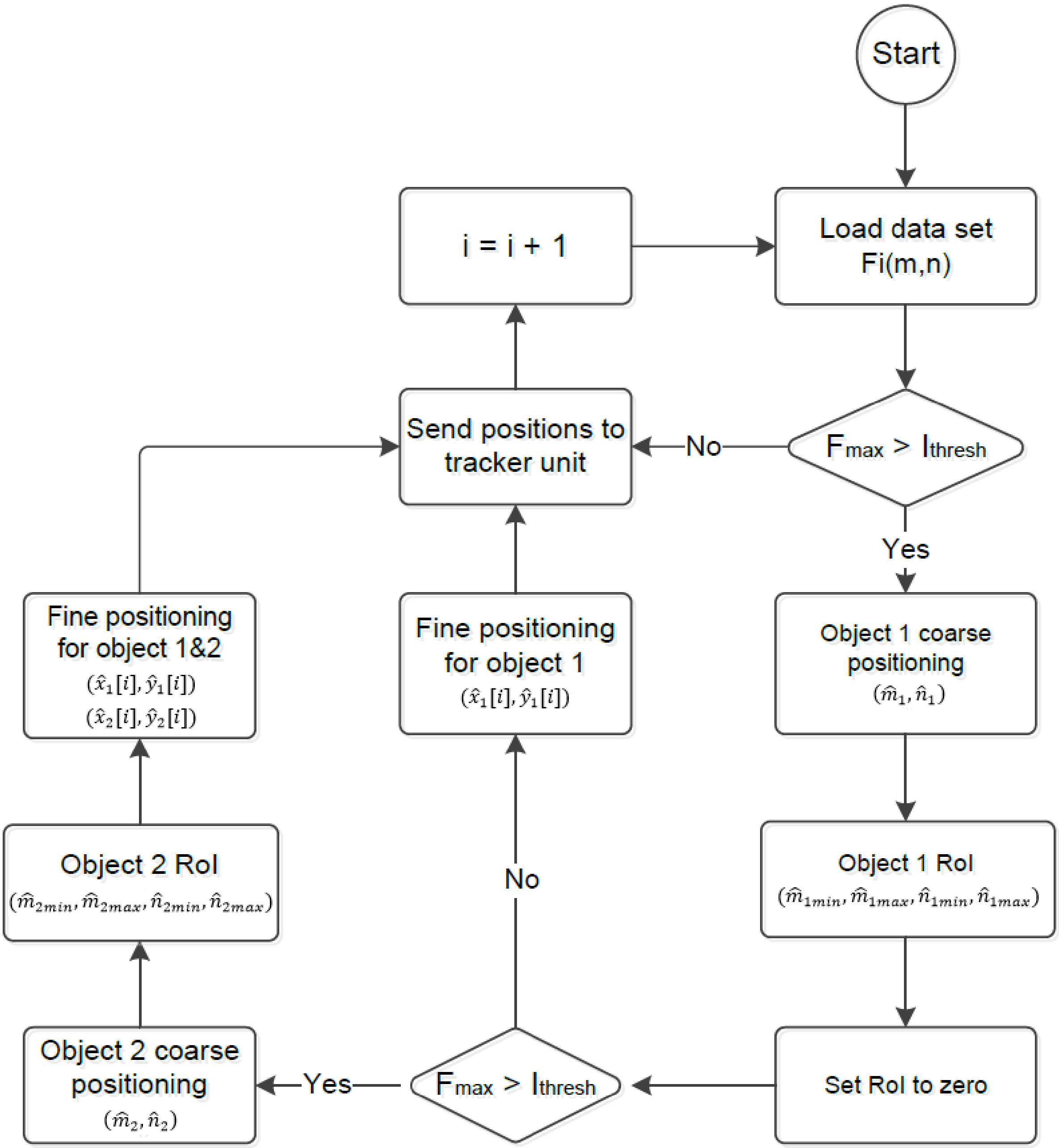

Start
Load data set Fi(m,n)
$F_{max} > I_{thresh}$
No
Yes
i = i + 1
Send positions to tracker unit
Object 1 coarse positioning $(\hat{m}_1, \hat{n}_1)$
Object 1 RoI $(\hat{m}_{1min}, \hat{m}_{1max}, \hat{n}_{1min}, \hat{n}_{1max})$
Set RoI to zero
$F_{max} > I_{thresh}$
No
Yes
Fine positioning for object 1 $(\hat{x}_1[i], \hat{y}_1[i])$
Object 2 coarse positioning $(\hat{m}_2, \hat{n}_2)$
Object 2 RoI $(\hat{m}_{2min}, \hat{m}_{2max}, \hat{n}_{2min}, \hat{n}_{2max})$
Fine positioning for object 1&2 $(\hat{x}_1[i], \hat{y}_1[i])$ $(\hat{x}_2[i], \hat{y}_2[i])$

Figure 4.11: Find 2-algorithm flow chart

4.4 Find two close objects

If the objects do not come too close to each other, and SNR is sufficient, successfully separating two objects is simple. However, if the distance is reduced enough, as illustrated by figure 4.12, the RoI algorithm is unable to separate the objects. When zooming in and out of a touch display, a common technique is to pinch the screen, and must be expected that this happens frequently. The solution to this problem is presented in this chapter.

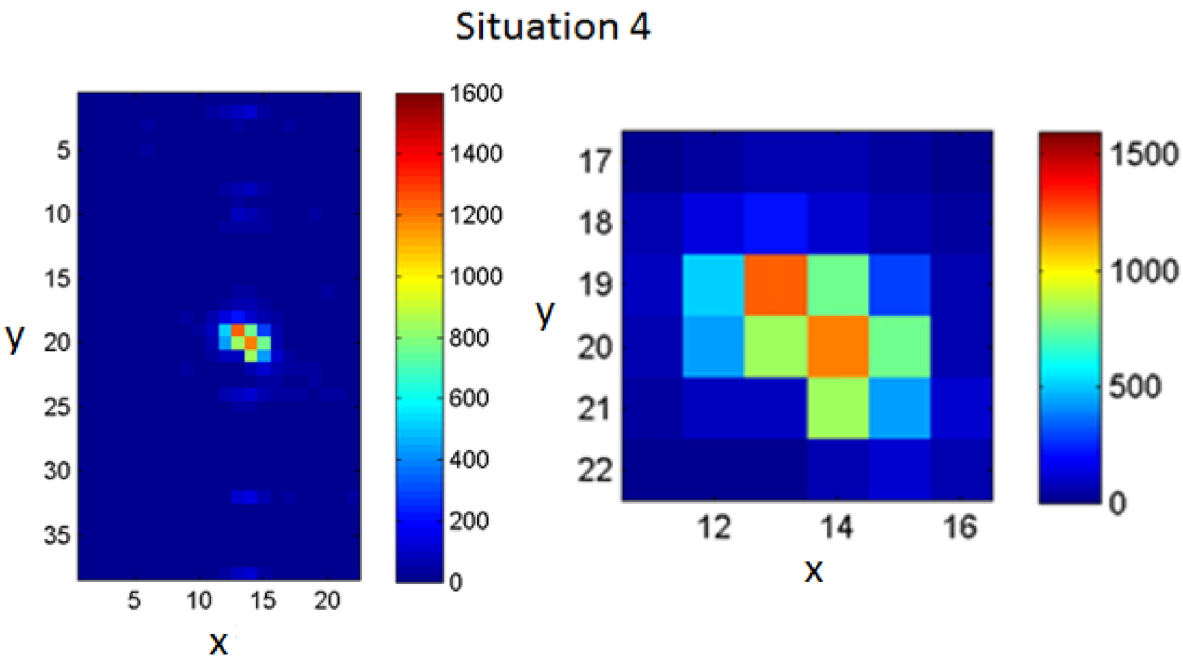


Figure 4.12: Two close objects

On figure 4.12, two objects have approached each other, and are assumed to be somewhere inside their respective orange pixel, such that $(\hat{m}_1, \hat{n}_1) = (13{,}19)$ and $(\hat{m}_2, \hat{n}_2) = (14{,}20)$. Two unfavorable situations may arise in this scenario. The first is that nodes are not assigned optimally to the objects. The result is that one object will lose important information, while the other over-estimates the subpixel position. This results in degraded accuracy for both objects and is shown by figure 4.13a.

The second situation that might occur is that the RoI-algorithm assigns all nodes above I_{thresh} to a single object. The objects “melt together” and are seen as one, and this result is shown by figure 4.13b. Both results are critical failures for a multi touch algorithm.

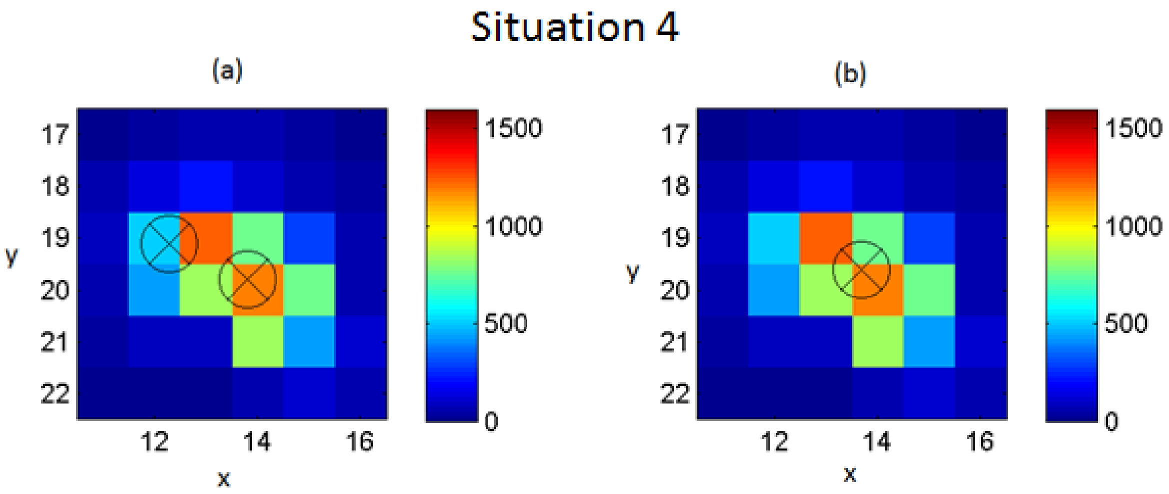
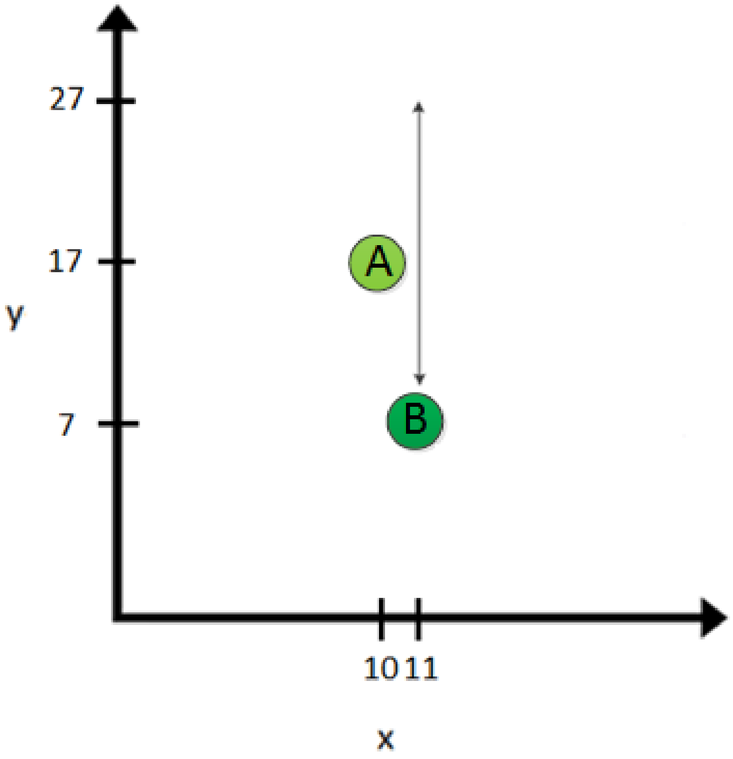


Figure 4.13: Below minimum distance: Failed situations

To quantify this inaccuracy a simulation is performed, as illustrated by figure 4.14. Here, two small objects are simulated, where object A is stationary and object B is moving object. The moving object passes the stationary object twice.



Figur 4.14: Simulated moving object and stationary object(skriv 1 og 2 istedenfor A/B)

The error distance ε, defined in formula (2.8), is measured as a function of distance. The intensity I is equal for both objects, and the simulation is performed in a 30dB SNR environment.

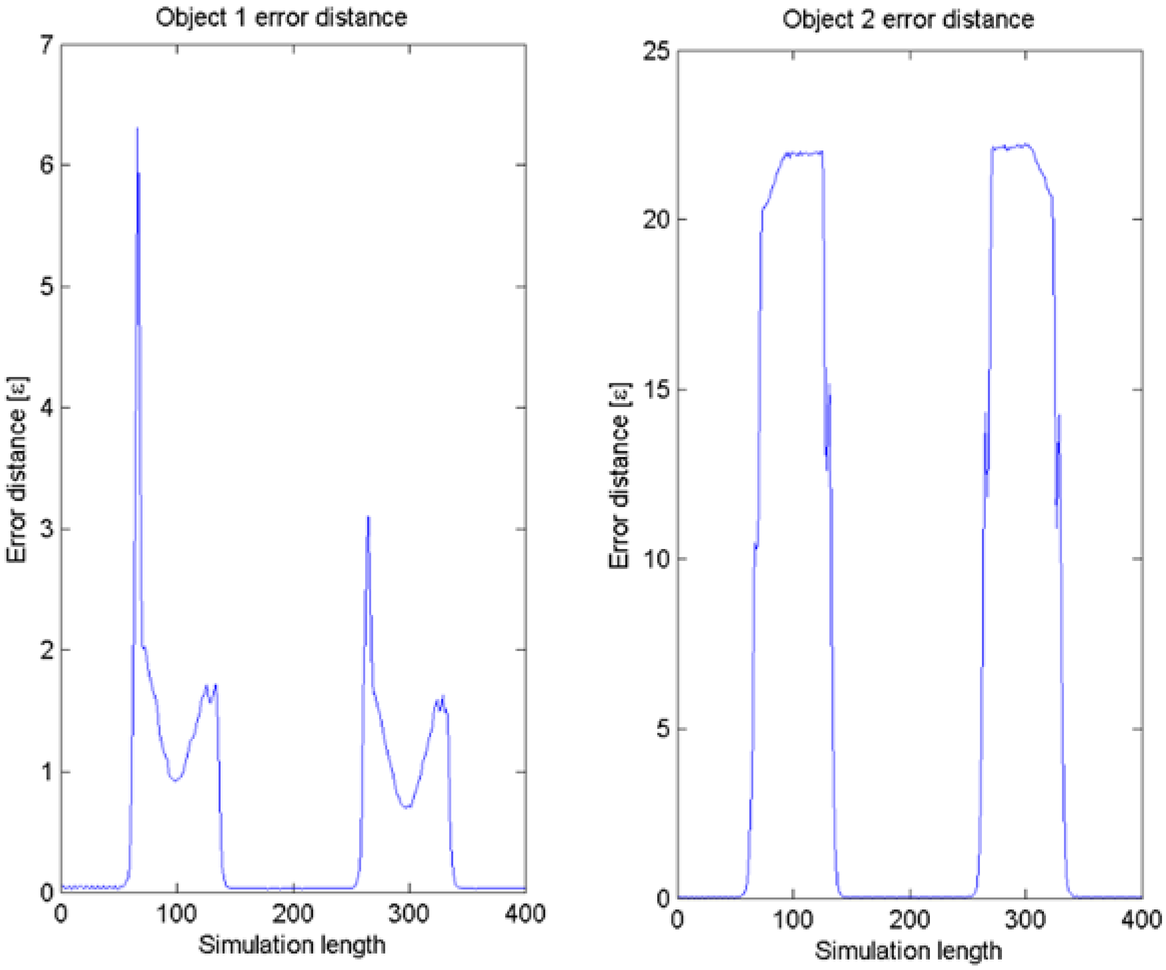
Object 1 is stationary at $(x_1, y_1) = (10\,, 17)$.

Object 2 is moving, starting from $(x_2[1], y_2[1]) = (11\,, 7)$.

Object 2 moves upwards and then downwards. It moves 0.1 nodes each data set, and turns after 200 simulations, at $(x_2[200], y_2[200]) = (11\,, 27)$.

The closest distance is found at the 100th and 300th simulation. In this position, we have $(x_2[100], y_2[100]) = (x_2[300], y_2[300]) = (11\,, 17)$. Object 1 and object 2 are now both on the same y-position, and the distance between the objects is $d = 1$.

A Monte Carl Simulation [10] with 1000 simulations is performed. The result is illustrated by figure 4.15, and ε is averaged for each position of the simulations.



Figur 4.15: Accuracy of simulated objects

At around the 55th step of the simulation, the error distance ε increases for both objects. The stationary object A is now assigned nodes belonging to object B. Object B eventually disappears and is set by default to $(\hat{x}_2, \hat{y}_2) = (0{,}0)$. At the 55th step of the simulation, the distance between the objects is

$$d \approx \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \approx \sqrt{4.5^2 + 1^2} \approx 4.6. \qquad (4.2)$$

This means that the scanner system, is unable to handle two inputs with the center closer than $d \approx 4.6$. This problem enforces the development of a new algorithm to separate close objects, which is presented next.

Close search algorithm

To be able to separate one large object from two close objects, the *Close search* algorithm is introduced. Note that this algorithm is executed by the scanner unit, but requested by the tracker unit, described in chapter 5.

The close search algorithm is activated if the distance d is reduced enough. This limit is set by the *minimum distance*, d_{min}. This is the distance between the estimated center of the objects where the RoI algorithm breaks down, by either assigning nodes incorrectly or falsely reporting one single object instead of two. Based on the result from equation (4.2), this distance should be set to above 4.6. d_{min} could also be a function of RoI, where one compares the closest node and not just the estimated center. This would require a more sophisticated algorithm and was not prioritized in this project.

Some assumptions was made in order to create a solution. The first is that the objects will never truly merge, shown by equation (4.3).

$$(x_1, y_1) \neq (x_2, y_2) \tag{4.3}$$

However, they can both share the same closest node, shown by equation (4.4) and (4.5).

$$(m_1, n_1) = (m_2, n_2) \tag{4.4}$$

$$(\hat{m}_1, \hat{n}_1) = (\hat{m}_2, \hat{n}_2) \tag{4.5}$$

The second assumption is that the objects are separable by the scanner, using the *Find two* algorithm, before they get closer to each other. In other words, we assume the RoI algorithm will successfully assign the nodes for each object, before the RoI algorithm fails. The third assumption is that the movement of two close objects is slow, such that the previously estimated position is useful for coarse positioning. Finally, it is assumed that objects below minimum distance are small objects.

Based on these assumptions a technical solution is developed. This involves changes to both the coarse-positioning algorithm and RoI algorithm. Instead of finding the node with F_{max}, the coarse positioning is replaced with receiving $(\hat{m}_1, \hat{n}_1)$ and $(\hat{m}_2, \hat{n}_2)$ from the tracker unit, based on the previous estimated position. The pseudo code is presented on the next page.

Pre-set main node pseudo code:

```
$for\ F_i(m,n)$
    $if\ d < d_{min}$
        $\hat{m}_1[i] = round(\hat{x}_1[i-1])$
        $\hat{n}_1[i] = round(\hat{y}_1[i-1])$
        $\hat{m}_2[i] = round(\hat{x}_2[i-1])$
        $\hat{n}_2[i] = round(\hat{y}_2[i-1])$
    $end$
$end$
```

This will occasionally result in an equal closest node for object 1 and object 2, as described by equation (4.5). A solution for separating them in this situation was not found in this project. Instead, this was solved by minimizing the likelihood of this scenario happening. The process starts by identifying the mutual nodes. Figure 4.16 shows the same data set as figure 4.12, where area of each of the two small objects are highlighted with a black frame respectively. Some of the nodes are mutual, highlighted by the hatched area.

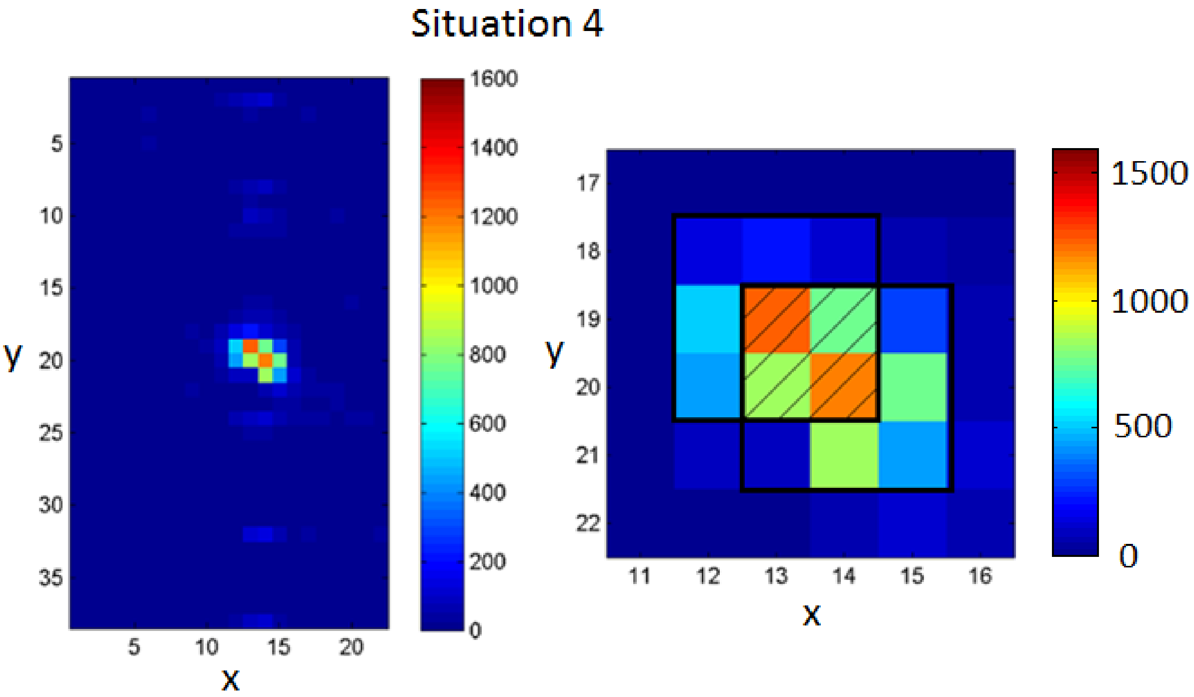


Figure 4.16: Mutual nodes on two objects

The two small objects are copied into their own 3x3 matrix, where the intensity value of the mutual nodes, except main node, is halved. Figure 4.17 illustrates the node values before and after adjustment, and this is also described by the pseudo code below.

Adjust node values pseudo code:

```
$if\ node\ of\ object\ 1\ (except\ main\ node) = object\ 2\ node$
    $node\ is\ halfed\ in\ value$
$end$

$if\ node\ of\ object\ 2\ (except\ main\ node) = object\ 1\ node$
    $node\ is\ halfed\ in\ value$
$end$
```

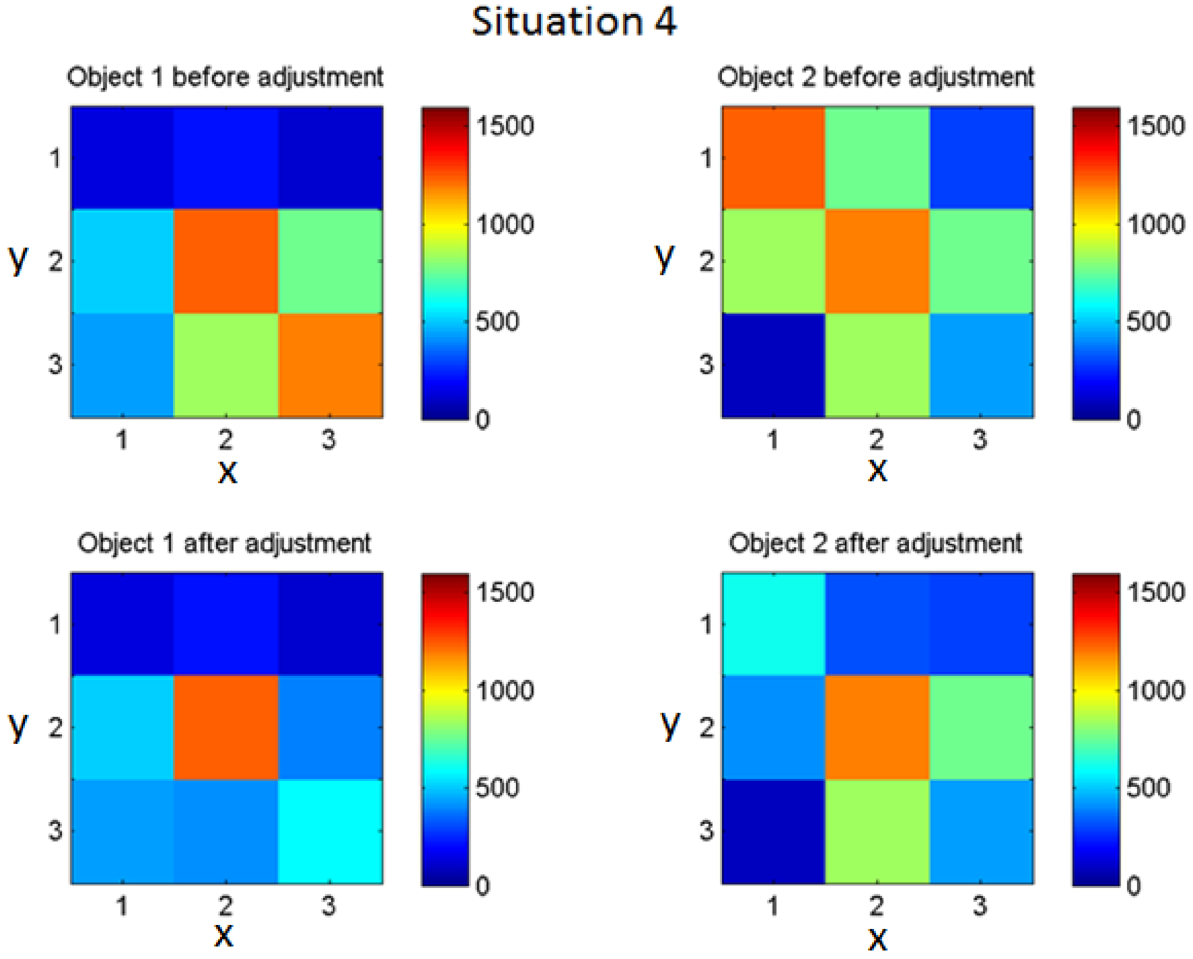


Figure 4.17: Object 1 and object 2

This algorithm removed occurrence of both objects getting same main node in all the real data sets. The scanner unit eventually performs the fine positioning for both objects, as small objects. Figure 4.18 demonstrates the new result on the previous data set. Both objects are successfully separated and identified with increased accuracy. Object 1, located at $(\hat{m}_1, \hat{n}_1) = (13,19)$, is arguably estimated too far down. This is likely a result of the blunt technique of halving the nodes.

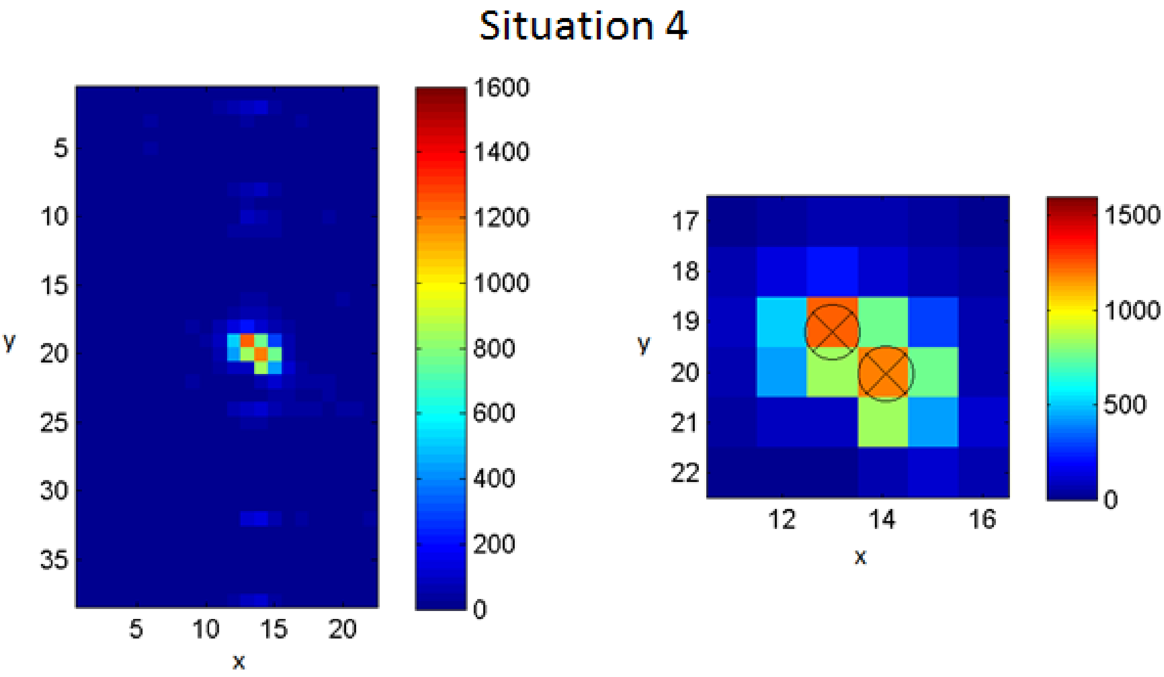


Figure 4.18: Successful separation of two close objects

A new simulation is performed to quantify the improvement of the close search-algorithm. Another Monte Carlo Simulation with the equal parameters is executed, and figure 4.19 illustrates the new results.

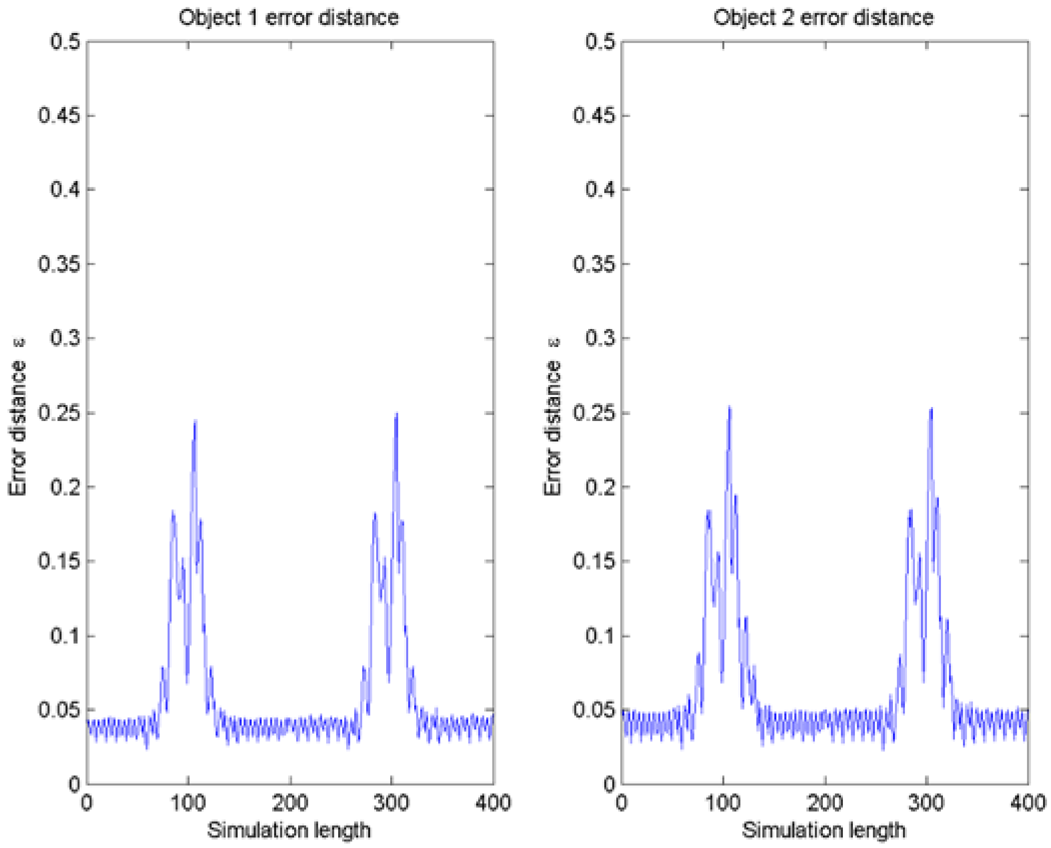


Figure 4.18: Accuracy of close search(y2 på begge x-akser, fiks [e])

Both objects are now successfully identified, with subpixel accuracy for the whole simulation, by controlling the distance between objects and looking at the previously estimated position.

The error distance ε still increases as the objects get closer, but this error is significantly reduced. The increased inaccuracy is, like in the real data set, likely caused by manipulating the node values. Halving the node values does not fairly distribute the values between the objects and this reduces the quality of the information in the fine-positioning estimation. A sophisticated algorithm that more fairly distributes the values might improve the accuracy.

The chapter is ended with the flow chart for the close search-algorithm (figure 4.19).

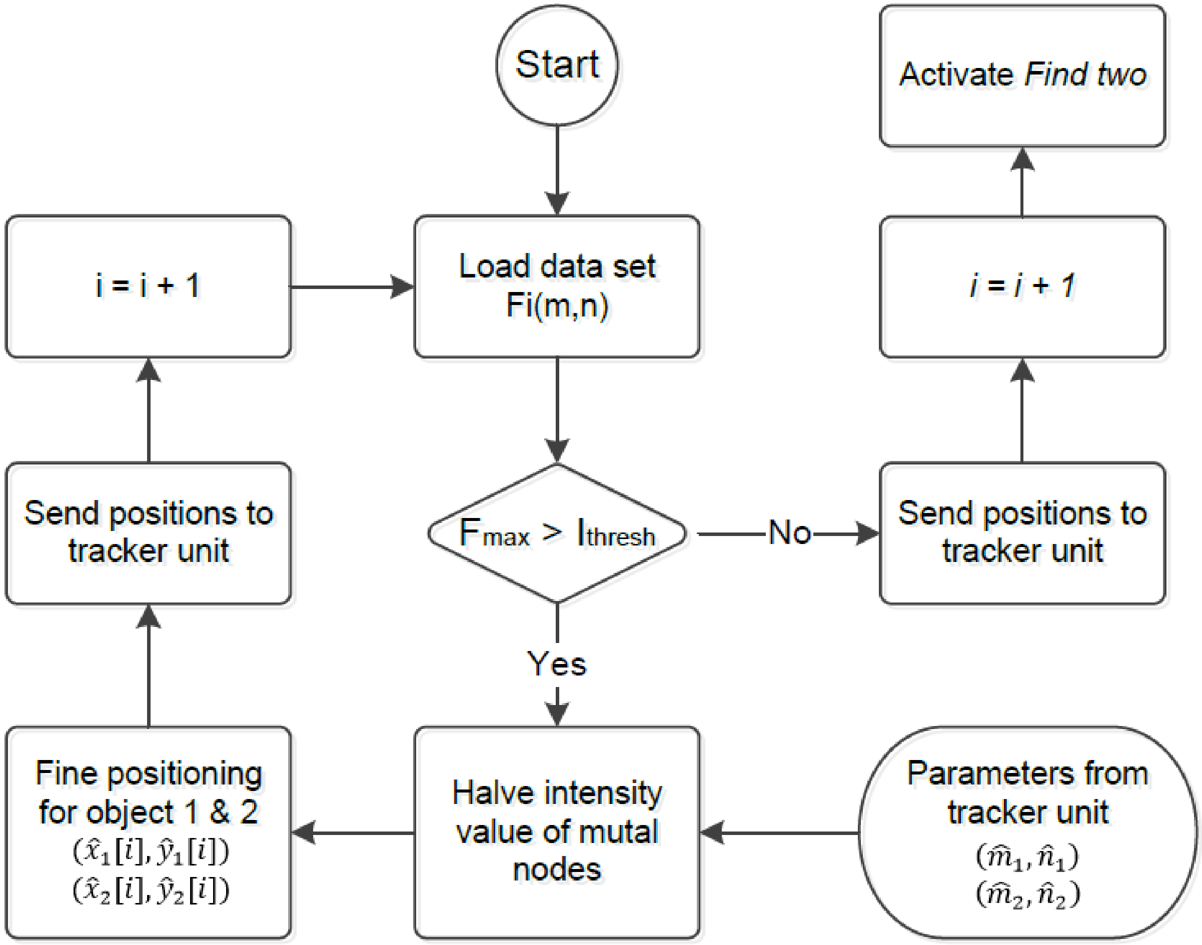

Start
Activate *Find two*
i = i + 1
Load data set
Fi(m,n)
i = *i* + *1*
Send positions to tracker unit
$F_{max} > I_{thresh}$
No
Send positions to tracker unit
Yes
Fine positioning for object 1 & 2
$(\hat{x}_1[i], \hat{y}_1[i])$
$(\hat{x}_2[i], \hat{y}_2[i])$
Halve intensity value of mutal nodes
Parameters from tracker unit
$(\widehat{m}_1, \hat{n}_1)$
$(\widehat{m}_2, \hat{n}_2)$

Figure: Find two close objects-algorithm flow chart

[This page is intentionally left blank]

5 Tracker unit

The data sets have been filtered into the positions of two objects, by the scanner unit. These positions are now sent to the tracker unit, not to be confused with the Tracker system, which is the complete system.

Before the object positions are sent to the application level, they must be viewed collectively instead of individually. Figure 5.1 shows a simplified model of the tracker unit.

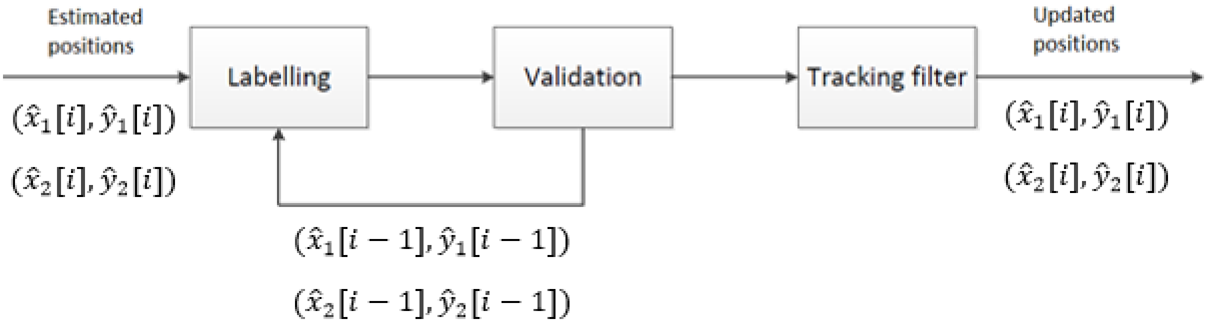

Estimated positions
$(\hat{x}_1[i], \hat{y}_1[i])$
$(\hat{x}_2[i], \hat{y}_2[i])$
Labelling
Validation
Tracking filter
Updated positions
$(\hat{x}_1[i], \hat{y}_1[i])$
$(\hat{x}_2[i], \hat{y}_2[i])$
$(\hat{x}_1[i-1], \hat{y}_1[i-1])$
$(\hat{x}_2[i-1], \hat{y}_2[i-1])$

Figure 5.1: Tracker unit flow chart(fiks tekst)

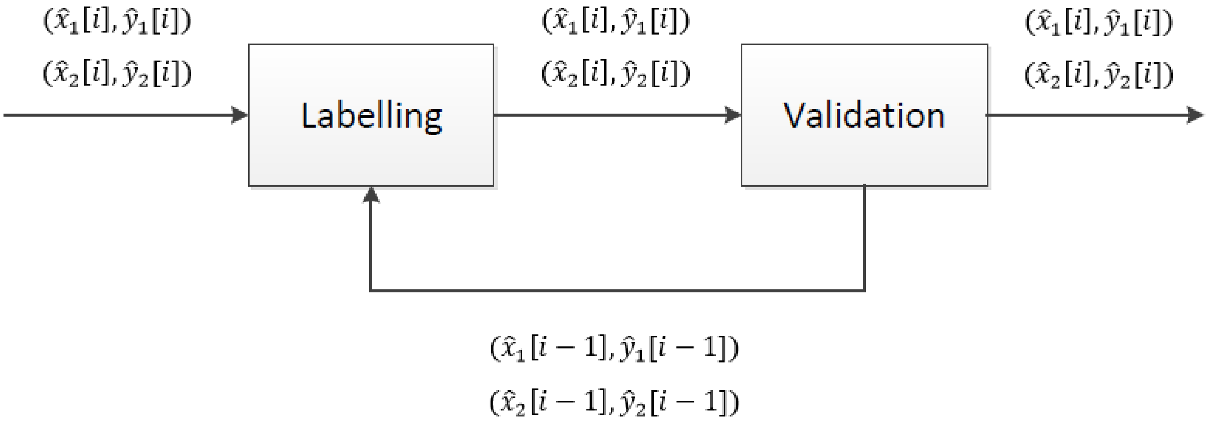
The tracker unit of this project is the final module before the application level and it has three main objectives: The first is a two-step process called *Identification*, consisting of *labelling* and *validation*. Here, the current reported positions, $(\hat{x}_1[i], \hat{y}_1[i])$ and $(\hat{x}_2[i], \hat{y}_2[i])$, are connected to the previous positions, $(\hat{x}_1[i-1], \hat{y}_1[i-1])$ and $(\hat{x}_2[i-1], \hat{y}_2[i-1])$. In addition they are verified that they are not false objects caused by noise. The tracker unit memory has size one, meaning it only remembers the previous positions.

The second task is sending the validated positions through a tracking filter, to achieve increased accuracy and smoothen the trajectory through the tracking filter.

Lastly, the distance between the objects are calculated and, if $d < d_{min}$, the tracker unit requests the scanner unit is to execute the close search-algorithm.

5.1 Identification

The tracker unit begins by identifying the inputs in a two-step process called *Identification*. Figure 5.2 shows the identification process, which consists of *labelling* and *validation.* Both of these processes will be presented in this chapter, starting with *labelling*.



$(\hat{x}_1[i], \hat{y}_1[i])$
$(\hat{x}_2[i], \hat{y}_2[i])$
Labelling
$(\hat{x}_1[i], \hat{y}_1[i])$
$(\hat{x}_2[i], \hat{y}_2[i])$
Validation
$(\hat{x}_1[i], \hat{y}_1[i])$
$(\hat{x}_2[i], \hat{y}_2[i])$
$(\hat{x}_1[i-1], \hat{y}_1[i-1])$
$(\hat{x}_2[i-1], \hat{y}_2[i-1])$

Figure 5.2: Identification process

This is where the tracker system has its memory, which is of length *one*, meaning that the current positions are only compared to the previous positions. A consequence of this is that if a sequence of data sets contains a corrupted data set, the identification will not be able to connect observations from previous observations. This is a feature that could be added later, or handled by the application level.

Labelling

The tracker unit receives two positions for each data set, but it does not automatically recognize which new position belongs to previously used positions. This happens because the scanner unit always labels the object with F_{max} as object 1, and since object 1 and object 2 vary in intensity the objects numbers might switch. For this reason an effective way to keep track of the identity of the reported objects is needed. This is shown using the fictional situation on figure 5.3.

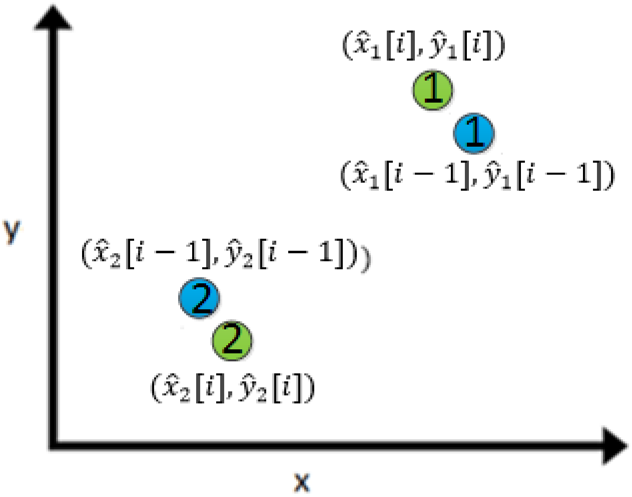

$(\hat{x}_1[i], \hat{y}_1[i])$
1
1
$(\hat{x}_1[i-1], \hat{y}_1[i-1])$
y
$(\hat{x}_2[i-1], \hat{y}_2[i-1]))$
2
2
$(\hat{x}_2[i], \hat{y}_2[i])$
x

Figure 5.3: Current positions (green) and previous positions (blue) recieved from scanner

The labelling process starts by calculating the distance between the current estimated positions and the previous positions. Four distances are produced, illustrated by figure 5.4, using the formula below.

Formula: Label objects, part 1

$$a = \sqrt{(\hat{x}_1[i] - \hat{x}_1[i-1])^2 + (\hat{y}_1[i] - \hat{y}_1[i-1])^2}$$

$$b = \sqrt{(\hat{x}_2[i] - \hat{x}_2[i-1])^2 + (\hat{y}_2[i] - \hat{y}_2[i-1])^2}$$

$$c = \sqrt{(\hat{x}_1[i] - \hat{x}_1[i-1])^2 + (\hat{y}_2[i] - \hat{y}_2[i-1])^2}$$

$$d = \sqrt{(\hat{x}_2[i] - \hat{x}_2[i-1])^2 + (\hat{y}_1[i] - \hat{y}_1[i-1])^2}$$

The distances that *a* and *b* represent, assumes that the previously reported object 1 and object 2 is the same as the current reported object 1 and object 2. The distance *c* and *d*, assumes that the previously reported object 1 is now reported as object 2, and vice versa.

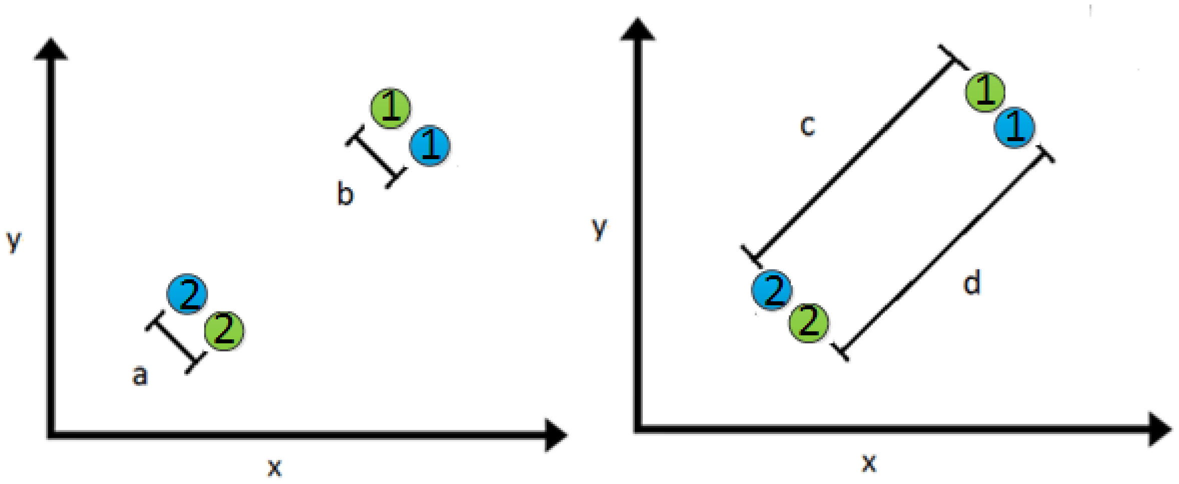


Figure 5.4: Labelling distances calculated

Next, distance *a* and *b* are added together, and distance *c* and *d* are added together. The total length of these combined are compared, and the current objects are labelled according to the formula below.

Formula: Label objects, part 2

$$\hat{x}_1[i] = \begin{cases} \hat{x}_1[i]\ for\ a+b<c+d \\ \hat{x}_2[i]\ for\ a+b>c+d \end{cases}$$

$$\hat{y}_1[i] = \begin{cases} \hat{y}_1[i]\ for\ a+b<c+d \\ \hat{y}_2[i]\ for\ a+b>c+d \end{cases}$$

$$\hat{x}_2[i] = \begin{cases} \hat{x}_2[i]\ for\ a+b<c+d \\ \hat{x}_1[i]\ for\ a+b>c+d \end{cases}$$

$$\hat{y}_2[i] = \begin{cases} \hat{y}_2[i]\ for\ a+b<c+d \\ \hat{y}_1[i]\ for\ a+b>c+d \end{cases}$$

Figure 5.4 shows that the initial labelling was correct, where clearly $(a+b<c+d)$. The labelling is now complete, and the next process is validation.

Validation

Chapter 4.1 showed that false objects can occur, depending on the size of the SNR. For this reason the reported objects must be validated to be true objects, and this is also done by comparing the positions to the previous reported position.

The false objects are assumed to be caused by random noise, evenly distributed on the entire touch display. By defining an area around each previously estimated positions, and comparing this to the current estimated positions, false objects can be separated from true objects. The validation distance, d_{val}, is introduced. This distance defines a circle around each object reported position, illustrated on figure 5.4. The estimated position is marked with x, and the grey area defines the area.

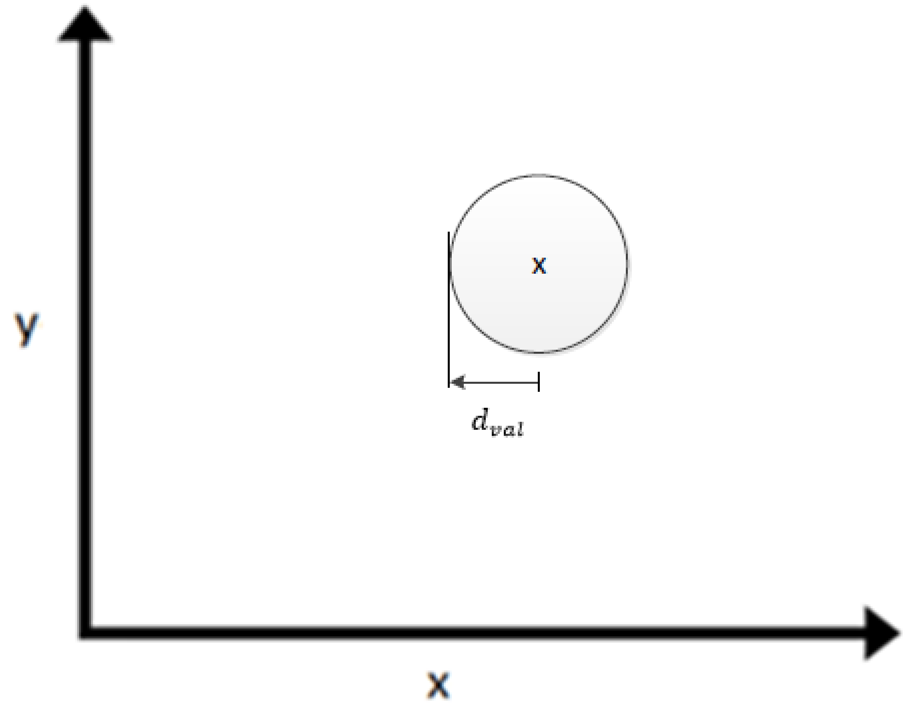


Figure 5.4: Validation distance

If the current estimated positions are assigned to a previous observation outside of this circle, it is dropped assumed to be a false objects. The length of d_{val} will depend on the sampling rate of the screen and assumed speed of object. For fast movement on low sampling rate d_{val} must be larger than for faster sampling rates.

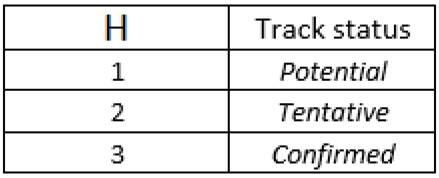
Depending on the number of consecutive observations, the status of an object has three stages: Potential, tentative and confirmed. The first time an object of a non-zero position is reported, it is set as potential. If it is reported a second consecutive time within the circle area it is set as tentative. Finally, the object is confirmed if it is observed three consecutive times. If it is not reported three consecutive times, the process is reset. If a data set is corrupted, the process also restarts. This is described below with pseudo-code and in table 5.1. The counter used for this purpose is denoted H.

Target validation for object 1 pseudo code:

```
$if\ distance\ between\ previous\ and\ currrent\ observation < d_{val}$
        $H_{object\ 1} = H_{object\ 1} + 1;$
$else$
        $object\_1 = false;$
        $H_{object\ 1} = 0;$
$end$

$if\ H_{object\ 1} == 3$
        $object\_1 = true;$
$end$
```

Table 5.1: Object validation



H	Track status
1	*Potential*
2	*Tentative*
3	*Confirmed*

5.3 Tracking filter

The tracking filter is the last chance to adjust the validated positions (figure 5.4). The positions must be sent to the application level in real time, but it is possible to implement a tracking filter and still be within the real-time requirement regarding output delay. This is done to further increase the accuracy, or at least provide a smoother trajectory.

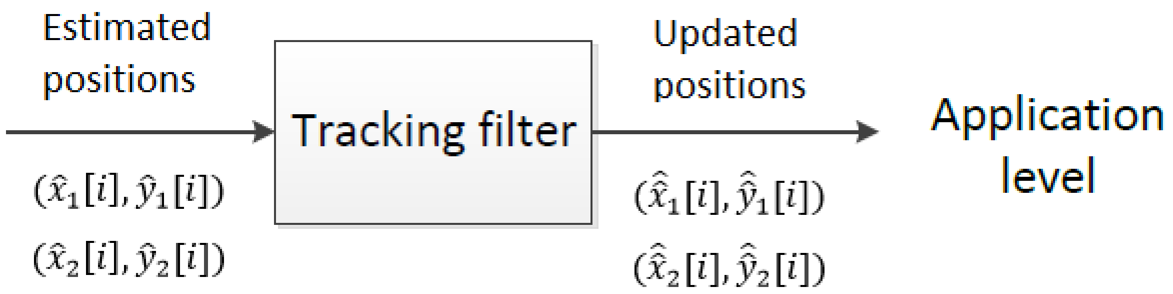

Estimated positions
$(\hat{x}_1[i], \hat{y}_1[i])$
$(\hat{x}_2[i], \hat{y}_2[i])$
Tracking filter
Updated positions
$(\hat{\hat{x}}_1[i], \hat{\hat{y}}_1[i])$
$(\hat{\hat{x}}_2[i], \hat{\hat{y}}_2[i])$
Application level

Figure 5.5: Tracking filter

The chosen filter is for this purpose is the Kalman filter, which is named after Rudolf E. Kálmán and developed in the 1950's. It is a recurseive state estimator that finds the statistically optimal estimate of noisy Gaussian input data if the noise model is exact[10]. It uses a combination of both current measurements, $(\hat{x}_0, \hat{y}_0)$, and a prediction, $(\bar{x}_0, \bar{y}_0)$, to turn the position into a better estimate, $(\hat{\hat{x}}_0, \hat{\hat{y}}_0)$, shown by figure 5.4.

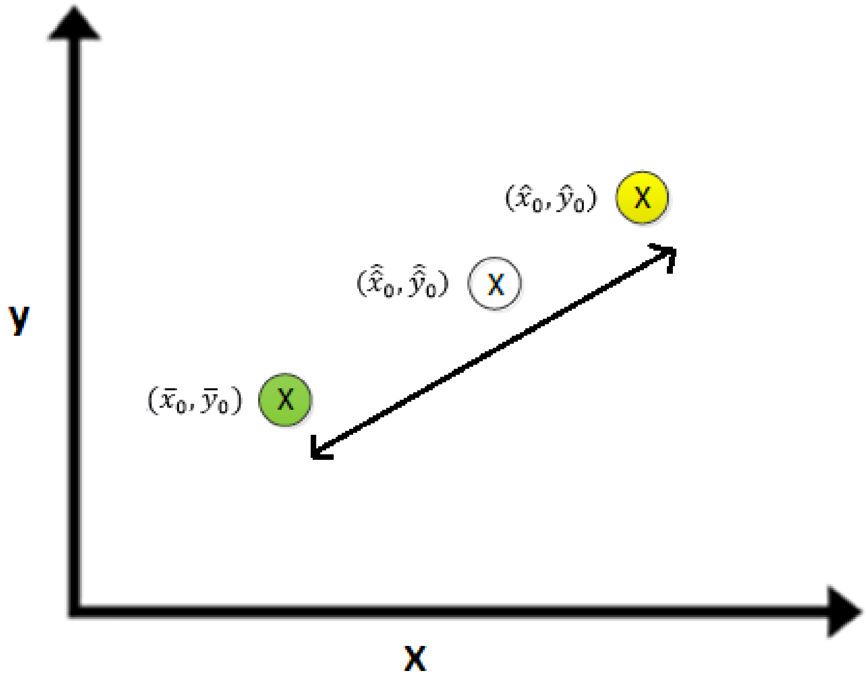


Figure 5.6: Principal behavior of the Kalman filter, with prediction (green), estimated position (yellow) and the resulting updated position(white)

The data stream has a fixed sampling rate, which allows the Standard Kalman Filter [10] to be used. A flow chart of this filter is presented below on figure 5.5. The reader may observe that the only input is the estimated x-coordinate, and that the only output is a filtered x-coordinate.

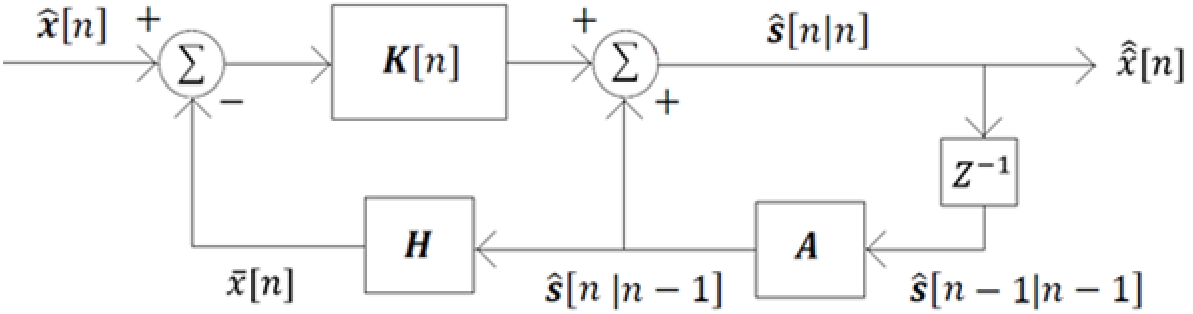


Figure 5.7: Kalman filter model

The filter is described for the x-component of the object 1 position. An equal filter is applied for the y-component of object 1, as well as the x- and y-component of object 2. The result is still the same as for a filter using two components.

The Kalman filter is based on estimating a state, denoted $\boldsymbol{s}[n|n]$. This is a two-dimensional vector that contains the object parameters. We assume acceleration is close to zero, and model the state using only the position and velocity. The state is defined as:

$$\boldsymbol{s}[n|n] = \begin{bmatrix} x[n] \\ v_x[n] \end{bmatrix}.$$

Where $x[n]$ is the current updated position that we want to estimate, and $v_x[n]$ is the velocity. When this filter is applied to the tracker system x-component, the following state estimate is produced:

$$\hat{\boldsymbol{s}}[n|n] = \begin{bmatrix} \hat{\hat{x}}[n] \\ \hat{v}_x[n] \end{bmatrix}.$$

The state vector, and the rest of the Kalman filter, evolves in time according to the following formulas [10]:

$$Prediction\text{: } \hat{\boldsymbol{s}}[n|n-1] = \boldsymbol{A}\hat{\boldsymbol{s}}[n-1|n-1]$$

$$\text{Minimum Prediction MSE matrix: } \boldsymbol{M}[n|n-1] = \boldsymbol{AM}[n-1|n-1]\boldsymbol{A}^T + \boldsymbol{Q}$$

$$Kalman\ Gain\text{: } \boldsymbol{K}[n] = \boldsymbol{M}[n|n-1]\boldsymbol{H}^T(\boldsymbol{HM}[n|n-1]\boldsymbol{H}^T + C)^{-1}$$

$$Correction\text{: } \hat{\boldsymbol{s}}[n|n] = \hat{\boldsymbol{s}}[n|n-1] + \boldsymbol{K}[n](\hat{x}[n] - \boldsymbol{H}\hat{\boldsymbol{s}}[n|n-1])$$

$$Minimum\ MSE\ matrix\text{: } \boldsymbol{M}[n|n] = (\boldsymbol{I} - \boldsymbol{K}[n]\boldsymbol{H})\boldsymbol{M}[n|n-1]$$

Next sample/repeat…

It starts with the *Prediction* of the estimated next state, given by

$$\hat{\boldsymbol{s}}[n|n-1] = \boldsymbol{A}\hat{\boldsymbol{s}}[n-1|n-1],$$

where **A** is a transition matrix, and $\hat{\boldsymbol{s}}[n|n-1]$ estimates (predicts) the next position $\bar{x}[n]$, using the previous estimated position and velocity.

$$\boldsymbol{A} = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix}$$

$$\hat{\boldsymbol{s}}[n|n-1] = \begin{bmatrix} \bar{x}[n] \\ \hat{v}_x[n-1] \end{bmatrix} = \begin{bmatrix} \hat{\hat{x}}[n-1] + \Delta T\hat{v}_x[n-1] \\ \hat{v}_x[n-1] \end{bmatrix}$$

The matrix calculated next is the error covariance matrix of the state estimate $\mathbf{M}[n|n-1]$, named the *Minimum Prediction MSE,* defined:

$$\boldsymbol{M}[n|n-1] = E[(\boldsymbol{s}[n] - \hat{\boldsymbol{s}}[n|n-1])(\boldsymbol{s}[n] - \hat{\boldsymbol{s}}[n|n-1])^T]$$

It is calculated by the formula:

$$\boldsymbol{M}[n|n-1] = \boldsymbol{AM}[n-1|n-1]\boldsymbol{A}^T + \boldsymbol{Q} \tag{5.1}$$

$$= \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \Delta T & 1 \end{bmatrix} + \begin{bmatrix} \sigma_{\hat{\hat{x}}}^2 & 0 \\ 0 & \sigma_{\hat{v}_x}^2 \end{bmatrix}.$$

Q represents the *process noise* of both the predicted speed and position. It can be constant or adaptive, but in this case it is fixed.

Next is the *Kalman Gain*, which is a factor that determines whether the filter relies more on the estimate, $\hat{x}[n]$, or the prediction, $\bar{x}[n]$. It is defined:

$$\boldsymbol{K}[n] = \begin{bmatrix} K_x[n] \\ K_{v_x}[n] \end{bmatrix}.$$

High values of **K** equals trusting the estimation more than the prediction, where:

$$0 \leq K_x[n] \leq 1$$

$$0 \leq K_{v_x}[n] \leq 1.$$

It is calculated by the formula:

$$\boldsymbol{K}[n] = \boldsymbol{M}[n|n-1]\boldsymbol{H}^T(\boldsymbol{HM}[n|n-1]\boldsymbol{H}^T + C[n])^{-1}. \tag{5.2}$$

Where **H** is an observation matrix and the scalar C is the estimation variance, or the quality of the estimation received from the *Scanner unit*. This is based on both hardware and algorithm inaccuracy and is connected to the error distance ε.

$$\boldsymbol{H} = [1\ 0]$$

$$C = \sigma_{x_{est}}^2$$

Finally, the current state is calculated in the *Correction*, where we find current updated estimate, $\hat{\hat{x}}[n]$, and the velocity $\hat{v}_x[n]$:

$$\hat{\boldsymbol{s}}[n|n] = \hat{\boldsymbol{s}}[n|n-1] + \boldsymbol{K}[n](\hat{x}[n] - \boldsymbol{H}\hat{\boldsymbol{s}}[n|n-1])$$

$$\hat{\boldsymbol{s}}[n|n] = \begin{bmatrix} \hat{\hat{x}}[n] \\ \hat{v}_x[n] \end{bmatrix}.$$

The current state is vital for the estimation of the *next* state. The filter ends the loop by calculating a second covariance matrix $\boldsymbol{M}[n|n]$.the minimum MSE matrix, defined:

$$\boldsymbol{M}[n|n] = E[(\boldsymbol{s}[n] - \hat{\boldsymbol{s}}[n|n])(\boldsymbol{s}[n] - \hat{\boldsymbol{s}}[n|n])^T],$$

using the formula:

$$\boldsymbol{M}[n|n] = (\boldsymbol{I} - \boldsymbol{K}[n]\boldsymbol{H})\boldsymbol{M}[n|n-1].$$

Initial values

Before the filter is activated it needs some initial values. The initial state vector is preset with zero velocity and the first *validated* estimate, as this is the most accurate position available:

$$\hat{\boldsymbol{s}}[0|0] = \begin{bmatrix} \hat{x}[1] \\ 0 \end{bmatrix}.$$

The minimum MSE matrix is set to zero, such that the first updated position mainly leans towards the estimate, $\hat{x}[1]$.

$$\boldsymbol{M}[0|0] = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

[This page is intentionally left blank]

6 Results

The previous chapter have shown how the main algorithms of the scanner unit and tracker unit works, and the results it has provided. In this chapter, the result of the complete Tracker system is presented. The results contains performance graphs of the labelled and validated positions of one or two objects. All of the results compare the estimate positions, $(\hat{x}_0, \hat{y}_0)$, from the scanner unit, with the filtered estimate, $(\hat{\hat{x}}_0, \hat{\hat{y}}_0)$, from the tracker unit. Four situations will be presented to highlight the performance, where three are simulations. The four situations are:

1. Real data set: Tracking one object from a real data set
2. Simulation: Tracking a stationary small object
3. Simulation: Tracking a small object that moves along the y-axis
4. Simulation: Tracking two close objects

Prior to testing, the Kalman filter was fine tuned to provide the best results on the real data sets. The parameters were then fixed, to better highlight the strengths and weaknesses of the Tracker system.

A note on accuracy may be added: While accuracy is imperative, it is important to remember that touch displays are operated by humans. For some applications, like typing, accuracy might be more important than smoothness. For other applications, like drawing, it could be more important that the motion is smooth. The human perception is not the scope of this project, but it is important to keep in mind.

6.1 Real data set: Subpixel resolution

The first situation shows the resulted tracking of a real object, using the tracking values of the x-component, shown by figure 6.1. While not quantifiable, the intention of this result is to show how much the resolution of the sensor is improved. The display used has a 50Hz data set sampling rate, which makes this simulation approximately 1.7 seconds.

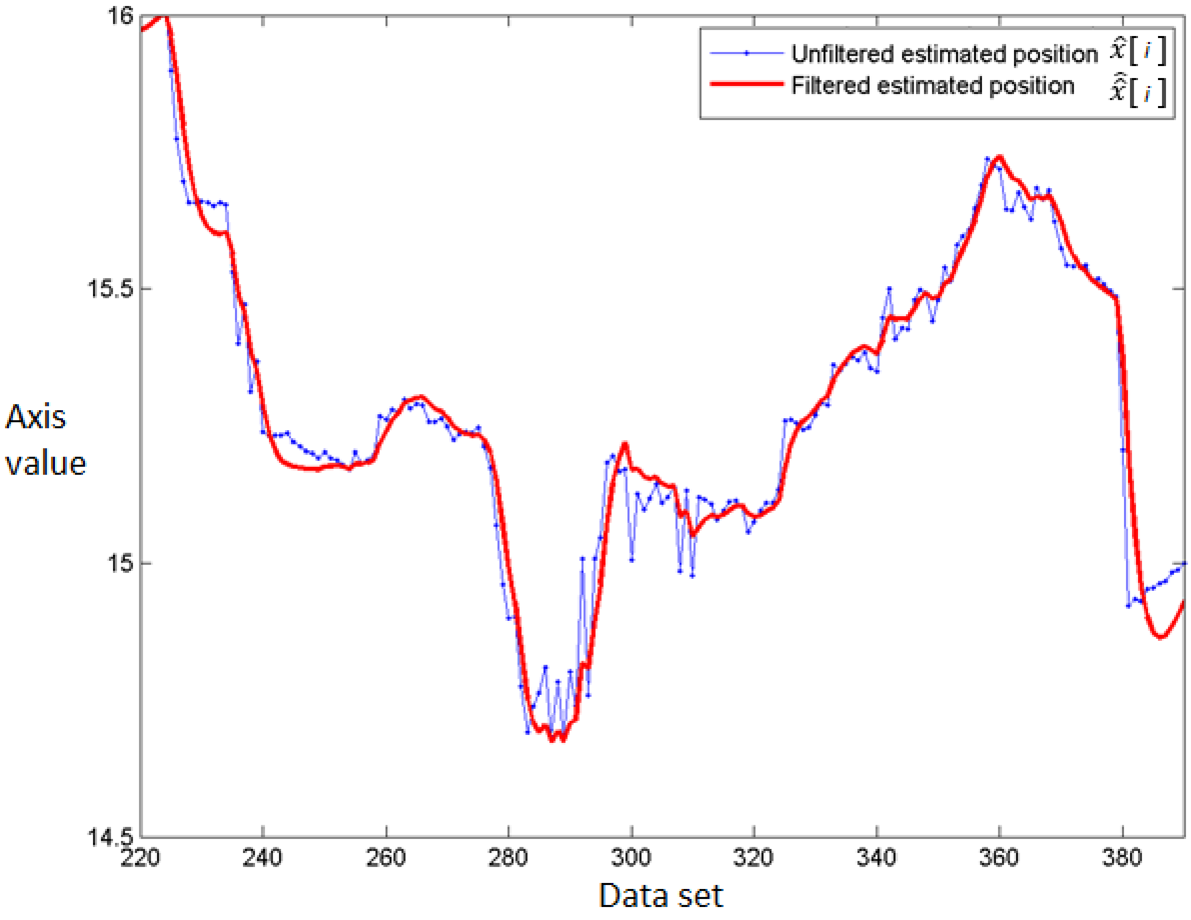


Figure 6.1: Tracker system accuracy of real object, x-component

The reader may observe the new resolution of the display is significantly greater than one pixel. Also, the Kalman filter provides a lot smoother trajectory for the object, but note that this does not equal to more accurate. One could argue that the tracking filter is more inaccurate, especially between sample 240-260 and 380-390. The filter parameters are adjustable for quicker reaction, but this will lead to a less smooth trajectory.

The application level can choose to smoothen the path further, but this might cause a time delay. For displays with high SNR it is hard to evaluate the effect, but the computational cost of the Kalman filter might not scale to the improvement of accuracy, when compared to the unfiltered estimation.

6.2 Simulation: Single stationary object

The first simulation is a single stationary small object. Six different positions was simulated, shown by figure 6.2.

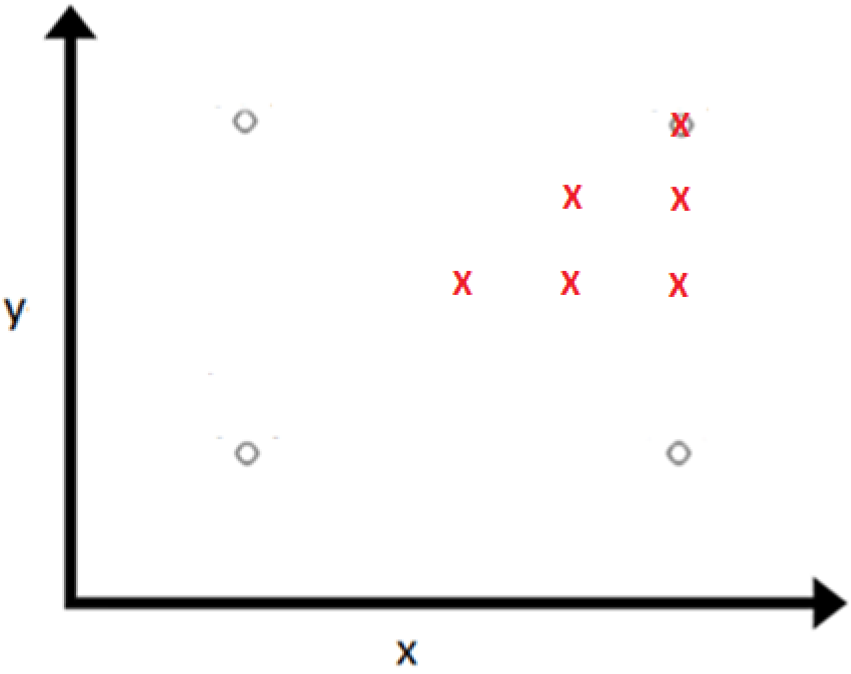


Figure 6.2: Test positions(red X's) and nodes (circles)

The six positions are located at:

$$F_1(x_1, y_1) = (5.00\,, 5.00)$$

$$F_2(x_2, y_2) = (5.00\,, 5.25)$$

$$F_3(x_3, y_3) = (5.00\,, 5.50)$$

$$F_4(x_4, y_4) = (5.25\,, 5.25)$$

$$F_5(x_5, y_5) = (5.25\,, 5.50)$$

$$F_6(x_6, y_6) = (5.50\,, 5.50)$$

This is because the accuracy varies slightly depending on the node position relative to the object position, which was shown in the specialization project [8]. The positions are on the top left corner of the simulated touch display. The accuracy of the Tracker system is tested by averaging the combined error distance of the six positions. Monte Carlo Simulations were performed with 1000 simulations per dB-level, with 1 dB steps, and the result is shown by figure 6.3.

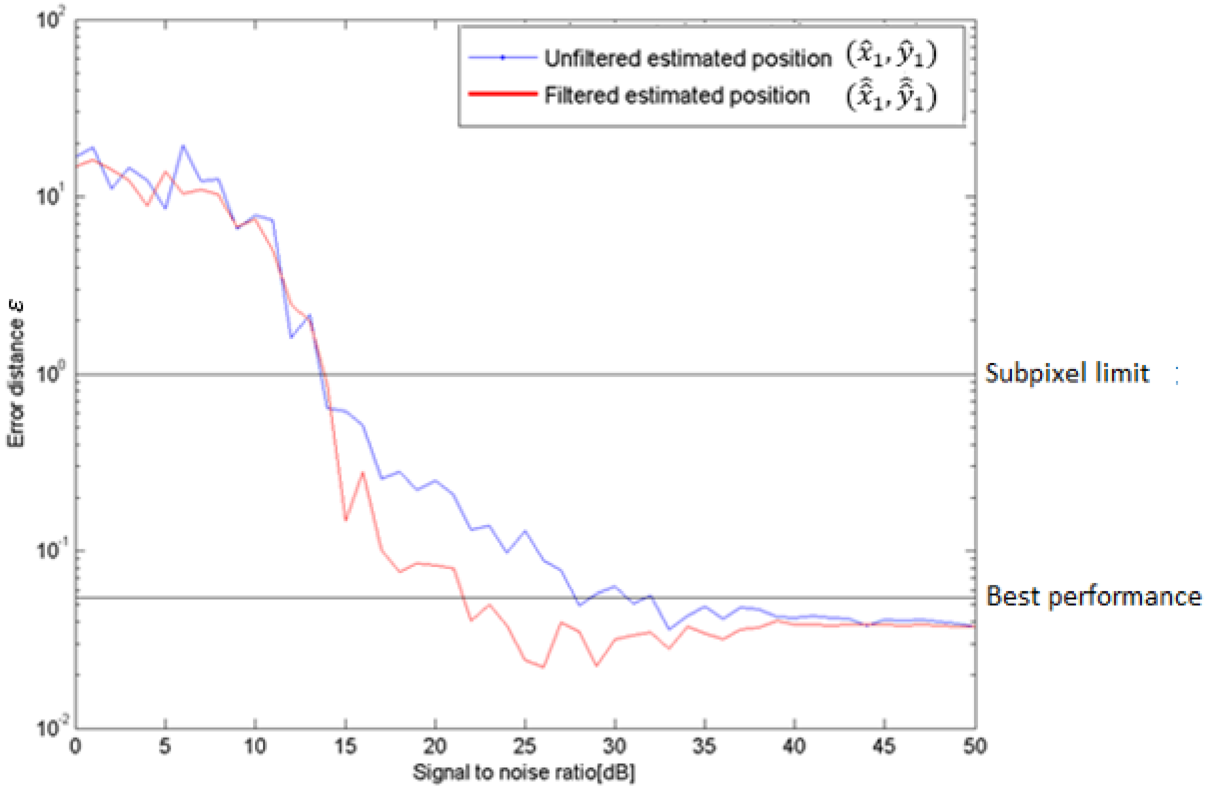


Figure 6.3: Tracker system accuracy of single stationary object

The filtered and unfiltered estimated position has approximately equal accuracy below 14dB and above 40dB. Below 14dB, the accuracy is not improved by the filter. However, the smoothening effect that it has might still make this a valuable addition. Above 40dB, the lower limit of performance seems to be reached, at $\varepsilon \approx 0.04$. For a touch display with 5mm space between each node, this transfers to 0.2mm, for the x-dimension. Note that these are simulated situations with symmetric objects, and that the result is likely better than what could be achieved on a real data set.

At approximately 14dB, when $\varepsilon = 10^0$, the Tracker system achieves subpixel accuracy for both filtered and unfiltered estimation. The filtered estimation is visibly better than the unfiltered graph from this point up to 40dB, but the gain varies. This is likely the result of the Kalman filter not being tuned for each SNR-level, which is done by adjusting C and Q in equation (5.1) and (5.2).

At around 30dB SNR, the tracking filter has the biggest gain compared to the unfiltered estimated position, highlighted by the grey horizontal line at $\varepsilon \approx 0.55$. The tracking filter gain is now equal to improving the display SNR in the order of 10dB.

6.3 Simulation: Single moving object

The second simulation is an object moving along the y-axis, identical to the situation from chapter 4.4. This time SNR-level is fixed at 30dB, where the tracking filter is shown to be the most effective, to more clearly highlight the strength and weakness of an optimal situation. Monte Carlo Simulations with 1000 simulations were performed (figure 6.4). The object moves 0.1 nodes per data set, and ε is averaged for each position.

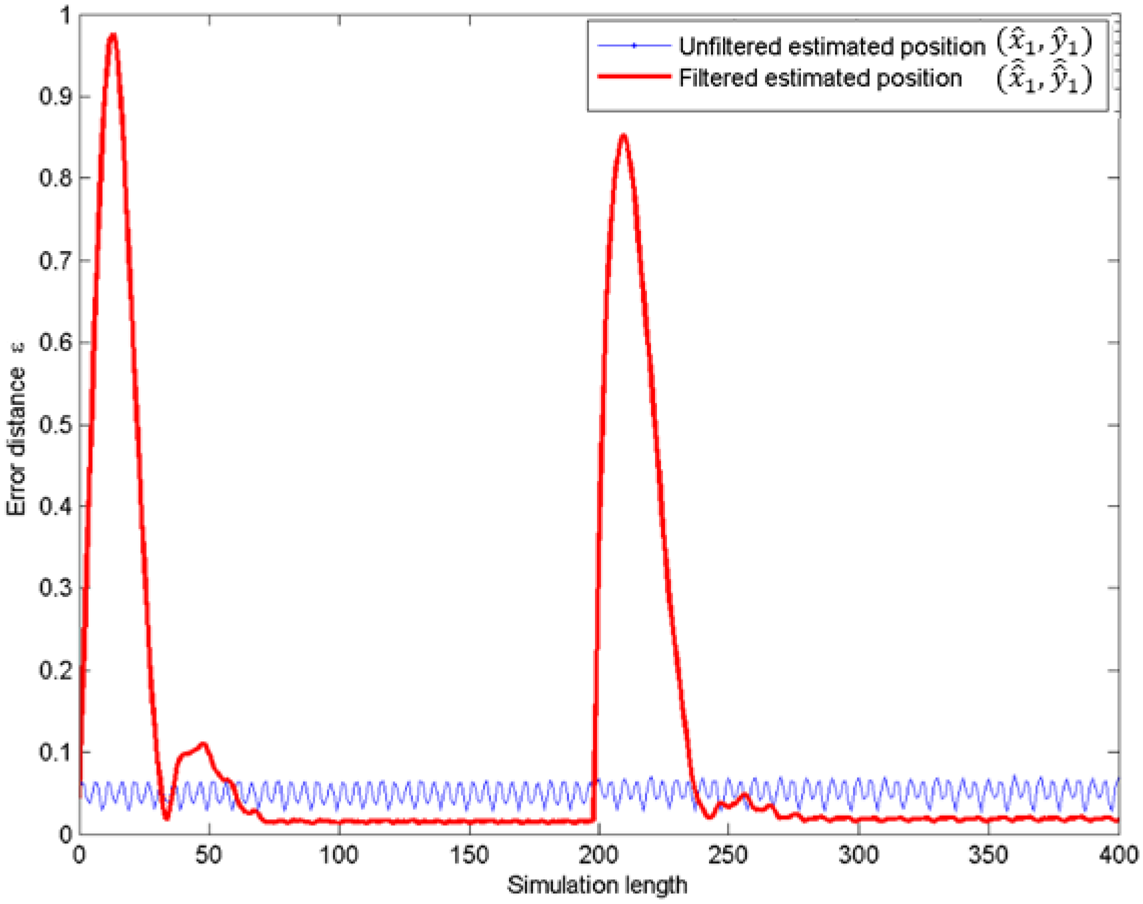


Figure 6.4: Tracker system accuracy of single moving object

Both the unfiltered and filtered estimation achieve subpixel accuracy for the entire simulation, but we observe that the filtered estimation has difficulties handling the start and middle of the simulation. This is the same effect visible on figure 6.1, for sample 380-390. There is a sudden change in velocity, and the tracking filter cannot keep up. Its initial velocity is set to zero, and the filter is not stabilized until around the 70th data set. At this point, performance is increased, and the filtered estimation is better than the unfiltered estimation. At the 200th simulation the object turns, which results in another 70 data sets before the filter stabilizes again.

Pre-setting the initial velocity to a more optimal estimate would likely reduce the inaccuracy at the beginning of the simulation, but not the inaccuracy at the middle. This would likely require a more advanced tracking filter, like the Extended Kalman Filter [10].

6.4 Simulation: Two close objects

This simulation compares the successful close search result from chapter 4.4, with and without the tracking filter. The result is shown by figure 6.5, where object 1 is once again stationary and object 2 is moving.

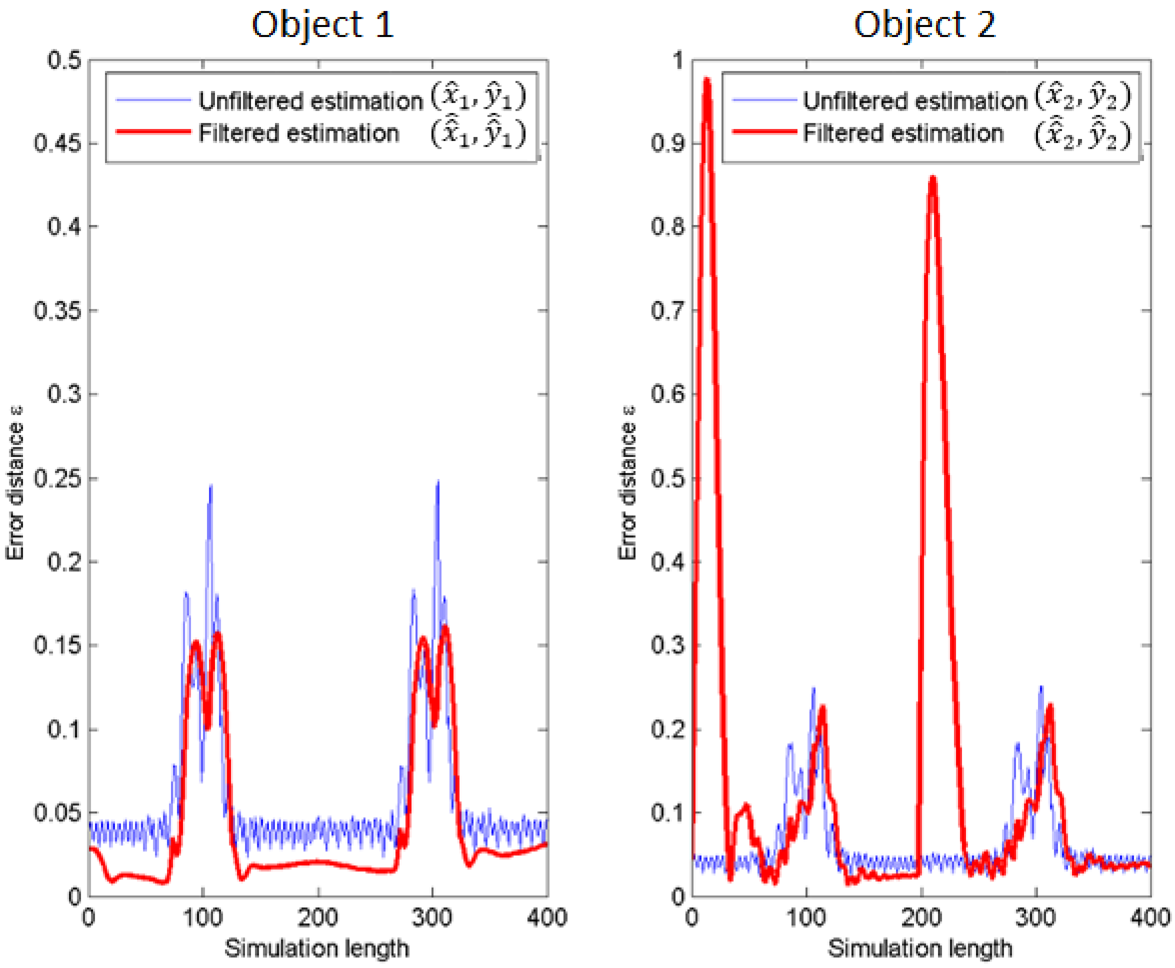


Figure 6.5: Tracker system accuracy of close search simulation

Subpixel accuracy was achieved again, for both objects and for the entire simulation. The stationary object achieves, overall, a more accurate tracking using the filtered position. The moving object still has two intervals with much higher inaccuracy, like in the previous simulation.

If the object is stationary for a short duration prior to pinching, or moving slower, the tracking filter might perform better. If an object often and quickly changes direction, it is not yielding better accuracy at this point. As mentioned earlier earlier in this chapter, smoothness might be valued higher than accuracy, depending on the application used.

7 Topics for further research

The Tracker system works as intended and it is ready to be implemented on a real display. However, there are still unanswered questions regarding performance. The most prominent topic for further research should be how the accuracy of the Tracker system changes with regard to different sizes and shapes of an object. The tested object in this thesis was a small object, and figure 6.3 (page 54) shows the accuracy of the Tracker system on this object. Testing how much the lower limit and the breakpoint for subpixel accuracy changes, for different sizes and shapes of objects, calls for further research.

Another topic for further research may be the tracking filter. Figure 6.4 (page 55) and 6.5 (page 56) showed that the Standard Kalman Filter was unable to effectively handle a rapid change of direction by the moving objects. Implementing a more advanced filter would likely improve the accuracy. Alternatively, since the estimated position is already highly accurate, a solution could be to find a smart way of toggling the tracking filter on and off.

If the Tracker system is implemented on a real touch display, it is important to remember that it was not designed for a specific display. No displays are alike, as shown in chapter 1.2, and it is advisable to tailor the Tracker system to the specific device. Tweaks of the parameters should include I_{thresh}, d_{min}, d_{val}, and Kalman filter settings: C, Q and possibly the initial conditions. This can be controlled by further development of the Tracker system, or the parameters could be sent downwards in the system from the application level, shown by the new stapled line on figure 6.6.

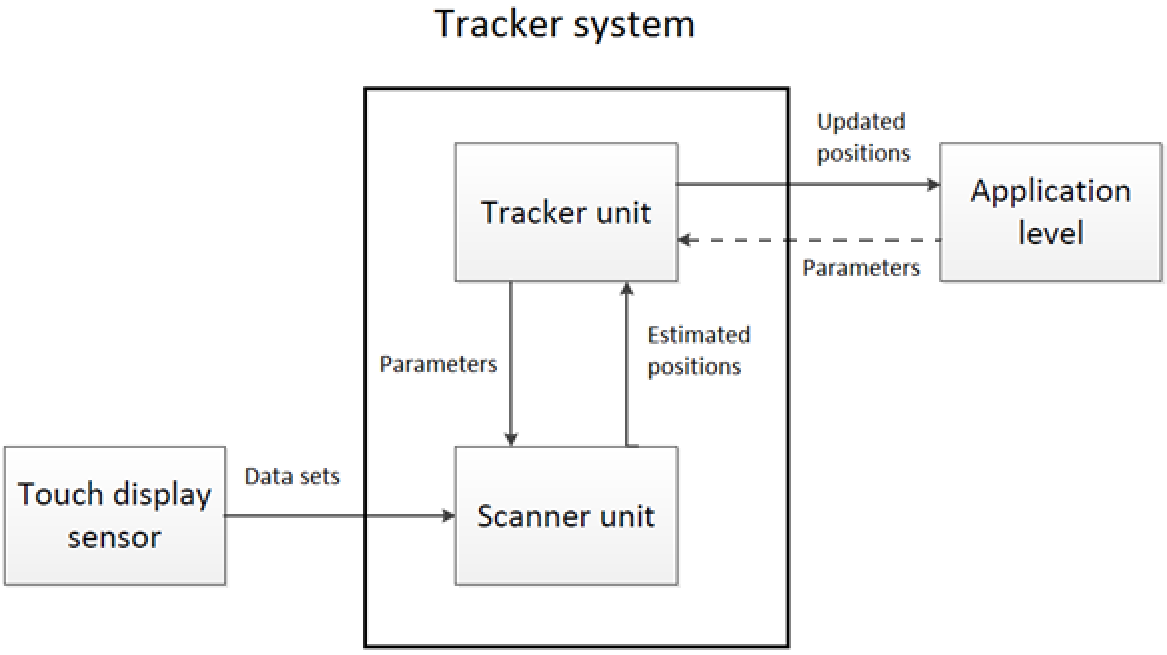


Figure 6.6: New Tracker system flow chart

Even though it was out of the scope of this project, the overall assessment of the Tracker system is not complete without a comprehensive analysis of the computational cost. This is particularly interesting for portable units, since these have limited battery life and limited computational power, which affects real-time requirements. Implementing the Tracker system on a real touch display would also make it possible to compare computational cost to the humans perceptive of this performance.

Some of the assumptions made in this thesis can be reduced by limiting the compatible touch inputs, even though this was a self-imposed requirement set in the problem description. Matching the touch display with a special touch stylus, with optimal size and conductive properties for the nodes, would likely make tracking easier.

Lastly, implementing the ability to track three or more objects creates new possibilities for the application level. This allows large groups to interact with the touch display at the same time, or for advanced touch motions.

8 Conclusion

This thesis set out to describe a program that tracks multiple objects in a low resolution environment. A subpixel, or high-resolution, position was estimated prior to tracking the objects. This was solved by developing a two-module solution named the Tracker system. The first module is the scanner unit that receives the data sets from the touch display sensor, and produces a two-dimensional coordinate for up to two objects. The second unit is the tracker unit, that first labels the objects and validates that the objects are not noise. Then, it sends the positions through a tracking filter, and these are now ready to be sent to an applications level.

Different scenarios for the scanner unit have been presented, and a solution for each scenario has been proposed and demonstrated. It has been shown that handling all situations requires some assumptions and limitations to be made, prior to developing a solution. These choices had to be made even with no self-imposed limit to computational cost. A big challenge was developing a good method of estimating the size of an object, combined with tracking two objects that are close to each other. These situations were eventually handled successful, albeit with reduced accuracy.

In a simulated 30dB SNR touch display environment, the tracking filter increased the accuracy equal to increasing the display SNR in the order of 10dB. The tracking filter did not always improve accuracy, but it might still be valuable as it smoothens the trajectory of the objects. Overall, the usage of the tracking filter depends on the application being used.

This thesis can serve as the foundation for further development, and there are two topics that stand out. The first topic is how the accuracy of the Tracker system changes for different sizes and shapes of the objects. The second topic is how the Tracker system performs on a real touch display, as the human perceptual performance does not necessarily correspond to the theoretical performance. If the implemented performance does not match the results presented in this thesis, this also points toward further emphasis on the first topic.

[This page is intentionally left blank]

9 Bibliography

[1] Store Norske leksikon, *Berøringsskjerm*, May 2015, https://snl.no/ber%C3%B8ringsskjerm%2FIT

[2] Wikipedia. *Touch screen*, February 2015, http://en.wikipedia.org/wiki/Touchscreen

[3] Wikipedia. *Smartphone*, February 2015, http://en.wikipedia.org/wiki/Smartphone

[4] Wikipedia. *Mutual Capacitance*, February 2015, http://en.wikipedia.org/wiki/Mutual_capacitance

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

[6] Illustration retrieved 1.11.2014, from http://m.eet.com/media/1046718/0909esd_Atmel01sm.gif

[7] Illustration retrieved 1.11.2014, from http://techielobang.com/blog/wp-content/uploads/2010/01/line-drawing-test-2.jpg

[8] Sveinung P. Røsok. TTT-4511: Digital Signal Processing specialization project: Detection of low-resolution objects in a grey-scale image, NTNU (2014)

[9] Atmel Norway AS, June 2015, www.atmel.com

[10] Stephen M. Kay, *Statistical Signal Processing volume I*, Prentice Hall, 1993

[This page is intentionally left blank]

9 Appendix

[A] Matlab code of Tracker system

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        %% TRACKING SYSTEM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initiations for testing
clear all;

minimum_distance_mode = 0;
counter = 0;
sim_length = 0;
reverse = 1;

object_1 = 0;
object_2 = 0;
validation_object_1 = 0;
validation_object_2 = 0;

**load data_set_Fi here**



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        %% SCANNER UNIT %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This is the first module of the program, which scans each data set Fi
% It scans the data ests based on parameters received from the tracker unit
% and performs the search based on these parameters.

% 2 main functions: Find 2, and Find 2 close objects


%% Find 2, input(data_set_Fi), output(x0y0/x1y1)
if minimum_distance_mode == 0
    object_positions = find_2(data_set_Fi);
    obj_1_pos = [object_positions(1) object_positions(2)];
    obj_2_pos = [object_positions(3) object_positions(4)];

%% Fine positioning, below minimum distance, either LS
elseif minimum_distance_mode == 1
    close_object_positions = find_2_close(data_set_Fi, prev_obj1_node(1),
prev_obj1_node(2), prev_obj2_node(1), prev_obj2_node(2));
    obj_1_pos(1) = close_object_positions(1); obj_1_pos(2) =
close_object_positions(2);
    obj_2_pos(1) = close_object_positions(3); obj_2_pos(2) =
close_object_positions(4);
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        %% TRACKER UNIT %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This is the second and final module of the tracker system
% First, it labels and validates the positions received from the scanner
% Then, it sends this through a tracking filter (Kalman)
% The distance between the positions are calculated, and the scanner unit
% is requested to perfom a close search if this d < d_min

%% Labelling - Connect to closest
if obj_1_pos > 0 && sim_length > 1
    connect_objects = labelling( obj_1_pos(1), obj_1_pos(2), obj_2_pos(1),
obj_2_pos(2), prev_obj_1_x(sim_length-1) , prev_obj_1_y(sim_length-1) ,
prev_obj_2_x(sim_length-1) , prev_obj_2_y(sim_length-1) ) ;
    obj_1_pos(1) = connect_objects(1);
    obj_1_pos(2) = connect_objects(2);
    obj_2_pos(1) = connect_objects(3);
    obj_2_pos(2) = connect_objects(4);
end

%% Validate
if obj_1_pos(1) == 0
    object_1 = 0;
else
    object_1 = 1;
end

if obj_2_pos(1) == 0
    object_2 = 0;
else
    object_2 = 1;
end

% Track 1
if object_1 == 0
    validation_object_1 = 0;
elseif object_1 == 1 && validation_object_1 < 3
    validation_object_1 = validation_object_1 + 1;
    object_1 = 0;
end

% Track 2
if object_2 == 0
    validation_object_2 = 0;
elseif object_2 == 1 && validation_object_2 < 3
    validation_object_2 = validation_object_2 + 1;
    object_2 = 0;
end


%%  Check minimum distance
if min_dist_mode == 1 && sqrt((track_1(1)-track_2(1)).^2 + (track_1(2)-
track_2(2)).^2) > 6
    min_dist_mode = 0;
elseif object_1 == 1 && object_2 == 1 && sqrt((track_1(1)-track_2(1)).^2 +
(track_1(2)-track_2(2)).^2) < 5;
    min_dist_mode = 1;
else
    min_dist_mode = 0;
end
```

```
%% Log current position

prev_obj_1_x(sim_length) = obj_1_pos(1);prev_obj_1_y(sim_length) =
obj_1_pos(2);
prev_obj_2_x(sim_length) = obj_2_pos(1);prev_obj_2_y(sim_length) =
obj_2_pos(2);

% Make sure they do not have the same main node
if round(obj_2_pos(1))==round(obj_1_pos(1)) &&
round(obj_2_pos(2))==round(obj_1_pos(2))
    prev_obj1_node = [round(obj_1_pos(1)) round(obj_1_pos(2))];
    prev_obj2_node = [round(prev_obj2_x) round(prev_obj2_y)];
else
    prev_obj1_node = [round(obj_1_pos(1)) round(obj_1_pos(2))];
    prev_obj2_node = [round(obj_2_pos(1)) round(obj_2_pos(2))];
end



% Object and estimation decomposed into x- and y-values
x1_sim(sim_length) = sim1(1);
x1_est(sim_length) = obj_1_pos(1);
y1_sim(sim_length) = sim1(2);
y1_est(sim_length) = obj_1_pos(2);
x2_sim(sim_length) = sim2(1);
x2_est(sim_length) = obj_2_pos(1);
y2_sim(sim_length) = sim2(2);
y2_est(sim_length) = obj_2_pos(2);



                            %% PLOT %%
% Print active nodes + object positions on display
data_set_Fi_result = zeros(matrix_size(1), matrix_size(2));
RoI_nodes = 0;

% Find RoI for object 1
if object_1 == 1
RoI_nodes = size_estimation(data_set_Fi, round(obj_1_pos(1)),
round(obj_1_pos(2)));
data_set_Fi_result(RoI_nodes(3):RoI_nodes(4),RoI_nodes(1):RoI_nodes(2))=
data_set_Fi(RoI_nodes(3):RoI_nodes(4),RoI_nodes(1):RoI_nodes(2)) ;
end

% Find RoI for object 2
if object_2 == 1
RoI_nodes = size_estimation(data_set_Fi, round(obj_2_pos(1)),
round(obj_2_pos(2)));
data_set_Fi_result(RoI_nodes(3):RoI_nodes(4),RoI_nodes(1):RoI_nodes(2)) =
data_set_Fi(RoI_nodes(3):RoI_nodes(4),RoI_nodes(1):RoI_nodes(2));
end

figure(1);
show_tracking(data_set_Fi, sim1(1), sim1(2), sim2(1), sim2(2),
data_set_Fi_result, object_1, obj_1_pos(1),obj_1_pos(2), object_2,
obj_2_pos(1),obj_2_pos(2));

% end of tracker system *
```

[This page is intentionally left blank]

[B] Least Squares formula derivation

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. The least squares (LSE) error model for a single touch input is defined here as:

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

This method assumes the object is a 3x3 matrix, where the node with the highest intensity is the center node, (2,2). 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, where x and y is used for familiarity:

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

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:

$a = F(m-1,n-1) + F(m,n-1) + F(m+1,n-1) - 2F(m-1,n) - 2F(m,n) - 2F(m+1,n) + F(m-1,n+1) + F(m,n+1) + F(M+1,N+1)$

$b = -F(n-1,n-1) + F(m+1,n-1) + F(m-1,n+1) - F(m+1,n+1)$

$c = -F(m-1,n-1) + F(m+1,n-1) - F(m-1,n) + F(m+1,n) - F(m-1,n+1) + F(m+1,n+1)$

$d = -F(m-1,n-1) + 2F(M,N-1) - F(m+1,n-1) + 2F(m-1,n) + 5F(m,n) + 2F(m+1,n) - F(m-1,n+1) + 2F(m,n+1) - F(m+1,n+1)$

$e = F(m-1,n-1) + F(m,n-1) + F(m+1,n-1) - 2F(m-1,n) - 2F(m,n) - 2F(m+1,n) + F(m-1,n+1) + F(m,n+1) + F(m+1,n+1)$

$f = F(m-1,n-1) + F(m,n-1) + F(m+1,n-1) - F(m-1,n+1) - F(m,n+1) - F(m+1,n+1).$

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

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

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

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

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

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

Where x and y is the displacement relative to the center node, (2,2).