Ambient Assisted Living Joint Programme

Project full title:

Development of a non-invasive CAPactive sensor oral MOUSE interface for the disabled elderly (CAPMOUSE, AAL-2008-1-203)

Deliverable report:

D2.3 & D4.2 Report describing the optimal processing algorithm including the output of normalized data

AAL project number:
Project starting date: 15/06/2009
Project duration: 30 months

Coordinator: Tomas Brusell
Coordinator e-mail: tomas.brusell@brusell-dental.com
Project website: http://www.brusell-dental.com/aal

Contributors: BD, HMC, LOTS
Planned delivery date: 31.03.2011
Actual delivery date: 12.04.2012

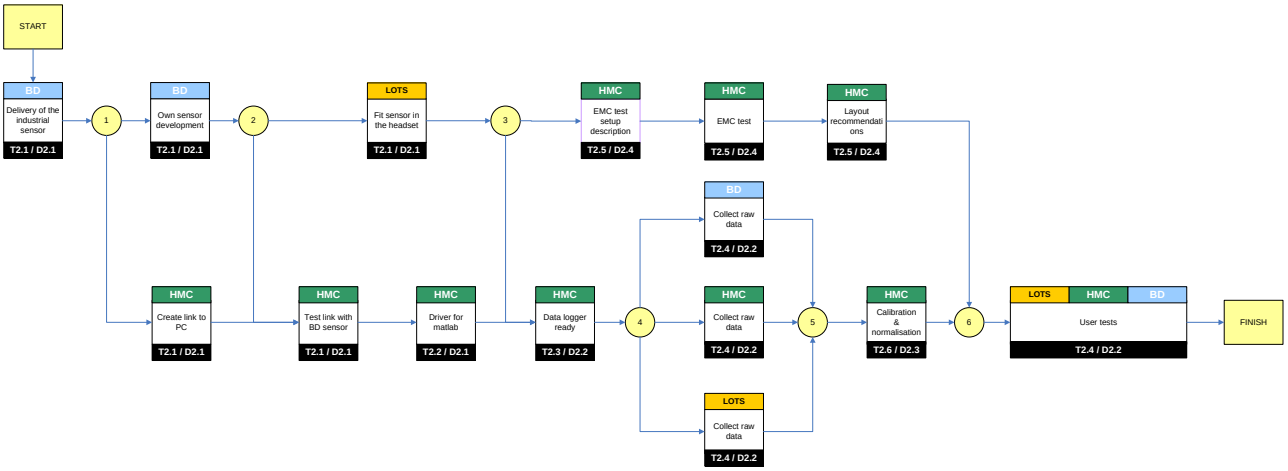
1. General

1.1. Task description

The content of deliverable D2.3 mainly consists of the description of the work done in T2.6.

T2.6 Designing a sensor signal normalisation method
Search for calibration and normalisation method to rule out sensor signal deviation; the measured signal should be hardware-sensor independent. Analysing and testing configurable parameters to customise the set-up for optimal usability of the sensor (e.g. tremor, vibrations, filtering, etc.)
Participants: BD 0,5pm, HMC 2pm, Lots 0pm, PRO 0pm

1.2. Work package 2: overview



This is the schematic overview of work package 2. Work package 2 contains the actions that have to be taken by the consortium to obtain a fully working sensor prototype. Work package 2 consists of 4 deliverables. D2.1 describes the hardware development of the software. D2.2 describes the data logger software and the test results gathered by use of the data logger. D2.3 describes the calibration and normalization method. D2.4 describes the tests on the hardware and the tests on the normalization and calibration method. This work package contains both research and development. Research jobs are always hard to plan due to their unpredictable nature. This may reflect itself in the information that is contained in the deliverables.

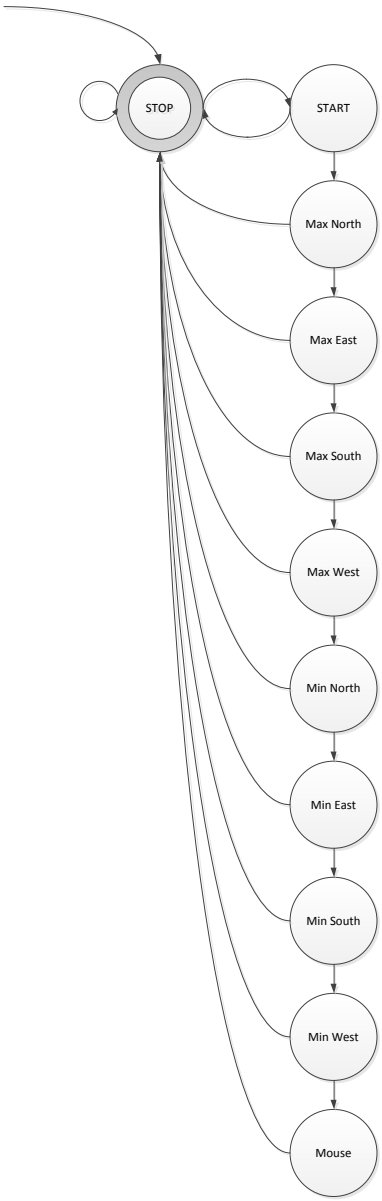
2. T2.6 Designing a sensor signal normalization method

2.1. Introduction

This section describes the software setup and the normalization method. But there is a fundamental difference between calibration and normalization. That is why these two subjects will be discussed separately. First the use of normalization will be discussed, next the way it is implemented in the software. Then the extra features in the software for basic mouse movement will be discussed. Finally the calibration routine is explained and later the implementation in the software.

2.2. Dataflow of the software

The basic setup of the software is a state machine containing 10 states. The initial state is the stop state. Every other state has to be called from its previous state. When the task of that specific state is done it continues onto the next state. If an error occurs in any of the states or if the sequence is aborted, the stop state is called. This state machine will be used trough out this deliverable as the base of the software.



2.3. The different states

2.3.1. The STOP state

Next state: STOP
START

Previous state: STOP
START
MAX NORTH
MAX EAST
MAX SOUTH
MAX WEST
MIN NORTH
MIN EAST
MIN SOUTH
MIN WEST
MOUSE

Description: Idle state. This state does not do anything except waiting to start the program.

2.3.2. The START state

Next state: STOP
MAX NORTH

Previous state: STOP

Description: In this state all the registers are downloaded into the AD7147.

2.3.3. The MAX NORTH state

Next state: STOP
MAX EAST

Previous state: START

Description: Ask the user to place his tong against the north sensor plate without touching any other sensor plate. Collect 100 consecutive samples and filter out the maximum value.

2.3.4. The MAX EAST state

Next state: STOP
MAX SOUTH

Previous state: MAX NORTH

Description: Ask the user to place his tong against the east sensor plate without touching any other sensor plate. Collect 100 consecutive samples and filter out the maximum value.

2.3.5. *The MAX SOUTH state*

Next state: STOP
MAX WEST

Previous state: MAX EAST

Description: Ask the user to place his tong against the south sensor plate without touching any other sensor plate. Collect 100 consecutive samples and filter out the maximum value.

2.3.6. *The MAX WEST state*

Next state: STOP
MIN NORTH

Previous state: MAX SOUTH

Description: Ask the user to place his tong against the west sensor plate without touching any other sensor plate. Collect 100 consecutive samples and filter out the maximum value.

2.3.7. *The MIN NORTH state*

Next state: STOP
MIN EAST

Previous state: MAX WEST

Description: User may not touch any of the sensor plates. Collect 100 consecutive samples of the northern sensor plate and filter out the minimum value.

2.3.8. *The MIN EAST state*

Next state: STOP
MIN SOUTH

Previous state: MIN NORTH

Description: User may not touch any of the sensor plates. Collect 100 consecutive samples of the eastern sensor plate and filter out the minimum value.

2.3.9. *The MIN SOUTH state*

Next state: STOP
MIN WEST

Previous state: MIN EAST

Description: User may not touch any of the sensor plates. Collect 100 consecutive samples of the southern sensor plate and filter out the minimum value.

2.3.10. *The MIN WEST state*

Next state: STOP
MOUSE

Previous state: MIN SOUTH

Description: User may not touch any of the sensor plates. Collect 100 consecutive samples of the western sensor plate and filter out the minimum value.

2.3.11. *The MOUSE state*

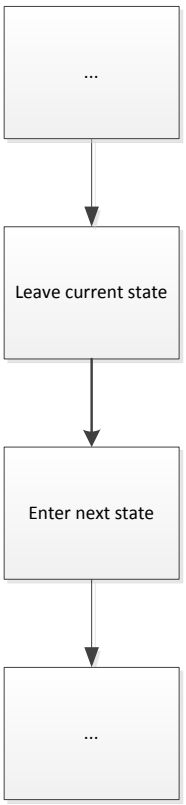
Next state: STOP

Previous state: MIN WEST

Description: Collect data and apply normalisation. Translate this data into mouse movement and mouse clicks. Use the data for real time calibration.

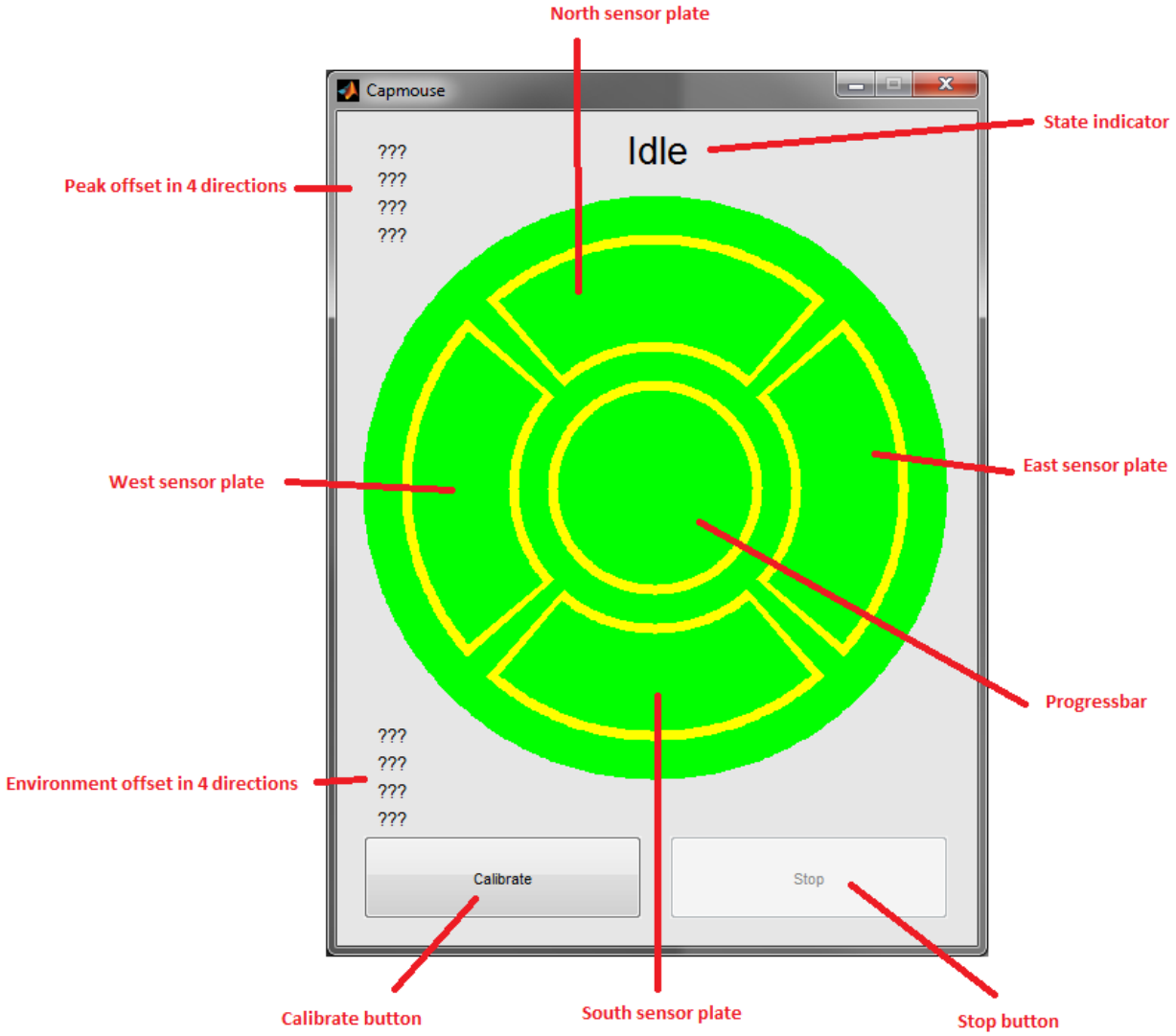
2.4. *General info about the states*

- Every state has an "Enter" and "Leave" method. These methods contain the initial settings of each state and the clean-up after a state has been completed. Each time a state change occurs these methods will be executed. These guarantee that each state initiates properly and closes down properly.



- Every state except the STOP state relies on the "BytesAvailable Callback" to gather the data. The "BytesAvailableFcn" is configured to execute a callback function when a bytes-available event occurs. A bytes-available event occurs when the number of bytes specified by the BytesAvailableFcnCount property is available in the input buffer, or after a terminator is read. This is determined by the BytesAvailableFcnMode. A watchdog timer function takes action when the data is not received within the predefined time limit. The latter still needs to be implemented. The STOP state will be triggered by the errors of other states or by button callbacks.

2.5. *The graphical user interface (GUI)*



North sensor plate: Yellow = active
Gray = inactive
Indicates if the north sensor plate is touched

East sensor plate: Yellow = active
Gray = inactive
Indicates if the south sensor plate is touched

South sensor plate: Yellow = active
Gray = inactive
Indicates if the south sensor plate is touched

West sensor plate: Yellow = active
Gray = inactive
Indicates if the south sensor plate is touched

Progressbar: Gives feedback to the user.
It displays how many samples are collected, and how many more are needed.

Calibrate button: Press to start the program.

Stop button: Press to stop the program

State indicator: Displays the current state the program is in.
It also gives feedback to the user on what to do.

Peak offset: Displays the peak offset in the 4 directions. (NESW)

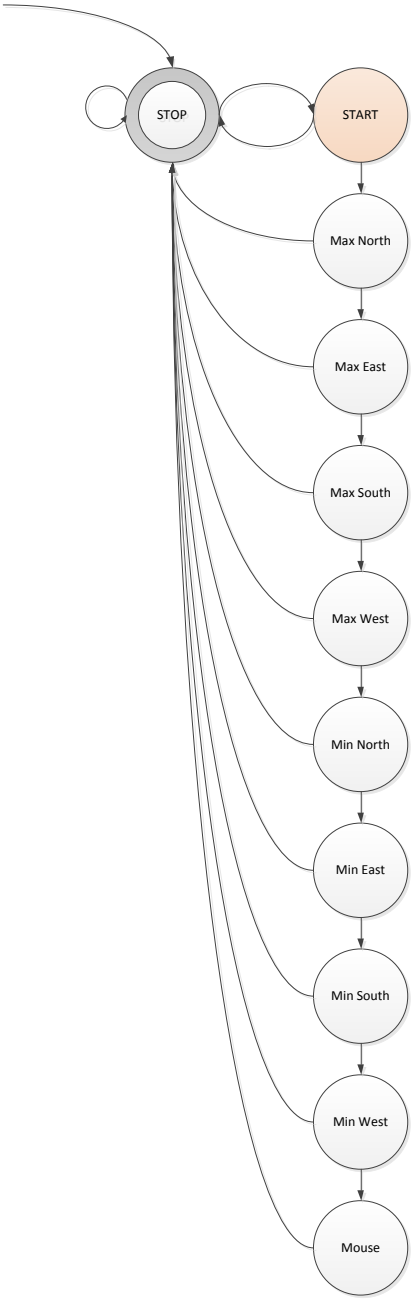
Environment offset: Displays the environment offset in the 4 directions. (NESW)

2.6. Downloading the registers into the AD7147

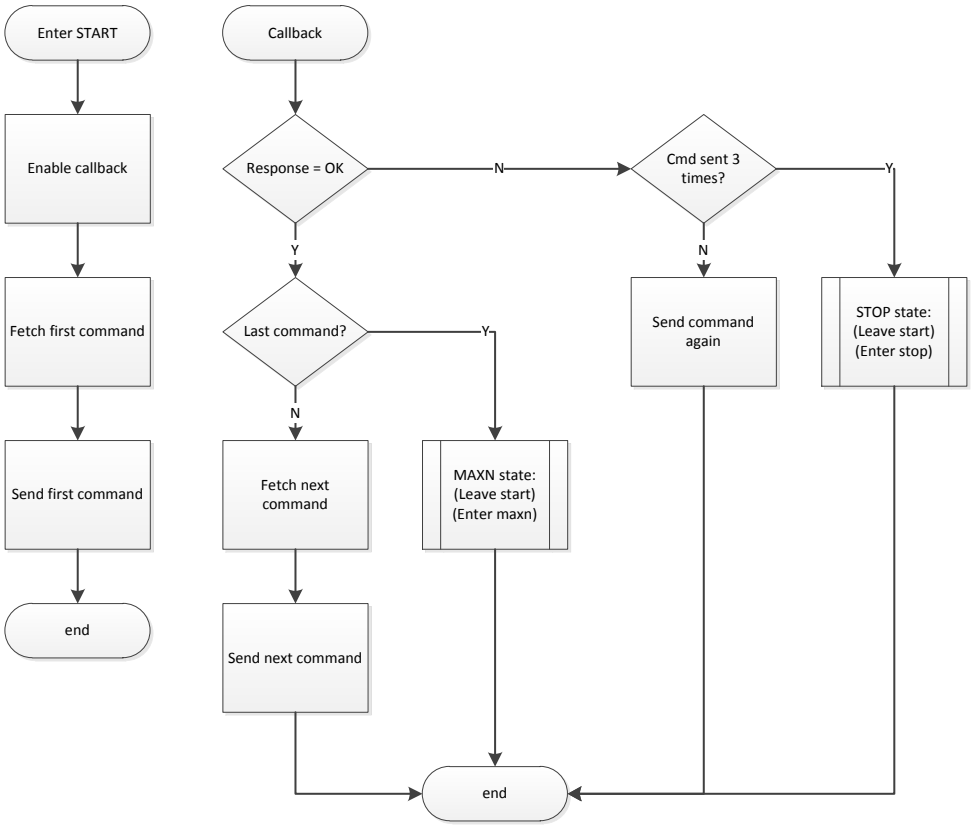
Downloading the registers into their initial configuration is being done in the START state. The communication with the dongle follows the protocol described in D2.4.

2.6.1. Location in the state machine

Downloading the register settings is done in the START st ate:

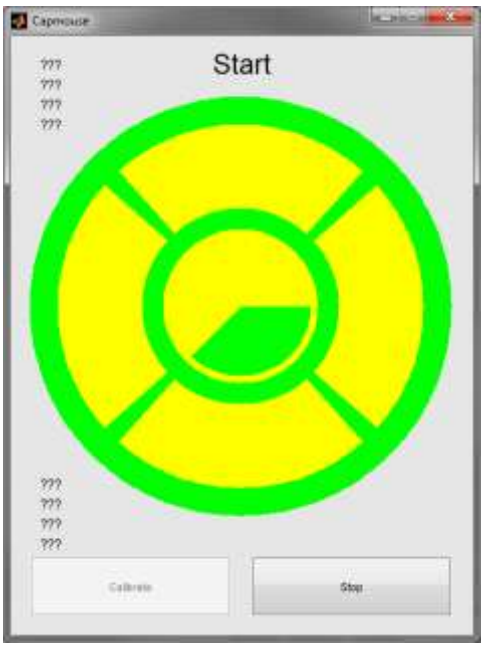


2.6.2. *Flowchart*



This chart describes the code when sending/receiving a command. The MATLAB software uses a FETCH and SEND routine to send each of the 104 commands. An OK is received each time a command is sent successfully. If a command is sent unsuccessfully an ERROR returns. Up to 3 ERRORS are tolerated before going back to STOP state.

2.6.3. *Graphical user interface*



The accompaning GUI for the START state.

2.6.4. The register settings

0x00 0x80 0xFF 0xFB	0x00 0x9A 0x00 0x00	0x00 0xB4 0x0F 0xA0	0x00 0xCE 0x0F 0xA0
0x00 0x81 0x1F 0xFF	0x00 0x9B 0x21 0x21	0x00 0xB5 0x0F 0xA0	0x00 0xCF 0x0F 0xA0
0x00 0x82 0x00 0x00	0x00 0x9C 0x0F 0xA0	0x00 0xB6 0x10 0x9A	0x00 0xD0 0xFF 0xFF
0x00 0x83 0x00 0x00	0x00 0x9D 0x0F 0xA0	0x00 0xB7 0x10 0x9A	0x00 0xD1 0x3F 0xFF
0x00 0x84 0x00 0x00	0x00 0x9E 0x10 0x9A	0x00 0xB8 0xFF 0xFF	0x00 0xD2 0x00 0x00
0x00 0x85 0x00 0x00	0x00 0x9F 0x10 0x9A	0x00 0xB9 0x3F 0xFF	0x00 0xD3 0x26 0x26
0x00 0x86 0x00 0x01	0x00 0xA0 0xFF 0xFE	0x00 0xBA 0x00 0x00	0x00 0xD4 0x0B 0xB8
0x00 0x87 0x00 0x01	0x00 0xA1 0x1F 0xFF	0x00 0xBB 0x21 0x21	0x00 0xD5 0x0B 0xB8
0x00 0x88 0xFF 0xEF	0x00 0xA2 0x00 0x00	0x00 0xBC 0x0F 0xA0	0x00 0xD6 0x0F 0xA0
0x00 0x89 0x1F 0xFF	0x00 0xA3 0x21 0x21	0x00 0xBD 0x0F 0xA0	0x00 0xD7 0x0F 0xA0
0x00 0x8A 0x00 0x00	0x00 0xA4 0x0F 0xA0	0x00 0xBE 0x10 0x9A	0x00 0xD8 0xFF 0xFF
0x00 0x8B 0x21 0x21	0x00 0xA5 0x0F 0xA0	0x00 0xBF 0x10 0x9A	0x00 0xD9 0x3F 0xFF
0x00 0x8C 0x0F 0xA0	0x00 0xA6 0x10 0x9A	0x00 0xC0 0xFF 0xFF	0x00 0xDA 0x00 0x00
0x00 0x8D 0x0F 0xA0	0x00 0xA7 0x10 0x9A	0x00 0xC1 0x3F 0xFB	0x00 0xDB 0x26 0x26
0x00 0x8E 0x10 0x9A	0x00 0xA8 0xFF 0xFF	0x00 0xC2 0x00 0x00	0x00 0xDC 0x0B 0xB8
0x00 0x8F 0x10 0x9A	0x00 0xA9 0x3F 0xFF	0x00 0xC3 0x21 0x21	0x00 0xDD 0x0B 0xB8
0x00 0x90 0xFF 0xBF	0x00 0xAA 0x00 0x00	0x00 0xC4 0x0F 0xA0	0x00 0xDE 0x00 0x00
0x00 0x91 0x1F 0xFF	0x00 0xAB 0x21 0x21	0x00 0xC5 0x0F 0xA0	0x00 0xDF 0x0F 0xA0
0x00 0x92 0x00 0x00	0x00 0xAC 0x0F 0xA0	0x00 0xC6 0x10 0x9A	0x00 0x00 0x00 0x80
0x00 0x93 0x21 0x21	0x00 0xAD 0x0F 0xA0	0x00 0xC7 0x10 0x9A	0x00 0x01 0x00 0x00
0x00 0x94 0x0F 0xA0	0x00 0xAE 0x10 0x9A	0x00 0xC8 0xFF 0xFF	0x00 0x02 0x32 0x32
0x00 0x95 0x0F 0xA0	0x00 0xAF 0x10 0x9A	0x00 0xC9 0x3F 0xFF	0x00 0x03 0x04 0x19
0x00 0x96 0x10 0x9A	0x00 0xB0 0xFF 0xFF	0x00 0xCA 0x00 0x00	0x00 0x04 0x08 0x32
0x00 0x97 0x10 0x9A	0x00 0xB1 0x3F 0xFE	0x00 0xCB 0x26 0x26	0x00 0x05 0x00 0x00
0x00 0x98 0xEF 0xFF	0x00 0xB2 0x00 0x00	0x00 0xCC 0x0B 0xB8	0x00 0x06 0x00 0x00
0x00 0x99 0x1F 0xFF	0x00 0xB3 0x21 0x21	0x00 0xCD 0x0B 0xB8	0x00 0x07 0x01 0x00

This is a list of all the register settings that have to be downloaded into the AD7147.

2.7. *Why need normalization?*

When developing a device or equipment, it is vital to understand that no two devices will ever be the same. This may be due to physical imperfections, tolerances and chance. In this case every sensor PCB will be different. The AD7147 has built in tolerances, the copper sensor surfaces might differ slightly etc. Therefor the data this sensor will output will slightly differ in comparison to another sensor. Since this data is key for our future calculations it is not a good idea to do our calculations with the absolute values this sensor delivers.

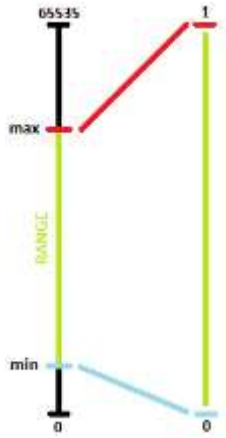
For example:
Let's say we have a built in threshold of 4pF. When this threshold is crossed, a switch is active. When this threshold is not crossed, the switch remains in its 'OFF' state. Sensor 1 sends a value of 3pF, Sensor 2 sends a value of 5pF under the same conditions. Sensor 1 will not trigger the switch and Sensor 2 will. Of course we want both sensors to give the same result under the same conditions. To make sure this happens, we need to filter out the tolerances and offsets. In this way relative values are distilled. These relative values will be used to do the calculations.

We found this out in the last user test session. Normalization was not implemented yet. Only the ability to change thresholds to trigger a mouse movement was implemented. The resulted in a constant shifting of thresholds for each test setup. Obviously this took a lot of time. This was the trigger to have a basic idea on how to implement this normalization. More details on the theory behind the normalisation are explained in the next chapter.

2.8. *The theory behind the normalization code*

The sensor has a 16 bit ADC. So it will always send a value between 0 and 65535. This value must become a value between 0 and 100% (or 0 and 1). The next big thing is to take the tolerances and offsets onto account. By measuring the minimum values and the maximum values we become 2 interesting parameters: the RANGE of the sensor and the BASE OFFSET.

BASE OFFSET = minimum value
RANGE = maximum value – minimum value



Expressed mathematically:

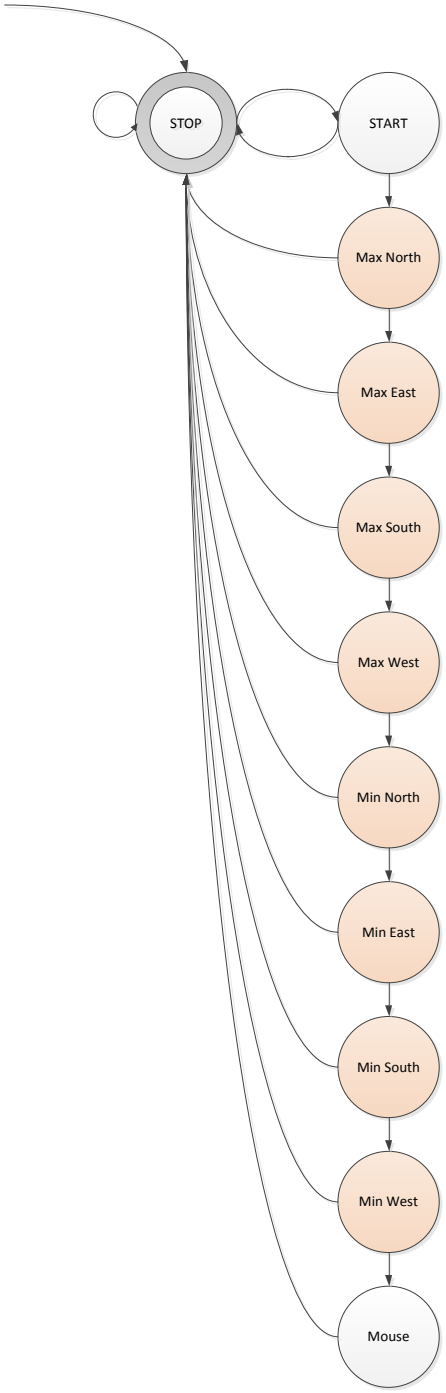
(DATA RECEIVED – BASE OFFSET) / RANGE = NORMALISED VALUE

2.9. Normalization in the software

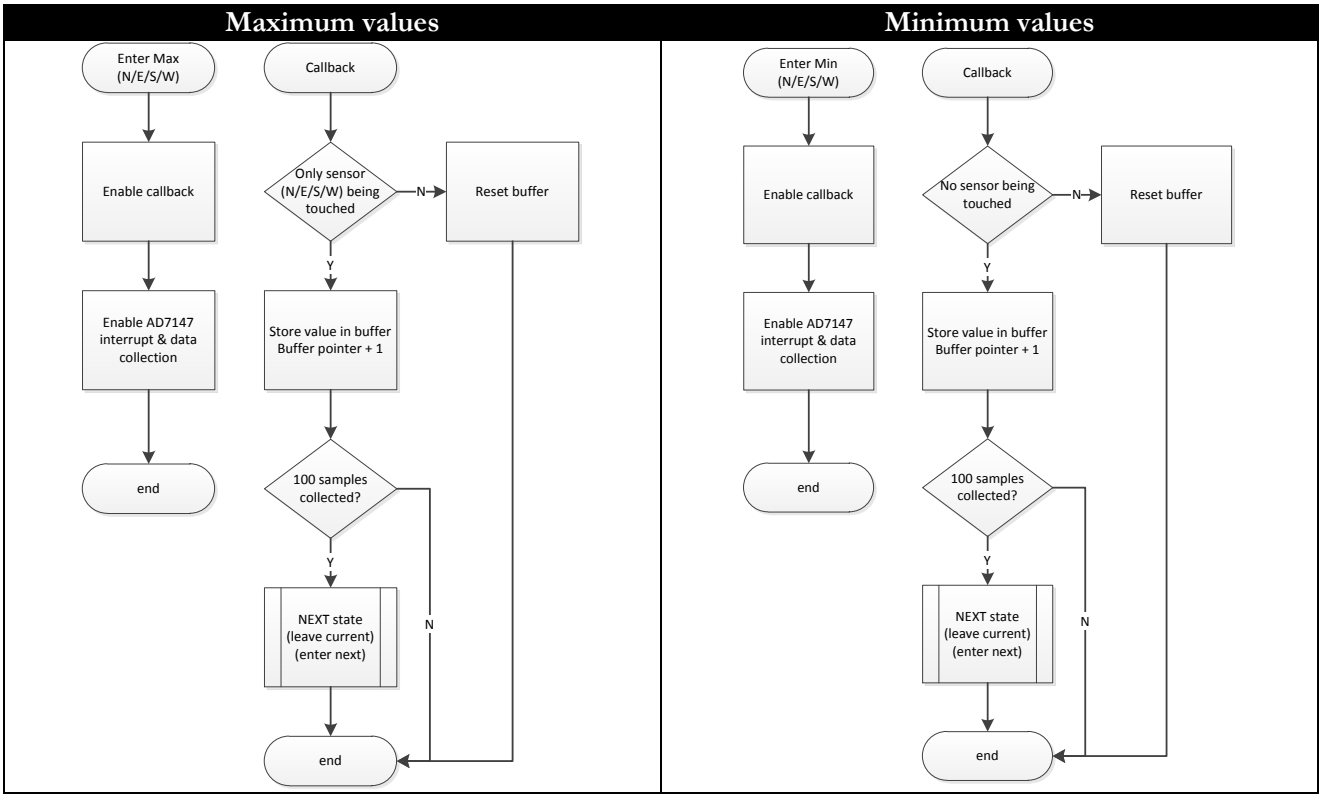
The essence of the normalization is to obtain the minimum and the maximum value. There are 4 directions: North, East, South and West. This implies that we need the minimum and maximum value for each of those directions. In total we need to collect 8 values. Once we have these, we can plug them into our equation and become normalized data.

2.9.1. Location in the state machine

We gather the maxima and minima in state Max North trough Min West

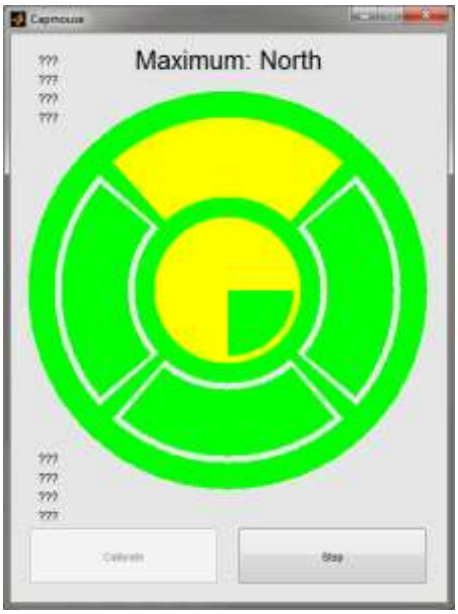


2.9.2. *Flowchart maximum values*



The callback routine first checks if the data it receives meets the conditions. For example, when getting the maximum value of north, only north may be touched by human skin. If this condition is valid for the next consecutive samples then north will be have collected all its samples. When it has all its samples, it moves on to the next state. If the condition is not met while collecting the samples, then it will restart the entire cycle. The same is valid for the minimum values, only the condition is different.

2.9.3. *Graphical user interface*



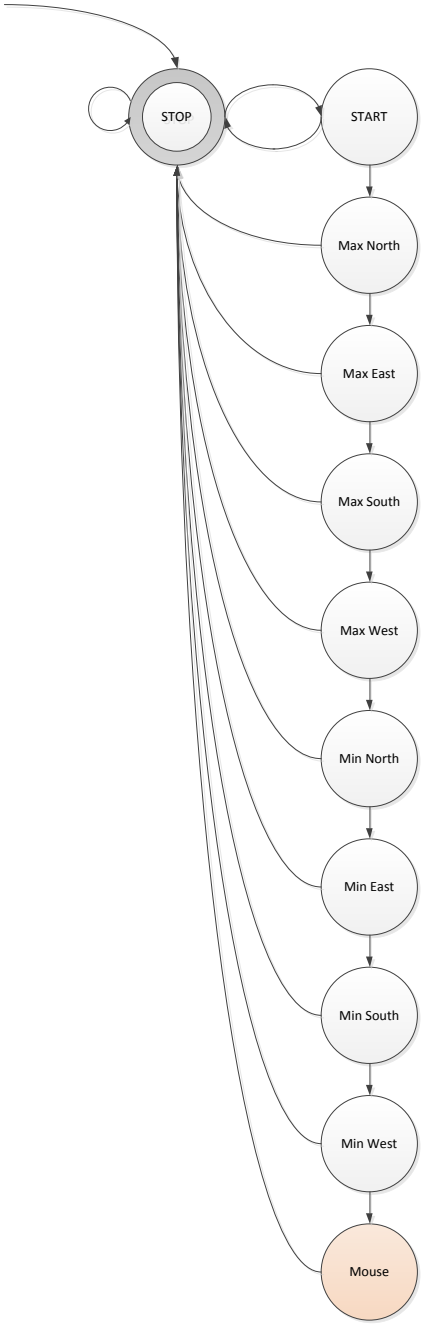
The accompaning GUI for the MAX and MIN states.

2.10. Mouse movement in the software

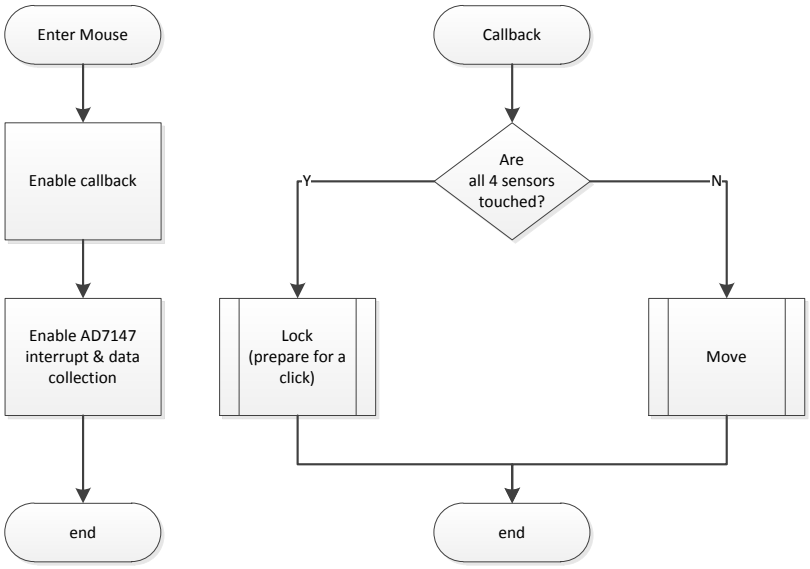
The mouse movement on the computer screen is a request by one of the partners (Brusell Dental). They were interested to see the mouse movement for demo purposes. It was sufficient to implement mouse movement and a single mouse click. One additional feature of the mouse movement was accelerated mouse movement. The feature behind accelerated mouse movement is: the longer the sensor plate is touched, the faster the mouse moves on the screen. It should also be possible to include a "dead band". For example, the first 3 seconds that the sensor plate is touched, the mouse should not move in that particular direction.

2.10.1. Location in the state machine

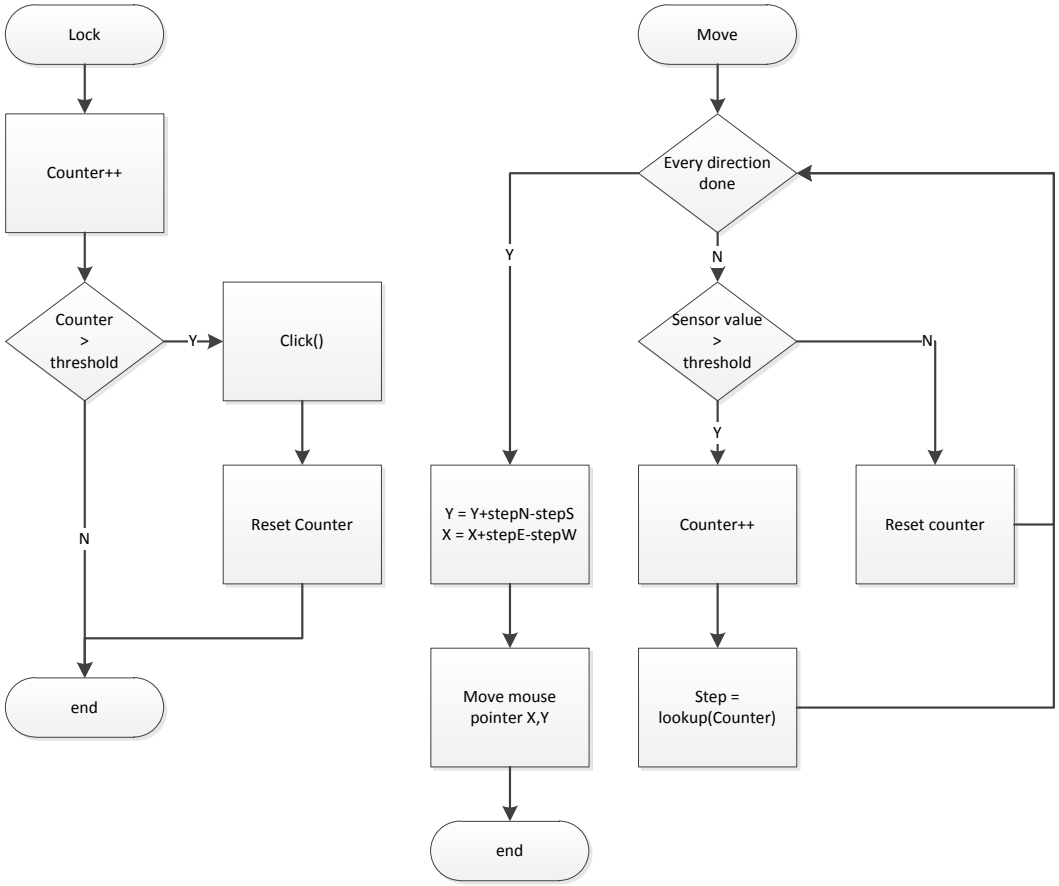
This is routine is located in the MOUSE state.



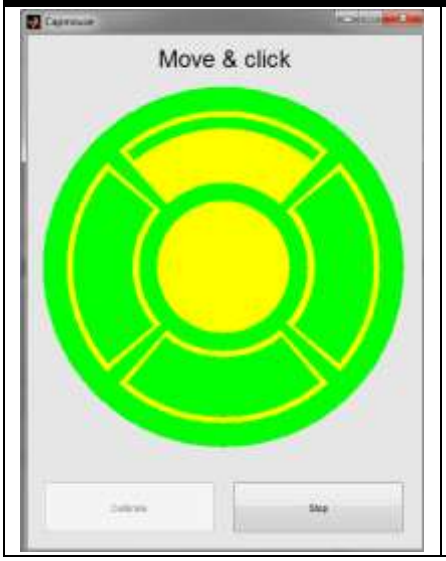
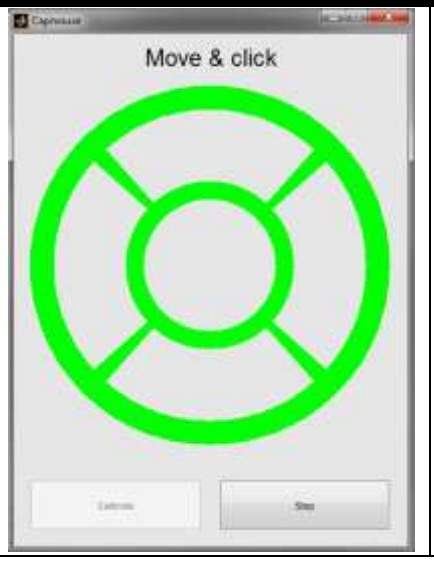
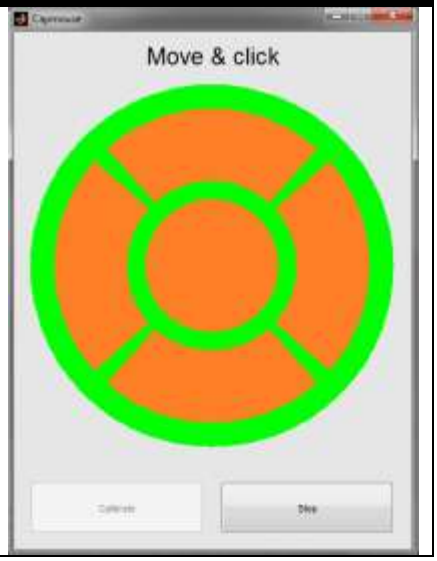
2.10.2. *Flow chart*



The move state uses data from the 4 directions to move the cursor on the screen. If all the sensors in the 4 directions are touched, then the mouse stops moving and enters the lock state. If they remain locked for a fixed period of time, a mouse click will be triggered. If the fixed period of time is not reached, then the mouse leaves the lock state and enters the move state again. To calculate step size of the mouse pointer movement on the screen, a lookup table is used. The counter acts as an index in the lookup table. The returned values from the lookup table are then added or subtracted from the current mouse pointer location. The new coordinates determine the new mouse pointer location and thus a movement on the screen.



2.10.3. *Graphical user interface*

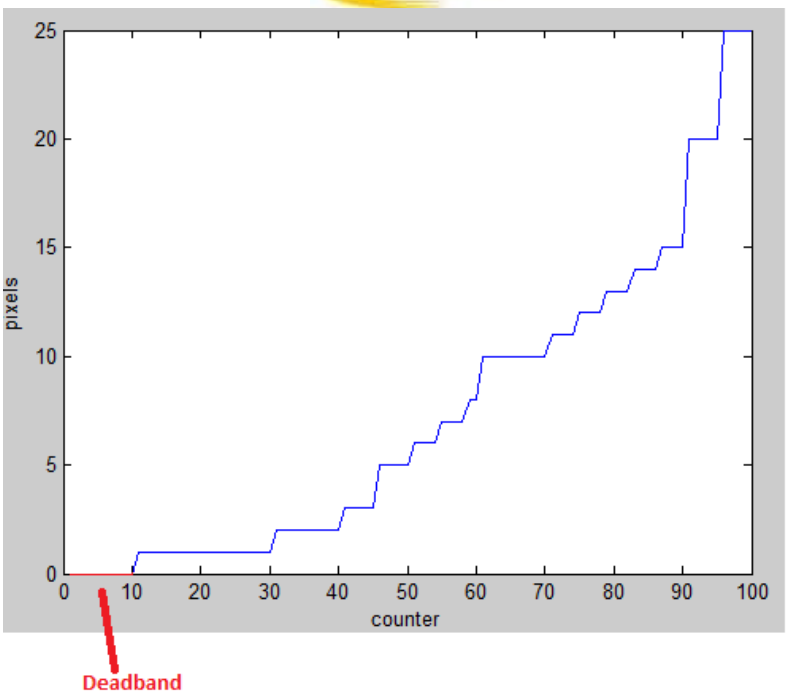
Move	Lock	Click
		

2.10.4. *Lookup table*

The lookup table is a table that uses the counter as an index. It contains 100 values. Each of those values represents the number of pixels the mouse will have to move. Since it contains 100 values, the index cannot be higher than 100. The dead band is also incorporated in the lookup table.

In the last user tests it was found that all four subjects found a movement of 1 to 10 pixels relatively slow, and a movement of 25 pixels pretty fast. These results were used to make a first draft of the lookup table.

The lookup table is hard coded into the software and determined empirically. In future releases it should be possible to make this lookup table adjustable trough a set of parameters.

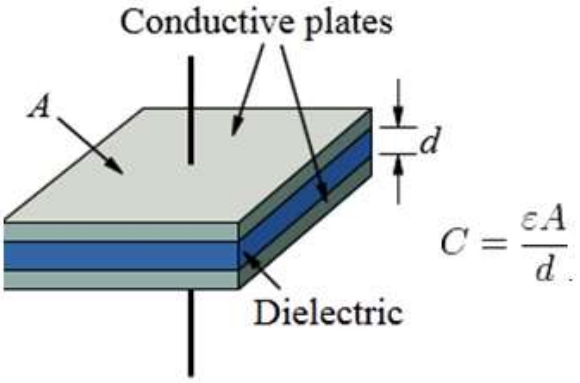

25
20
15
10
5
0
pixels
0
10
20
30
40
50
60
70
80
90
100
counter
Deadband

2.11. Why need "real time" calibration?

Capacitive sensors tend to be sensitive to temperature and humidity. Since the sensor will be placed close to the skin of the user it is highly probable that the values might deviate over time due to skin temperature and sweat. If the values deviate over time this means that the received values become unreliable. Unreliable data can become a potential danger especially in a wheel chair environment.

2.11.1. The theory behind the calibration code

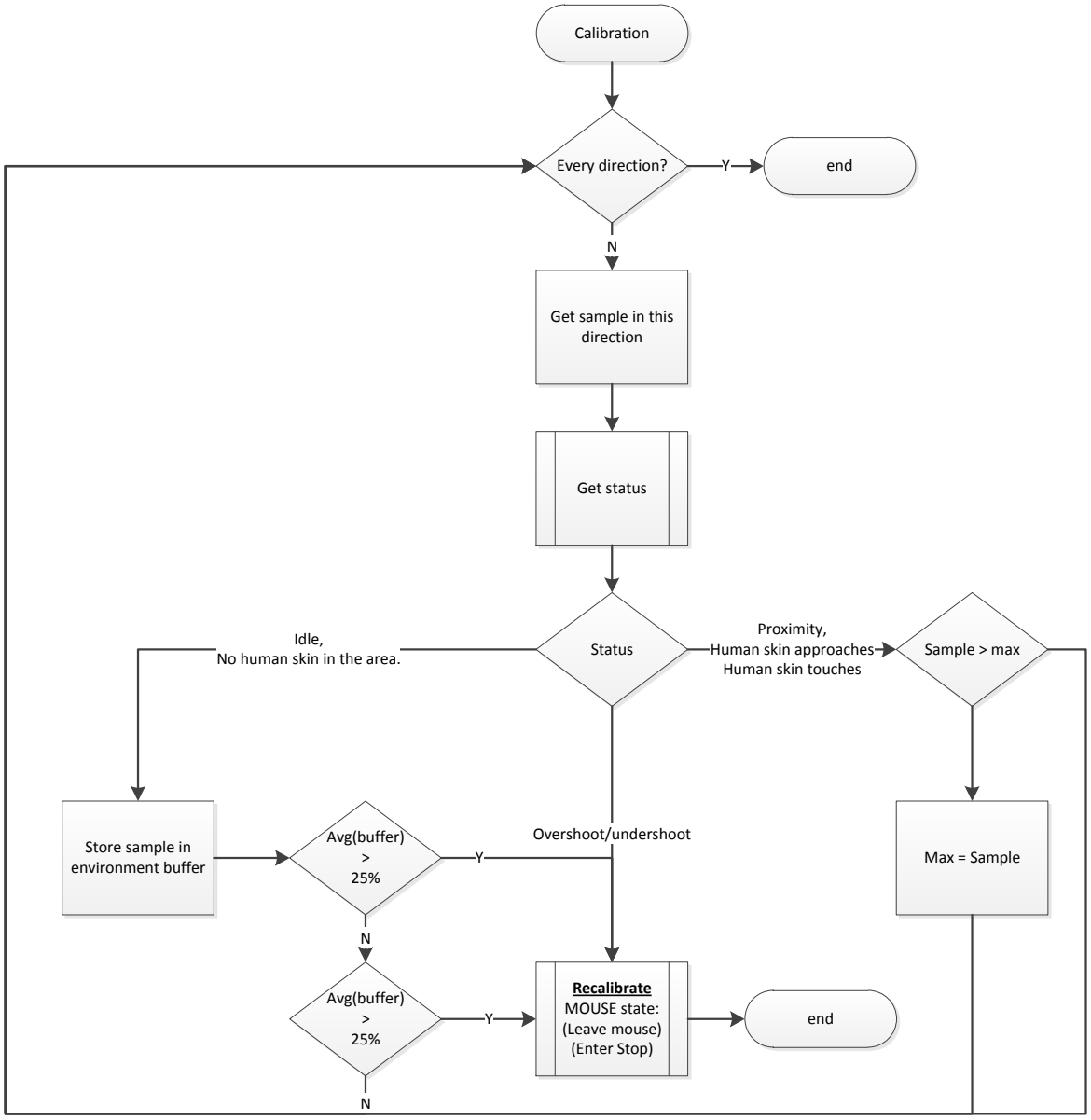
Thinking out a calibration routine for this problem is tricky. It is a tricky problem because of the nature of capacitive sensing. The air between the human skin and the copper sensor plate is a dielectric. If the dielectric remained constant there would not be a problem. Unfortunately especially the humidity in the air can vary. Since the impurities in water are good conductor, the electric field lines will find it easier to penetrate the dielectric and thus the dielectric constant (or permittivity) changes. If the dielectric constant changes, the capacitance changes also. This is a mathematical certainty since a parallel plate capacitor can be described by the following equation.



So if the environment changes, the capacitance will change. Of course if the distance from the finger to the plate varies, then the capacitance also changes. The main problem is that the algorithm will have to determine if the capacitance change is due to an environmental change or due to a finger that is approaching the sensor plate. There is one parameter that can be exploited to decide if the it is an environmental change or not. It can be assumed that environmental conditions change relatively slow over time. The touch of human skin is relatively fast. We can exploit this parameter but the bottom line is that this is a relative parameter. We would like to define this parameter with a number. That is the tricky part. Because a approaching the sensor in a relative slow manner, will make the calibration algorithm think that it is dealing with an environmental change.

2.11.2. *The theory in a flowchart*

First try to determine if a change is due to the environment or not. If it is due to the environment, save the average environmental value. Store the peak values when touching. If the peak value goes over the 150% of the predefined maximum, it is time to recalibrate. If the environmental value is greater or smaller than 25% of the estimated minimum, it is time recalibrate. Finally let the user know that it is time to recalibrate. Later on instead of letting the end user know of that the sensor needs recalibration, the sensor will automatically use these values to correct its own offset.



Four samples need to be collected. Based on this sample, the algorithm will decide if:

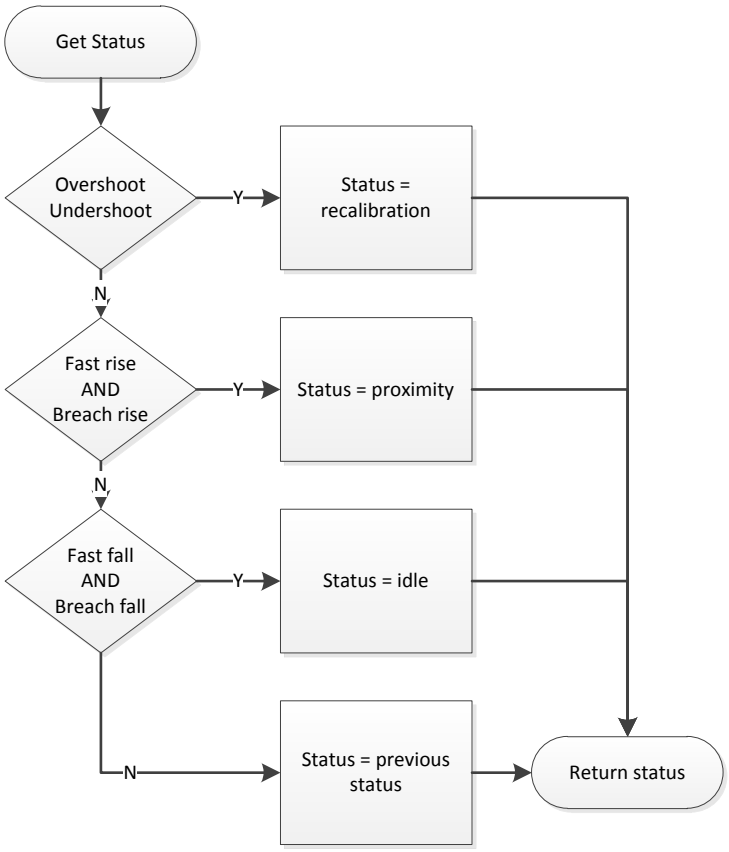
- No conductive surface is in the nearby area of the sensor plate. → status = IDLE
- A conductive surface approaches the sensor plate. → status = PROXIMITY
- A conductive surface touches the sensor plate. → status = PROXIMITY
- The sample is out of range. (sample > 150%, sample < -50%) → status = OVER/UNDERSHOOT

Based on the status of the pulse, the appropriate action is executed. Recalibration is in this stage is shutting down and let the user recalibrate by going through the entire program. This means downloading, normalisations etc…

Determining the status of the pulse is done through 6 methods that return a boolean.

- isOvershoot() = sample value > 150% of max
- isUndershoot() = sample value < -50% of min
- isFastRise() = (sample value – previous value) > 0.01
- isBreachRise() = sample > 25%
- isFastFall() = (sample value – previous value) < -0.01
- isBreachFall() = sample < 25%

To get an idea of the speed it is necessary to take the derivative of the input data in real time. A derivative is defined as the change in y divided by the change in x. This derivative expresses speed in the form of a number. The threshold that determines "slow" from "fast" was determined by the last user tests. The value 0.01 was found based on the last user tests. When the skin of the subject approached the sensor plate a deviation greater than 0.01 took place between consecutive values. The percentages are determined empirically since you have to start somewhere.



Proximity:
If an action is seen as a "fast rise" movement AND the sample value is greater than a certain threshold then it is highly probable that it is a conductive area that is approaching the sensor plate. The algorithm switches into "proximity mode".

Idle:
If an action is seen as a "fast fall" movement AND the sample value is smaller than a certain threshold then it is highly probably that the conductive area has moved away from the sensor plate. The algorithm switches to "idle mode".

Recalibrate:
If a value is detected that crosses the extremes then immediate action is taken by recalibrating

3. Index