PRESSURE SENSOR ARRAY INFUSED MATTRESS WITH NEURAL NETWORK POSTURE CLASSIFICATION

A thesis presented in partial fulfilment of the requirements for the degree of
Master of Engineering
in
Mechatronics
at Massey University, Albany,
New Zealand.

Matthew Krull

2022

Abstract

Sleep is an essential physiological process and is vital for human health and well-being. There is an undesirable trend of poor sleep hygiene, especially insufficient sleep. Sleep monitors can help bring to light this essential nighttime activity. The Comfort Group® (TCG), Australasia's largest mattress manufacturer seeks to embed a sleep posture monitor into a mattress as this is a platform used in almost every home. A full body posture monitor can provide richer information about the body than most movement-based sleep monitors on the market. There is particular interest in embedding a Pressure Sensor Array into a mattress because of the amount of sleep information that can be extracted from a pressure image. The renewed interest in this already-proven technology comes with the growth of computing power and machine learning algorithms. Today, features such as body shape, limb position, joint angles and even breathing rate can be inferred from the sleeper. However, they are not cost-effective for the consumer market. Here, we demonstrate a cost-effective pressure sensor array that is embedded into a consumer mattress. Basic posture detection of left-lateral, right-lateral and supine is shown using an artificial neural network. An accuracy rate of 99.1% is achieved. Having a cost-effective mattress-infused platform for the consumer will increase sleep hygiene in society and open the doors to a larger dataset for further analyse. For the scientific community, this larger dataset has the potential to produce a higher fidelity of insights into societal sleep.

Acknowledgements

Professor Johan Potgieter	-	Primary Mentor
Doctor Steven Dirven	-	Initial Mentor
Anna Wilson	-	Former employee of The Comfort Group - Asia Pacific
Simon Rogers	-	Former employee of The Comfort Group - Asia Pacific
Ashe Seatter	-	Business Manager at The Comfort Group - Asia Pacific
Stefan Geertsema	-	Chief Financial Officer at The Comfort Group - Asia Pacific

Contents

List of Tables

List of Figures

Acronyms

ANN Artificial Neural Network. x, 27–29, 63, 66, 69–71, 78, 79

API Application Programming Interface. 67, 71, 78

ASCII American Standard Code for Information Interchange. x, 60–62, 64

BLE Bluetooth Low Energy. 55, 61, 62

BMI Body Mass Index. 2, 27

BOM Bill of Materials. 48

BoMI Body Mass Imaging. x, 63, 70, 71

CDC Capacitance-to-Digital Converter. 48, 50, 56, 59, 60, 62, 77, 79

CNN Convolutional Neural Network. 18, 28, 29, 79

COM Communication. 64

CPU Central Processing Unit. 59, 62

CSI Channel State Information. 18

ECG Electrocardiogram. 14, 16, 20

EEG Electroencephalogram. 15

EMD Earth Mover's Distance. 27

EMG Electromyography. 14

FMCW Frequency-Modulated Continuous-Wave Radar. 18

FMRI Functional Magnetic Resonance Imaging. 15

FSR Force Sense Resistor. 19

GERD Gastroesophageal Reflux Disease. 28

GPIO General Purpose Input/Output. 55, 56, 59, 62

GPIOTE General Purpose Input/Output Tasks and Events. 59

GUI Graphical User Interface. x, 57, 63, 64, 70, 72, 78

IC Integrated Circuit. 26, 48–51, 55–57, 59, 77

ICSD International Classification of Sleep disorders. 9, 11

IDE Integrated Development Environment. 57

IMU Inertial Measurement Unit. 4, 15, 16, 55

IoT Internet of Thigs. 2

IRM Inception-Residual Module. 18

KNN K-nearest Neighbours. 28

LHS Left-Hand Side. 66

MCU Microcontroller Unit. 52, 55–57, 77

MDF Medium-Density Fibreboard. ix, 34–36

ML Machine Learning. 27–29, 71, 78, 81

MRI Magnetic Resonance Imaging. 15

nRF Nordic Radio Frequency. 57, 62, 77

OSA Obstructive Sleep Apnea. 11, 12, 15

PCB Printed Circuit Board. 39–41, 43, 44, 48, 55, 73, 79

PI Pressure Image. 2, 3, 6, 8, 20, 78

PLMD Periodic Limb Movement Disorder. 28

POSA Postional Obstructive Sleep Apnea. 12, 13, 28

PPI Programmable Peripheral Interconnect. 59, 62

PSA Pressure Sensor Array. ii, 2–4, 6, 8, 15, 20–23, 25, 27–29, 50, 51, 56, 62, 70, 77–79, 81

PSAM Pressure Sensor Array infused Mattress. x, 4, 31, 36, 38, 39, 45, 46, 48, 51, 55, 56, 58, 63, 64, 67, 70, 71, 74, 78, 80, 81, 110

PSG Polysomnography. v, 3, 11, 14, 15

PU Pressure Ulcer. 11, 13, 16

RF Radio Frequency. v, 18, 19

RHS Right-Hand Side. 66

RLS Restless legs syndrome. 11, 28

SDK Software Development Kit. 57, 62, 77

SES Segger Embedded Studio. 57

SOC System On a Chip. 55, 57, 61, 62

SPDT Single Pole Double Throw. 49, 52, 55, 56, 62

SPI Serial Peripheral Interface. 55, 59

SRBDs Sleep-Related Breathing Disorders. 14

TCG The Comfort Group®. ii, 1–6, 8, 26, 37, 38, 73, 78

TDC Time-to-Digital Converter. 50

TPU Thermoplastic Polyurethane. 31, 37, 45, 77

UI User Interface. 63, 64

USB Universal Serial Bus. 59, 60, 62, 64

WPF Windows Presentation Foundation. 63, 70

XAML Extensible Application Markup Language. 63

Chapter 1

Introduction

1.1 Quality Sleep Is Vital

Sleep is vital for our health and well-being. There is no doubt that sleep is an essential physiological process for survival. Good sleep is essential for many bodily functions to perform optimally. This includes functions such as cellular restoration in muscles, tissue growth, long-term memory formation and effective immune responses. On another level, sleep affects our emotional well-being and provides essential mental composure to help subdue stress and anxiety.

It is difficult for humans to monitor their sleeping behaviours. It is only once they have awoken that they become more aware of the quality of their sleep for that night. Having a system monitor a person's sleep and report it to them in the morning would be valuable and even enlightening. Over time, having access to those results could curb unhealthy trends. On a societal level, such a system could prevent higher rates of disease, stroke and mortality. It could lower rates of depression and next-day thoughts of suicide. It could also help maintain high levels of cognition and long-term memory formation (see section 2.2.1).

Mattresses are synonymous with sleep and are ubiquitous in the developed world. As such, there is a platform already available in which to embed a sleep monitoring system.

1.1.1 Working with The Comfort Group

The Comfort Group® (TCG) is Australasia's largest mattress and foam manufacturer, with brands such as Sleepyhead®, SleepMaker® and Dunlop Foams®. They provide foam and flooring underlay for an industrial market as well as manufacture their own mattresses and bases. They also provide other consumer products such as pillows, toppers and bean bags. They are interested in infusing sensors into their mattresses to

gather data to improve their product proposition, create a better consumer purchasing experience, better match consumers with the correct mattress and provide their users with information about their sleep.

Figure 1.1: The Comfort Group® and subsidiary companies.

Sleep is an active area of investigation and TCG would like to be part of it. They are especially concerned with sleep position, postural alignment and its effects on sleep and quality of life. Industries are heading toward sensor-infused smart devices that are connected to the internet, known as the Internet of Thigs (IoT). They have expressed particular interest in infusing Pressure Sensor Arrays (PSAs) into their commercial mattresses for monitoring posture. This interest stems from the many physiological factors that can be measured or inferred from the images they produce.

1.1.2 Pressure Image Potential

Pressure Images (PIs) from a PSA are a contact and weight image of a resting body on the sensor. Each pixel has a location and pressure associated with it. A number of physiological and behavioural features could be extracted from the images. These are:

- Areas of high and/or prolonged pressure. This information is useful in decubitus ulcer prevention in hospitals and old age care [1], [2].
- Body Mass Index (BMI) estimations and changes over time. Fat or muscle change could be inferred based on the change in location and body shape.
- Other mass measurements such as individual limbs, torso, pelvis stomach and so forth.
- Long-term mass gain and body shape development in childhood and adolescence.
- Body dimensions and positions such as limb length, limb position and joint angle. Spinal alignment is a realm for future investigations.
- Overall posture with changes in them and duration of them. The proportion between each posture can also be used as an analysis tool, especially for long-term sleep studies.
- Partner movement and disturbance for a couple's mattress

- Inferences about sleep stages and circadian rhythm.
- Breathing rate[3], [4] and basic lung capacity estimations.
- Pre-sleep or in-bed non-sleep activities such as multimedia device usage, reading or general seated activities. The posture of these activities could be inferred as well.
- Other mattress uses not identified in the items above.

All these items highlight the great potential a PSA has, especially for monitoring sleep. Such information is useful for TCG, the mattress user and the scientific community.

The Comfort Group®: In the short term, TCG would primarily use the information to more appropriately match current bed designs with body shapes and weights. Using a live PI, they can recommend an appropriate mattress to a customer that provides the best pressure distribution for their body. In the long term, collecting user feedback would refine their recommendation. It would bring about better mattress endorsements, increase the perception of comfort, and bring about long-term satisfaction for their consumers. Mattresses in the past have largely been developed based on subjective feedback. The data provided by a PSA would greatly ground their future design decisions with objective information. Future mattress designs would reflect the knowledge of a customer's real needs.

Device Users: Such a sleep monitoring system would present valuable insights about a multitude of physiological occurrences during their sleep and behaviours surrounding mattress use. It would bring personal awareness and help to educate individuals about what might be causing poor rest. It could become their sleep coach with best practices recommended to them through their smartphones. This would be especially important for those who are in their developing years such as children and adolescents. For couples, it could help them see what partner disruptions might be contributing to improper rest. It will replace several sleep monitoring systems and take up no extra space in the bedroom because it is built directly into their mattress. Sleep trackers that use actigraphy or ballistography would mostly be superseded with a full body pressure image.

The Scientific Community: The scientific collective will greatly benefit from the wealth of information from such a system. As mentioned earlier, it is anticipated that this will be a cost-effective in-home solution that can provide more information than many other in-home sleep monitoring systems. Much of the data used to determine sleep quality today is sourced through a Polysomnography (PSG)

exam, which is usually conducted in a sleep lab with many sensors - most of which are attached to the body. Actigraphy is an in-home option that measures general movements. Smartphones and watches use their built-in Inertial Measurement Unit (IMU) to act in this way too. Long-term sleep studies rely on patients to keep a sleep diary [5], requiring vigilant entry efforts making it vulnerable to human foibles. A Pressure Sensor Array infused Mattress (PSAM) could take the place of many of these sensors as a non-contact 'set and forget' system. It would be used anytime someone lies on their bed. Additionally, with the approach of making it cost-effective for the average mattress consumer, it will be accessible to more people. In return, this would supply a large data set of sleep behaviours that would provide insights into societal sleeping patterns and trends that have otherwise been difficult or impractical to investigate.

The aforementioned reasons establish that an effective posture monitoring system like an PSAM has benefits for many parties. PSAs are commercially available but are prohibitively expensive to the average mattress consumer. There have been none to be found that are embedded into a mattress, only ones that cover a mattress. These are useful because they can be placed over any mattress, but such a topper can be felt or easily damaged. Having them built into a mattress will give it an invisible feel and be no extra hassle.

1.2 Scope and Motivation

As sleep monitors are a new area of investigation for TCG and they have expressed interest in PSAs, a broader scope has been given to cover other potential posture monitoring devices that have not been investigated. That scope can be expressed as:

> **Investigate and produce a cost-effective posture monitoring platform that can be embedded into a mattress. Its purpose is to monitor sleep behaviours (especially posture) and present the data to the user in a way that leads them to obtain a higher quality of sleep.**

It has a secondary purpose in assisting TCG with making an improved bed-to-body recommendation, thereby creating a better overall consumer purchasing experience (as discussed in section 1.1.1 above).

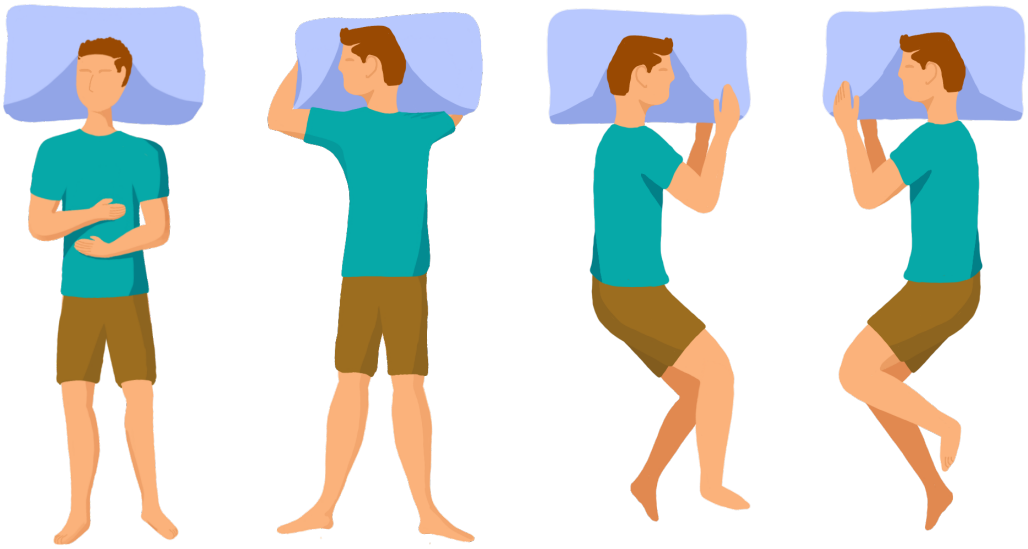
To establish clear objectives, it is imperative that the terms 'cost-effective' and 'posture monitoring' are defined and outlined.

1.2.1 Defining Cost-Effective

A mattress is a high-value asset that can be used for five to ten years. The average mattress produced by TCG and sold in the New Zealand market costs approximately $700 NZD to manufacture. Such mattresses sell for approximately $2500 - $3000 NZD. Exact values are not supplied due to market sensitivity. Internal discussions with TCG's product development division have concluded that an acceptable upper bound for a posture monitor would be $250 NZD. Naturally, creating the monitor for the lowest price possible means it will be accessible to a greater proportion of consumers. Still, this price helps to define what cost-effective is within the New Zealand commercial mattress market.

1.2.2 Defining Posture Monitoring

On a rudimentary level, sleeping positions can be categorized into three predominant postures: supine (fig 1.2a), prone (fig 1.2b) and lateral (fig 1.2c & 1.2d). Other postures can be considered derivatives of these, such as (but not limited to) left-lateral (fig 1.2c), right-lateral (fig 1.2d), left and right foetus [6], or fowler (inclined) for supine posture. The same, or very similar positions have different names in literature [7, 8, 9]. For the quest of determining posture in this investigation, at least three of the basic categories should be identified and classified. These are left and right lateral and supine. It is an important preliminary milestone for any posture monitoring device.



(a) Supine (b) Prone (c) Left-lateral (d) Right-lateral

Figure 1.2: Basic sleeping postures

1.3 Objectives & Deliverables

From the thesis focus (section 1.2) and desires from TCG (section 1.1.1) the key objectives are set as follows;

1. Investigate and develop a cost-effective sleep posture monitoring unit that can be embedded into a consumer mattress.
2. Investigate pressure sensor arrays as a potential and sufficient posture monitoring system.
3. Perform an analysis of the data to determine basic sleeping postures.
4. Review the effectiveness of such device and data analysis technique in determining posture and other physiological sleep factors.

The project should then deliver;

- A prototype of the monitoring device.
- Software to analyse and display the pressure image and identify basic sleep postures.
- A basic cost analysis for producing such a device including the potential manufacturing methods to achieve it.

1.4 Summary

Sleep is an essential physiological process and vital for human health and well-being. Sleep monitors can help bring to light behaviours that occur while asleep. The Comfort Group® (TCG), Australasia's largest mattress manufacturer is seeking to embed a sleep posture monitoring device into mattresses. It will provide more information than the standard sleep monitor and will help educate any user about their sleep behaviours. There is particular interest in Pressure Sensor Arrays (PSAs) because of the richer set of physiological information they could provide.

Multiple parties will benefit from the data such a device will provide, such as the device user, The Comfort Group® (TCG) and the scientific community. Seeking a cost-effective platform for the average consumer would increase the volume of pressure image information. The greater volume of data would produce insights into societal sleeping patterns and trends that have otherwise been difficult or impractical to investigate. The deliverables for a successful project are a prototype system with analytical software for posture monitoring. The software should have the ability to display the PI and identify a basic posture.

1.5 Thesis Outline

The thesis is outlined in the following way. First, a literature review is performed. The focus of the review begins with a broad investigation of sleep, what affects it and how it is studied. It then narrows towards sleep posture monitoring techniques and technologies. Then, an investigation regarding pressure sensor arrays is conducted next and lastly, data analysis techniques are identified to assist the final goal of basic posture recognition and classification.

The chapters after this cover the hardware and software components of the monitoring device. Sections 4 and 3 discuss hardware development and sections 5 and 6 discuss the software.

Later chapters review the system with its associated costs and potential manufacturing techniques.

Each chapter walks down the development path, listing out possible solutions to meet the design requirements and investigating which are sufficient. A choice is then made about which is the most appropriate. Justifications are given throughout the whole process. At the end of each chapter is a summary containing key achievements as well.

Chapter 2

Literature Review

2.1 Introduction & Structure

This literature review begins by briefly exploring sleep and the effect it has on humans. The factors that influence it are listed; the beneficial and detrimental factors. Sleep is predominantly a physiological process that can produce behavioural manifestations. These indications can be evident in body movement and posture. As the objective of this project is to establish an adequate sleep posture monitoring method, particular interest is shown in body movement and posture. All this information will provide a baseline understanding of sleep and solidify the need for monitoring systems. These findings are presented in section 2.2.

Sleep monitoring systems are then investigated and their underlying technologies are identified. There is a gold standard for sleep studies today and it requires many physiological monitoring devices. These are listed and discussed.

The search then narrowed towards posture monitoring devices. Their sensing principles are explored with examples demonstrated in literature. The end of each section contains justifications for the techniques discussed as well as their applicability for use in the project. As The Comfort Group® has taken particular interest in Pressure Sensor Arrays (PSAs), they receive particular attention during the exploration. The search did not begin at PSAs, even though it is where the interest resided. This is to ensure that there is not a more effective method of sleep posture monitoring. This work is presented in section 2.4.

Later in the review, a PSA is justified as an adequate and appropriate solution for posture monitoring. From here, the search is directed towards finding the most appropriate pressure sensing technology for the desired use case. Their fundamental sensing principles are discussed along with their overall characteristics.

Lastly, the Pressure Image (PI) data from a PSA needs to be analysed for sleep postures. These findings are presented in section 2.7

2.2 Understanding Sleep

Sleep is an essential biological function with major roles in recovery, energy conservation, and survival [10]. Optimal sleep is vital for maintaining good physical and mental health [11]. It ensures that an individual can sustain a high quality of life. The American Thoracic Society has stated the following; "sleep appears to be important for vital functions such as neural development, learning, memory, emotional regulation, cardiovascular and metabolic function, and cellular toxin removal" [10]. To understand sleep we must consider the factors that influence its quality.

2.2.1 Sleep Quality

Many aspects influence sleep quality. These include internal factors - such as stress levels [12] and physical wellbeing [13], external factors - such as the resting environment[14] and diet [15], as well as behavioural factors - such as sleep habits and nightly routines[16].

Amongst all of these potential factors, the ones relating to movement and posture are identified and listed below. Ones that could also be identified by a posture monitoring device are listed as well.

Sleep Duration

The duration of an individual's sleep is one essential component of quality sleep. There is a goldilocks zone for sleep. For adults (at a population level), it is 7-9 hours [10]. Less than 6 hours or more than 9 hours per 24-hour period is associated with adverse effects [17]. For newborns to teenagers varies and is best outlined in figure 2.1

Insufficient sleep is one of the key contributors to poor sleep. As of 2012, 29.2% of US adults were sleeping less than the recommended 7-9 hours [10] at $\leq$6 hours [19], which is up 6.9% from 1985, when it was 22.3%.

Insufficient sleep has been defined as a curtailed sleep pattern that has persisted for at least three months for most days of the week. It is normally accompanied by complaints of sleepiness during the day [20]. It is also associated with the experience of unfavourable mental well-being [21]. Reduced sleep duration has been linked to 7 of the 15 leading causes of death in the United States, including cardiovascular disease [22], malignant neoplasm, cerebrovascular disease, accidents, diabetes, septicemia, and hypertension [20, 23]. Further, it has been found to contribute to a higher severity of next-day suicidal ideation [24]. Polling based on a validated questionnaire of ICSD revealed that 43.2% of participants suffered from insufficient sleep [25]. Using multivariate statistical analysis, Magee et al. [26] found that short sleep durations among Australian adults (between 18 to 64 years) were associated with longer working hours,

Figure 2.1: Sleep duration recommendations across the life span [18].

lower education levels, high levels of alcohol consumption, obesity, depression and anxiety. Overall, insufficient sleep in society can lead to adverse consequences in school and labour market performance [20].

Conversely, the effects of a surplus of sleep include poor cognitive function[27], diabetes, cardiovascular disease, heart disease, stroke and a higher risk of mortality [28]. Both an excess and lack of quality sleep lead to increased symptoms of depression [29] and obesity [10].

From all this, it can be seen that the right proportion of sleep is essential for proper human psyche and bodily operations.

Body posture and movement

There are a few aspects of body posture that affect the quality of sleep an individual has. Incorrect sleep positions held for a considerably long duration can result in spinal alignment problems. It has also been found that sleeping in a supine position can reduce back pain since it is the position in which the muscles have the least amount of work to do to maintain one's posture against the force of gravity [30].

It has been suggested that lateral sleeping might allow for more efficient brain activity. The glymphatic system is the brain's mechanism for waste clearance. The brain uses approximately 20% of the body's energy [31] and thus produces a considerable amount of metabolic waste. Cerebrospinal fluid is a clear fluid that flows into and out of the brain carrying waste material with it. This mental flush predominantly occurs

during sleep [32]. There is limited research on how sleeping position influences glymphatic operations in humans. However, studies in mice have shown that glymphatic transport is most efficient when the rodents sleep in a lateral position (on their side) [33].

Gastroesophageal Reflux Disease (GERD) is a condition when stomach acid and contents flow back up into the esophagus. It is a condition that can disrupt sleep [34]. Studies have shown that respondents with nighttime GERD were more likely to experience sleep difficulties, including induction and maintenance of sleep. [35] GERD sufferers typically experience more esophageal acid exposure while on their right recumbent position relative to the left. Gastroesophageal reflux is also more frequent in the supine position [36, 37].

It has also been said that sleeping in the supine position is more likely to bring on symptoms of sleep paralysis [38].

2.2.2 Sleep Disorders

sleep disorders are conditions that prevent an individual from sleeping well. They can affect the duration, timing and overall quality of sleep obtained. It results in daytime tiredness and decreased cognitive ability. The ICSD is a key reference work in diagnosing sleep disorders. Michael J. Sateia produced a highlights publication [39] based on ICSD. Sleep-related movement disorders are discussed in it. Some of them are listed in the subsection to follow. Later subsections will discuss other position-related disorders such as Obstructive Sleep Apnea (OSA) and Pressure Ulcer (PU)s.

Sleep-Related Movement Disorders

Certain Body movements during sleep can be a physical manifestation of a sleeping disorder and can aid in the diagnosis of the disorder. Table 2.1 is a compilation of sleep-related movement disorders. The first two are highlighted and discussed in more detail below.

Restless legs syndrome (RLS): This is a newly termed sleep disorder [40] in children that is characterized by at least five body movements per hour that last throughout the night [41]. It reduces sleep time, increases the number of awakenings and has a significant impact on daytime energy and behaviour. Studies have shown that 7.7% of children who have undergone clinical sleep studies have RLS [41].

Periodic Limb Movement Disorder (PLMD): PLMD is a frequent finding in sleep studies such as Polysomnography (PSG). It is estimated to affect $4-11\%$ of adults

Disorder
Restless legs syndrome
Periodic limb movement disorder
Sleep-related leg cramps
Sleep-related bruxism
Sleep-related rhythmic movement disorder
Benign sleep myoclonus of infancy
Propriospinal myoclonus at sleep onset
Sleep-related movement disorder due to a medical disorder
Sleep-related movement disorder due to a medication or substance
Sleep-related movement disorder, unspecified

Table 2.1: Sleep-related movement disorders [39]

[42]. This disorder is diagnosed when adults experience greater than fifteen movements per hour or more than five for children. It must also be accompanied with sleep disturbances or other impairments to be considered a disorder [39]. PLMD is associated with cramping, twitching or jerking of the lower limbs. It disrupts sleep even if a person is not awoken. It typically leads to daytime fatigue.

Sleep Apnea

Sleep apnea is a disorder where breathing briefly pauses during sleep. It is the most common sleep disorder tested for in sleep centers [43]. The most common form of sleep apnea is Obstructive Sleep Apnea (OSA). It consists of recurrent episodes of partial or complete upper airway collapse during sleep [44]. It causes sleep disruption due to increased ventilatory effort as a response to the air closure [45]. It mostly occurs when there is hypotonia, or decreased muscle tone in the upper-airway muscles. An episode of paused breathing can last for several seconds.

It is reported to affect 4% of adult males and 2% of adult females [46]. In New Zealand, it is twice as common in Maori males compared with non-Maori males [47]. Maori males also tend to present more severe forms of OSA Syndrome [47].

When most apneic episodes can be attributed to a sleeping position, it is known as Postional Obstructive Sleep Apnea (POSA) [48]. Most POSA patients experience their breathing abnormalities while sleeping in the supine position [49]. Lying in such a position, along with gravity, alters the shape and size of the upper airway causing impeded airflow. It is thought that over 50% of people with obstructive sleep apnea have positional obstructive sleep apnea that is supine related [50]. Untreated POSA can lead to low blood oxygen levels, elevated blood pressure levels leading to strokes, heart attacks, type 2 diabetes and reduced lung volume [48].

Pressure Ulcers

Pressure Ulcers (PUs) are chronic skin wounds that pose a significant threat to public health and are a burden to a nation's economy. They develop when prolonged pressure is exerted on an area of skin, causing restricted blood flow to the cells (ischemia) and eventually leading to cell death. They can easily be prevented by reducing or relieving the pressure on stressed skin. Most medical institutions prevent PU formation in their patients through periodic inspections and continual repositioning of pressure-prone areas [51]. This requires caretakers and hospital staff to be constantly mindful of their patient's condition. This is not in and of itself, but when this task is accompanied by a myriad of other patients and responsibilities, it can easily be lowered on the priority scale.

PUs cost the United States 11 Billion per year [52], Australia an estimated A$983 million [53] and the UK, €1.4 - €2.1 billion [54]. These figures highlight the fact preventing PUs has a significant potential for health care savings around the world.

Somnambulism

Somnambulism or sleepwalking is one of the leading causes of sleep injury. It is a transitionary condition that is usually grown out of when aging out of childhood, but if it isn't it can become especially dangerous for adults. Individuals with this disorder can get themselves into hazardous situations, such as walking into doors, tables, chairs or even leaving the house [55]. Sleepwalking is mainly diagnosed on the patient's clinical history [56]. Sleep posture devices can play a key part in observing individuals with this disorder by monitoring bed occupency throughout the night.

2.2.3 Section Review

There are many factors that affect the quality of an individual's sleep. The duration of an individual's sleep is a key factor for quality sleep. Optimal sleep is between 6-9 hours over a 24-hour period. Insufficient sleep leads to daytime drowsiness and unfavourable mental well-being. Body posture during sleep contributes to proper spinal alignment, metabolic waste removal and aid in reducing Gastroesophageal reflux episodes. Body movements throughout the night can also be behavioural manifestations of sleeping disorders such as restless leg syndrome and periodic limb movement disorder. Postional Obstructive Sleep Apnea (POSA) is a condition characterised by brief breathing pauses in sleep and which can be significantly reduced when maintaining a non-supine sleeping position.

A movement and posture monitoring system would be useful in detecting and keeping track of these disorders.

2.3 Sleep Monitoring

Sleep is arguably an unconscious state of the mind [57]. It is only after the individual has awoken that they become fully aware of their slumber. There are many sleep monitoring devices that record and quantify different physiological factors during this night-time rest and the behaviours surrounding it.

2.3.1 Polysomnography (PSG)

Polysomnography (PSG) is the gold standard for diagnosing sleep disorders, especially Sleep-Related Breathing Disorders (SRBDs) [43]. A Type I (Level I) sleep study involves recording body position, limb movements, airflow, respiratory efforts, heart rate and rhythm, oxygen saturation and sleep stages. The sensors that are used to measure these physiologic parameters are:

- For airflow:
  - Two qualitative **airflow sensors** are used to measure the airflow and respiratory effort for apneas.
  - An **oronasal thermal sensor** is used to measure changes of inspired air vs expired breath.
  - A **nasal pressure transducer** is used to measure pressure changes during inspiration and expiration. It is also used to monitor airflow and identify hypopnea.
- For oxygen:
  - A **pulse oximetry** is used to measure the oxygen content in the blood and assist with scoring hypopneas.
- For heart rate and rhythm:
  - An **Electrocardiogram (ECG)** is used by taking electrical measurements of the heart with electrodes.
- For limb movement:
  - An **Electromyography (EMG)** is used by making electrical measurements of muscles with electrodes.
- For body position/posture, one or many of the following are used:
  - Analysis of video recording by a sleep technician.
  - A body position monitor.

  - A Pressure Sensor Array (PSA).

  body position monitoring is important for patients with OSA.

The following are additional physiological measurements that can be taken but are not required to be considered a PSG. These include:

- **Carbon dioxide** measurements for hyperventilation at the end of a breath or transcutaneously.
- An **Electroencephalogram (EEG)** measurement of brain wave activity using up to 256 electrodes. It is mostly used for patients who suffer from seizures.

2.3.2 Actigraphy

Actigraphy is a monitioring method that involves attaching a motion sensor such as a peizoelectric transducer [58], accelerometer or Inertial Measurement Unit (IMU) to a patient's body. It is a cost-effective method of capturing body movements and, and is a simplified in-home alternative to PSG. It is a non-invasive method that can be used for long-term sleep analysis in a natural setting [59]. Algorithms have indicated that sleep cycles can be inferred that are consistent with polysomnography studies. Dick et al. demonstrated an overall accuracy of 85.9% [60]. Today, there are smart phone apps and smart watches that use the inbuilt Inertial Measurement Unit (IMU) to monitor sleep like an actigraph [61, 62].

2.3.3 Neuroimaging

Functional Magnetic Resonance Imaging (FMRI) is a method of imaging brain activity associated with blood flow, similar to MRI. As many physiological processes occur during sleep, FMRI scans are another tool to help understand what occurs within the brain. It is a large medical device that often disrupts natural sleep because of loud noises [63] (for some machines) and the potential for claustrophobia to be induced due to a confined space. They are room-scale pieces of equipment that are costly to run.

2.4 Sleep Posture Monitoring

Body movement and posture monitoring are two of the many aspects involved with PSG. To reiterate, these two monitoring techniques provide useful information about bodily movements that stem from internal physiological occurrences and behaviours surrounding sleep.

In this next section of the literature review, posture monitoring systems are focused on. They have been grouped based on their proximity to the user and how the devices

interact with them. These groups are attached sensors, unattached sensors and contactless sensors. They are then subgrouped based on the major sensing principle they use. Some involve sensor fusion or multiple sensing mechanisms to determine posture. Each is analyzed for its suitability within the focus of the project and if it can be embedded into a mattress for commercial use .

2.4.1 Attached Sensors

These are sensors that must be attached to the subject being monitored. They can also be attached to an item that is attached to the subject, such as a piece of clothing.

Accelerometer or IMU An accelerometer measures the acceleration that is subjected to it. It can detect the change in movement and determine the direction of gravity. An Inertial Measurement Unit (IMU) is an accelerometer with the addition of a gyroscope. A gyroscope measures rotational acceleration. Basic postures have been detected using an accelerometer by attaching it to the subject's chest and observing the amplitude of each axis as the patient rotates[64]. An average accuracy of 99.16% was achieved for 5 postures. An IMU has also been used to determine in-bed postures for PU prevention [65]. Chang et al. also demonstrated posture detection based on an accelerometer's orientation with gravity [66].

Sleep guard is a software-based system that uses the IMU inside of a smartwatch to determine four basic sleep postures. It can also determine hand positions, micro body movements, illumnination conditions, rollovers and acoustic events such as snoring [67].

An accelerometer and IMU have the benefit of being small and low-cost. They are sensitive to movements and can determine sleep orientation. However, they must be attached to the wearer each night. Additionally, they can only give coarse posture information - up to 5 postures.

Electrocardiogram An electrocardiogram ECG measures electrical signals of the heart while attached to the skin. A two-channel ECG and accelerometer have been used to demonstrate that the electrical heart axis is affected by body positions. Body position classification accuracy with just two heart axis features was 52.59% [68].

Another example of an electrocardiogram is discussed as an unattached sensor in section 2.4.4 later on.

The major downfall of using an electrocardiogram is that it is not consumer friendly. It requires electrodes to be placed around the body for each use. This is impractical to do every night. The cables can also disrupt sleep and become potential sites for PU formation.

2.4.2 Contactless Sensors

These sensing methods are not attached to the body of the user. Rather, they are placed in the proximity of the user to observe their activity. Their primary sensing mechanism is electromagnetic radiation. What differentiates them is the wavelength used and whether it is image-based or not.

Image Based Systems

Vision Systems: These sensors use a camera of visible light[69] (400nm - 1000nm [70]) or of near-infrared wavelengths [71, 72] ($900nm$ - $2\mu m$, camera dependent). The image produced is evaluated using machine learning or other algorithms to determine sleep posture. Monitoring cameras have always raised privacy concerns and are widely perceived to be intrusive. They also suffer in low light situations unless they are accompanied by a light source. Akbarian et al. [71] demonstrated head and posture detection using an infrared camera. Camera monitoring systems are also less effective when the subject uses body covers such as a blanket. Akbarian et al were able to determine lateral versus supine head position with 92% accuracy, and 94% F1-Score and lateral versus supine body position with 87% accuracy and 87% F1-Score.

Thermal Imaging Cameras: This imaging-based system (($2\mu m - 12\mu m$, camera dependent)) can detect wavelengths emitted by the human body ($\approx 9.3\mu m$). The emitted radiation and overall body temperature can also heat up the immediate surroundings such as coverings and cause them to emit thermal radiation as well. The insulating ability of the covers determines how defined a thermal signature for posture inferencing can be. Chen et al. [7] demonstrated posture detection with a thermal image even with a covering. No information regarding the thickness and type of covering was given. A thermopile array was used to produce the image, which is a lower-cost alternative than full infrared cameras. They can be considered low-resolution infrared cameras.

Depth cameras produce an image whose pixel values represent the distance of a point away from a camera. Usually, this point represents a surface Tam et al. [73] demonstrated the use of an infrared depth camera to determine coarse-grained (four-posture) and fine-grained (seven-posture) classification with an F1-score of 88.9% accuracy. A thick blanket was also demonstrated with the accuracy of classification dropping by 8.9% in the worst case.

Li et al. [8] demonstrated posture classification with a depth map and CNN classification. Grimm et al. [74] also demonstrated posture recognition of subjects lying on a bed without a blanket.

Vision-based posture monitoring systems are non-contact and can provide great detail about the monitored subject. However, as mentioned earlier, cameras can be perceived to invade privacy, especially while someone is sleeping. They are also required to be placed a distance away from the subject. Thus, they cannot be embedded into a mattress.

Radio Frequency (RF) based Systems

These systems use radio waves as their means of sensing.

Millimeter-Wave Radar: This is a recent development as a posture monitoring system. Tao et al. [75] utilized the 77GHz - 81GHz waves for radar echo signals reflected by the human body, as well as range spectrum, Doppler spectrum, range-Doppler spectrum, azimuth angle spectrum and elevation angle spectrum. All of these items compose a 5 single-channel 2D radar map by splicing them together (within a fixed frame window). The data is then classified using a multichannel Convolutional Neural Network (CNN) and Inception-Residual Module (IRM). This achieved an overall posture classification accuracy of 87.2%. This is a promising technology. However, it cannot be integrated into a mattress as it needs to be placed at a 1.5m distance due to the sensor's field of view.

RF Signals (5.7GHz - 7.2GHz): Yue et al. showed the use of a Frequency-Modulated Continuous-Wave Radar (FMCW) to identify sleep position, sleep latency, sleep efficiency, sleep onset, sleep stage classification, respiration of multiple individuals and bed entries and exits[76]. It is a major advancement in sleep monitoring using radio waves. It is accomplished by recording the reflected radio waves off the human body and isolating the signal from other objects in the vicinity. Many layers of signal analysis were performed with data classification done by neural networks.

Similar to a millimeter-wave radar, it has to operate at a distance and cannot be integrated into a mattress. It is a technology that has yet to be proven in many non-scientific environments as well.

Wifi Signals: Using Wifi signals, Lui et al. [77] were able to determine sleeping posture and respiration in all but the prone position. It was achieved by using a transmitter-receiver pair and analyzing the Channel State Information (CSI). CSI describes how a RF link propagates from the transmitter to the receiver and reveals the combined effect of scattering, fading and power decay with distance [77]. More recently, Cao et al. [78] demonstrated 6 body movements. They were: turning over, lying down, raising an arm, raising a leg, curving legs and sitting down. This was achieved using CSI to train a bidirectional recurrent neural

network. Average accuracies of greater than 93.5% were achieved. The drawback of this system is that the analyzed body needs to be in-between the transmitter and receiver, which were 2.5 m apart.

A RF posture monitoring system has the benefits of being non-contact and can operate in the dark. However, they require complex signal analysis and can be computationally expensive. They also have to be a distance away from the user and, as such, cannot be placed inside a mattress. However, they do not cause the same privacy concerns that surround a vision-based system.

2.4.3 Unattached Sensors

These are sensors that make contact with the monitored subject but are not directly attached to them. For example, sensors in or on top of a mattress may come in contact with a subject, but they remain stationary when the subject leaves the sensor's proximity. Force and Pressure sensors are the main sensors in this category.

Force Sensing

Hsia et al. [79] demonstrated the use of 16 Force Sense Resistor (FSR) to determine three basic sleeping postures: left and right lateral and supine. The Force Sense Resistor (FSR) sensors are placed on top of the mattress in a line that spans the length of it. The sensors deform under pressure and provide the reading. The data was evaluated with statistical analysis and basian classification.

Barsocchi et al.[80] demonstrated another arrangement with 48 FSRs used on the staves of a bed frame. The pressure data was analysed with a multi-class logistic regression and a training dataset to determine 4 postures: left and right lateral, prone and supine. An accuracy rate of up to 95.8% was achieved.

Single FSRs are low-cost items however they cannot be placed in an array format with shared connections. However, They are thin sensors that can be embedded into a mattress. This can be a good low-cost solution for basic sleep monitoring by using just a few of them, but it is not feasible for higher fidelity posture detection. The high number of them means a higher number of connections and makes for a more labor-intensive manufacturing process.

Pressure Sensing

These are sensors that convert pressure stimuli into electrical signals. Arrays of them produce an image of the contact surface an object makes with it. They can be constructed ultilising a variety of principles, such as piezoresistive, piezoelectric, triboelectric and capacitive.

PIs from Pressure Sensor Array (PSA) are primarily used for detecting high areas of skin pressure for decubitusch ulcer prevention[2, 1, 81] and sleep posture recognition [82, 83, 84, 85, 8, 86]. Posture information can be extracted from moderate to high sensor count arrays. Clever et al. demonstrated this by making human pose and body shape estimation [87]. First from a simulated array, then demonstrated with an actual sensor array.

Pressure sensors can also be used for monitoring some physiological features such as breathing rate [3] and other non-sleeping behaviours surrounding bed use.

The benefits of PSAs are; they can be integrated into a mattress, they do not raise the same level of privacy concerns as vision-based systems and they do not emit radio signals which allow them to be used in RF-sensitive areas such as hospitals. PSAare a proven technology with commercial versions available. They are, however, still costly.

Thermal Sensing

An array of temperature sensors give a contact thermal image from the body heat that is produced. The amount of heating is determined by ambient temperature, the subject's temperature and the thermal conductivity of materials between the sensor and the body [**?**].

2.4.4 Electrocardiogram

An ECG is usually attached to a subject's skin, but Lee et al. demonstrated posture classification by placing a strip of 12 electrodes horizontally on a mattress surface. With signal analysis and artificial neural networks, four postures were able to be determined with an accuracy of 98.4% [88].

2.4.5 Pressure Sensing: The Most Appropriate Method

When considering all of the possible posture monitoring devices, a PSA is the most appropriate sensor type to embed into a mattress. This is because it does not require attachment to the user and can provide high-fidelity data for fine-grained posture information such as limb position and joint angles. Body shape estimation and breathing rate have also been detected using a PSA. This level of detail for posture monitoring is unmatched by any other mattress-embeddable sensing system. Vision systems come close, but they are required to be above the user and can suffer when the subject is occluded by a covering.

With the monitoring method determined, the different types of pressure sensor arrays can be investigated to find the most appropriate solution.

2.5 Pressure Sensor Arrays

This section looks a pressure sensor arrays in more detail.

To reiterate, a PSA is a sensor array of pressure-detecting elements. The data produced is a contact image, where each pixel of the image represents the location and amount of pressure that is exerted on it. Each element in the array is called a sensel [89] and the pixels in the produced image are called "pressure pixels" [90]. Some of the many applications of pressure sensor arrays include artificial skin[91] fingerprint sensors [92], posture monitoring for sitting [93],[94] posture monitoring for sleeping [82],[86] and smart surfaces [95]. Their scale ranges from the micro[96] to the macro[95].

They can be constructed in two ways and operate on different sensing principles.

2.5.1 Sensor Construction

Pressure sensor arrays can be made up of individual sensing elements [97], or constructed in a way where rows and columns of the sensor share access lines or wires. A raw pressure sensor is an analog sensor and needs to be connected to some form of a processing system. Here we will define the "access lines" as the connections from the sensor to the processor. A sensor array of unshared access lines would have at least the same amount of wires as there are sensors - double if they require a return path. For an array of n rows and m columns, this results in at least $m * n$ access lines. Therefore, a 20x20 element array would have 200 elements and $\geq$400 wires if they require a return path. Thus we can see the wire count becomes significantly larger with increased rows and columns and can be deemed impractical for large arrays. Sensor arrays with shared access lines will have a wire count of $n + m$, so for the previously given example, that would be 40 wires. This is one-tenth of the unshared count for this example.

2.5.2 Underlying Pressure Sensing principle

PSAs utilize a variety of different material properties or electrical principles to sense pressure. The sections to follow classify the sensor arrays based on these principles.

Piezoresistivity

Piezoresistivity is a change in a material's electrical resistance to pressure. It can be seen in the name itself: Piezo - to press and resistive - to oppose (referring to electricity). Many pressure sensor arrays utilize this principle. They are widely adopted because of their simple design requirement and electrical measurement circuitry. This makes them lowwer-cost than some of the other arrays. However, they suffer from hysteresis and can have lower resolution compared to other PSAs.

They are constructed by placing electrical contacts on either side of a piezoresistive substance. A voltage is placed on one of the contacts. As pressure is applied to the piezoresistive substance the electrical resistance decreases causing more current to flow. This change in current flow can then be measured. A relationship between the compression of the material to current flow can be modeled to make the pressure sensor.

Conductive foams are commonly used as the piezoresistive material. This is because Foam is a porous material that has collapsable cells. When pressure is applied, the cells collapse and shorten the electrical path, thereby lowering the resistance.

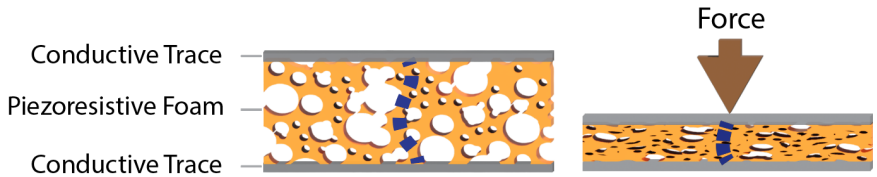


Figure 2.2: Piezoresistive Foam

A number of peizoresistive systems have been demonstrated as general PSAs [1, 98] and for use in beds [99, 100]. Electromechanical hysteresis and non-linearity are reported to be major drawbacks of resistance-type sensors [101, 102] and require calibration for each session of use [103]. They can also suffer from crosstalk. This is current flowing through other paths of the resistive medium. Methods have been demonstrated to reduce or emliminate cross talk [104, 105]

Optical Occlusion

The pressure sensing principle behind this method is optical occlusion or diffusion. One method of creating an optical pressure sensor array is demonstrated by S4 Sensors[106] embedded with Kinotex fiber-optic sensors [107]. Light is transmitted to a cavity or a scattering medium through an optical fiber. This is a site that changes the intensity of light with pressure. Another optical fiber connects to a photodiode for measuring the intensity change. Figure 2.3 demonstrates this technique. The output is a voltage that is proportional to the pressure. A similar design is demonstrated by Lang et al.[108]. [109]

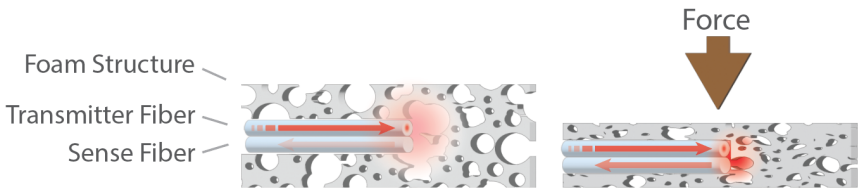


Figure 2.3: Kinotex^{TM} optical pressure sensor operating principles.

Bennet et al used a 24-pixel array (3 x 8) sensel array produced by S4 sensors, to measure the position and transitions of a subject. The array was quoted at less than $100 USD for volume manufacturing.

Another optical PSA construction option is to use a waveguide with a bragg grating to alter the wavelength of the light passing through. [109]. Only small arrays for artificial skin have been demonstrated using this approach. Mattress-sized versions are yet to be exhibited.

Optical sensors are unaffected by electrical noise, humidity and temperature. They are also said to be highly responsive. The ones demonstrated in literature have a low resolution in comparrison to other mentioned arrays in this section but are cost-effective for these smaller arrays. An inherent downside to an array of optical sensors is that they require two fiber optic cables per sensing element. This equates to $2(m * n)$ for n rows and m columns, which is a significant amount of optical cable for mattress-sized arrays of high resolutions.

Electrical Generation

This sensing method utilizes some materials' ability to generate electrical energy from mechanical energy such as vibration or a kinetic impulse. They are called active sensors because, in principle, no external energy is needed to obtain an output signal [110]. They transduce the kinetic energy to electrical energy, which is then measured as the signal.

Piezoelectric materials: A Piezoelectric material is an anisotropic crystalline substance. They generate a small electric potential when the internal dipole moments have pressure exerted on them [111]. For a flexible array, that material is a polymer-based poly vinylidene fluoride (PVDF) [112]. An array of individual sensors have been demonstrated to measure physiological signals of respiration, heart rate and body movements [113].

electrostatic generator: Electromechanical films (EMFi) [114] are another type of generator. It only produces an electric charge when a force is applied perpendicular to its surface. They are constructed by injecting charges into an electret (cellularized polypropylene film in this case) and placing electrodes on either side [115]. It is the mutual movement of these injected static charges that generates electrical energy used for measurements[116]. This sensing approach is well suited to measure pulsatile and transient forces, but it is unsuitable for static forces. It has been demonstrated for use in neonatal ballistography measurements[115].

Triboelectric Sensor Array: These are like electrostatic generators in that they convert low-frequency mechanical energy into electrical energy based on the coupling

of triboelectrification and electrostatic induction [117]. The triboelectric effect is also called contact electrification. It is a process of charge transfer during the separation of two materials. A few sensor arrays have been demonstrated [118], [117], with the largest being by Wang et al. [119]. It was 38 x 38cm in area and had a pressure sensitivity from 0.1 to 37.5 kPa. The sensor's composition is demonstrated in figure 2.4.

No mattress-sized array has been demonstrated in literature for these aforementioned techniques. The construction process for this is currently more complex than other sensor arrays such as capacitive or resistive arrays. However, it has the potential be to produced using cost-effective materials and could be a viable option with more development.

Figure 2.4: Triboelectric sensor array construction [119].

Capacitance

A capacitive pressure sensor is mainly based on a parallel plate capacitor. It is a well-known electrical principle because the ubiquitous electrolytic capacitor seeks to maximize this effect. A capacitor is created when two electrodes of a high surface area (or conductive plates) overlap each other and are separated with a non-conductive material. This inner material is known as the dielectric medium. When a voltage is applied to one electrode, electrons are forced either onto or off of it. This creates a first electric field that attracts or repels electrons in the other plate. A potential difference between them is created and a charge is stored. If the dielectric medium has the ability to be polarised it reduces the electric field strength between the two plates and allows more charge to be forced onto the plates. Equation 2.1 expresses the relationship between these factors.

$$C = \frac{Q}{V} = \frac{\varepsilon_0 \varepsilon_r A}{d} \tag{2.1}$$

ε_0 is the absolute dielectric constant, ε_r is the relative permittivity of the dielectric medium - the material between the electrodes, A is the overlapping area of the electrodes

and d is the distance between them.

This equation reveals three variables that can be adjusted to change the capacitance: ε_r,A, d.

The change in conductive plate distance (d) is the most common method used in capacitive pressure sensors [120, 121, 122]. A compressible dielectric medium allows the distance between the conductive electrodes to decrease, causing the capacitance to increase. This increase is defined by equation 2.1. The pressure sensing characteristics of each sensel are determined by the mechanical properties of the dielectric medium.

Changes in the overlapping area (A) of the electrodes have also been used for pressure measurements. This is a less common method of pressure sensing. However, Zhang et al. demonstrated a 4 x 4 pressure sensor array using a liquid metal film electrode with an icicle-shaped structure [123]. Pressure on the system caused the stretchable elastomer encapsulation to deform and increase the overlapping area and separation distance of the electrodes.

There are only a few mattress-sized sensor arrays that operate on capacitance. Mayer et al. [97] demonstrated a larger body sized array. This array had a grounded configuration, in that each sensel had a wire connected to the electrode and the other electrode was connected to ground. All grounded sensels shared a common ground. This means that for every sensel there was one access line, resulting in many access lines for this array.

Other sensor arrays share row and column lines to reduce the number of connections to the array [121]. Many utilizing this style are demonstrated in literature. However, they are of a smaller scale and mostly demonstrated for use in artificial skin [120], wearable applications or as "proof of concept" arrays [122]. Xsensor[124] is a company that produces a commercially available PSAs that is layed on top of a mattress[125, 126].

Dielectric mediums of high constants have also been used to improve the sensitivity of capacitive-based pressure sensor arrays. [127]

Capacitive sensors show promise because they do not suffer from non-linearity like resistive sensors do. However, the electronic sensing mechanism is usually more complex [128] and can suffer from stray electrical interference.

2.5.3 Capacitive Sensing: The Desireable Option

After considering all of the PSA sensing principles, capacitive sensing has been selected for the following reasons:

- A high level of sensitivity can be achieved.
- A mattress-sized array can be constructed with high resolutions and minimal connections required for high resolutions.

- The pressure measurement characteristics are based on the mechanical properties of the dielectric medium. In this way it can have more linear pressure readings.
- The dielectric medium can be easily produced at the TCG foam plant.

Aspects of this sensing principle that need to be considered are:

- It is susceptible to electric interference.
- The capacitive sensing circuitry can be complex.

A potential solution for electrical interference would be to encapsulate the whole sensor array in a conductive shield, such as a faraday cage. To keep the capacitance sensing circuitry a simple Integrated Circuit (IC) "off the shelf" solution will be investigated.

2.6 Pressure Sensor Array Market Research

There are a number of pressure sensor arrays on the market, some of which, are listed in table 2.2. Many of the prices given were difficult to accurately acquire and are possibly outdated. However, this list demonstrates the price range of pressure sensor arrays available to purchase. The manufacturing costs of each are unknown, but the table highlights the need for a low-cost solution. None of the examples provided are built into a mattress. They are a sheet-like layer that sits on top of the mattress. This has the convenience of fitting the sensor to any mattress but it does not offer the same level of protection as being embedded into a mattress.

Pressure Sensor Array	Sensing Principle	Sensing Elements	Cost (USD)	Ref
Tactilus High Performance Mattress Pad Sensor System	Piezoresistance	1,728	£5,730.00	[129]
Teckscan Body Pressure Measurement System	Resistive	1,558	$5,350	[130], [131]
Vista Medical FSA Boditrak Bed System	Piezoresistive	1,728	$5,895	[132], [133]
XSENSOR X3 Retail Mattress Sensor Pad	Capacitive	1,664	$9,750	[124] [134]
Wellsense Advanced Pressure Visualization System™	Unspecified	Unspecified	Unspecified[1]	[135]

Table 2.2: Mattress sized commercially available pressure sensing systems

[1]The Previous Wellsense sensor M.A.P was said to be $3,500 [132]

2.7 Pressure Image Analysis

This last section investigates methods of pressure image analysis for extracting physiological information such as body posture and shape.

2.7.1 Statistically Based Posture Analysis

Xu et al. [86] presented a matching-based approach called Body-Earth Mover's Distance (BEMD). First the images are transformed into histograms with three descriptors (planar, polar and projection). Then, a histogram normalization is applied. Then weight normalization and space alignment are required to remove the offset of weight, position, and size for different subjects. BMI is also used for normalization scaling. Lastly, a combination of Earth Mover's Distance (EMD) and euclidean distance to evaluate the similarity of sleep postures. Six sleeping postures were considered. They are left/right log and fetus, prone and supine. An accuracy of 91.21% was achieved. The fastest runtime per image sample was 3.27 seconds for the projection descriptor. This approach requires many processing steps which is what causes it to be slow, but it is quite accurate.

2.7.2 Region Based Posture Analysis

Sun et al. [103] created a posture recognition algorithm for physical rehabilitation exercises. This algorithm was only designed using images of subjects on their backs. First, each image was subjected to an alignment step, then all the pixel values were normalized so the pressure sum equated to one. Next, region-based clustering was performed. The image was split into 6 regions representing different parts of the body. The centroids of pressure clusters were found and euclidean distance was taken. The end result was basic limb movement detection for exercise. It achieved 97.8% accuracy on fifteen test subjects.

2.7.3 Machine Learning-based Posture Analysis

Machine Learning (ML) refers to a computer's ability to learn without being explicitly programmed [83]. The purpose of ML is, as the name suggests, to learn from a given set of data. All ML systems have a training dataset and a learning model. The dataset that a PSA provides are pressure images. Artificial Neural Networks (ANNs) are a subset of machine learning.

Sarah et al. demonstrated posture and limb detection on a 27 x 64 array. Three main posture classes were determined: left lateral, right lateral and supine. A total of thirteen unique postures defined by limb location were identified. First, the image was filtered, blurred and turned into a binary image. It was also adjusted to be vertical.

A Gaussian Mixture Model was used to segment the body pressure data into clusters. This was then combined with previously recorded pressure data and a KNN linear classifier to obtain posture classification. The center of the clusters were identified and labeled as the limbs. The overall posture classification detection accuracy was 98.4%.

This example by Davoodnia et al. [136] is a feature extraction-based approach with ML to assist in classification. Other ML approaches leave the feature extraction up to the model. Deep learning is a class of ANNs. It is defined by the use of three or more neural network layers for the training and inference model. Convolutional Neural Network s (CNNs) is a deep learning algorithm often used for image classification. It determines what features are important in the dataset without any human supervision. CNNs scale well and are highly performant. They have been used to demonstrate posture detection [84, 82, 8] and object detection [85] from a pressure image.

A recent publication by Keison et al. [82] clearly demonstrates CNN basic posture recognition on a simple PSA. They made use of the artificial intelligence library Tensorflow [137]. They utilized a dataset of 200 images for six possible cases: "right lateral", "left lateral", "face up", "face down", unoccupied and "closer to the edge". Their accuracy detection was 80%, 85%, 93%, 90%, 100%, 95% respectively.

2.7.4 Section Summary

A pressure image can be analysed in a variety of different ways from simple image segmentation and region-based clustering, to ML and CNNs. With the advent of ML libraries such as Tensorflow, is easier to perform image based classification using deep ANN and CNNs. Using a ML library is the approach that will be taken to analyse and classify postures from the pressure images during this project. This method is sufficient to demonstrate that the developed PSA is capable of posture identification and classification. A labeled dataset will need to be created as part of the training process and a ML model used to train.

2.8 Chapter Summary

Quality sleep is important for physical and mental well-being. A proper sleep duration of 7 - 9 of sleep in adults is key for obtaining quality sleep. However, almost a third of adults are reported to be sleeping less than this recommended time.

The sleeping posture of an individual can affect the quality of their rest. An example to re-highlight is sleeping in a left lateral position reduces acid reflux episodes in GERD sufferers. Another is Postional Obstructive Sleep Apnea (POSA), which is a disorder that can be solved by maintaining a lateral sleeping position. Other sleep disorders become evident with unconscious body movement such as RLS PLMD and sleep

walking. These reasons are why posture monitoring system are beneficial in society.

The investigation for a posture monitoring system that can be embedded into a mattress has led to utilizing a capacitive based Pressure Sensor Array (PSA). PSAs are an adequate sensing solution that can provide basic posture monitoring such as prone, supine and lateral positions, but also has the to produce detailed bodily information such as limb positions, joint angles and body shape.

Capacitive sensing was selected out of all of the potential PSA sensing principles because of its high sensitivity and linearity. The pressure image produced by the PSA would be best analysed using modern Machine Learning (ML) algorithms because they are effective and efficient at identifying features in the data. Artificial Neural Network (ANN), especially Convolutional Neural Network (CNN), are most appropriate for classifying images.

Research into the market of commercially available pressure sensor arrays has shown that they are not priced for the average consumer. The prices of the listed devices range from $5,350 to $9,750 USD.

The next chapters of this thesis are based on these conclusions. A capacitive-based PSA is constructed for embedding into a mattress. It is designed to meet the cost-effective requirements outlined in section 1.2.1. Elements such as material selection, manufacturing process and an overall design have been considered when meeting the given requirements.

Chapter 3

Hardware: The Soft Sensor Array

The next two chapters comprise the hardware portions of the posture monitoring device.

The soft sensor array, or soft sensor for short, are the layers of foam and conductive fabric that make all of the sensels (pressure sensing elements). It's what the user interacts with when they lay on the mattress. There are a few requirements that the soft sensor needs to meet in order to be consumer-ready. It needs to:

- Be durable enough to withstand the repeated loading that a mattress experiences from daily use. The conductive traces should not tear or come in contact with one another for at least the 5 year warranty period.
- Be able to deform as the mattress deforms and not significantly change the comfort feel.
- Take into account material usage and manufacturing methods that are cost-effective and meet requirements outlined in section 1.2.1.

3.1 Sensor Construction

The array of sensels in the soft sensor is constructed by placing two soft circuit layers above and below a non-conductive layer, known as a dielectric medium (figure 3.1). The soft circuits are comprised of conductive strips that run most of the length of the sensor. The two are angled perpendicular to each other. The upper circuit has 16 strips running vertically (figure 3.2a) and the lower circuit has 41 strip running horizontally (figure 3.2b). A sensel is created where the strips overlap. This equates to 656 sensels. The dielectric medium material that is used should be the first material in the mattress

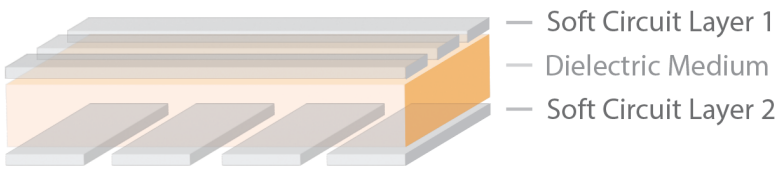
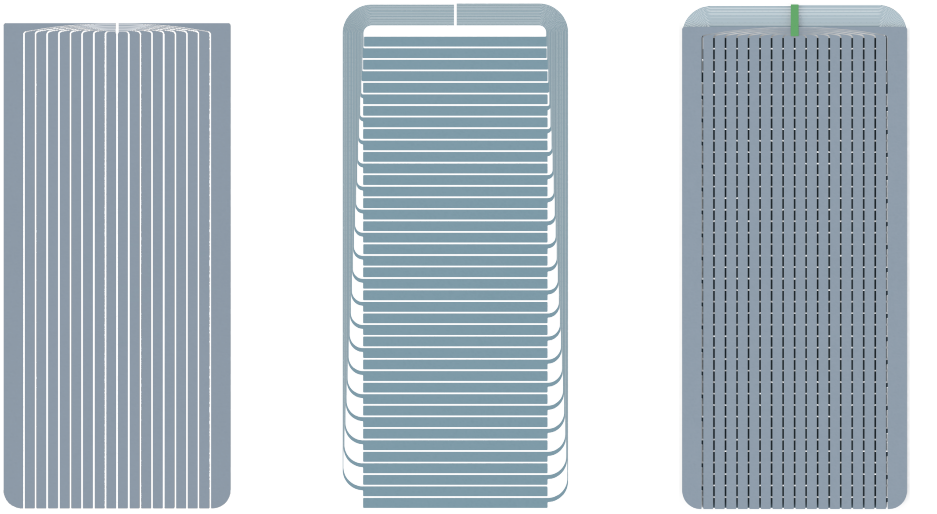


Figure 3.1: Basic pressure sensor array layer stack

to compresses. This will result in the highest pressure sensitivity. Therefore, a low hardness TPUs foam (VF42-50) was used for this project. In general, the mechanical properties of the dielectric medium determine the sensitivity of the sensor and linearity in pressure measurements.



(a) Upper soft circuit (b) Lower soft circuit (c) Combined soft circuits

Figure 3.2: Soft circuit layers

Working from the basic stack of figure 3.1, further TPU foam spacers are placed above and below to electrically isolate a conductive fabric wrap. This wrap is used to shield the inner soft sensor from external electrical noise and parastatic capacitance. Any other foam layers required to complete the mattress are placed above and below the wrap. The final layer stack is shown in figure 3.3. This stack is also what has been referred to as a Pressure Sensor Array infused Mattress (PSAM).

3.1.1 Soft Circuit Design

As previously mentioned, the circuits are designed to have 16 strips running vertically and 41 strips running horizontally (figures 3.2a & 3.2b). They can be seen in greater detail and with basic dimensions in appendix A. Each strip has a routing back to the attachment point of the hard circuit. This is the top green rectangle on figure 3.2c.

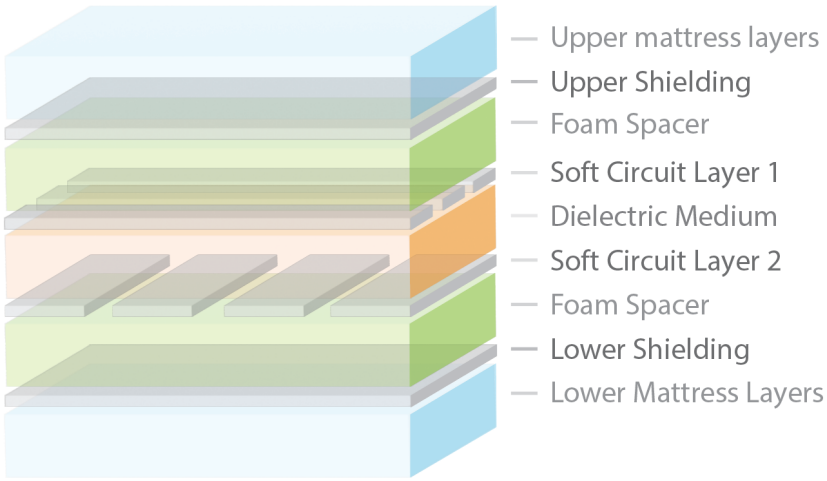


Figure 3.3: Mattress-infused pressure sensor array layer stack

The upper circuit also has extra wide strips on the outer edges to act as a shield to the routings on the layer below. This, however, became redundant when the decision to encompass both soft circuits with a shield was made.

The overall sensor size is 1910mm long by 855mm wide, which is to fit a king single of 2030mm long by 1070mm wide. It was originally designed to fit a more narrow medical mattress but was later repurposed for a consumer mattress. The strips are 34mm wide and are spaced at 43.75 increments, giving 9.75mm between them. The traces leading back to the hard circuit are 2.7mm wide with 1.3mm spacing between them. The circuits were designed in Solidworks [138] and Adobe Illustrator [139]. Illustrator performed better with all of the line entities, but Solidworks could perform volume calculations for conductive ink estimations.

There were a few trade-offs to make when designing the soft circuit. Wider conductive strips allow for a larger overlapping area between the circuits which in turn improves pressure sensitivity. However, they bring adjacent strips closer to each other with the increased risk of short-circuiting, especially as the mattress deforms under loads. The material and shape of the strip also influence mattress deformation and comfort characteristics. This is discussed in more detail in 3.1.3.

3.1.2 Methods of Creation

A few different methods of soft circuit creation were investigated.

Cut Conductive Fabric: As the name suggests, this is a conductive sheet made up of threads that can be woven or non-woven. Wires can be spun in with the thread and made into a fabric, or a non-conductive fabric can be coated with a conductive material. Low-cost conductive fabrics are nylon weaves coated in

metals such as copper, copper-nickel or silver.

A soft circuit layer is made by cutting out the desired shape from the fabric. Price investigations have been able to source them for as low as $6 per linear meter for non-woven and $8 for woven (as of September 2019). This is the lowest cost method of all the options that will be listed. However, since many parts of the circuit design are narrow and disjointed, great care needs to be taken when holding and gluing it in place. It would also produce a fair amount of material off-cuts and waste. This is important to consider from a long-term sustainability perspective.

Conductive Ink & Screen Printing: Screen printing is a well-established technology to print an image on surfaces such as cloth or paper. Printing a circuit using conductive inks onto a stretch fabric or directly on foam could be one of the simplest production methods. However, this is dependent upon the ink properties and price. It would need to withstand repetitive deformation forces without cracking. The price of many inks is too high for the amount required to make the circuits. For example, Bare Conductive Electric Paint [140] is one of the more cost-effective conductive paints on the market at £250 or $273 USD per liter. The datasheet lists the resistance as $55\Omega/sq$ at $50\mu m$. Working with this thickness in the current sensor design would equate to a total volume of 188.7 cm^3. The volume was calculated using solid works. This would be approximately 188.7 ml of ink required per soft circuit pair. A one-liter bucket of ink would print 5.3 pairs with an equated cost of $51.51 USD per soft circuit pair. This is a conservative estimate because the volume calculation assumes no solvent evaporation during the drying period. Such information was unavailable during research. However, it is sufficient for rough ink requirement estimations - it could be considered a lower price bound. It is unclear if this ink can withstand strain stresses without cracking. Dycotec materials provide a stretchable ink [141] at £440 or $709 USD per kg and $22\Omega/sq$ at $25\mu m$. The datasheet specifies 27-33% solid content. If 30% is assumed for calculations and an ink thickness of $25\mu m$, that equates to a wet or uncured volume of 83.33 μm. At that thickness, the soft circuit patterns require 313.2 cm^3 of ink. At a listed density of 0.93 g/cm^3, 1kg of ink will supply a total of 1075 cm^3 in volume. Thus, the total would be 3.43 soft sensor circuits per kg of ink and $206 USD each.

This is too costly but is an area to periodically check as lower-cost solutions are in development with the rise of graphene and carbon nanotubes. Optimizations in the circuit pattern would also reduce ink volume usage.

Kitting and Emboroiding with Conductive Thread: Soft circuits can be knitted or embroidered into fabric. Myant [142] is a Canadian textile company specializing in knit and embroidered sensors, which they call textile computing. The comfort group consulted with them to produce the soft circuit designs, but it would be hundreds of dollars per circuit. Embroidering a soft circuit at the scale of a mattress is also inefficient and would require high levels of machine time.

Cut conductive fabric was selected as the method to produce the soft circuits. The main determining factor was the cost of the conductive fabric.

3.1.3 Soft Circuit Fabrication

The soft circuit strips or traces are cut out of one single sheet of conductive fabric. The traces are then adhered to a nylon stretch fabric. Ideally, the cut fabric would be adhered directly to the dielectric foam but having the flexibility to move or remove the circuit or dielectric during the development phase was desirable.

The Conductive Fabric

The conductive fabric (SY-PF37B000A09) used to make the conductive strips is a copper and nickel coated plain weave polyester. It was acquired from Saintyear Electronic Technologies [143] for $8 USD per linear meter (as of September 2019). It was ordered with an adhesive backing and transported on a wax paper separator. The roll was 1300mm wide. Its datasheet can be found in appendix B. Each soft circuit requires 2 linear meters of conductive fabric. With two sensors, that equates to a material cost of $32 USD.

Fabric Cutting

Two methods of cutting the fabric were investigated: CNC knife cutting and laser cutting. Laser cutting is the better approach because it is faster and seals the edges to prevent fraying. The blade of a CNC knife needs to be changed regularly, thereby increasing cut costs. However, in laser cutting the fumes from the cut fabric require appropriate extraction and handling.

Prototype Soft Circuit Construction

The prototype's soft circuits were cut using a laser cutter and constructed using the method outlined below. They apply to both layers of the soft circuit.

Fabric Preparation: The conductive fabric was pulled off the wax paper backing and placed non-adhesive side down on a Medium-Density Fibreboard (MDF)

platform. This is a sacrificial platform that had a light coating of 3M Super 77 spray adhesive to help ensure no movement during the cut. A sheet of nylon stretch fabric large enough to cover the cut area was also prepared.

Laser Cutting: The CO_2 laser's power and speed was tunned to cut through the conductive fabric layer but not through the MDF underneath, as demonstrated in figure 3.4. High cutting speeds (100mm/s) could be achieved because of the low material thickness. The undesired fabric was removed as waste, leaving behind the desired strips and traces. Figure 3.5 below demonstrates an early prototype using this method.

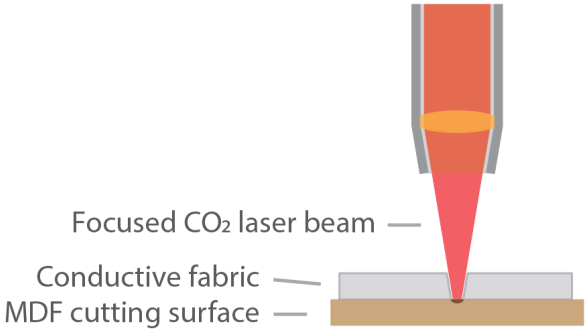


Figure 3.4: Laser cutting of conductive fabric

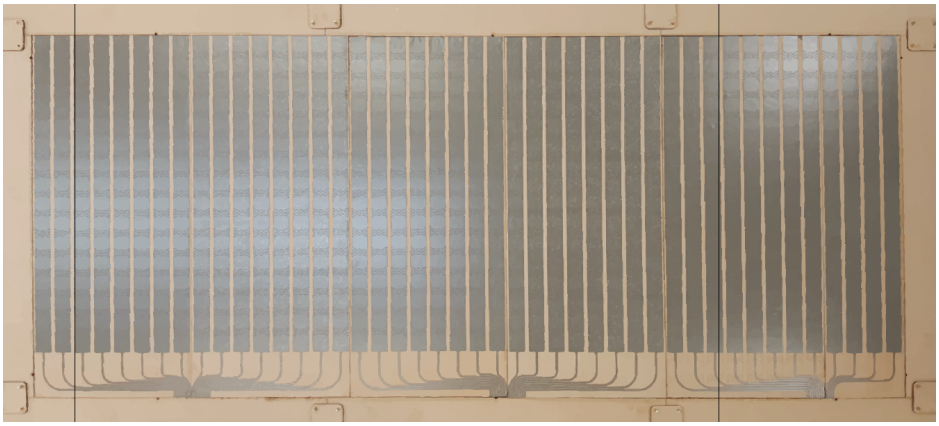
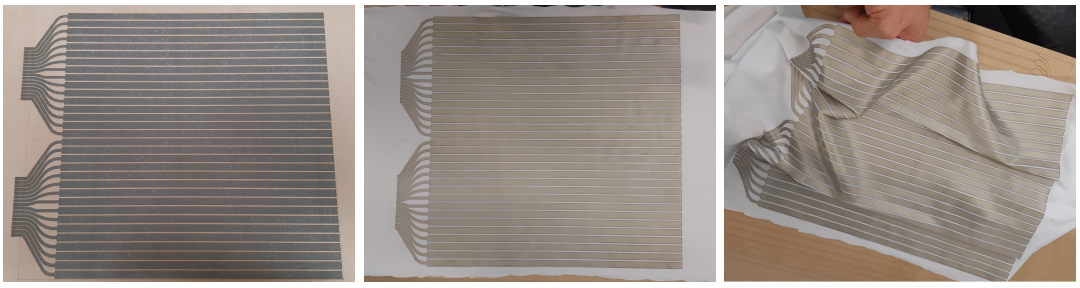


Figure 3.5: Early prototype of a soft circuit tacked to and cut on MDF

Substrate Attachment: The non-conductive nylon stretch fabric was then placed on top and pressed down firmly to adhere the layers together. The newly formed circuit was then peeled off of the MDF. Figure 3.6 are photos of this assembly process on a smaller test design.

Post processing: A visual inspection was undertaken to ensure that no strips were



(a) Laser cut conductive fabric on MDF. (b) Laser cut conductive fabric with nylon backing. (c) Flexibility demonstration of the soft circuit

Figure 3.6: First test soft circuit and construction photos.

touching. Strips of nylon fabric wide enough to cover the thin traces leading back to the hard circuit were spray glued as a covering. This was to prevent potential electrical shorts if the fabric folded over. This can be seen in figure 3.19d later on.

Mass Manufacturing Approach

There is a more effective approach for large-scale manufacturing. As previously mentioned, the fabric is coated with an adhesive on a wax paper backing. This means the conductive fabric also has a supportive surface on which it can be cut. Cutting only the fabric and not the backing would allow the undesired areas to be pulled away leaving the cut shape behind (steps 1-3 of figure 3.7). A spray adhesive can then be applied to the top side of the cut fabric to prepare it for being attached to the dielectric foam. The area where the hard circuit will be attached needs to be masked off to allow a clean contact surface. After the fabric is attached to the dielectric foam, the paper backing can be pulled away(steps 4-6 of figure 3.7). Next, the hard circuit can be attached and another layer of spray adhesive (if required) can be put down to allow for the placement of a foam spacer (steps 7-8 of figure 3.7). These steps must be repeated for the other soft circuit layer to complete the inner part of the PSAM.

Non-stretch Problem

The plain weave conductive fabric used to make the soft circuit is flexible but not easily stretched. When placed inside, this can slightly alter the feel of the mattress. It also puts the strips under a high degree of tension. There are stretch fabrics that could potentially be used, but sourced prices were quadruple the cost and none came with an adhesive backing. Adding the adhesive backing could be performed in-house but it is not ideal due to increased machinery costs.

Another solution would be to make shaped cuts or perforations in the current strips

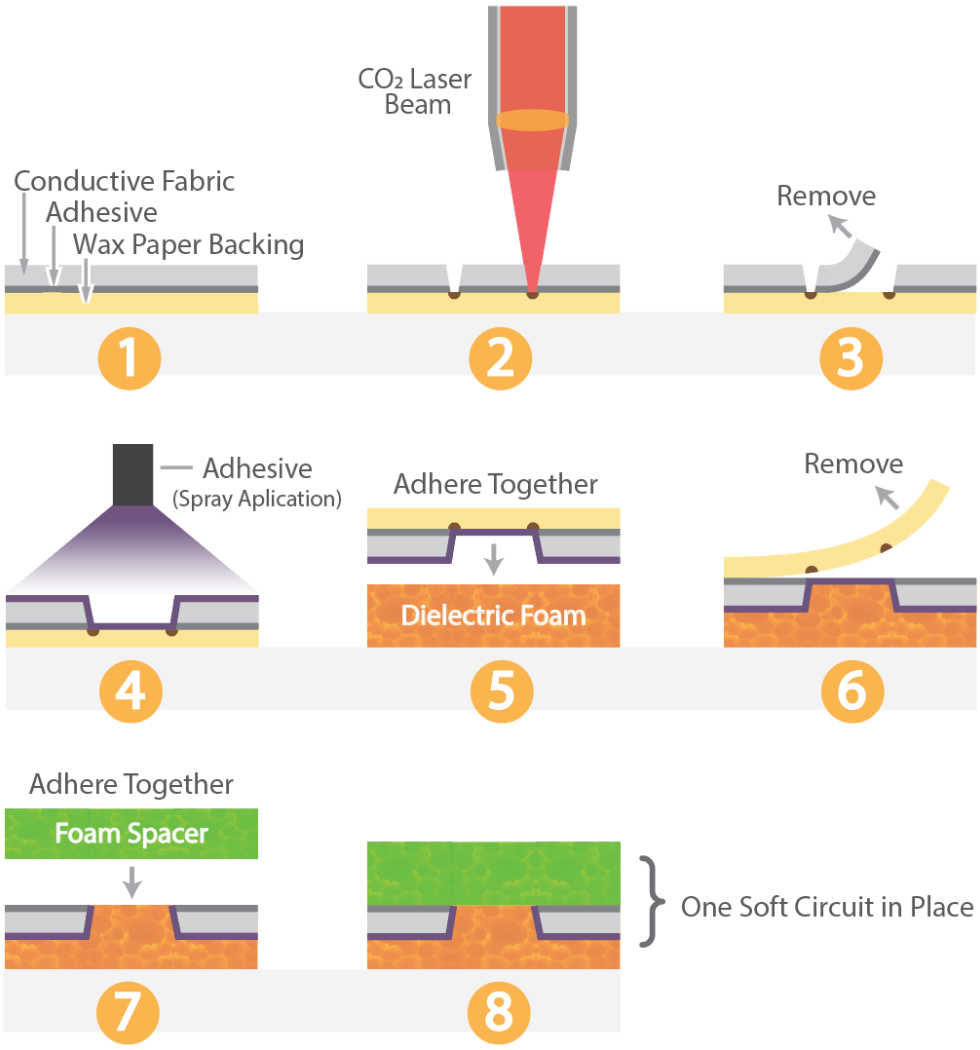


Figure 3.7: Potential Production process of soft circuits

(figure 3.8). The perforated cuts have been implemented in a queen-sized version of the sensor. The other cut strip types have yet to be tested. It is unclear what effects the opening and closing of relief cuts might have on the electrical characteristics of the sensor as the mattress deforms. As it does so, the resistance in the strips will increase and decrease. Further investigation into how this will add to the noise or transient response is yet to be conducted. However, no observable effect has been seen in other test sensor arrays to date.

Polyurethane Foams

The dielectric medium between the soft circuits is a 10mm thick TPU foam produced at TCG. It is coded as VF52-40 which is a Visco-elastic Foam (VF) with a density of

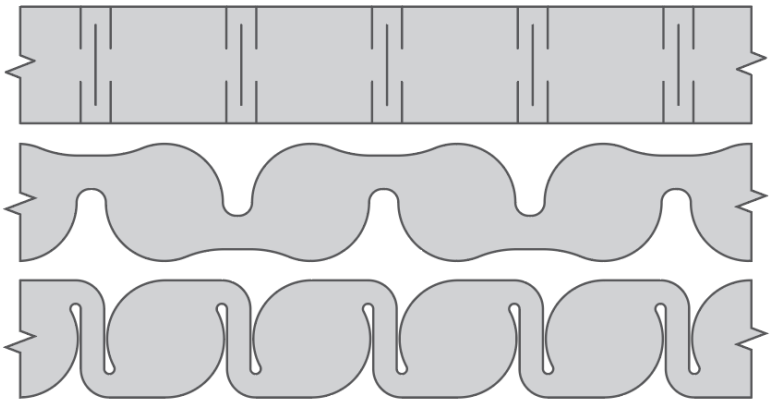


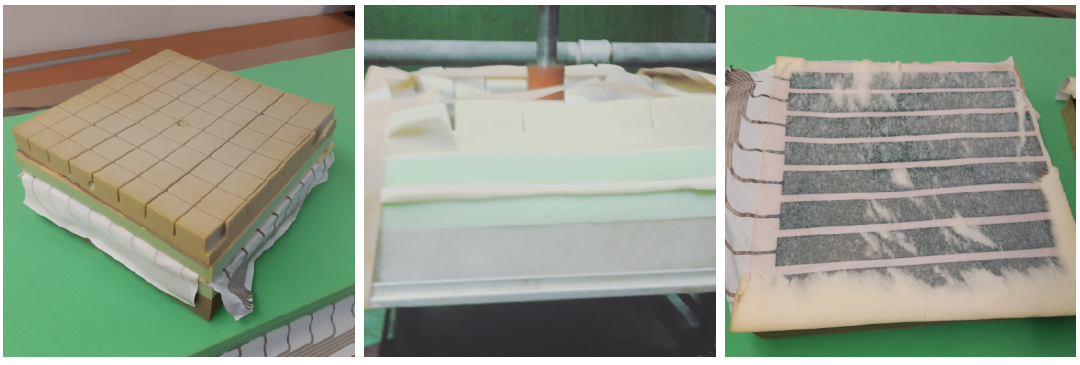
Figure 3.8: Stretchable shapes in conductive traces

52 kg/m^3 (50 min - 54 max) and a hardness of 40 newtons (35 min - 55 max). The hardness is measured as the force required to compress a foam sample by 40%. More information on foam properties can be found in B

3.1.4 Initial Durability Tests

Durability is a key requirement of the soft sensor. One test that is performed at TCG is a pound test. It is a cyclical force that pounds the top of a foam sample 80,000 times. The foam sample is a square 400mm x 400mm wide.

A 400x400 cut out of the PSAM was constructed and glued together (figure 3.9a). It was subjected to 78,756 cycles of pounding (figure 3.9b). Measurements of hardness before and after were taken. To pass the test, a harness loss had to be below 15 percent. This test sample passed with 14.8 percent (table 3.1). This test is primarily used to test a foam's durability, but it was also used to test the conductive fabric lines. The layers of foam were pulled away and the lines were inspected for any tears (figure 3.9c). No observable tears were found during a close-up inspection. As a preliminary durability test, it produced promising results.



(a) psam cutout for pounding (b) Pound test (c) Sensor lines after pounding

POUNDING TEST	80,000 CYCLES	BEFORE POUNDING	AFTER POUNDING	% HARDNESS LOSS
78,756 CYCLES	Height (mm)	151.11	148.26	1.89%
	Hardness at 10% (N)	76.43	67.94	11.11%
	Hardness at 20% (N)	114.18	97.27	14.81%
	Hardness at 30% (N)	153.78	136.35	11.33%
	Hardness at 40% (N)	215.12	201.63	6.27%
	Hardness at 50% (N)	311.07	296.89	4.56%
	Recv. (%)	272.45	305.21	-12.02%
	Sag	1.88	2.07	-0.1011

Table 3.1: Hardness test results for a PSAM test sample showing percentage of compression for before and after.

3.2 Soft Circuits to Hard Circuit Connection

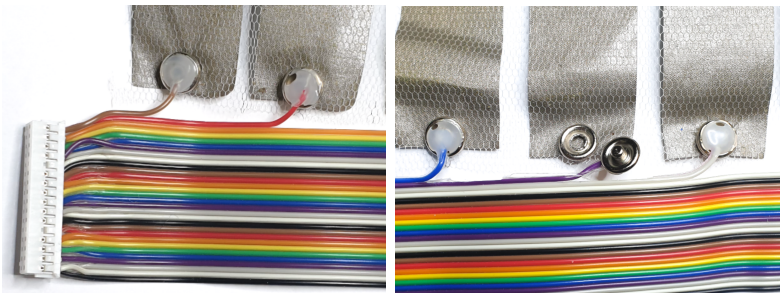
This is the connection point between the soft circuits and the hard circuit PCB (discussed in chapter 4). It needs to maintain reliable electrical contact while dealing with mechanical stress from mattress use. Any momentary disconnections will appear as little to no pressure in one or more sensels. Conversely, short circuits will appear as very large pressure values. Given its importance, there were many different connection methods explored. The following sections demonstrate what was investigated and the approximate order in which this was done.

3.2.1 Metal Snap Fastener

Early investigations led to using metal snaps. It is a common solution among the smart textile hobbyists [144], [145]. They were easy to source from online or local fabric shops. They produce a reliable connection between the soft and hard circuits. Although this may not be the best approach for mass manufacturing, they are proven to work and as such was used as the first connection method for the project. Wires from a ribbon cable were soldered directly to the button snaps and reinforced with hot glue. They are detachable, more cost-effective than other dedicated connectors and produce a secure connection. However, they are thick and labour intensive to make and attach. Thus, they are adequate for prototyping but not an effective solution for a consumer product.

3.2.2 FPC Connector

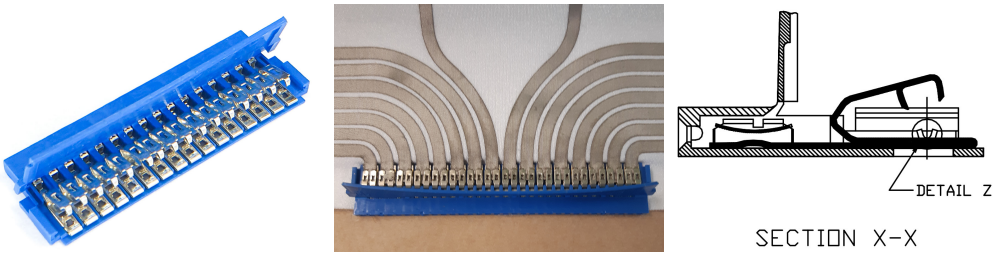
In the search for existing connectivity solutions, Amphenol provides a connector series called the Clincher™[146] for FPC or FFC cables. Its unique design seen in figure 3.11C allows the contact to push down and latch onto the fabric. It is designed for applications where shock and vibrations are a concern.



(a) Ribbon cable and JST connector to hard circuit

(b) Conductive fabric connection with snaps

Figure 3.10: Soft to hard circuit connection with snap fasteners



(a) Clincher™ connector

(b) Connected to soft circuit

(c) Side profile [147]

Figure 3.11: Amphenol clincher™ connector.

When implementing the connector, two contacts were pressed down on one fabric trace. This was to ensure mechanical and electrical contact. One contact could have been sufficient, but concerns about the traces being too thin prevailed. At the time of testing (July 2019), a 24-position connector from Digikey cost $7.34 USD. The sensor array at the time required four, thus totaling $29.36 USD. Ultimately, this pricing is reasonable but still costly.

Positioning the conductive fabric into the mouth of the connector was troublesome due to the protruding pieces of metal in the contact area. Each fabric trace had to be guided into the clamping area and pressed down. This connector initially seemed like a promising solution but in the end, it was not pursued because of high connector prices and difficulty lining up the conductive traces.

3.2.3 PCB Rivets

PCB rivets are small, round, metal tube-like structures that are predominantly used for non-through hole plated boards. It is used to connect a trace on one side of the board to another on the opposite side. Figure 3.12a demonstrates PCB rivets and a punch. The punch has a conical structure in the center to spread the rivet end and secure it to the board. To make a mechanical and electrical connection of the hard circuit to soft circuit, a conductive trace needs a hole or slit cut into it. The rivet is then threaded

through before it is secured into the PCB. Figure 3.13 shows this process.

(a) A PCB rivet(lower) and punch (upper)

(b) Rivet connecting a conductive fabric trace (top).

(c) Rivet connecting a conductive fabric trace (bottom).

Figure 3.12: PCB rivets for securing the soft circuit to the hard circuit.

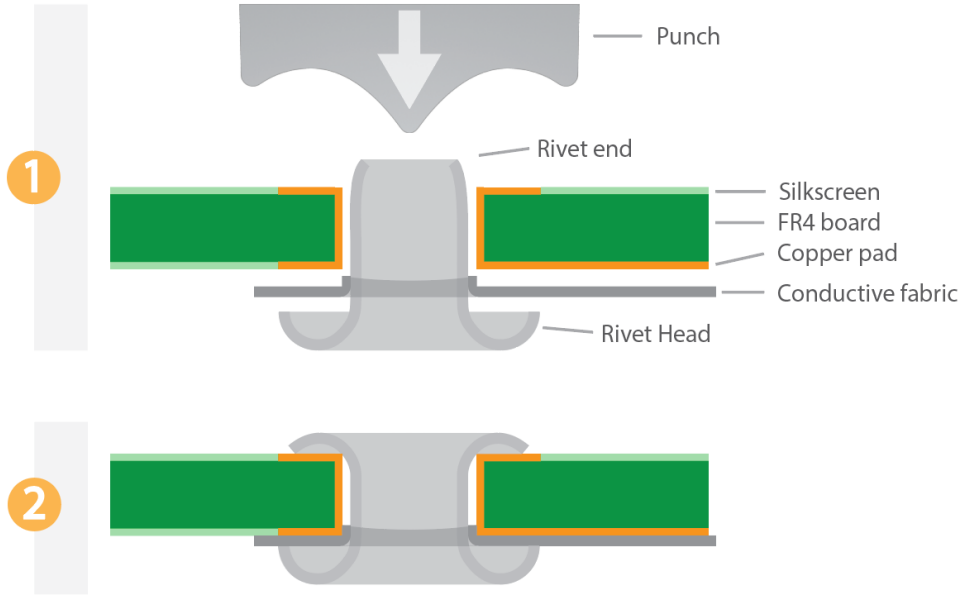


Figure 3.13: The riveting process

Rivets are inexpensive and secure the conductive fabric well. However, as each trace would need a rivet, this adds up to a significant amount of punching. Tooling and an automated process could be utilized, but this is another capital cost. It is still an option if soldering the conductive fabric is not sufficient in future durability tests.

3.2.4 Soldering

Directly soldering a wire to the conductive fabric was used in the early stages of investigating a board to fabric connection. It was performed with a non-temperature

controlled soldering iron and lead to poor results. The high temperature of the soldering iron would melt the fabric or, at the very least, degrade the fabric causing it to be brittle and crack or break off. It was deemed unreliable. However, it was revisited in a quest to find the most cost-effective attachment method.

Direct attachment of the conductive fabric to PCB pads is the most cost-effective of all the connection methods explored. It requires no other connector or securing mechanism besides solder paste and heat. During this revisit, a more thorough investigation involving solder paste and a temperature-controlled hot plate was employed. Solder pastes can come in a varying range of melting temperatures. A common one such as MG Chemicals no-clean solder paste (Sn63 Pb/37 [148] - appendix B) shows to have a reflow temperature range from just below 200^{o}C to 230^{o}C. Even lower temperature pastes can reflow from approximately 150^{o}C to 180^{o}C [149]. These ranges are near the melting point of nylon.

There are a few chemical variations of nylon (nylon 5, nylon 6, nylon 6-6, nylon 6-10, ect[150]), each with a different melting range. No information is given regarding the form of Nylon used in the conductive fabric. It was through correspondence with Zhejiang Saintyear Electronic Technologies that the substrate fabric was even known to be nylon. Nylon 6 and nylon 6,6 are used in fabrics and have a melting point of approximately 223 [151] and 270 [152] respectively.

Optimal Soldering Conditions Investigation

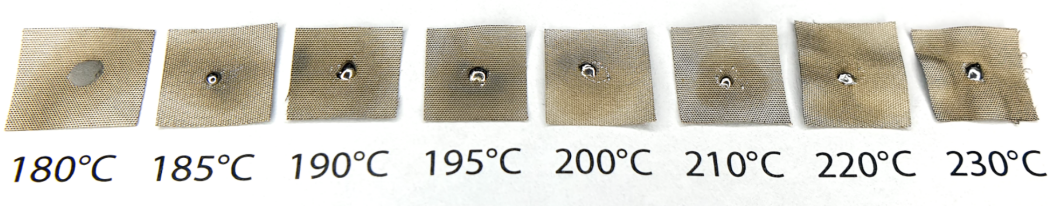
A series of tests were undertaken to validate whether solder paste can adhere well to the fabric's surface and form a sufficient connection. Eight 12mm x 12mm squares were cut out of a plain weave non-adhesive backed fabric. Each had a small amount of solder paste placed in the center of the square. They were then placed on the hot plate of an "MHP30 mini hot plate preheater" [153], as seen in figure 3.14. The hot plate has a temperature range of 100-350^{o}C and a claimed stability of 3%. The temperature of the surface was not validated with any external measurement equipment.

The temperature subjected to each square started from 180^{o}C, eventually reaching 230^{o}C. Increments of 5^{o}C were given until it reached 200^{o}C. The increments were changed to 10^{o}C until 230^{o}C was achieved. This change was because reflow had occurred well after 200^{o}C and there was interest regarding how the temperature would affect the fabric over the recommended solder reflow range. The squares were placed directly onto the hot surface when the desired temperature was stable. There was no preheat, soak or reflow stage in the heating process as given in the solder pastes datasheet (appendix B).

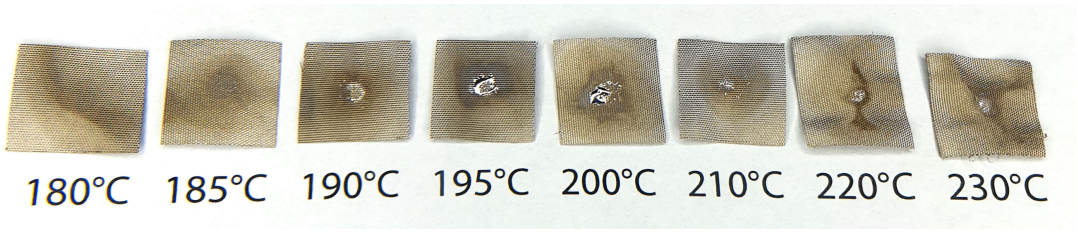
Figure 3.15 demonstrates the results.

180^{o}C shows no melting of the solder paste. Melting occurred at 185^{o}C. It took a

Figure 3.14: MHP30 mini hot plate with conductive fabric test square.



(a) above



(b) below

Figure 3.15: Solder paste melting tests on conductive fabric.

few seconds for all of the paste to become shiny - a sign of a full melt. Temperatures above this caused the paste to melt more quickly. Fux in the paste would melt and begin evaporating before the tin and lead and seep into the fabric. This is the wet stained looking area in the fabric around the solder ball. At higher temperatures such as 200^{o}C, the solder would bead up quickly and then settle into the fabric. However, this effect was inconsistent with repeated tests. Therefore, it is believed that the amount of solder paste and the way in which it is spread play a part in the melting process. The beading effect does not appear to cause an issue when soldering the fabric to a PCB pad.

In figure 3.15b test squares of 190^{o}C - 230^{o}C show solder seeping through to the other side of the fabric. This is desirable because it forms a strong mechanical bridge between both surfaces and becomes more difficult for the fabric to pull away from its soldered surface. If the solder only adheres to one surface, the metal coating can pull

away from the nylon fabric.

As a matter of interest, higher temperature tests were performed on more test squares. Figure 3.16 shows the squares being subjected to temperatures from 240^{o}C to 260^{o}C. Significant deformation or crinkling can be seen. The samples also shrunk while the edges appeared to fray in the heat. As a note, the samples were cut with scissors and not a laser cutter. At 260^{o}C the fabric is soft when handling. After heating and cooling, these samples were more easily torn than the original or other lower heated samples.

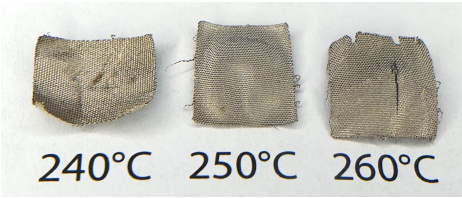
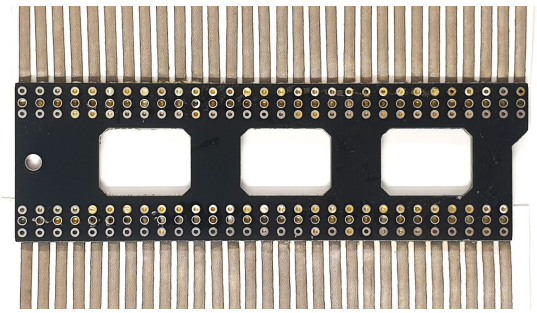


Figure 3.16: Conductive fabric heated to 260^{o}C

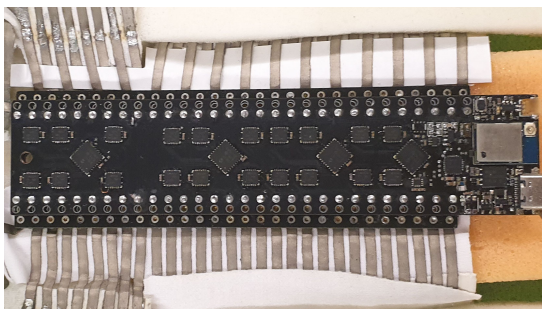
From these tests, it can be concluded that 190^{o}C - 200^{o}C is an adequate temperature at which to perform reflow on this conductive fabric (SY-PF37B000A09 from Saintyear Electronic Technologies). At these temperatures, the solder adheres to it without causing any apparent shrinkage or crinkling.

As a final test, traces were laser cut and soldered to an intermediate PCB (figure 3.17a). The hard circuit stacks on top of this and connects each of the lines through mall pin-to-receptacle joins. It is only used during the development phase. The traces will connect directly to the hard circuit in the production stage. Figure 3.17b demonstrates the soft circuit soldered to the intermediate board and the hard circuit.

Pre-punched experimenters board (FR2) was secured with 3M Super 77 spray adhesive underneath the traces so to remove mechanical stress away from the solder joints to further down the traces.



(a) Intermediate board with soldered traces.



(b) Hard circuit mounted to intermediate board.

Figure 3.17: Conductive fabric traces soldered to a intermediate PCB

Inital Strength Test

A single trace was tested for durability by pulling and peeling at the connection. A fair amount of effort was required. Surprisingly, the soldered connection did not break. Rather, the conductive fabric coating pulled away from the nylon fabric itself. This is demonstrated in figure 3.18 by the copper color on the corner pad. A qualitative strain test would be the next step in properly characterizing its strength.

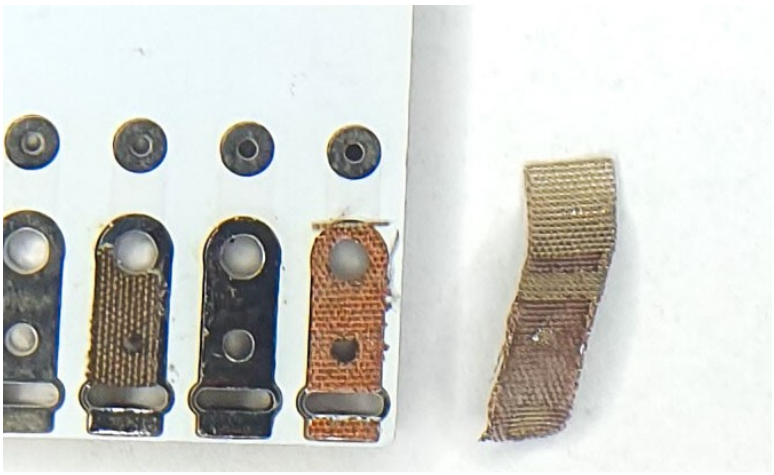


Figure 3.18: Peel test on the solder join

3.3 Completed Mattress-Infused Pressure Sensor Array

To create the final PSAM the hard circuit (discussed in the next chapter) and the soft sensor need to be attached together. Then the remaining foam layers need to be added. Each layer of the PSAM is demonstrated in figure 3.19.

Just beneath the quilted panel sits the conductive 4-way stretch fabric shield (a). The shield had a zip sewn around it for easy access inside during prototyping. The consumer product will not require a zip. Below that layer sits a green 20mm TPU foam spacer(b). Next sits the upper soft circuit layer (c). The laser cut pattern can clearly be seen - vertically aligned strips with conductive traces leading back to the hard circuit. The lower soft circuit and white dielectric medium are below this (d), with horizontally aligned strips and traces leading back to the hard circuit. The traces have a nylon fabric adhered to the top of them to prevent short circuits. For commercial production of the PSAM, the hard circuits would be glued between the foam layer. The last two images show another 20mm foam layer with the bottom half of the fabric shield and springs underneath.

(a) Conductive 4-way stretch fabric shield.

(b) 20mm green TPU foam spacer.

(c) Upper soft circuit layer and hard circuit.

(d) Lower soft circuit & 10mm white dielectric foam.

(e) 20mm foam spacer & lower conductive fabric shield

(f) pocket springs - the rest of the mattress.

Figure 3.19: Pressure sensor array infused mattress.

3.4 Summary

The soft sensor array is the main sensor that is compressed and produces a pressure image. It is composed of two conductive fabric soft circuits that are separated by a 10mm TPU foam. A capacitive pressure sensor or sensel is created where the strips of the soft circuits overlap. The foam that separates them is also the dielectric medium of the sensor. Further foam spacers are placed above and below these layers and it is encompassed in a conductive fabric shield. This is then placed in a mattress to become a Pressure Sensor Array infused Mattress (PSAM).

Currently, the most cost-effective way to produce a soft circuit is by laser cutting it out of a sheet of conductive fabric. However, it is not the simplest method. A simpler option would be to screen print using conductive inks. This would be faster, requires fewer production steps and produces less material waste. Conductive inks are

an active area of research. Future inks may become more cost-effective and become a more feasible option for replacing the laser cutting method.

There are a few different ways to connect the hard circuit to the soft circuit including button snaps, a dedicated connector, pcb rivets or soldering. Each of these methods was explored but soldering was proven to be the most cost-effective and make the best electrical contact. It was found that any solder connection below 200^{o}C would maintain the integrity of the fabric in that it would not shrivel. A solder connection showed to be the most durable. The conductive coating would pull off of the nylon backing before the connection would break. Durability can also be increased with a stiffer backing behind the connection. Initial durability tests have shown that the sensor can withstand repeated loading. This was through 80,000 pounding cycles on a test sample. No tears in the fabric were observed. Longevity tests as a Full sized mattress need to be undertaken to definitively prove its durability.

Chapter 4

Hardware: The Hard Circuit

The hard circuit is the term used to describe the PCB containing all the electronic components for selecting a sensel and reading it. It is the other portion of hardware that attaches to the soft sensor to create the Pressure Sensor Array infused Mattress (PSAM). The circuit is designed to select a particular row and column of the soft sensor and perform a capacitance measurement. The capacitive sensing method is the most essential aspect of the sensor. It plays one of the key roles in determining the sensitivity and stability of pressure readings.

4.1 Capacitive Measurements

Capacitance is one of the three basic components of the electrical system, resistance and inductance being the other two. There are numerous ways to measure the capacitance of a circuit element. Many are discussed by Wei et al. [97]. However, a single chip or Integrated Circuit (IC) solution was desirable for this project because it would reduce the circuit complexity, the Bill of Materials (BOM), and potential costs. It is also more likely to have increased performance and a reduction in system noise [154]. Two Capacitance-to-Digital Converter (CDC) IC were investigated. The PCAP04 Capacitance-to-Digital Converter (CDC) from ScioSense [155] and the AD7147 [156] from Analog Devices. Both of these have high data capture rates (>100hz) and low capacitance measurement values(±9nF & 1 - 100nF respectively).

4.1.1 Analog Devices AD7147

The AD7147 IC is a programmable controller for single-electrode capacitive sensors (grounded sensors). The IC also outputs an active shield or guard signal to minimize noise pickup in measurements. The initial appeal of this IC came from its low cost (≈$2.6 USD at 1k [157]) and the built switch matrix of 12 lines, which could be used

for columns of the array and remove the need for external switches. The theory behind using this IC was to connect the C_{in} lines on the chip to the columns of the array. The shield signal would then be applied to all the rows excluding the one with the pixel of interest. That row would be set to 0 Vdc or ground. This would be done by connecting each row to a Single Pole Double Throw (SPDT) switch and having the two selectable inputs be the shield signal and ground. In the sensor's makeup, the row layer is placed above the column layer. As a force compresses the dielectric medium where the active column and grounded row cross, the capacitance increases. All of the other rows should essentially be capacitively invisible to the active column because they would be at the same voltage as the measurement signal and have no potential difference between them. It was hoped that the rows of shielded lines would also shield the columns below from parastatic capacitance above, like a human body. This working principle is demonstrated in figure 4.1. The active column (red) and grounded row (black) work through an array to measure the capacitance. All of the other rows and columns are connected to the shield signal.

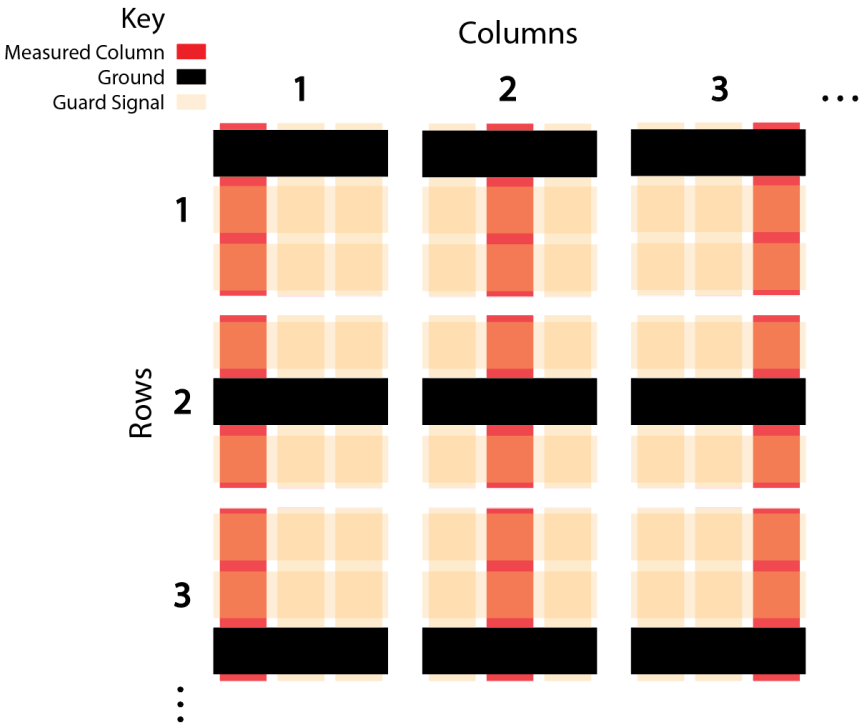


Figure 4.1: Capacitive measurements using an AD7141 working through an array.

A prototype was created using the AD7147 and in the aforementioned configuration. It produced a pressure image and seemed quite sensitive. When small areas of pressure were exerted on the array a clean image was produced. However, when a large pressure was exerted on multiple sensels in the same column, such as a resting body, a bleeding effect occurred. All of the pixels in the column showed that they had pressure exerted

on them. It seemed as though the guard lines were not protecting the active column from the parastatic capacitance caused by the body. The reason why was unclear. This was an issue that was not solved for this setup. A new PSA was developed with a new IC, the PCAP04.

4.1.2 ScioSense Pcap04

The PCAP04 [158] IC is a low cost (≈$3.7 USD at 1k [159]) Capacitance-to-Digital Converter (CDC) IC from ScioSense [155]. Its capacitive sensing principle is based on accurately measuring the discharge time of a capacitive element. It is measured with a high-resolution Time-to-Digital Converter (TDC). The element is charged up to 1.8v, held for a moment, then discharged through a resistor. As the element begins to discharge, a timer for the TDC begins. It measures the time it takes for the voltage to cross a lower threshold value (figure 4.2). Essentially, the IC is measuring the discharge time of an RC circuit.

The measurements are ratiometric in that they are compared to a fixed reference capacitor. This can be an internal reference capacitor or an external one. The internal capacitor has a programmable range from 1pF to 31pF in increments of 1pF. As elaborated in Pcap04's data sheet [158] (p.47), the discharge time is defined by the capacitor and the discharge resistor.

$$\frac{\tau_N}{\tau_{ref}} = \frac{C_N}{C_{ref}}$$

$$\tau = k \times R \times C$$

There are also a set of discharge resistors (10k / 30k / 90k / 180k ohm) that can be selected from to alter the Discharge rate. An external discharge resistor can be selected as well. The capacitive sensor arrangement for this PSA is a single floating sensor with a 10K internal resistor and 31pF reference capacitor. Lastly, it has an onboard digital signal processor and can handle single or differential sensors in grounded or floating connections.

4.1.3 Measurement Cycle

The measurement principle of the Pcap04 is based on a three-step cycle seen in figure 4.2

1. **Pre-charge phase:** The capacitor is charged up via a series resistor to a level close to V_{dd}. The resistor reduces the initial charge current as a short circuit detection method. If there is a short, the voltage level never increases. In this way, the internal circuitry can be protected.

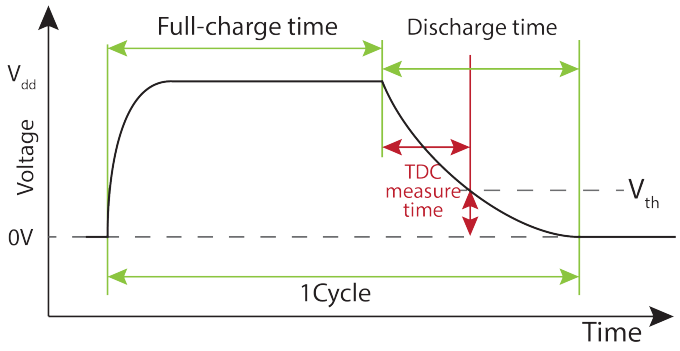


Figure 4.2: A capacitance measurement cycle (based on [158]).

2. **Full-charge phase:** The capacitor is then fully charged up to Vdd without any series resistor.

3. **Discharge phase:** The capacitor is discharged through the discharge resistor down to 0V. The CDC measures the time interval from the initiation of discharge to a threshold voltage crossing.

All of these steps constitute a single "cycle". In applications that require the highest possible conversion rate, it is possible to disable the pre-charge option and directly charge up the element without a series resistor. This is the mode of operation that is used in the final PSAM.

4.1.4 Measurement Sequence

A measurement sequence is a number of measurement cycles. At least four measurements are made per sequence for this PSA setup. Figure 4.3 demonstrates a full read sequence for a floating sensor with full compensation. The orange circled numbers specify the sequence order. First, a measurement is performed on pin PC4 then on PC5, then a measurement is performed on both. This combined measurement is used for parastatic capacitance compensation. The sequence ends with the internal compensation measurement C_{int}. Figure 4.4 demonstrates a floating capacitance sensor being measured with the first three steps of the sequence.

This measurement sequence can repeat a select number of times to produce a collected average reading. This reduces the number of measurements per second but increases the effective number of bits in the reading.

4.2 The Shield Signal

The shield signal is a copy of the measurement cycle but is isolated from it and is on a different pin on the IC. It is used to "remove" any unused rows or columns in the

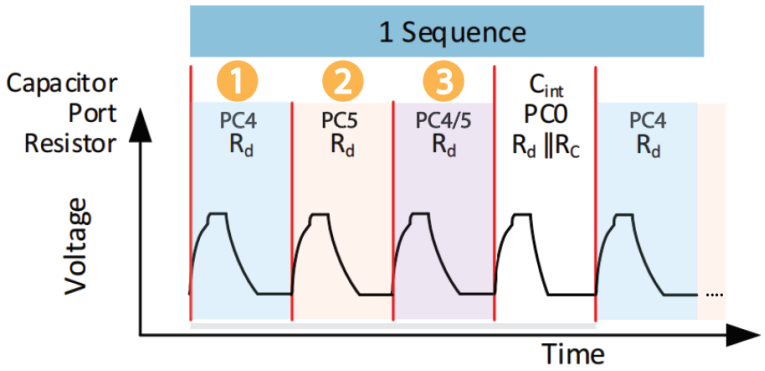


Figure 4.3: Pcap04 CDC measurement sequence ([158] coloring added).

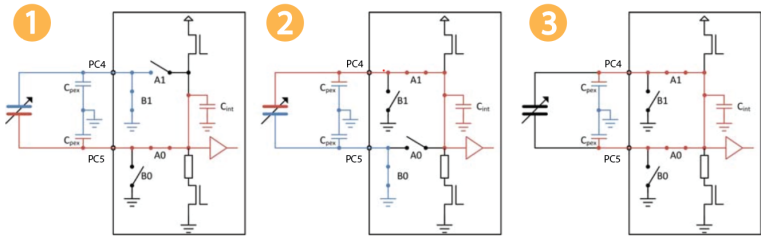


Figure 4.4: Circuit operations of CDC measurement sequence [158].

sensor array by holding them at the same voltage as the measured row or column line. Its operational behavior is demonstrated in figure 4.5. When enabled, it also causes a copy of the measurement sequence to occur on the PC4 or PC5 when they are not being measured. This, however, is undesirable because to make a capacitance measurement on any row or column in the sensor array, there needs to be a ground reference to make a capacitor. Figure 4.6 demonstrates what the read sequence is like on a sensor array using just the IC's shield pin. Note that the signal mirrors the measurement signal, so all of the rows and columns are at the same voltage potential and no capacitance is formed. This was a realization that came during the early development stages. There needs to be a potential difference between lines to make a capacitor (as demonstrated in figure 4.7). A separate shield signal needs to be produced that has the characteristics shown in figure 4.8.

To accomplish this, some extra circuitry and control logic was utilized. The internal guard signal of the ic was disabled.

The circuit consists of a Single Pole Double Throw (SPDT) selector switch and a buffer. The switch is controlled by the Microcontroller Unit (MCU) to select the active measurement port (PC4 or PC5). An internal timer and comparator of the MCU are used to assist in the precise switching times required. This is discussed in more detail in section 5.3. The buffer's high input impedance separates the sensitive measurement signal from the large guard load. It capacitively isolates the measurement signal from

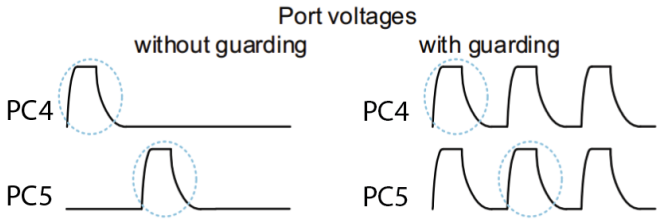


Figure 4.5: Measurement port sequence with shield signal enabled [158].

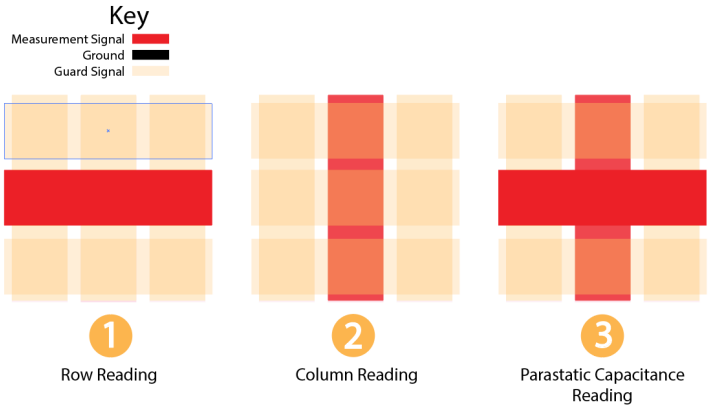


Figure 4.6: Sequence measurement on a sensel with Pcap04 guard enabled.

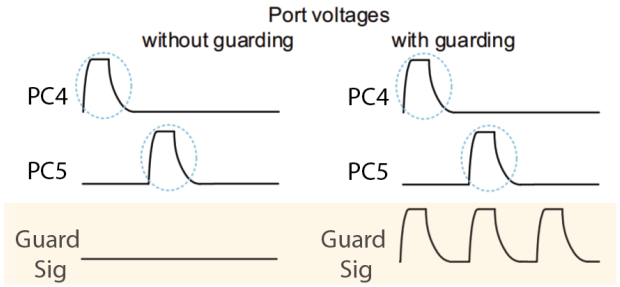


Figure 4.7: Port voltages of a measurement sequence with custom shield signal.

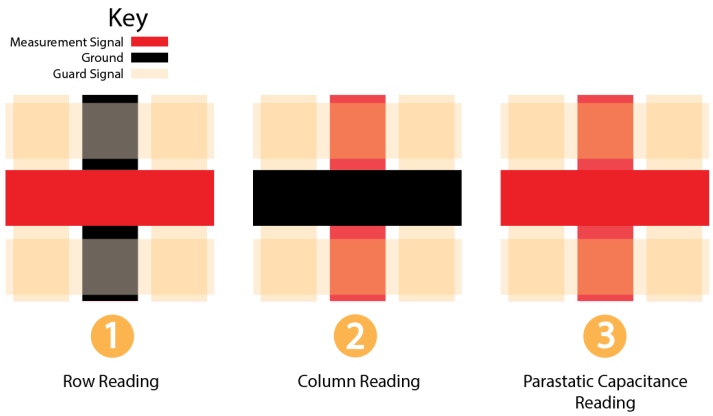


Figure 4.8: Sequence measurement on a on a sensel with custom shield signal (compare with figure 4.5).

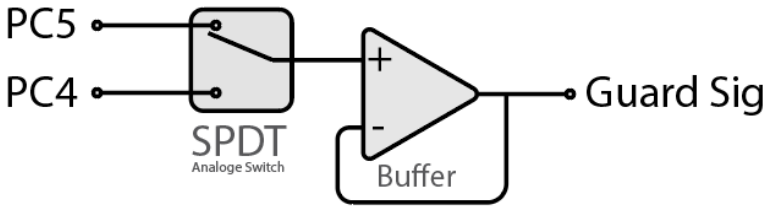


Figure 4.9: Guard creation circuitry.

the guard load. When it all performs correctly, the output signal is what is shown in figure 4.7.

4.3 Hardware Overview

A functional overview of the hardware can be seen in figure 4.10 below. The circuit

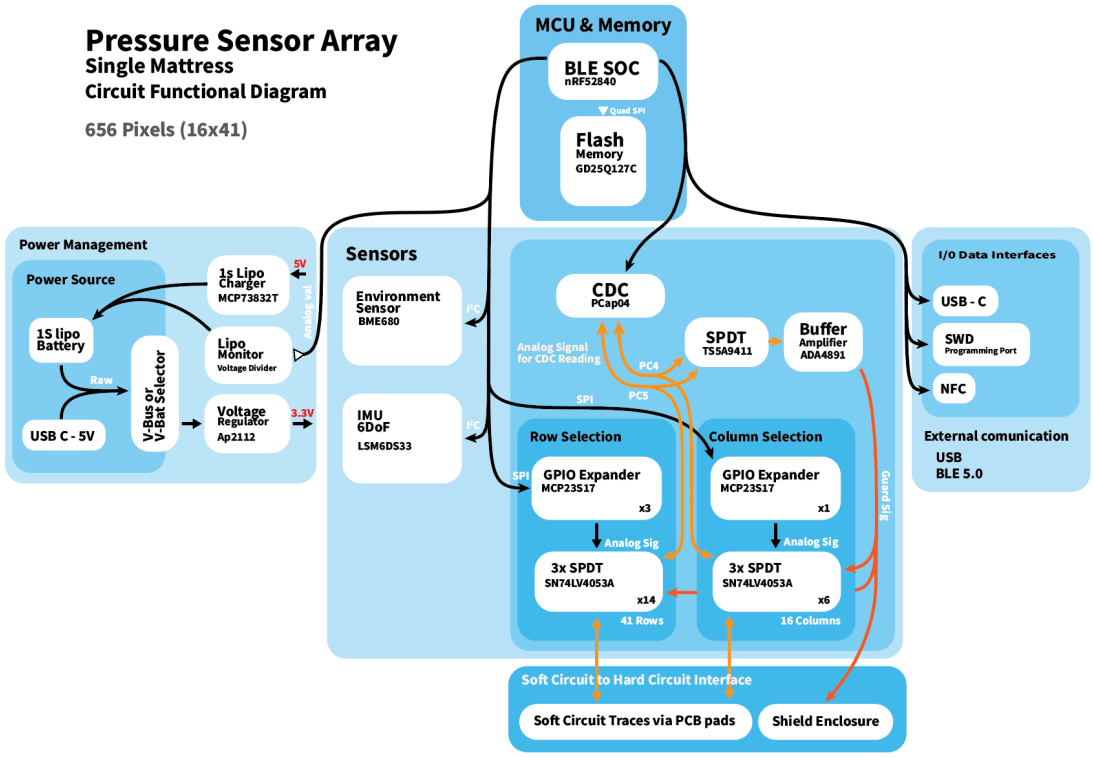


Figure 4.10: Functional overview of the hardware design.

elements and components are represented with white boxes and the blue shaded areas are groupings of type or function. The black arrows represent general connection, control and data. The orange represents capacitive measurement signals and the red is a shield or guard signal.

As previously mentioned, the main capacitive sensing IC is the Pcap04. Its measurement lines are connected to a total of 20 SN74LV4053A Single Pole Double Throw (SPDT) switches (14 for the rows and 6 for the columns). Each IC has 3 switches which is sufficient to cater for the 57 required connections (16 rows + 41 columns). When a switch is activated it connects to the Pcap04 measurement lines. When it is not, it is connected to the shield signal. There are two measurement lines, one for the rows and another for the columns. The output lines of the switches go straight to the pads that connect to the soft sensor.

The Microcontroller Unit (MCU) does not have enough General Purpose Input/Output (GPIO) to cater for all the SPDT switches. Thus, four MCP23S17 General Purpose Input/Output (GPIO) expanders are employed to control all of the switches. Three are for the rows and the other is for the columns. These ICs are then communicated to through a high-speed Serial Peripheral Interface (SPI).

The Microcontroller Unit (MCU) is a Nrf52840 [160] - bluetooth 5.0 System On a Chip (SOC). It was chosen because it has wireless capabilities through Bluetooth Low Energy (BLE). BLE is a communication method that all modern smartphones and tablets have built into them. This makes it a viable way for the data to be presented to a user. It is also anticipated that the PSAM system will be further developed for medical mattresses. As such, the device needs to be able to operate without power for an extended period. BLE has one of the lowest power consumptions for a data transfer rate that meets the requirements for live pressure imaging.

Extra flash memory is added to have enough memory storage for at least one night's worth of pressure images. Other Sensors such as the environment sensor (BME680) and IMU(LSM6DS33) are listed in the overview. They were designed in for potential future sensor fusion in sleep analysis. The other listed elements provide regulated power, battery charging, other communication interfaces(USB) and MCU programming.

The PCB is a 4 layer board designed in Altium designer software [161] and manufactured using JLBPCB Fabrication Services [162]. The components were acquired from various vendors and assembled in-house. Figure demonstrates a completed hard circuit.

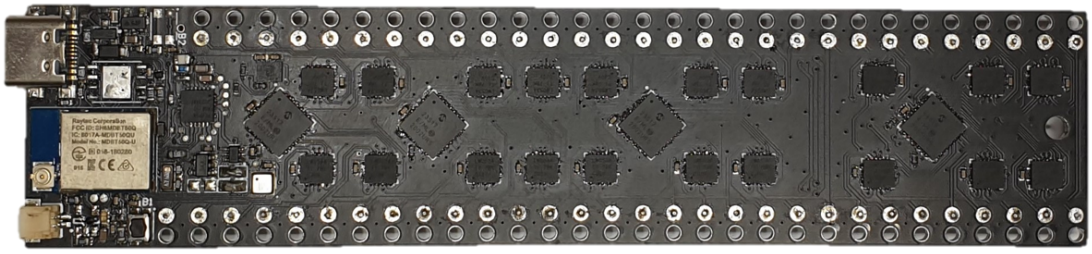


Figure 4.11: Hard circuit prototype fully assembled

4.4 Summary

The hard circuit is the portion of the PSAM that contains all of the electronic components used to select and read a sensel and make a pressure measurement.

A dedicated IC was chosen over a design of passive components because it would reduce circuit complexity, noise, bill of materials and costs. Two chips were investigated The AD7147 and the Pcap04. First, a pressure sensor array was designed with the AD7147 ic. It worked well for small single points of pressure but not with large areas of pressure. The cause of this issue remains unresolved. It is thought that the shield signals on the top rows could insufficiently shield from parastatic capacitance with the body.

Another pressure sensor array was designed utilizing the Pcap04 IC. The configuration of the IC was set to "floating" with regards the capacitive element arrangement. It performed better and allowed for higher sensel reading rates and a wider capacitance sensing range. When using the built-in shield or guard signal, undesirable operations in the measurement lines occurred. It caused all unused measurement lines to be held to the same voltage as the measured ones. For some applications this is desirable but for a PSA based on parallel plate capacitance, one of the electrodes needed to be held to a ground reference. A separate signal was created with a Single Pole Double Throw (SPDT) switch, buffer and internal MCU comparator and timer to form the desired shield signal.

A SPDT switch was used to connect each row and column to the CDC measurement line or shield signal. When a specific sensel is to be read the corresponding row and column are connected to the CDC lines. The rest are connected to the shield signal. As there weren't enough GPIO pins on the nRF52840 to control each SPDT switch, a GPIO expander was used.

Chapter 5

Software: PSAM Firmware

This chapter and the next contain the software components of the project. The hard circuit's firmware is discussed in this chapter and the Graphical User Interface (GUI) and posture classification in the next.

5.1 Development Environment

Since the Nordic Semiconductor nRF52840 SOC was chosen as the MCU, its primary development library is the nRF Software Development Kit (SDK). The specific version used was 17.0.2. Segger Embedded Studio (SES) [163] was the Integrated Development Environment (IDE) that was used to write and compile the device firmware code. All code is written in C. Early device firmware was written with Arduino libraries for rapid prototyping but was quickly changed to nordic's native SDK. It was deemed a better practice to release the final product with the manufacturer's libraries rather than other 3^{rd} party libraries, like Arduino. The development time increased but more control over the ICs peripherals and operation were gained. This was beneficial for controlling the external circuitry to produce the shield signal discussed in 5.3 and later on in 4.2. To note, arduino libraries for the nRF52840 involve the nRF SDK.

5.2 Firmware Structure

An overview of the firmware's structure can be seen in figure 5.1. Detailed documentation of the code can be found in appendixC. The file names for the code are labeled for a queen-sized mattress but still apply to this project as a king single.

5.2.1 Setup Sequence

The program begins by setting up peripherals, external IC initializations and other underlying SDK requirements. They are as follows:

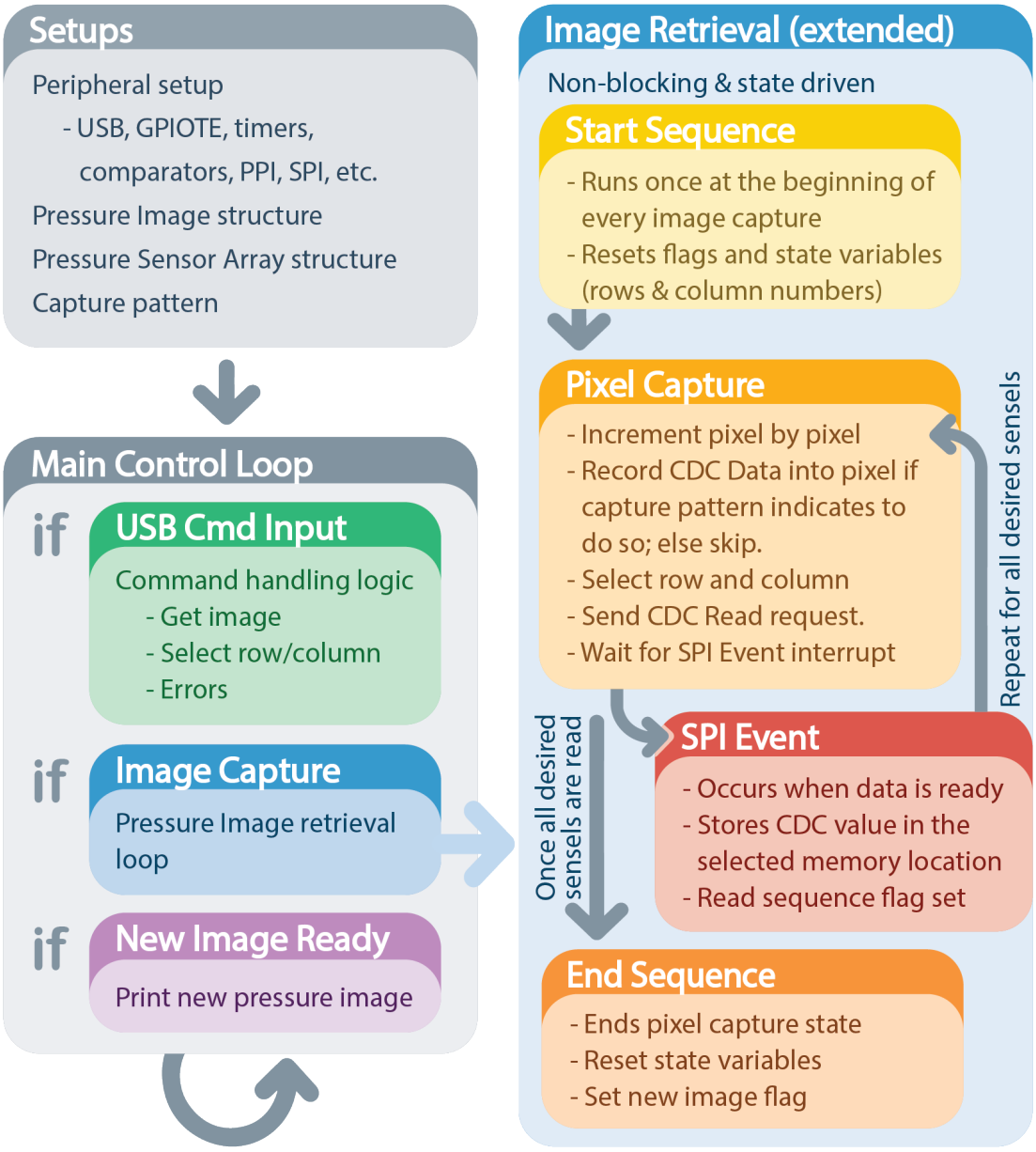


Figure 5.1: Functional overview of PSAM firmware

- **Timers:** Used for the global clock and timing in shield signal generation.
- **USB:** Used for communicating with a host PC.
- **Low Power Comparator:** Used in shield signal generation.
- **Serial Peripheral Interface (SPI):** Used for communicating with external IC, such as the PCAP04 CDC and SN74LV405A GPIO expanders.
- **Programmable Peripheral Interconnect (PPI):** Used to connect internal peripherals together to operate independently of the CPU via tasks and events. They also require an additional enable step after all the event-to-task connections are made.
- **Connected ICs:** After all of the inter-IC communication methods are initialized, each ICs goes through their own initialization.
- **General Purpose Input/Output Tasks and Events (GPIOTE)** Used for autonomously switching GPIOs pins.

5.2.2 Main Loop

After initializations, the program then enters a main continuous loop. A paradigm used throughout the loop is to have minimal blocking code. This means no infinite while loops to wait for a peripheral response. If the code requires a response within a short amount of time then the wait is combined with a timed break out and an error condition is raised. The timed break is based on the passing of a timer used as the global clock.

There are three main sections within the loop. All are accessed through flags, (boolean variables) checked by if conditions. The first of these branches is called when an input is received over the USB, such as a message or command. Some examples of the command message are: get a pressure image, select a column or row and a raised error. The second branch is an image capture subroutine which will be discussed in more detail in the section below. The third and final branch is called when a new pressure image is fully captured and ready to be sent out to the host PC via USB.

The Image Capture Branch

This branch acts like a state machine but continues with the non-blocking paradigm. It uses state flags that determine a state to be entered in and exited each time the main loop is traversed. There are three states in which the image capture branch can be in. They are the start sequence, the pixel capture state and the end sequence.

Start Sequence: This sequence runs once at the beginning of every new "get image" command. It resets all internal state variables or flags.

End Sequence: This sequence also runs once at the end of every "get image" command. It ends the pixel capture state as well as resets the flag for the image capture branch. It also sets the new image-ready flag used for printing out the image.

Pixel Capture Sequence: This is where this image capture branch spends most of its time. An index variable is used to increment pixel by pixel and record the corresponding sensels capacitance.

There is a separate data structure involved in the reading process that acts as an image recording mask. It defines what sensels are read and which are skipped for the frame. This is determined by a separate data structure called a capture pattern. Consider figure 5.2; starting from the top left sensel of the array, each sensel is checked for whether it has been masked. If so, it is skipped, if not, then the row and column of the sensel is selected and a capacitive measurement is taken.

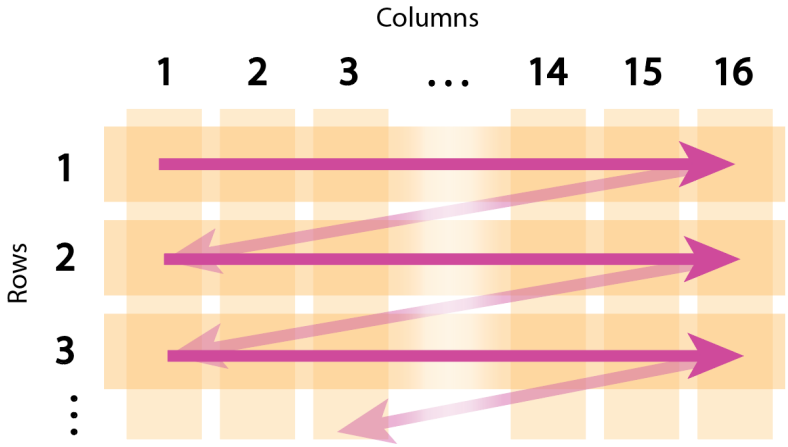
To make a capacitive measurement and record the sensels value, a "start CDC request" is sent. This is a non-blocking request so the program flows back to the main control loop to perform other tasks. It waits for an SPI event interrupt containing capacitive data to occur. Once this has occurred, a subroutine is performed to record the raw CDC value as a pixel in the pressure image. The program then flows back into pixel capture state and another sensel is checked whether to be read and the process repeats.

The pixel masking system was implemented for dynamic pixel reading. This function allows an area of interest to be sampled at a much higher rate. For instance, focusing on the chest area could assist in determining breathing rate and lung capacity. Faster readings also mean they could be averaged to get even more sensitive readings. The default pixel mask is to not skip any pixels (figure 5.2a).

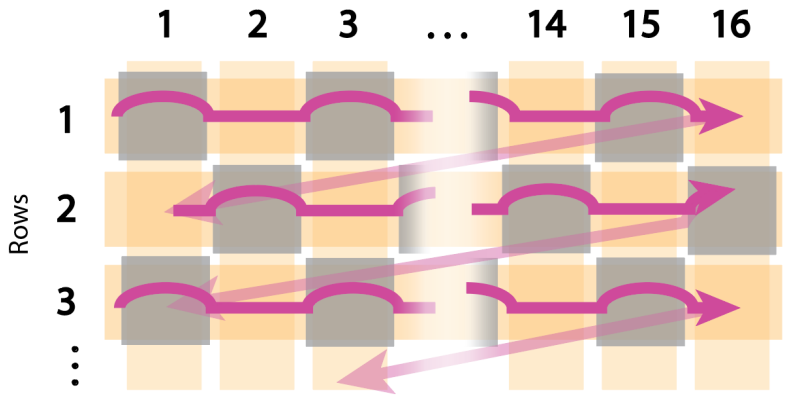
Inputs & Outputs

Capture occurs when the ASCII string "read \n\r" pressure image has been received as a message through the USB

The pressure image is outputted to any USB host through a string of numerical ASCII characters. Each pixel is formatted into a key-value pair. The key is a numerical address that begins at 0 and increments to the number of columns multiplied by the



(a) Without Mask (default capture pattern).



(b) With capture pattern mask (checkerboard shape).

Figure 5.2: Sensel read sequence.

number of rows minus one; $m * n - 1$. The value is the raw chip reading of the capacitance. In the string, the key is separated from the value with a vertical line character "|" and each pixel (key-value pair) is separated with a comma ",".

The data string is surrounded with an element reference similar to an HTML. The header token is "< p_image >" and the footer token is "< /p_image >.

The pressure image data as a string can be seen in figure (5.3). The coloring and grouping are to help identify the key-value (green & blue respectively) pairs in the string.

Figure 5.3: Pressure image data as an ASCII character stream

One of the main reasons the nRF52840 SOC was chosen is because of its Bluetooth

Low Energy (BLE) capabilities. However, it was not implemented in time with the nRF SDK libraries. It was only implemented with Arduino libraries during the early stages of development.

5.3 Shield Signal Generation

A useful feature of the nRF52840 is its Programmable Peripheral Interconnect (PPI). This system enables peripherals to interact autonomously with each other by using tasks and events [164]. This function is independent of the CPU. It is used in generating the guard signal discussed in section 4.2.

When a CDC conversion is requested by the CPU, a number of peripherals work independently to create the shield signal discussed in section 4.2. A comparator is used to determine a rising edge on the PCAP04's PC4 Line. That event triggers a task in a timer to count the time taken for one measurement cycle (4.1.3). After this, an event is fired to set the SPDT GPIO line high causing the selected input line to the buffer to switch. It switches from PC4 to select the other capacitance measurement line PC5.

Once a full sequence has occurred, as demonstrated back in figure 4.7, everything is reset for another reading. This process can happen up to one thousand times per second. Using the PPI allows for precise timing to occur and because it happens autonomously, the CPU is not regularly interrupted.

5.4 Summary

The Firmware for the nRF52840 SOC is based on the nRF SDK and written in C. The framework is based on a non-blocking loop - that is, no infinite wait states. The main loop has branches to deal with inputs and outputs and for acquiring a pressure image. Currently, the inputs and outputs are communicated through the USB. The image is serialized as a human-readable ASCII character string that is set in key-value pairs. Each pixel has a numerical key and raw CDC measurement as the value.

Creating a pressure image uses a state machine-like structure to read the PSA sensel by sensel. The nRF52840 has peripherals and a Programmable Peripheral Interconnect (PPI) is used for generating a shield signal. The Peripherals utilised were a comparator, timer and GPIOs. They were mostly connected together through the PPI to operate independently from the CPU and give the precise timing required to generate the signal.

Chapter 6

Software: GUI and Posture Classification

The Graphical User Interface (GUI) or UI for short, is what displays the pressure image from the PSAM and where most of the image processing and analysis occurs. It is also used for Artificial Neural Network (ANN) training dataset creation by recording and labeling images. The GUI application has been named BoMI for Body Mass Imaging. It is intended for research and development purposes rather than consumer use. It is to serve as a proof of concept only. The eventual consumer GUI would be deployed on a smartphone or tablet.

6.1 GUI Platform and Layout

Windows Presentation Foundation (WPF) [165] was chosen as the UI framework. It is a well established GUI platform that was released in late 2006 with the .Net 3.0 framework [166]. It utilizes the .Net set of libraries and has a large online community, thus making development easier for the novice.

Figure 6.1 demonstrates the layout of the BoMI app. The windows app has two main areas: the pressure image display and the control panel. The display is the large blue rectangular area on the left of figure 6.1 and the control panel is everything on the right. The GUI is written in Extensible Application Markup Language (XAML). There are three main areas in which the app functions and UI elements operate. They are:

- Process and display the pressure image.
- Record and label the pressure image for ANN dataset creation.
- ANN training and posture classification.

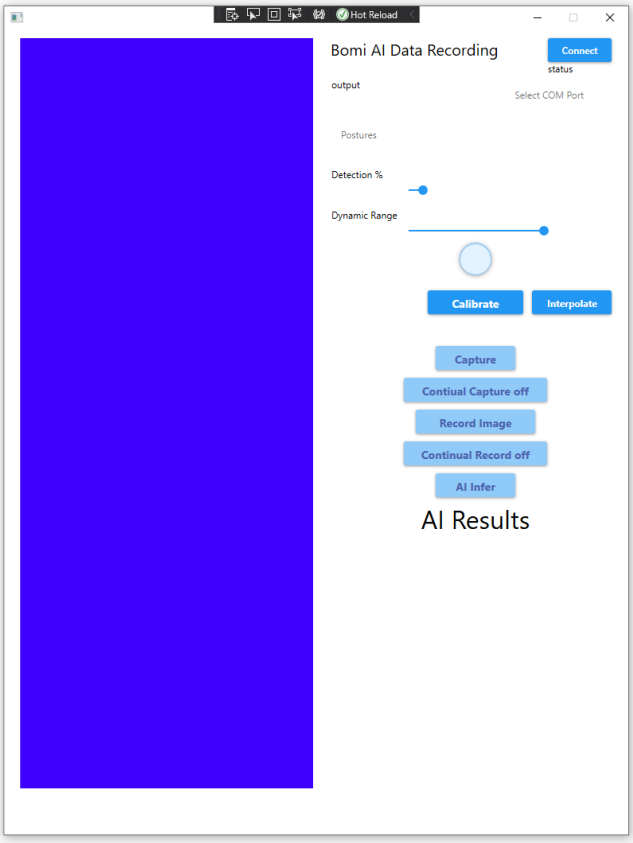


Figure 6.1: Graphical User Interface (GUI) for PSAM

Each of these areas is discussed in the following subsections.

As an architectural note, the application is event driven, that is, when a UI element has been clicked or changed, a registered callback function is executed. Many of the functions discussed are executed with callback functions. All of the key functions will be mentioned hereafter, but for the complete implementation see appendix D - "BoMI App Software"

6.1.1 Image Processing & Display

This feature-set processes the pressure data and displays it to the user as a live image.

Input Image

The pressure image data is received through the COM port via the SerialPort class found in the .Net libraries. It comes as an ASCII text string packet as shown earlier in figure 5.3. The PSAM is first connected to the host PC through a USB, then the appropriate COM port is chosen and the connect button is clicked. When it is clicked

the function "ConnectClick" is called and sets up all the necessary parameters for the serial device.

When a message is received, such as a pressure image, the callback function "OnDataReceived" is executed. This is where most of the image processing and recording occurs. Many of the buttons in the GUI set flags or conditional booleans that determine the flow of code in this function.

The pressure image arrives in sections because it is too large for one usb data packet. Therefore, a pressure image is reconstructed when the starting token and ending token are identified in the serial stream. The function first looks out for the start token "< p_image >". Once found, it records the data into a string until the end of the image token "< /p_image >. The string is parsed into its key-value pairs and then into a pressure image datatype. This is a raw, unprocessed image.

Image Processing

Ideally, the pressure image pixel values should represent a standard unit of pressure measurement, but this is yet to be calibrated. It is not required for the purpose of posture recognition.

A baseline pressure image will act as a "no-pressure image" because each pixel has a non-zero raw value. This image will be used to help calibrate the final image. This image is taken when the device is first connected or when the calibrate button is pressed. A normalized pressure image is a term used to describe an image that is the difference between a current image and the recorded calibration image. Any pixel values that are below zero after normalization are clamped to zero.

To display the pressure image it needs to work within the colour space of a screen's pixels. One standard 8-bit colour channel has 256 intensity levels. A raw capacitance reading from the PSAM can contain up to 24-bits of information. Although this is not the effective bits of the image, there needs to be some scaling of the values so that it can fit in a screens color space. This is done by way of the "Dynamic_Resolution_Scaler" variable. It scales down each pixel by its value which can be changed with the "Dynamic Range" slider on the UI. Any values above 255 and below zero are clamped to those respective values.

Image Display

A custom class called "PressureDisplay" is used to draw and display the image. It does this by drawing coloured squares on a canvas. Each time a new pressure image is received the display is redrawn.

Once normalization and scaling of the raw image have occurred, the image is converted to the HSV colour space. It is easier for humans to determine the difference in

pressure values by colour variations than by intensity changes. The pressure values are only mapped to the hue channel. This image is transformed into the RGB colour space to be rendered to the squares in the pressure display.

The display has the ability to interpolate the image to double the resolution with bilinear interpolation. The interpolate "button" accesses this feature. There is also the option to overlay the pixel value and the row and column numbers. This assists in image debugging.

6.1.2 Image Labeling & Recording

A dataset needs to be created on which a neural network model can be trained. To produce that dataset, recording functionality was implemented.

Two buttons are used to record a pressure image - "Record Image" and "Continual Record on/off". The first button only records one image (the displayed image), and the latter continuously records the stream of live images. The images are labeled based on what item is selected in the "Postures" ComboBox. An item in the ComboBox needs to be selected before recording the image. It is a label for the image. The label name is automatically saved in a folder. There are four image labels in total: three are postures (Left-Hand Side, Right-Hand Side, Back) and the fourth is for the absence of pressure (none). This last category was used for the model to recognize no pressure and the noise floor.

A total of four subjects were recorded in the three different sleeping postures.

The subjects were asked to repeatedly move around the mattress adjusting their position, angle and limb orientation. This produced a variety of images used to train a neural network. A total of 5106 images were recorded within the 4 categories.

- **1,402** back (prone) images.
- **1,895** LHS (left-lateral) images.
- **1,466** RHS (right-lateral) images.
- **343** none (no pressure) images.

The dataset does not have a great variety of subjects but has a large number of samples from each. Images were recorded as a bitmap file. This format did not add any compression and maintained the data's integrity. The pressure image data was recorded in the first (red) of the three colour channels. No values were recorded in the other two. In hindsight, it would have been better to save it as a single channel grey scale image. This would have decreased the size of the input layer of the ANN

Figure 6.2 demonstrates sample images from the four different subjects in the three different positions. The non-inverted set of images on the left show red as areas of

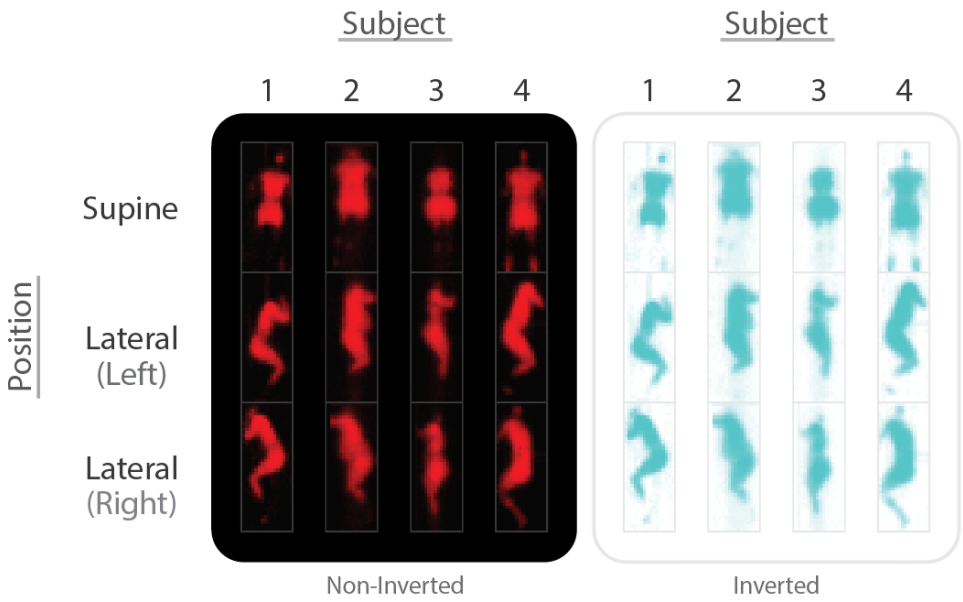


Figure 6.2: Example images of the four different subjects in three different sleeping postures.

pressure and black as areas of no pressure. The inverted image on the right is another colour perspective with cyan areas of pressure and white areas of no pressure.

6.1.3 ANN Training & Posture classification

A machine learning algorithm is used to classify the pressure images for sleep posture. Tensorflow [137] is a popular open source machine learning library developed by google researchers. Keras [167] is a higher level API that allows the user to switch between different backends or machine learning frameworks, like TensorFlow. Keras is simple and it minimizes the number of user actions for common use cases [167]. The keras api with the tensorflow backend are sufficient to perform image classification on the PSAMs pressure images. Python was the programming language used to create the model.

Artificial Neural Network Training

The whole program can be found in appendix E, but an overview is covered hereafter. First, the data set is loaded into the program. It is split into two datasets - a training set and a validation set. 75% of the recorded images are used for the training dataset and the remaining 25% for a validation set. The neural network model is specified by sequentially supplying the desired layers to the keras function "Sequential" 6.3. First, the input layer is specified and given details such as image dimensions and the number of color channels. Next, it is flattened to 1968 nodes for connecting to the hidden layers.

```
model = keras.Sequential([
    layers.Input((41, 16, 3), name="input"),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(8, activation='relu'),
    layers.Dense(4, activation='softmax')
])
```

Figure 6.3: Keras: sequential function

This number comes from each pixel and color channel in the image ($41 \times 16 \times 3 = 1968$). Two hidden layers are then specified. The first layer has 64 nodes and the second has eight. It is a simple network to reduce the potential for over-fitting. It is also a dense network in that each node of a layer is connected to every node in the next. A visual depiction can be seen in figure 6.5. The colored circles are the nodes and the light grey lines are the connections. The number below the groups of nodes are the actual amount used in the layer for this model. A console printout of the model (figure 6.4)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 1968)              0

 dense (Dense)               (None, 64)                126016

 dense_1 (Dense)             (None, 8)                 520

 dense_2 (Dense)             (None, 4)                 36

=================================================================
Total params: 126,572
Trainable params: 126,572
Non-trainable params: 0
_________________________________________________________________
```

Figure 6.4: Console printout of Keras neural network model

verifies the type, shape and number of connections (parameters).

The model is then compiled with a loss function of sparse categorical cross entropy and an Adam (adaptive moment estimation) optimizer as seen in the code of figure 6.6. It is trained with 15 epochs and a batch size of 10.

Finally, the network is tested with the validation dataset. The final epoch of the training results was a loss of 0.1090 and an accuracy of 0.9911 or 99.1%. The validation test results are a loss of 0.1009 and an accuracy 0.9969 or 99.6% Lastly, the model is saved so it can be used in the classification program.

The Keras API keeps the user's code simple and clean. Here, all of the training is performed in less than 100 lines of user code.

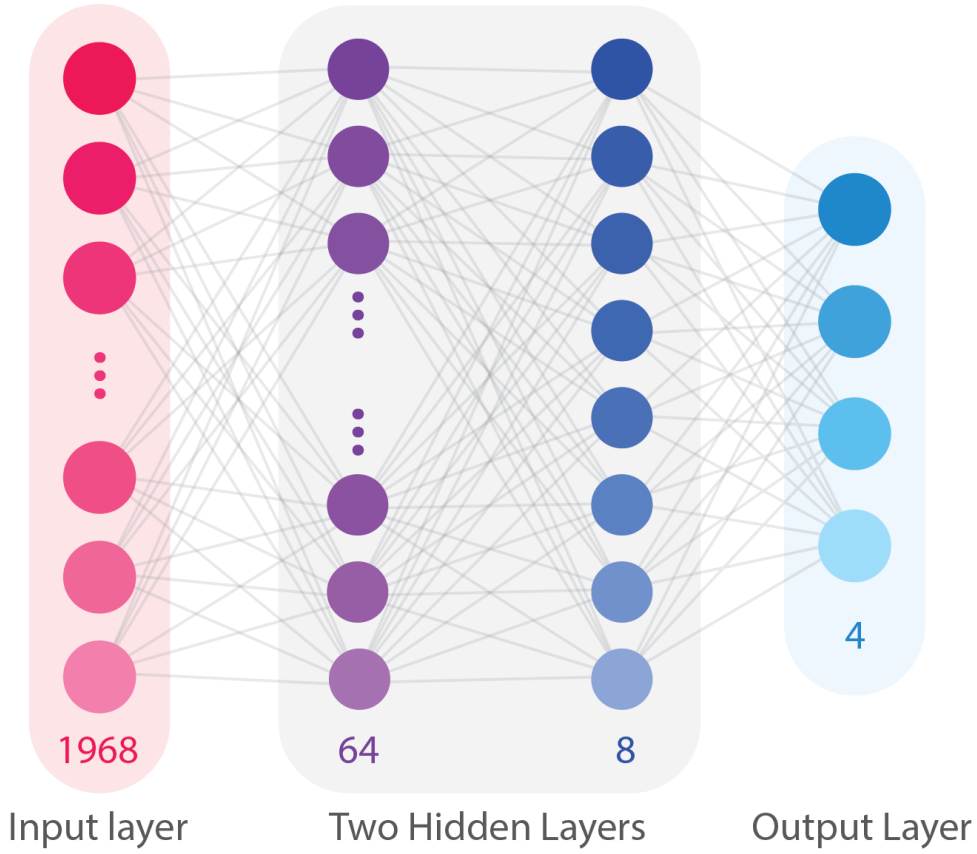


Figure 6.5: Posture monitoring ANN model

```
model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=keras.optimizers.Adam(lr=1e-4),
    metrics=["accuracy"]
)
```

Figure 6.6: Keras: model compile function

Posture Classification

Live posture classification was the last feature added to the acsbomi app. Investigations into using the pre-trained model natively in the WPF app and .Net framework proved inoperable in the allotted time. A simple solution involving the acsbomi app and a dedicated python program communicating through a tcp socket was utilized. ZeroMQ [ZeroMQ:Online] is an open source universal messaging library and has libraries for both .Net and python.

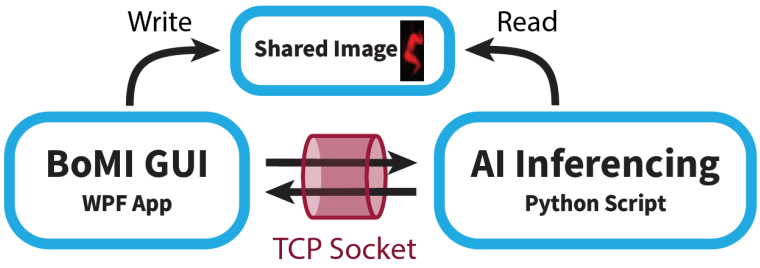


Figure 6.7: BoMI app and python AI program interaction

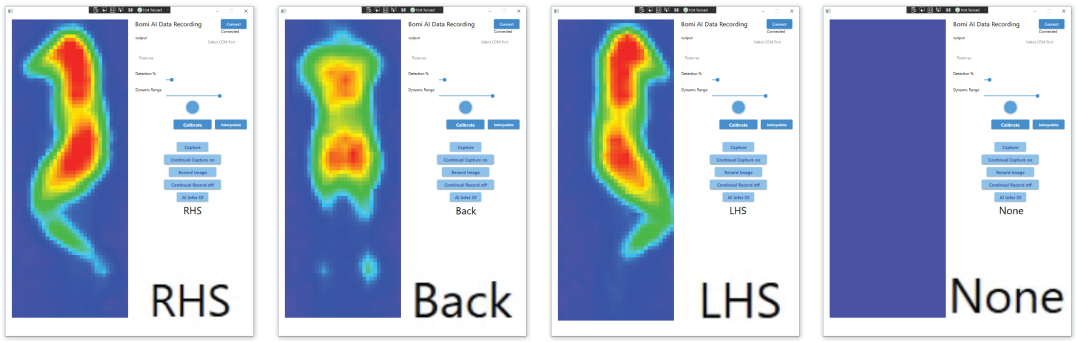
The infer button enables a flag that causes the newly received pressure image to be saved into specified folder. Each time a new image is received it is saved over the previous one. That folder is a shared folder to a separate python program. The python program reads in the image, performs an inference and sends a message to the acsbomi app through the TCP socket. See figure 6.7 for a visual representation. The code for the python classification program is demonstrated in appendix E.

It would be more desirable to have The GUI and inference app to work in the same environment, but the method employed was still able to allow live classification. Figure 6.8 demonstrates four sleeping postures being correctly classified. The displays have interpolation enabled to produce a higher resolution. The label for each has been magnified for legibility purposes. Figure 6.9 is a photograph of a subject lying on the PSAM with the live pressure image overlaid and the classified posture above it.

6.2 Summary & Discussion

A desktop GUI application called BoMI was developed to display the pressure image from the PSA device.

It was also programmed to have the ability to record and label images for ANN dataset creation. A total of 5106 images were recorded for use in the dataset. 75% of the data was used for training and the other 25% was used for validation. The images were of four different body types and gender in different angles and orientations. The data was labeled with 4 classes, supine (back), prone (front), lateral (side) and no



(a) Right-Lateral (b) Supine (c) Left-Lateral (d) No pressure

Figure 6.8: BoMI app with AI inference of the four classifications

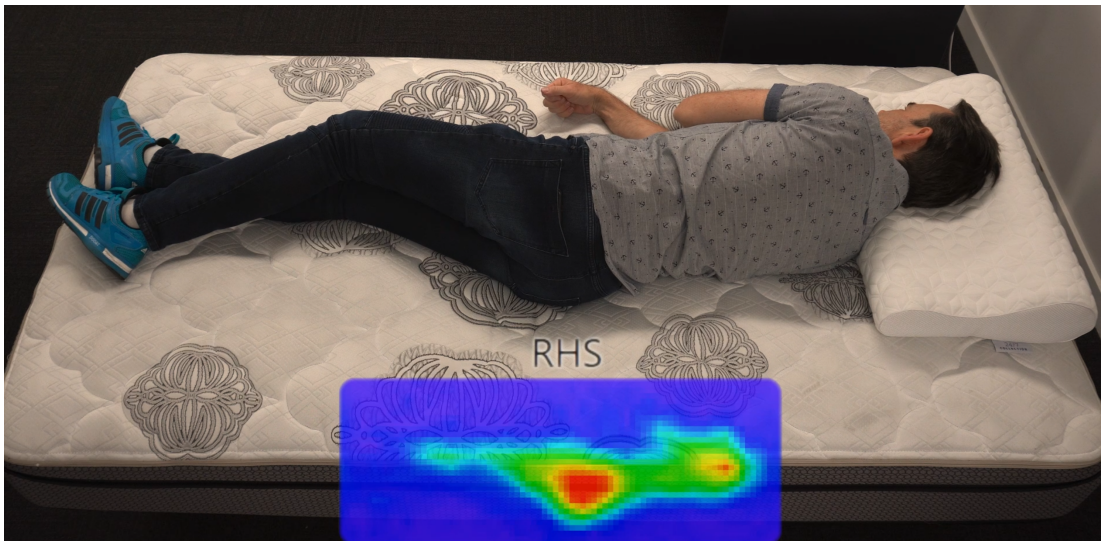


Figure 6.9: Image of subject resting on PSAM with a live pressure image overlay and classified posture above.

pressure (none).

The ML that was used for inferring posture was a standard dense Artificial Neural Network (ANN). It was created with the Keras API on a Tensorflow backend. Training of the network utilized an Adaptive Moment Estimation Optimizer along with a sparse categorical cross-entropy loss function. Validation tests resulted in an accuracy of 99.1% The pressure image was saved as a three-channel RGB bitmap image. It would have been more appropriate to record the images as a single-channel "grey scale" image. This would have decreased the nodes of the input layer in the neural network model from 1968 to 656. This naturally also reduces the note parameter count. The high level of accuracy does bring up concerns of overfitting.

Classification of a live pressure image was also demonstrated. It was achieved by connecting the BoMI app to a separate python classification program through a TCP

Socket. A shared file with the current image was used as well. This method was adequate as a proof of concept for live classification but not a long-term solution. It would be better to have the feature built into the main GUI application.

Chapter 7

Cost & Feasibility Analysis

One of the main objectives of this project was to design the pressure sensor array so that it would be cost-effective. Cost-effective was defined with an acceptable upper bound of $250 NZD (section 1.2.1). The analysis to follow is not all-encompassing. It is an educated estimate based on material costs.

7.1 Hard Circuit Costs

Table 7.1 below lists the cost of a hard circuit as of march 2021. This is a collection of individual component costs and PCB manufacturing costs. It does not include assembly costs. The table reflects the fact that component costs decrease as you increase the purchase amount. The component costs are mostly based on digikey [168] listings and JLCBCB [162] board manufacturing costs.

Quantity	Unit Price NZD	Extended Price NZD
50	≈ 28.7	≈ 2000
500	≈ 20.6	≈ 10300
1000	≈ 18.4	≈ 18400
10000	≈ 16.98	≈ 169800

Table 7.1: Hard circuit cost estimation based on quantity created

7.2 Soft Circuit Costs

The material for a soft circuit was quoted to TCG as $8 USD per linear meter by Saintyear Electronic Technologies [143]. Each soft circuit needs two linear meters and there are two circuits per sensor array. This equates to four linear meters of fabric and a total cost of $32 USD. The conductive mesh for shielding is $7.87 USD per linear

meter from Xing Tech Company Limited[169] and 4 meters are required. This equates to $31.48 USD.

Using an exchange rate of 1.44 USD to NZD, the converted expenses come to $46.08 NZD and $45.33 NZD respectively. The conductive material equates to a combined cost of $91.44 NZD

7.3 Potential Production Plan

A potential PSAM production plan based on the production steps discussed in section 3.1.3 can be seen in figure 7.1 The soft sensor requires two soft circuits and a dielectric

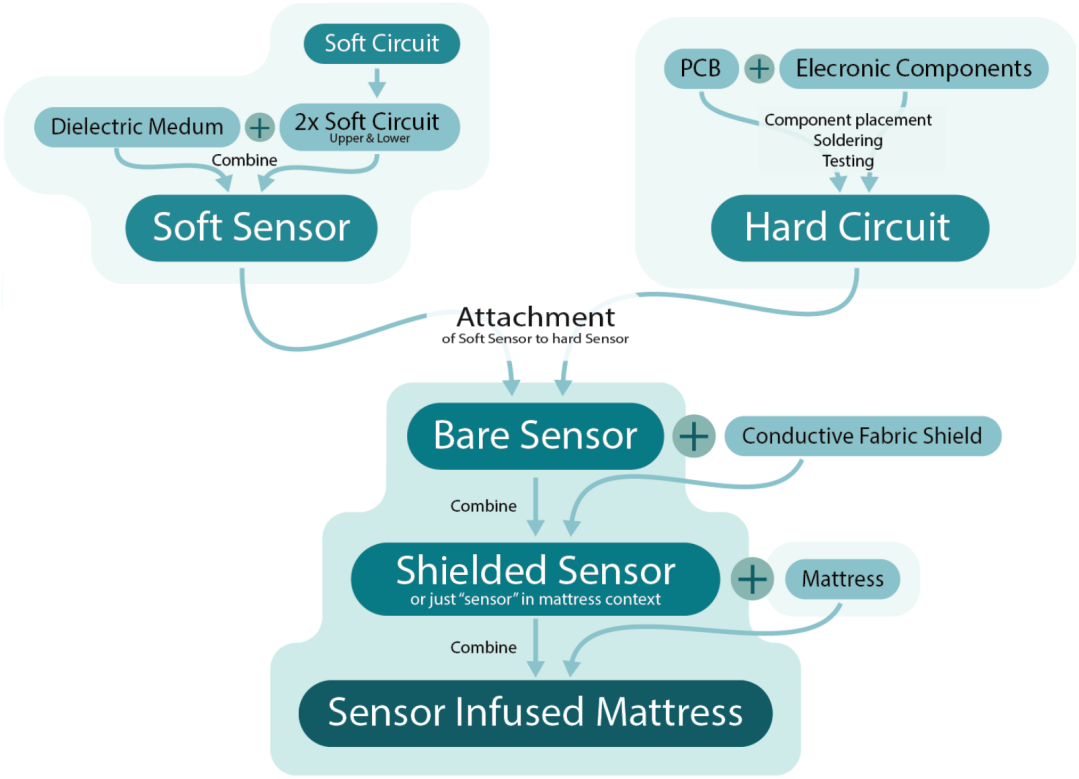


Figure 7.1: Production steps required to manufacture a PSAM

medium. The hard circuit is composed of electronic components and a PCB. The bare internal sensor is the combination of these two. This is then electronically protected with a conductive fabric shield. Then this shielded sensor is placed into a mattress to become an PSAM

7.4 Total Estimated Cost

A potential production plan was discussed above and material costs were taken to the comfort group. After internal discussions, they returned with foam and labor costs as follows.

- Hard circuit raw materials: $16.98
- Soft circuits raw materials: $46.08
- Shield fabric costs: $45.33
- Dielectric foam: $19.34
- Assembly labour and overheads for sensor array assembly and embedded into the mattress: ≈$100

All prices have been converted to NZD using a conversion rate of 1.44.

The total cost equates to $227.73 NZD per sensor unit. It is below the upper cost-effective limit of $250 NZD. This is an additional cost above any mattress cost and is based on a production of 10,000 units.

This is a promising estimate and demonstrates that a low-cost pressure sensor array can be embedded into a consumer mattress. There is the potential for this cost to be reduced even further through optimizations. For example, finding a capacitive measuring method that is more immune to noise and parastatic capacitance would allow the conductive fabric shield to be removed.

Chapter 8

Conclusions, Discussions & Future Work

8.1 Review of objectives

The objectives outlined in section 1.3 were as follows:

1. Investigate and develop a cost-effective sleep posture monitoring unit that can be embedded into a consumer mattress.
2. Investigate pressure sensor arrays as a potential and sufficient posture monitoring system.
3. Perform an analysis of the data to determine basic sleeping postures.
4. Review the effectiveness of such device and data analysis technique in determining posture and other physiological sleep factors.

And that the project should deliver:

- A prototype of the monitoring device.
- Software to analyse and display the pressure image and identify basic sleep postures.
- A basic cost analysis for producing such a device including the potential manufacturing methods to achieve it.

All of these items have been achieved a demonstrated.

8.2 Results

A cost-effective posture monitoring system that can be embedded into a consumer mattress has been developed and demonstrated. It is a capacitive-based Pressure Sensor Array (PSA) that utilises low-cost materials and a simple construction. A software solution was also developed to demonstrate posture classification abilities.

The hardware of the pressure sensor array is composed of a hard circuit and a soft sensor. The soft sensor contains all of pressure sensing elements (sensels) and the hard circuit contains all of the line selection and capacitive measurement circuitry. Two Capacitance-to-Digital Converter (CDC) IC were investigated for use in an array: the AD7147 and Pcap04. The Pcap04 Capacitance-to-Digital Converter (CDC) IC was selected because it produced a better sensor array for large areas of pressure.

In working with the Pcap04, a guard signal was desired to capacitively decouple all non-measured rows and columns and shield the array from parastatic capacitance. The internal guard signal of the IC produced undesirable operations within the two measurements lines(row and column) when enabled. It would not allow the measurement of a sensel because it would not produce a voltage difference between the sensels electrodes. To make a capacitor, there need to be a voltage difference. The desired operation was achieved by disabling the internal shield signal of the IC and creating a separate one with some external circuitry and the MCUs peripherals.

The soft sensor was constructed using low-cost conductive fabric and TPU foam. The conductive copper and nickel coated nylon fabric was laser cut into basic circuit shapes for use as the upper and lower layers of the sensor. A Low hardness vf52-40 foam was used as the dielectric medium.

To test the durability of the soft sensor, 80,000 pound-cycle test was performed and showed no signs of fabric tearing. The hard circuit to soft circuit interface was investigated using a variety of connections, but directly soldering them together was the simplest and most cost-effective. Experiments were conducted to determine optimal reflow conditions on the fabric. The optimal soldering temperature for MG Chemicals no-clean solder paste (Sn63 Pb/37) on the conductive fabric was between 190^{o}C - 200^{o}C. It was determined that temperatures above 200^{o}C caused the shape of the conductive fabric to deform. It would begin to shrink and crinkle.

The software components of the project include the device firmware and a graphical user interface. The firmware was developed using the MCU native firmware nRF SDK. It contained essential features for addressing each sensel in the soft circuit and performing capacitive measurements. The pressure image was then serialized and outputted to a host computer via a USB connection. The MCU (nrf52840) provided adequate performance to measure and output a live pressure image. The ICs peripherals also allowed for reduced external components for generating the shield signal.

The GUI was developed to process and display the pressure image. It was also programmed with the ability to record and label pressure images for use in training an Artificial Neural Network (ANN) for sleep posture detection and classification. A dense ANN is used as the Machine Learning (ML) model. It was built using the Keras API and on a Tensorflow backend. The Posture classification had an accuracy of 99.6% for three postures; left-lateral, right-lateral and supine. It was also trained to detect no pressure as well. The high accuracy rate is promising but raises concerns about overfitting.

A cost analysis was performed and estimations came to approximately $227.73 NZD. This is below is below the upper bound set by TCG for an acceptable expense for a mattress (section 1.2.1). A Potential mass manufacturing method has also been put forward and demonstrated in section 7.3

8.3 Discussions

The use of a PSA as a sleep posture monitoring device is effective because of the pressure image it produces. A Pressure Image (PI) contains the shape of the subject and quantifies the pressure at each point. A collection of PI can reflect the relative redistribution of mass in the short term, and mass gain or loss in the long term. The redistribution of mass could reflect an arm or leg moving which can be easily seen by anyone inspecting an image. Additionally, the pressure redistribution may occur in a localized area, such as breathing movements within the chest area. On this note, the monitoring of a subject's breathing rate has been demonstrated in literature. However, by employing a more sensitive array and in-depth analysis of chest area movements, lung capacity has the potential to be inferred.

The PSAM that has been developed for this project has produced a clean pressure image that can be visually validated. The resolution of the array is sufficient to determine a human's figure, including its limbs and body shape. The trained ANN also validates that there are sufficient features to determine three sleep postures on any point of the array.

Only basic posture recognition has been demonstrated by the classification algorithm in this project. This array has become a springboard for future investigations in the realm of sleep monitoring.

One of the next essential steps for advancing the sensor array is to characterize its performance. This would involve determining its accuracy and sensitivity for pressure measurements as well as proving th long-term stability in readings. This is discussed further in the next section.

Regarding the ML model, the dataset had many images of the test subject on which it was trained. However, there were multiple images of similar positions. This had the

consequence of over-fitting. A greater variety of images including different subjects with varying body shapes would be needed to create a more robust model. The dense ANN model could also have undergone pruning of its parameters to assist in over-fitting. A CNN model was also trained but did not produce the same level of results that a basic dense ANN achieved. This could be due to the low resolution of the array.

An important lesson was learned during the investigation of an adequate soft circuit to hard circuit connection. The first attempt of connecting the conductive fabric to the hard circuit through a wire or directly to the hard circuit was deemed inadequate. This was a premature conclusion. This was due to poor soldering practices and lack of temperature-controlled heating. It was only after months of investigating other connection solutions and dealing with difficulties maintaining good electrical connections that soldering was revisited. This time, it was done with more care and careful consideration. It lead to the final result of using solder paste and a temperature-controlled platform to solder directly to a PCB pad.

Overall, the researchers of this project are content with the development of the low-cost, medium-resolution sensor array presented in this thesis.

8.4 Ongoing Work

There are a few avenues of future work that could be investigated. This last section discusses the areas of potential development.

8.4.1 Hard Circuit

CDC Measurements Without a Shield: There are capacitive sensing PSA commercially available that are not encompassed in a farady cage or shield. Investigating a capacitive sensing method that could operate without a shield would remove that expense. However, there are patents for specific capacitance measurement methods that are used in pressure sensor arrays. These also need to be considered.

Wifi Communication Utilizing wifi communication would allow the pressure images to be transferred to a cloud server through the home network. This would allow a larger history of pressure data to be stored and analysed. It could also be accessed faster than via Bluetooth because of its limited bandwidth.

8.4.2 Soft Sensor

No Laser Cutting: Laser-cutting conductive fabric is a viable solution for soft circuit creation. However, it would be more effective to avoid cutting out the shapes

altogether. This would reduce the number of manufacturing steps. Screen printing is a way to achieve this, but as discussed in section 3.1.2, for the amount of ink required, it is too costly. An alternate solution would be to mask the fabric before it is coated in a conductive substance. This could be a surface mask or an impregnated one. These techniques are demonstrated by Wu et al. [170] Conversely, an etching process could remove undesirable areas of coating [171].

Shaped Sensor Lines: It is important that the soft circuits do not alter the comfort factor of the mattress. If a low-cost conductive stretch fabric can not be attained, relief cuts or shaped sensor lines should be considered. Initial tests mentioned in section 3.1.3 have been conducted but further qualitative analysis is needed.

8.4.3 Pressure Data Analysis

Pressure Data Calibration: This item involves modeling the mechanical properties of the dielectric medium so that raw capacitance measurements are turned into standardized pressure units. It is essential in determining the long-term stability of the sensor and what pressure level it can resolve. A test platform is already in development to determine the accuracy, sensitivity, hysteresis and drift (see figure 8.1). Machine learning has also been employed for accurate pressure reading

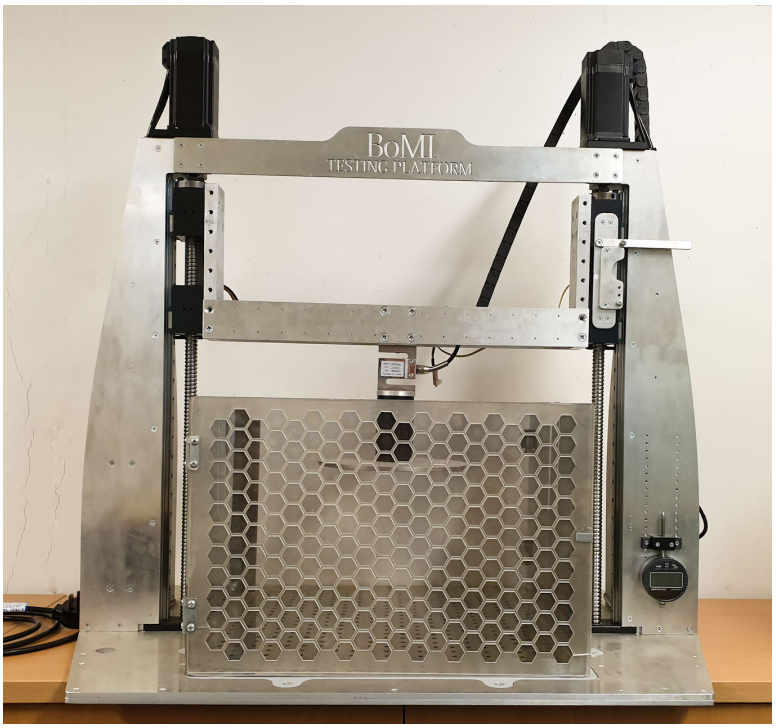


Figure 8.1: Testing platform to be used for measuring and determining the characteristics of the PSAM

calibrations [101]. It was performed on piezoresistive PSAs. This is also an avenue for investigation in calibrating the PSAM.

Extraction of More Physiological Data The ML model used in this project only demonstrates three postures. Adding more detailed postures is an option. However, the publication by clever et al. [87] demonstrates that pose estimation and body shape can be achieved. This is another avenue of development. Another potential area of investigation would be that of determining spinal alignment from a pressure image.

Smartphone App The desktop application is a suitable environment for research and development but at some point, the pressure image and posture data need to be presented in a consumer-friendly way. A smartphone app is a way to present that to the user, as smartphones are the more prevalent form of human-computer interaction.

Human Body Detection This would be to determine the difference between human and non-human figures such as pets or other heavy objects.

Glossary

apnea When airflow is absent or nearly absent for at least 10 seconds[172]. 12

ballistography Record forces generated by the body due to body movement. This could include breathing, motion and the mechanical action of the beating heart. 3, 23

electret dielectric material with the presence of quasi-permanent real charges on the surface or in the bulk of the material. 23

electrocardiogram electrical measurements of the heart. xi, 14, 16

electroencephalogram Is a record of the oscillations of brain electric potentials recorded from 20 to 256 electrodes attached to the human scalp [173]. xi, 15

electromyography Measurements in electrical activity in response to a nerve's stimulation of a muscle. xi, 14

HSV hue, saturation, value. A color model used in graphics software. 65

hypopnea When airflow is at least 30 percent lower (or thoracoabdominal movement) than baseline values for at least 10 seconds[172]. 14

hypotonia decreased muscle tone. 12

Magnetic Resonance Imaging An imaging technology that produces 3D images of the inside of a human body. xii

polysomnography Is a sleep study that records oxygen levels in the blood, brain waves, heart rate, breathing and leg movement[43]. v, xiii, 3, 11, 14

transcutaneously acros the depth of the skin. 15

Bibliography

[1] P. Chung, A. Rowe, M. Etemadi, H. Lee, and S. Roy, "Fabric-based pressure sensor array for decubitus ulcer monitoring," in *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2013, pp. 6506–6509.

[2] M. Yip, D. Da He, E. Winokur, A. G. Balderrama, R. Sheridan, and H. Ma, "A flexible pressure monitoring system for pressure ulcer prevention," in *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, 2009, pp. 1212–1215.

[3] S. S. Gilakjani, H. Azimi, M. Bouchard, R. A. Goubran, and F. Knoefel, "Improved sensor selection method during movement for breathing rate estimation with unobtrusive pressure sensor arrays," in *2018 IEEE Sensors Applications Symposium (SAS)*. IEEE, 2018, pp. 1–6.

[4] G. Matar, G. Kaddoum, J. Carrier, and J.-M. Lina, "Kalman filtering for posture-adaptive in-bed breathing rate monitoring using bed-sheet pressure sensors," *IEEE Sensors Journal*, vol. 21, no. 13, pp. 14 339–14 351, 2020.

[5] "Sleep diary: How and why you should keep one," Feb 2021. [Online]. Available: https://www.sleepfoundation.org/sleep-diary

[6] S. Mansfield, K. Obraczka, and S. Roy, "Pressure injury prevention: a survey," *IEEE reviews in biomedical engineering*, vol. 13, pp. 352–368, 2019.

[7] Z. Chen and Y. Wang, "Remote recognition of in-bed postures using a thermopile array sensor with machine learning," *IEEE Sensors Journal*, vol. 21, no. 9, pp. 10 428–10 436, 2021.

[8] Y.-Y. Li, Y.-J. Lei, L. C.-L. Chen, and Y.-P. Hung, "Sleep posture classification with multi-stream cnn using vertical distance map," in *2018 International Workshop on Advanced Image Technology (IWAIT)*. IEEE, 2018, pp. 1–4.

[9] P. Jeng and L.-C. Wang, "Stream data analysis of body sensors for sleep posture monitoring: An automatic labelling approach," in *2017 26th Wireless and Optical Communication Conference (WOCC)*. IEEE, 2017, pp. 1–4.

[10] S. Mukherjee, S. R. Patel, S. N. Kales, N. T. Ayas, K. P. Strohl, D. Gozal, and A. Malhotra, "An official american thoracic society statement: the importance of healthy sleep. recommendations and future priorities," *American journal of respiratory and critical care medicine*, vol. 191, no. 12, pp. 1450–1458, 2015.

[11] T. W. Strine and D. P. Chapman, "Associations of frequent sleep insufficiency with health-related quality of life and health behaviors," *Sleep medicine*, vol. 6, no. 1, pp. 23–27, 2005.

[12] K. Wunsch, N. Kasten, and R. Fuchs, "The effect of physical activity on sleep quality, well-being, and affect in academic stress periods," *Nature and science of sleep*, vol. 9, p. 117, 2017.

[13] A. N. S. Bisson, S. A. Robinson, and M. E. Lachman, "Walk to a better night of sleep: testing the relationship between physical activity and sleep," *Sleep health*, vol. 5, no. 5, pp. 487–494, 2019.

[14] Z. A. Caddick, K. Gregory, L. Arsintescu, and E. E. Flynn-Evans, "A review of the environmental parameters necessary for an optimal sleep environment," *Building and environment*, vol. 132, pp. 11–20, 2018.

[15] M.-P. St-Onge, A. Mikic, and C. E. Pietrolungo, "Effects of diet on sleep quality," *Advances in nutrition*, vol. 7, no. 5, pp. 938–949, 2016.

[16] J. A. Mindell and A. A. Williamson, "Benefits of a bedtime routine in young children: Sleep, development, and beyond," *Sleep medicine reviews*, vol. 40, pp. 93–108, 2018.

[17] F. P. Cappuccio, L. D'Elia, P. Strazzullo, and M. A. Miller, "Sleep duration and all-cause mortality: a systematic review and meta-analysis of prospective studies," *Sleep*, vol. 33, no. 5, pp. 585–592, 2010.

[18] M. Hirshkowitz, K. Whiton, S. M. Albert, C. Alessi, O. Bruni, L. DonCarlos, N. Hazen, J. Herman, P. J. A. Hillard, E. S. Katz *et al.*, "National sleep foundation's updated sleep duration recommendations," *Sleep health*, vol. 1, no. 4, pp. 233–243, 2015.

[19] E. S. Ford, T. J. Cunningham, and J. B. Croft, "Trends in self-reported sleep duration among us adults from 1985 to 2012," *Sleep*, vol. 38, no. 5, pp. 829–832, 2015.

[20] V. K. Chattu, M. Manzar, S. Kumary, D. Burman, D. W. Spence, S. R. Pandi-Perumal *et al.*, "The global problem of insufficient sleep and its serious public health implications," in *Healthcare*, vol. 7, no. 1. Multidisciplinary Digital Publishing Institute, 2019, p. 1.

[21] D. P. Chapman, J. B. Croft, Y. Liu, G. S. Perry, L. R. Presley-Cantrell, and E. S. Ford, "Excess frequent insufficient sleep in american indians/alaska natives," *Journal of environmental and public health*, vol. 2013, 2013.

[22] S. Ai, J. Zhang, G. Zhao, N. Wang, G. Li, H.-C. So, Y. Liu, S. W.-H. Chau, J. Chen, X. Tan *et al.*, "Causal associations of short and long sleep durations with 12 cardiovascular diseases: linear and nonlinear mendelian randomization analyses in uk biobank," *European Heart Journal*, vol. 42, no. 34, pp. 3349–3357, 2021.

[23] J. Xu, K. D. Kochanek, S. L. Murphy, and E. Arias, *Mortality in the United States, 2012*. US Department of Health and Human Services, Centers for Disease Control and . . . , 2014, no. 168.

[24] D. L. Littlewood, S. D. Kyle, L.-A. Carter, S. Peters, D. Pratt, and P. Gooding, "Short sleep duration and poor sleep quality predict next-day suicidal ideation: an ecological momentary assessment study," *Psychological medicine*, vol. 49, no. 3, pp. 403–411, 2019.

[25] G. A. Kerkhof, "Epidemiology of sleep and sleep disorders in the netherlands," *Sleep medicine*, vol. 30, pp. 229–239, 2017.

[26] C. A. Magee, D. C. Iverson, and P. Caputi, "Factors associated with short and long sleep," *Preventive medicine*, vol. 49, no. 6, pp. 461–467, 2009.

[27] J. C. Lo, J. A. Groeger, G. H. Cheng, D.-J. Dijk, and M. W. Chee, "Self-reported sleep duration and cognitive performance in older adults: a systematic review and meta-analysis," *Sleep medicine*, vol. 17, pp. 87–98, 2016.

[28] M. Jike, O. Itani, N. Watanabe, D. J. Buysse, and Y. Kaneita, "Long sleep duration and health outcomes: a systematic review, meta-analysis and meta-regression," *Sleep medicine reviews*, vol. 39, pp. 25–36, 2018.

[29] B.-P. Liu, X.-T. Wang, Z.-Z. Liu, Z.-Y. Wang, D. An, Y.-X. Wei, C.-X. Jia, and X. Liu, "Depressive symptoms are associated with short and long sleep duration: A longitudinal study of chinese adolescents," *Journal of affective disorders*, vol. 263, pp. 267–273, 2020.

[30] G. Desouzart, R. Matos, F. Melo, and E. Filgueiras, "Effects of sleeping position on back pain in physically active seniors: A controlled pilot study," *Work*, vol. 53, no. 2, pp. 235–240, 2016.

[31] S. Herculano-Houzel, "Scaling of brain metabolism with a fixed energy budget per neuron: implications for neuronal activity, plasticity and evolution," *PloS one*, vol. 6, no. 3, p. e17514, 2011.

[32] L. Xie, H. Kang, Q. Xu, M. J. Chen, Y. Liao, M. Thiyagarajan, J. O'Donnell, D. J. Christensen, C. Nicholson, J. J. Iliff *et al.*, "Sleep drives metabolite clearance from the adult brain," *science*, vol. 342, no. 6156, pp. 373–377, 2013.

[33] H. Lee, L. Xie, M. Yu, H. Kang, T. Feng, R. Deane, J. Logan, M. Nedergaard, and H. Benveniste, "The effect of body posture on brain glymphatic transport," *Journal of Neuroscience*, vol. 35, no. 31, pp. 11 034–11 044, 2015.

[34] S. M. Harding, "Sleep-related gastroesophageal reflux: Evidence is mounting. . . ," *Clinical Gastroenterology and Hepatology*, vol. 7, no. 9, pp. 919–920, 2009.

[35] R. Mody, S. C. Bolge, H. Kannan, and R. Fass, "Effects of gastroesophageal reflux disease on sleep and outcomes," *Clinical Gastroenterology and Hepatology*, vol. 7, no. 9, pp. 953–959, 2009.

[36] R. M. Khoury, L. Camacho-Lobato, P. O. Katz, M. A. Mohiuddin, and D. O. Castell, "Influence of spontaneous sleep positions on nighttime recumbent reflux in patients with gastroesophageal reflux disease," *The American journal of gastroenterology*, vol. 94, no. 8, pp. 2069–2073, 1999.

[37] C. Loots, M. Smits, T. Omari, R. Bennink, M. Benninga, and M. Van Wijk, "Effect of lateral positioning on gastroesophageal reflux (ger) and underlying mechanisms in ger disease (gerd) patients and healthy controls," *Neurogastroenterology & Motility*, vol. 25, no. 3, pp. 222–e162, 2013.

[38] J. A. Cheyne, "Situational factors affecting sleep paralysis and associated hallucinations: position and timing effects," *Journal of Sleep Research*, vol. 11, no. 2, pp. 169–177, 2002.

[39] M. J. Sateia, "International classification of sleep disorders," *Chest*, vol. 146, no. 5, pp. 1387–1394, 2014.

[40] L. M. DelRosso, O. Bruni, and R. Ferri, "Restless sleep disorder in children: a pilot study on a tentative new diagnostic category," *Sleep*, vol. 41, no. 8, p. zsy102, 2018.

[41] L. M. DelRosso and R. Ferri, "The prevalence of restless sleep disorder among a clinical sample of children and adolescents referred to a sleep centre," *Journal of sleep research*, vol. 28, no. 6, p. e12870, 2019.

[42] M. Hornyak, B. Feige, D. Riemann, and U. Voderholzer, "Periodic leg movements in sleep and periodic limb movement disorder: prevalence, clinical significance and treatment," *Sleep medicine reviews*, vol. 10, no. 3, pp. 169–177, 2006.

[43] J. V. Rundo and R. Downey III, "Polysomnography," *Handbook of clinical neurology*, vol. 160, pp. 381–392, 2019.

[44] W. De Backer, "Obstructive sleep apnea/hypopnea syndrome." *Panminerva medica*, vol. 55, no. 2, pp. 191–195, 2013.

[45] P. J. Strollo Jr and R. M. Rogers, "Obstructive sleep apnea," *New England Journal of Medicine*, vol. 334, no. 2, pp. 99–104, 1996.

[46] S.-J. Paine, R. B. Harris, and K. M. Mihaere, "Managing obstructive sleep apnoea and achieving equity: implications for health," *The New Zealand Medical Journal (Online)*, vol. 124, no. 1334, 2011.

[47] bpacnz, "Obstructive sleep apnoea in adults." *bpacnz*, vol. 48, pp. 191–195, november 2012. [Online]. Available: https://bpac.org.nz/bpj/2012/november/apnoea.aspx

[48] A. Authors, amp; ReviewersSleep Physician at American Sleep Association Reviewers, and W. certified sleep M.D. physicians, "Positional obstructive sleep apnea." [Online]. Available: https://www.sleepassociation.org/sleep-apnea/positional-sleep-apnea/

[49] A. Oksenberg, D. Silverberg, D. Offenbach, and E. Arons, "Positional therapy for obstructive sleep apnea patients: a 6-month follow-up study," *The Laryngoscope*, vol. 116, no. 11, pp. 1995–2000, 2006.

[50] S. A. Joosten, D. M. O'Driscoll, P. J. Berger, and G. S. Hamilton, "Supine position related obstructive sleep apnea in adults: pathogenesis and treatment," *Sleep medicine reviews*, vol. 18, no. 1, pp. 7–17, 2014.

[51] S. Sprigle and S. Sonenblum, "Assessing evidence supporting redistribution of pressure for pressure ulcer prevention: a review," *J Rehabil Res Dev*, vol. 48, no. 3, pp. 203–13, 2011.

[52] S. L. Swisher, M. C. Lin, A. Liao, E. J. Leeflang, Y. Khan, F. J. Pavinatto, K. Mann, A. Naujokas, D. Young, S. Roy *et al.*, "Impedance sensing device

enables early detection of pressure ulcers in vivo," *Nature communications*, vol. 6, p. ncomms7575, 2015.

[53] K.-H. Nguyen, W. Chaboyer, and J. A. Whitty, "Pressure injury in australian public hospitals: a cost-of-illness study," *Australian Health Review*, vol. 39, no. 3, pp. 329–336, 2015.

[54] C. K. Sen, G. M. Gordillo, S. Roy, R. Kirsner, L. Lambert, T. K. Hunt, F. Gottrup, G. C. Gurtner, and M. T. Longaker, "Human skin wounds: a major and snowballing threat to public health and the economy," *Wound repair and regeneration*, vol. 17, no. 6, pp. 763–771, 2009.

[55] P. K. Rauch and T. A. Stern, "Life-threatening injuries resulting from sleepwalking and night terrors," *Psychosomatics*, vol. 27, no. 1, pp. 62–64, 1986.

[56] A. Zadra, A. Desautels, D. Petit, and J. Montplaisir, "Somnambulism: clinical aspects and pathophysiological hypotheses," *The Lancet Neurology*, vol. 12, no. 3, pp. 285–294, 2013.

[57] R. P. Vertes and S. B. Linley, "No cognitive processing in the unconscious, anesthetic-like, state of sleep," *Journal of Comparative Neurology*, vol. 529, no. 3, pp. 524–538, 2021.

[58] M. T. Smith, C. S. McCrae, J. Cheung, J. L. Martin, C. G. Harrod, J. L. Heald, and K. A. Carden, "Use of actigraphy for the evaluation of sleep disorders and circadian rhythm sleep-wake disorders: an american academy of sleep medicine systematic review, meta-analysis, and grade assessment," *Journal of Clinical Sleep Medicine*, vol. 14, no. 7, pp. 1209–1230, 2018.

[59] C. McCall and W. V. McCall, "Comparison of actigraphy with polysomnography and sleep logs in depressed insomniacs," *Journal of sleep research*, vol. 21, no. 1, pp. 122–127, 2012.

[60] R. Dick, T. Penzel, I. Fietze, M. Partinen, H. Hein, and J. Schulz, "Aasm standards of practice compliant validation of actigraphic sleep analysis from somnowatch™ versus polysomnographic sleep diagnostics shows high conformity also among subjects with sleep disordered breathing," *Physiological measurement*, vol. 31, no. 12, p. 1623, 2010.

[61] "Sleep cycle: Sleep tracker, monitor and alarm clock," Nov 2021. [Online]. Available: https://www.sleepcycle.com/

[62] "Sleep as android," Sep 2021. [Online]. Available: https://sleep.urbandroid.org/

[63] H. A. Chapman, D. Bernier, and B. Rusak, "Mri-related anxiety levels change within and between repeated scanning sessions," *Psychiatry Research: Neuroimaging*, vol. 182, no. 2, pp. 160–164, 2010.

[64] H. Yoon, S. Hwang, D. Jung, S. Choi, K. Joo, J. Choi, Y. Lee, D.-U. Jeong, and K. Park, "Estimation of sleep posture using a patch-type accelerometer based device," in *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2015, pp. 4942–4945.

[65] E. B. Monroy, A. P. Rodríguez, M. E. Estevez, and J. M. Quero, "Fuzzy monitoring of in-bed postural changes for the prevention of pressure ulcers using inertial sensors attached to clothing," *Journal of Biomedical Informatics*, vol. 107, p. 103476, 2020.

[66] K.-M. Chang and S.-H. Liu, "Wireless portable electrocardiogram and a triaxis accelerometer implementation and application on sleep activity monitoring," *Telemedicine and e-Health*, vol. 17, no. 3, pp. 177–184, 2011.

[67] L. Chang, J. Lu, J. Wang, X. Chen, D. Fang, Z. Tang, P. Nurmi, and Z. Wang, "Sleepguard: Capturing rich sleep information using smartwatch sensing data," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 3, pp. 1–34, 2018.

[68] T.-W. Shen, F.-C. Liu, Y.-T. Tsao, and S.-C. Chang, "A body position detection method by fusing heterogeneous information from surface ecg," in *2010 Computing in Cardiology*. IEEE, 2010, pp. 517–520.

[69] S. Liu and S. Ostadabbas, "A vision-based system for in-bed posture tracking," in *Proceedings of the IEEE international conference on computer vision workshops*, 2017, pp. 1373–1382.

[70] E. P. Devine, A. Ahamed, A. S. Mayet, S. Ghandiparsi, C. Bartolo-Perez, L. McPhillips, A. F. Elrefaie, T. Yamada, S.-Y. Wang, and M. S. Islam, "Optimization of cmos image sensors with single photon-trapping hole per pixel for enhanced sensitivity in near-infrared," *arXiv preprint arXiv:2110.00206*, 2021.

[71] S. Akbarian, G. Delfi, K. Zhu, A. Yadollahi, and B. Taati, "Automated noncontact detection of head and body positions during sleep," *IEEE Access*, vol. 7, pp. 72 826–72 834, 2019.

[72] S. Akbarian, N. M. Ghahjaverestan, A. Yadollahi, B. Taati *et al.*, "Noncontact sleep monitoring with infrared video data to estimate sleep apnea severity and distinguish between positional and nonpositional sleep apnea: Model development

and experimental validation," *Journal of medical Internet research*, vol. 23, no. 11, p. e26524, 2021.

[73] A. Y.-C. Tam, B. P.-H. So, T. T.-C. Chan, A. K.-Y. Cheung, D. W.-C. Wong, and J. C.-W. Cheung, "A blanket accommodative sleep posture classification system using an infrared depth camera: A deep learning approach with synthetic augmentation of blanket conditions," *Sensors*, vol. 21, no. 16, p. 5553, 2021.

[74] T. Grimm, M. Martinez, A. Benz, and R. Stiefelhagen, "Sleep position classification from a depth camera using bed aligned maps," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 319–324.

[75] T. Zhou, Z. Xia, X. Wang, and F. Xu, "Human sleep posture recognition based on millimeter-wave radar," in *2021 Signal Processing Symposium (SPSympo)*. IEEE, 2021, pp. 316–321.

[76] S. Yue, "Enabling contactless sleep studies at home using wireless signals," Ph.D. dissertation, Massachusetts Institute of Technology, 2021.

[77] X. Liu, J. Cao, S. Tang, and J. Wen, "Wi-sleep: Contactless sleep monitoring via wifi signals," in *2014 IEEE Real-Time Systems Symposium*. IEEE, 2014, pp. 346–355.

[78] Y. Cao, F. Wang, X. Lu, N. Lin, B. Zhang, Z. Liu, and S. Sigg, "Contactless body movement recognition during sleep via wifi signals," *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 2028–2037, 2019.

[79] C.-C. Hsia, Y.-W. Hung, Y.-H. Chiu, and C.-H. Kang, "Bayesian classification for bed posture detection based on kurtosis and skewness estimation," in *HealthCom 2008-10th International Conference on e-health Networking, Applications and Services*. IEEE, 2008, pp. 165–168.

[80] P. Barsocchi, M. Bianchini, A. Crivello, D. La Rosa, F. Palumbo, and F. Scarselli, "An unobtrusive sleep monitoring system for the human sleep behaviour understanding," in *2016 7th IEEE international conference on cognitive infocommunications (CogInfoCom)*. IEEE, 2016, pp. 000 091–000 096.

[81] J. J. Liu, M.-C. Huang, W. Xu, and M. Sarrafzadeh, "Bodypart localization for pressure ulcer prevention," in *2014 36th annual international conference of the ieee engineering in medicine and biology society*. IEEE, 2014, pp. 766–769.

[82] K. Tang, A. Kumar, M. Nadeem, and I. Maaz, "Cnn-based smart sleep posture recognition system," *IoT*, vol. 2, no. 1, pp. 119–139, 2021.

[83] B. Mahesh, "Machine learning algorithms-a review," *International Journal of Science and Research (IJSR).[Internet]*, vol. 9, pp. 381–386, 2020.

[84] Q. Hu, X. Tang, and W. Tang, "A real-time patient-specific sleeping posture recognition system using pressure sensitive conductive sheet and transfer learning," *IEEE Sensors Journal*, vol. 21, no. 5, pp. 6869–6879, 2020.

[85] J. M. Gandarias, A. J. Garcia-Cerezo, and J. M. Gomez-de Gabriel, "Cnn-based methods for object recognition with high-resolution tactile sensors," *IEEE Sensors Journal*, vol. 19, no. 16, pp. 6872–6882, 2019.

[86] X. Xu, F. Lin, A. Wang, Y. Hu, M.-C. Huang, and W. Xu, "Body-earth mover's distance: A matching-based approach for sleep posture recognition," *IEEE transactions on biomedical circuits and systems*, vol. 10, no. 5, pp. 1023–1035, 2016.

[87] H. M. Clever, Z. Erickson, A. Kapusta, G. Turk, K. Liu, and C. C. Kemp, "Bodies at rest: 3d human pose and shape estimation from a pressure image using synthetic data," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 6215–6224.

[88] H. J. Lee, S. H. Hwang, S. M. Lee, Y. G. Lim, and K. S. Park, "Estimation of body postures on bed using unconstrained ecg measurements," *IEEE journal of biomedical and health informatics*, vol. 17, no. 6, pp. 985–993, 2013.

[89] D. Zhou and H. Wang, "Design and evaluation of a skin-like sensor with high stretchability for contact pressure measurement," *Sensors and Actuators A: Physical*, vol. 204, pp. 114–121, 2013.

[90] W. Yao, R. Mao, W. Gao, W. Chen, Z. Xu, and C. Gao, "Piezoresistive effect of superelastic graphene aerogel spheres," *Carbon*, vol. 158, pp. 418–425, 2020.

[91] S. Wang, K.-H. Huang, and Y.-J. Yang, "A highly sensitive capacitive pressure sensor with microdome structure for robot tactile detection," in *2019 20th International Conference on Solid-State Sensors, Actuators and Microsystems & Eurosensors XXXIII (TRANSDUCERS & EUROSENSORS XXXIII)*. IEEE, 2019, pp. 458–461.

[92] P. Rey, P. Charvet, M. Delaye, and S. Abou Hassan, "A high density capacitive pressure sensor array for fingerprint sensor application," in *Proceedings of International Solid State Sensors and Actuators Conference (Transducers' 97)*, vol. 2. IEEE, 1997, pp. 1453–1456.

[93] X. Ran, C. Wang, Y. Xiao, X. Gao, Z. Zhu, and B. Chen, "A portable sitting posture monitoring system based on a pressure sensor array and machine learning," *Sensors and Actuators A: Physical*, vol. 331, p. 112900, 2021.

[94] W. Xu, Z. Li, M.-C. Huang, N. Amini, and M. Sarrafzadeh, "ecushion: An etextile device for sitting posture monitoring," in *2011 International Conference on Body Sensor Networks*. IEEE, 2011, pp. 194–199.

[95] J. Cheng, M. Sundholm, B. Zhou, M. Hirsch, and P. Lukowicz, "Smart-surface: Large scale textile pressure sensors arrays for activity recognition," *Pervasive and Mobile Computing*, vol. 30, pp. 97–112, 2016.

[96] K. B. Balavalad, B. Sheeparamatti, and V. B. Math, "Design and simulation of mems capacitive pressure sensor array for wide range pressure measurement," *Int J Comput Appl*, vol. 163, no. 6, pp. 39–46, 2017.

[97] J. Meyer, "Textile pressure sensor: Design, error modeling and evaluation," Ph.D. dissertation, ETH Zurich, 2008.

[98] C. Sun, W. Li, and W. Chen, "A compressed sensing based method for reducing the sampling time of a high resolution pressure sensor array system," *Sensors*, vol. 17, no. 8, p. 1848, 2017.

[99] J. J. Liu, W. Xu, M.-C. Huang, N. Alshurafa, M. Sarrafzadeh, N. Raut, and B. Yadegar, "A dense pressure sensitive bedsheet design for unobtrusive sleep posture monitoring," in *2013 IEEE international conference on pervasive computing and communications (PerCom)*. IEEE, 2013, pp. 207–215.

[100] M. B. Pouyan, J. Birjandtalab, M. Heydarzadeh, M. Nourani, and S. Ostadabbas, "A pressure map dataset for posture and subject analytics," in *2017 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*. IEEE, 2017, pp. 65–68.

[101] M. Kim, H. Choi, K.-J. Cho, and S. Jo, "Single to multi: Data-driven high resolution calibration method for piezoresistive sensor array," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4970–4977, 2021.

[102] J. H. Kang, J. Y. Kim, Y. Jo, H.-S. Kim, S. M. Jung, S. Y. Lee, Y. Choi, and S. Jeong, "Three-dimensionally printed pressure sensor arrays from hysteresis-less stretchable piezoresistive composites," *RSC Advances*, vol. 9, no. 68, pp. 39 993–40 002, 2019.

[103] Q. Sun, E. Gonzalez, and Y. Sun, "On bed posture recognition with pressure sensor array system," in *2016 IEEE SENSORS*. IEEE, 2016, pp. 1–3.

[104] A. Bybi, C. Granger, S. Grondel, A.-C. Hladky-Hennion, and J. Assaad, "Electrical method for crosstalk cancellation in transducer arrays," *NDT & E International*, vol. 62, pp. 115–121, 2014.

[105] W. Lin, B. Wang, G. Peng, Y. Shan, H. Hu, and Z. Yang, "Skin-inspired piezoelectric tactile sensor array with crosstalk-free row+ column electrodes for spatiotemporally distinguishing diverse stimuli," *Advanced Science*, vol. 8, no. 3, p. 2002817, 2021.

[106] S. Bennett, Z. Ren, R. Goubran, K. Rockwood, and F. Knoefel, "In-bed mobility monitoring using pressure sensors," *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 8, pp. 2110–2120, 2015.

[107] S. S. Gilakjani, M. Bouchard, R. A. Goubran, and F. Knoefel, "Long-term sleep assessment by unobtrusive pressure sensor arrays," in *Proceedings of the 2018 3rd International Conference on Biomedical Imaging, Signal Processing*, 2018, pp. 23–35.

[108] P. Lang, "Design and prototyping of a fiber optic tactile array," in *2008 Canadian Conference on Electrical and Computer Engineering*. IEEE, 2008, pp. 000 373–000 376.

[109] M. Ramuz, B. C.-K. Tee, J. B.-H. Tok, and Z. Bao, "Transparent, optical, pressure-sensitive artificial skin for large-area stretchable electronics," *Advanced Materials*, vol. 24, no. 24, pp. 3223–3227, 2012.

[110] G. Gautschi, "Piezoelectric sensors," in *Piezoelectric Sensorics*. Springer, 2002, pp. 73–91.

[111] J. Li, R. Bao, J. Tao, Y. Peng, and C. Pan, "Recent progress in flexible pressure sensor arrays: from design to applications," *Journal of Materials Chemistry C*, vol. 6, no. 44, pp. 11 878–11 892, 2018.

[112] Y. Watanabe, S. Uemura, and S. Hoshino, "Printed pressure sensor array sheets fabricated using poly (amino acid)-based piezoelectric elements," *Japanese Journal of Applied Physics*, vol. 53, no. 5S3, p. 05HB15, 2014.

[113] G. G. Mora, J. M. Kortelainen, E. R. P. Hernández, M. Tenhunen, A. M. Bianchi, and M. O. Méndez, "Evaluation of pressure bed sensor for automatic sahs screening," *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 7, pp. 1935–1943, 2014.

[114] M. Paajanen, J. Lekkala, and K. Kirjavainen, "Electromechanical film (emfi)—a new multipurpose electret material," *Sensors and Actuators A: Physical*, vol. 84, no. 1-2, pp. 95–102, 2000.

[115] R. Joshi, B. L. Bierling, X. Long, J. Weijers, L. Feijs, C. Van Pul, and P. Andriessen, "A ballistographic approach for continuous and non-obtrusive monitoring of movement in neonates," *IEEE Journal of Translational Engineering in Health and Medicine*, vol. 6, pp. 1–10, 2018.

[116] M. Cohen-McFarlane, S. Bennett, B. Wallace, R. Goubran, and F. Knoefel, "Bed-based health monitoring using pressure sensitive technology: A review," *IEEE Instrumentation & Measurement Magazine*, vol. 24, no. 2, pp. 13–23, 2021.

[117] D. Yang, H. Guo, X. Chen, L. Wang, P. Jiang, W. Zhang, L. Zhang, and Z. L. Wang, "A flexible and wide pressure range triboelectric sensor array for real-time pressure detection and distribution mapping," *Journal of Materials Chemistry A*, vol. 8, no. 45, pp. 23 827–23 833, 2020.

[118] J. Tao, R. Bao, X. Wang, Y. Peng, J. Li, S. Fu, C. Pan, and Z. L. Wang, "Self-powered tactile sensor array systems based on the triboelectric effect," *Advanced Functional Materials*, vol. 29, no. 41, p. 1806379, 2019.

[119] H. L. Wang, S. Y. Kuang, H. Y. Li, Z. L. Wang, and G. Zhu, "Large-area integrated triboelectric sensor array for wireless static and dynamic pressure detection and mapping," *Small*, vol. 16, no. 2, p. 1906352, 2020.

[120] S.-J. Woo, J.-H. Kong, D.-G. Kim, and J.-M. Kim, "A thin all-elastomeric capacitive pressure sensor array based on micro-contact printed elastic conductors," *Journal of Materials Chemistry C*, vol. 2, no. 22, pp. 4415–4422, 2014.

[121] M. Sergio, N. Manaresi, M. Tartagni, R. Guerrieri, and R. Canegallo, "A textile based capacitive pressure sensor," in *SENSORS, 2002 IEEE*, vol. 2. IEEE, 2002, pp. 1625–1630.

[122] H. Vandeparre, D. Watson, and S. Lacour, "Extremely robust and conformable capacitive pressure sensors based on flexible polyurethane foams and stretchable metallization," *Applied Physics Letters*, vol. 103, no. 20, p. 204103, 2013.

[123] Y. Zhang, S. Liu, Y. Miao, H. Yang, X. Chen, X. Xiao, Z. Jiang, X. Chen, B. Nie, and J. Liu, "Highly stretchable and sensitive pressure sensor array based on icicle-shaped liquid metal film electrodes," *ACS applied materials & interfaces*, vol. 12, no. 25, pp. 27 961–27 970, 2020.

[124] XSENSOR, "Reveal mattress recommendation system," Accessed March 2022. [Online]. Available: https://www.xsensor.com/solutions-and-platform/sleep-improvement/reveal-recommendation-system

[125] H. Kirkland-Walsh, O. Teleten, M. Wilson, and B. Raingruber, "Pressure mapping comparison of four or surfaces," *AORN journal*, vol. 102, no. 1, pp. 61–e1, 2015.

[126] I. Main and D. Jack, "capacitive pressure sensor," Oct 2013, patent no. US20110120228A1.

[127] Y. Xia, H. Gu, L. Xu, X. D. Chen, and T. V. Kirk, "Extending porous silicone capacitive pressure sensor applications into athletic and physiological monitoring," *Sensors*, vol. 21, no. 4, p. 1119, 2021.

[128] C. Ashruf, "Thin flexible pressure sensors," *Sensor Review*, 2002.

[129] "Tactilus high performance mattress pad sensor system," [Online; accessed 26 Feb. 2022]. [Online]. Available: https://www.johnpreston.co.uk/tactilus-high-performance-mattress-pad-sensor-system

[130] Tekcan, "Body pressure measurement system (bpms)," Accessed March 2022. [Online]. Available: https://www.tekscan.com/products-solutions/systems/body-pressure-measurement-system-bpms

[131] S. Swangarom, T. Takuya, A. Takehiko, and K. Haruhiko, "Development of sleep disorder detection system for solitary person using boditrak sensor," in *2017 International Conference on Biometrics and Kansei Engineering (ICBAKE)*. IEEE, 2017, pp. 66–69.

[132] S. B. Higer, "Pressure-time profile analysis to select surfaces that effectively redistribute pediatric occipital pressure," Ph.D. dissertation, Tufts University, 2015.

[133] G. D. Carol Vermeer, *FSA 4.0 User Manual 3rd edition*, Vista Medical.

[134] X. T. C. via SlideServe, "Xsensor vs. competitors," uploaded Nov 18, 2013, accessed March 2022. [Online]. Available: https://www.comfortcompany.com/product/Comfort%20Rehab/Accessories/BodiTrak2%20LT

[135] wellSense, "VŪ is the world's first and only advanced pressure visualization system (apvs)." Accessed March 2022. [Online]. Available: https://www.wellsensevu.com/technology

[136] V. Davoodnia, S. Ghorbani, and A. Etemad, "In-bed pressure-based pose estimation using image space representation learning," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 3965–3969.

[137] TensorFlow, "Tensorflow, an end-to-end open source machine learning platform," Accessed March 2022. [Online]. Available: https://www.tensorflow.org/

[138] D. Systems, "Solidworks," Accessed March 2021. [Online]. Available: https://www.solidworks.com

[139] Adobe, "Elecric paint," March 2022. [Online]. Available: https://www.adobe.com/products/illustrator.html

[140] B. Conductive, "Bare conductive faq," Accessed March 2022. [Online]. Available: https://faqs.bareconductive.com/hc/en-gb/categories/200769382-Electric-Paint

[141] D. Materials, "Dm-cap-1060s," Accessed March 2022. [Online]. Available: https://www.dycotecmaterials.com/product/dm-cap-1060s/

[142] Myant, "Myant website," Accessed March 2019. [Online]. Available: https://myant.ca/

[143] S. E. TECHNOLOGIES, "Zhejiang saintyear electronic technologies co.ltd," Accessed October 2019. [Online]. Available: http://www.saintyoo.com/en/about.aspx

[144] H. T. G. W. Y. Want, "Connections," Accessed Jan 2019. [Online]. Available: https://www.kobakant.at/DIY/?cat=32

[145] L. Jones, S. Nabil, and A. Girouard, "Swatch-bits: Prototyping e-textiles with modular swatches," in *Proceedings of the Fourteenth International Conference on Tangible, Embedded, and Embodied Interaction*, 2020, pp. 893–897.

[146] "Clincher flex connectors," [Online; accessed Feb. 2019]. [Online]. Available: https://www.amphenol-icc.com/product-series/clincher-flex-connectors.html

[147] *Clincher Receptacle Assemly*, Amphenol ICC, 3 2012. [Online]. Available: https://cdn.amphenol-icc.com/media/wysiwyg/files/drawing/65801.pdf

[148] M. chemicals, "4860p - sn63/pb37 solder paste - no-clean," Accessed July 2021. [Online]. Available: https://www.mgchemicals.com/products/soldering-supplies/solder-paste/sn63-solder-paste/

[149] ——, "4902p - sn42/bi57/ag1 low temperature solder paste t3," Accessed July 2021. [Online]. Available: https://www.mgchemicals.com/products/soldering-supplies/solder-paste/low-temperature-solder/

[150] C. Ramesh, "New crystalline transitions in nylons 4, 6, 6, 10, and 6, 12 using high temperature x-ray diffraction studies," *Macromolecules*, vol. 32, no. 11, pp. 3721–3726, 1999.

[151] C. Ramesh and E. B. Gowd, "High-temperature x-ray diffraction studies on the crystalline transitions in the α-and γ-forms of nylon-6," *Macromolecules*, vol. 34, no. 10, pp. 3308–3313, 2001.

[152] C. Ramesh, A. Keller, and S. Eltink, "Studies on the crystallization and melting of nylon 66: 2. crystallization behaviour and spherulitic morphology by optical microscopy," *Polymer*, vol. 35, no. 24, pp. 5293–5299, 1994.

[153] miniware, "Mhp30 mini hot plate preheater," Accessed Feb 2022. [Online]. Available: http://www.miniware.com.cn/product/mhp30-mini-hot-plate-preheater/

[154] R. Mena, "System advantages of mixed signal integration," in *Texas Instruments Incorporated*, 2014.

[155] "Sciosense website," Feb 2021, [Online; accessed 2. Feb. 2021]. [Online]. Available: https://www.sciosense.com/

[156] *CapTouch Programmable Controller for Single-Electrode Capacitance Sensors*, Analog Devices, rev. E. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/AD7147.pdf

[157] Digikey, "Ad7147wpacpz-500r7," Accessed March 2022. [Online]. Available: https://www.digikey.com/en/products/detail/analog-devices-inc/AD7147WPACPZ-500R7/2700116

[158] *PCap04: Capacitance-to-Digital Converter*, AMS/ScioSense, 4 2018, v1-03. [Online]. Available: https://www.sciosense.com/wp-content/uploads/documents/PCap04-Datasheet.pdf

[159] Mouser, "Pcap04-aqfm-24," Accessed March 2022. [Online]. Available: https://www.mouser.com/c/?marcom=19678184

[160] N. Semiconductor, "Nrf52840 - advanced bluetooth 5, thread and zigbee multiprotocol soc," Accessed March 2021. [Online]. Available: https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52840

[161] Altium, "Altim designer software," Accessed March 2021. [Online]. Available: https://www.altium.com/altium-designer/

[162] JLBPCB, "Jlbpcb fabrication services," Accessed September 2019. [Online]. Available: https://jlcpcb.com/

[163] Segger, "Embbed studio - the all-in-one embedded development solution," Accessed March 2022. [Online]. Available: https://www.segger.com/products/development-tools/embedded-studio/

[164] N. Semiconductor, "Programmable peripheral interconnect," Accessed March 2022. [Online]. Available: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.nrf52832.ps.v1.1%2Fppi.html

[165] Microsoft, "What is windows presentation foundation (wpf)?" Accessed March 2022. [Online]. Available: https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf?view=vs-2022

[166] Follow, p. r. N. 14, P. Rastogi, I. L. N. 20, and I. Landwerth, "The roadmap for wpf," Apr 2019. [Online]. Available: https://devblogs.microsoft.com/dotnet/the-roadmap-for-wpf/

[167] Keras, "Keras - simple. flexible. powerful," Accessed March 2022. [Online]. Available: https://keras.io/

[168] Digikey, "Digikey home page," Accessed March 2022. [Online]. Available: https://www.digikey.com/

[169] X. T. C. Limited, "Real silver coat knit jersey conductive fabric mesh mattress," Accessed October 2021. [Online]. Available: https://cnxingtech.en.alibaba.com

[170] R. Wu, L. Ma, A. Balkrishna Patil, C. Hou, Z. Meng, Y. Zhang, X. Liu, and W. Yu, "A facile method to prepare a wearable pressure sensor based on fabric electrodes for human motion monitoring," *Textile Research Journal*, vol. 89, no. 23-24, pp. 5144–5152, 2019.

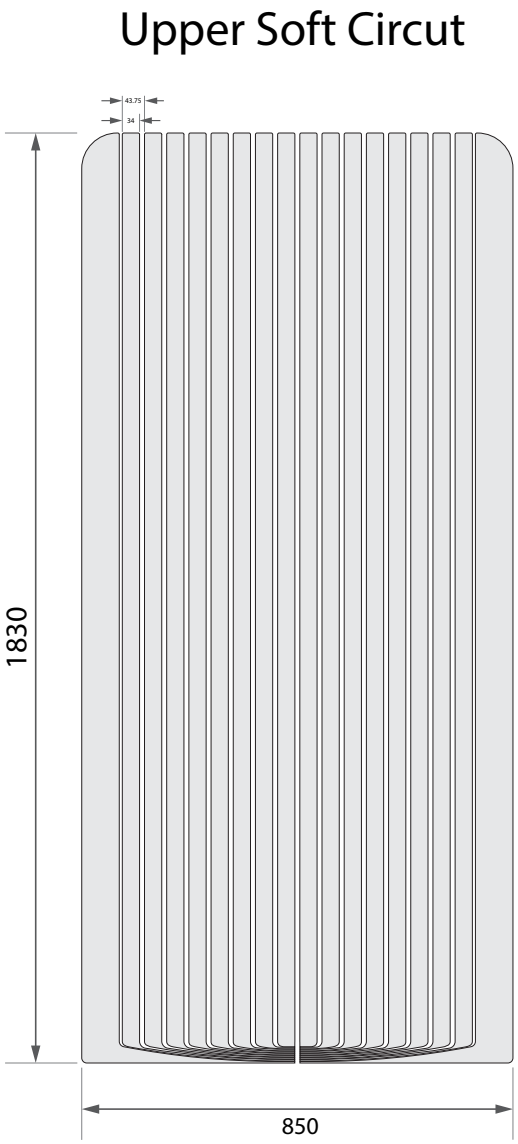
[171] C. Usma, A. Kouzani, J. Chua, A. Arogbonlo, S. Adams, and I. Gibson, "Fabrication of force sensor circuits on wearable conductive textiles," *Procedia Technology*, vol. 20, pp. 263–269, 2015.

[172] N. M. Punjabi, A. B. Newman, T. B. Young, H. E. Resnick, and M. H. Sanders, "Sleep-disordered breathing and cardiovascular disease: an outcome-based definition of hypopneas," *American journal of respiratory and critical care medicine*, vol. 177, no. 10, pp. 1150–1155, 2008.

[173] P. L. Nunez and R. Srinivasan, "Electroencephalogram," *Scholarpedia*, vol. 2, no. 2, p. 1348, 2007.

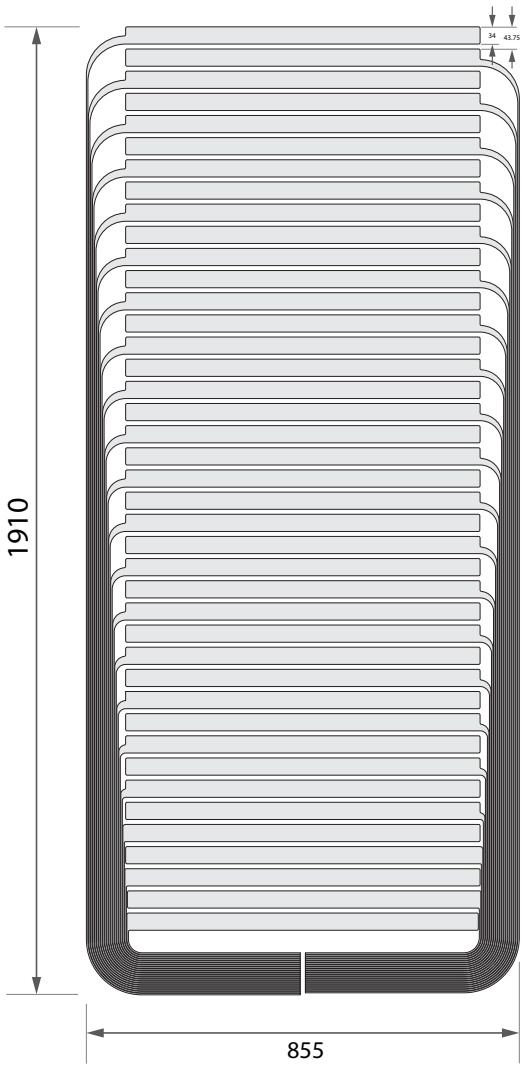
Appendix A

Soft Circuits & Dimensions



Dimentions in mm

Lower Soft Circut



Dimentions in mm

Appendix B

Materials

浙江三元电子科技有限公司

ZHEJIANG SAINTYEAR ELECTRONIC TECHNOLOGIES Co.,LTD.

Product Data Sheet

1、Product Name: Plain weave conductive fabric

2、Part Number: SY-PF37B

3、Typical Property Data

Item	Unit	Spec.	Test standard
Weight	g/m^2	95±10	GB/T 4669-1995
Thickness	mm	0.11±0.02	FZ/T01003-1991
Width	mm	1400±5	GB/T4667-1995
Length	m	N/A	GB/T 4666-1995
Fabric Density	T	290±10	ASTMD 3775
Shielding Effectiveness	dB	60(min)in 10MHz~3GHz	SJ20524-1995
Salt Spray（5% 48H 35℃）	Ω/sq	<0.5	ASTM B117-03
Surface Resistivity	Ω/sq	≤0.05	ASTM F390
Metal Adhesion	grad	Four above	/

4、Store transportation

4.1 Store condition：Stored in a cool dry place in the condition of (-20℃--40℃)、(≤65%RH) below, avoid direct sunshine. Using within two year can get the best effect.

4.2 Protect from moisture and crash, don't be packed under heavy cargo during transportation.

5、Suggestions and statement

We suggest user do adaptability test before the formal use of this product. Due to the diversity of practical application, our company do not guarantee that the problem appear in a specific condition, thus our company will not be responsible for any direct, indirect or accidental damage. You can contact my company's after-sales service when problems encountered in the use, we will try to help you.

2017-04-17

4860P

Sn63/Pb37 Solder Paste—No-Clean

Sn63/Pb37 solder paste, also known as Sn63 solder paste or 63 37 solder paste, is a no-clean solder paste that is made from a blend of high purity, non-recycled tin and lead alloy powder combined with a no-clean flux to form a paste. It is designed for surface mount applications and provides high tack force and good wettability. The post-soldering residues are transparent, non-conductive, non-corrosive, and highly insulated. ("No-clean" means that residues are not harmful to assemblies.)

It is designed for use in high-speed printing and is an ideal choice for SMT solder paste printers. It can yield brick-like prints even when using ultra-fine pitch stencils as small as 0.3 mm.

Features & Benefits

- Alloy exceeds J-STD-006C and meets ASTM B 32 purity requirements
- Flux meets J-STD-004B
- Non-corrosive
- Non-conductive residue
- Halide-free
- Good wettability
- Type 3 (45–25 µm)

Available Packaging

Cat. No.	Packaging	Net Vol.	Net Wt.
4860P-35G	Syringe	4.21 mL	35 g

Contact Information

MG Chemicals, 1210 Corporate Drive
Burlington, Ontario, Canada L7L 5R6

Email: support@mgchemicals.com

Phone: North America: +(1)800-340-0772
International: +(1) 905-331-1396
Europe: +(44)1663 362888

Properties

Flux Classification	REL0	
Flux Type	Resin	
Flux Activity	Low	
Copper Mirror	No removal	
Corrosion Test	Pass	
Electromigration	Pass	
Solder Ball Test	Pass	
Slump Test		
@ 25 ˚C, 0.63 vert./horiz.	No bridges	
@ 150 ˚C, 0.63 vert./horiz.	No bridges	
@ 25 ˚C, 0.33 vert./horiz.	0.20/0.20	
@ 150 ˚C, 0.33 vert./horiz.	0.20/0.20	
Viscosity, poise	850–1100	
Acid Number (mgKOH/g sample)	100	
Halides (by weight)	<0.05	%
Post Reflow Flux Residue	45	%
Metal Loading	88	%
Surface Insulation Resistance (SIR)	2.4×10^{10}	Ω
Bellcore (Telecordia)	4.1×10^{10}	Ω
Tack		
Initial	85	g
Retention @ 24 h	90	g
Retention @ 72 h	92	g

4860P

Application Instructions

Read the product SDS before using this product (downloadable at www.mgchemicals.com).

1. Take solder paste out of refrigerator and allow it to reach room temperature prior use.
2. Remove the cap from the syringe. Do not discard cap.
3. Insert plunger to the back of the syringe. For better control, insert needle to the tip.
4. Dispense paste onto the desired area and place component on top.
5. Apply heat using a heat gun.
6. Clean tip to prevent contamination and material buildup.
7. Replace the cap on the syringe.
8. (Optional) Clean residue with MG #8241-T or #8241-W Isopropyl Alcohol Wipes.

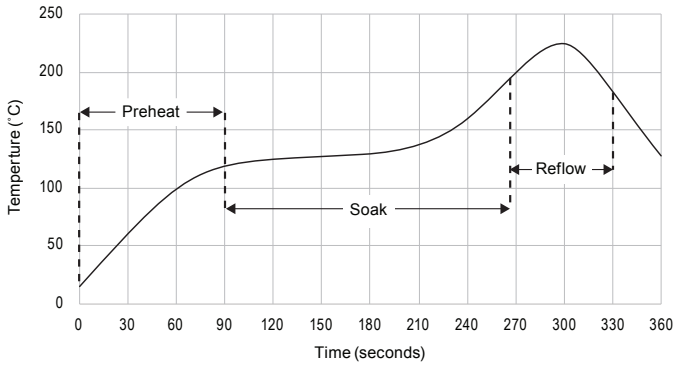
Reflow

Best results are achieved when the paste is reflowed in a forced air convection oven with a minimum of 8 zones. The following is a recommended profile for a forced air convection reflow process.

Preheat Zone—It is the ramp zone, which elevates the temperature of the PCB to the desired soak temperature. The rate of temperature rise should not exceed 2.5 °C/s to avoid thermal shock stress.

Soak Zone—It exposes the PCB to a stable temperature that allows the components to reach a uniform temperature. It allows the flux to concentrate and the volatiles to escape from the paste.

Reflow Zone—It is the spike zone, which elevates the temperature of the PCB assembly from the activation temperature to the recommended peak temperature.



Storage and Handling

Store refrigerated between 2–10 °C in an upright position with tip down to prevent flux separation and air entrapment.

Unopened Container

Shelf Life @ 2–10 ˚C	2 years
Shelf Life @ 20–25 ˚C	1 year

Disclaimer

This information is believed to be accurate. It is intended for professional end-users who have the skills required to evaluate and use the data properly. M.G. Chemicals Ltd. does not guarantee the accuracy of the data and assumes no liability in connection with damages incurred while using it.

FOAM PROPERTY KEY

Property	Definition	Sources of possible error & effects	AS2281 Specifications
Density	This refers to the weight of foam per cubic metre.This is the most important measure when looking at foam quality. The denser the foam, the more foam per cubic metre for comfort and support. The first number in the foam grade represents the density, eg. EN36-130 foam density is 36kg/m3 which is a high-density foam.	• Sample size – small sample large error • Balance (± 0.05 kg) • Operator • Accuracy ± 0.5 –1.0 kg/m^3	Various – eg. for Hardness 130 N, min. D= 23 kg/m^3
Hardness	This is a measure of how the foam feels, not a measure of quality. Lower quality foams can lose up to 40% of their hardness in the first few months. E.g. Higher quality foam grades such as EN36-100, have a nominal hardness of 100 Newtons, when tested to Australian Standard AS2282.	• Sample size (surface area & thickness) • Mechanical (height, force & time) • Operator (preflex) • Environmental (high temp & humidity – softer readings) • Accuracy ± 5 %	Various - eg. for 100mm Mattress, min Foam Hardness =130 N
Support / Indentation / Sag Factor	This is the ratio of forces required to compress the foam to **65%** and **25%** of its original height. Quality foam for seating applications will have an indentation or sag factor of at least 2.0 for support. A foam with 2.0 or higher Sag factor makes hardness variation less noticeable. However the Sag factor needs to be balanced with Comfort factor (see below).	• Same errors as for hardness • A 100mm thick sample would give higher factor than a 50mm sample • Varies from 1.8-2.5 for conventional and 2.3-3.5 for HR • An excessively high Sag factor would result in a high comfort factor, high pressure and discomfort • An excessively low Sag factor may result in too much sinking or 'bottoming out'	Min 1.75 for Hardness <160 N Min 1.80 for Hardness >190N
Comfort Factor	This lesser known factor is the ratio of forces required to compress the foam to **25%** and **5%** of its original height.This is an indication of initial 'feel' of foam. Quality foam for seating applications will have an indentation or comfort factor of about 1.1 (conventional) or 1.5 (HR)	• Same errors as for hardness • Varies from 1-1.3 for conventional and 1.3-1.9 for HR • An excessively high Comfort factor may result in high pressure and discomfort • An excessively low Comfort factor may result in too much sinking or 'bottoming out'	N/a

Property	Definition	Sources of possible error & effects	AS2281 Specifications for Types N & H
Resilience	This gives a measure of the bounce or spring of the foam grade.The higher the resilience percentage, the better the foam recovery. Resilience of 40% or higher indicates good recovery. Conventional foams have a resilience of 35-55% and HR foams 45 % or higher. Viscoelastic foams have a slow recovery and a max resilience of 15%	• Sample thickness • Mechanical (ball size & weight) • Operator eye • Accuracy ± 5 %	Min. 40% for Hardness <160N
Porosity	The rate of air-flow through the foam is a measure of the ratio of closed to open cells or 'windows' in the foam structure. HR foams generally have low porosity and feel harder when produced. They need to be crushed before use. This property varries greatly depending on foaming process (chemicals & machine) and other variables.	• Sample size • Mechanical – Flow-meter, pump • Foam cell structure can influence air-flow (coarse cells -high porosity, fine cells -low porosity). • In conventional / HR foams, porosity vary between 30-120 L/min • Vicoelastic foams can have porosity as low as 5 L/min	N/a
Compression Set	Compression Set is measured by compressing a 25mm thick samples by 75%, for 22 hours at 70 °C.Then the percentage thickness loss is measured 30 minutes after releasing the samples. AS2281 the max thickness loss for foams used in bedding & furniture.	• Thickness measurements (operator) • Oven temp & uniformity • Timing of compression & recovery • Accuracy ± 1 %	Max. 15 % for Hardness < 80N Max. 10% for Hardness >100 N
Tensile Strength	This is a measure of the force required to strech a standard specimen until it breaks.	• Thickness measurements (operator) • Mechanical (force, speed,,) • Specimen with voids or nicks • Accuracy ± 5 KPa	Min. 50 for Hardness <80N Min. 70 for Hard. 100-130 N Min. 85 for Hard.>130 N
Elongation	This is measured alongside the Tensile Srength and is the max. percentage extension of the standard specimen until just before it breaks.	• Same as Tensile Strength • Accuracy ± 20 %	Min. 175 for Hardness <190 N Min. 125 for Hard. 240-300N
Tear Resistance	This is a measure of the resistance of a standard specimen to tear propagation .	• Same as Tensile Strength • Accuracy ± 10 N/m	Min. 250 N/m

Property	Definition	Sources of possible error & effects	AS2281 Specifications for Types N & H
Roller Shear Fatigue	The height and hardness loss of foam is determined after 20,000 cycles with a roller set for 65% compression (can also be set for fixed load).	• Same as Hardness • Accuracy ± 1 % height, ± 5% IF40	Max Height loss 5% Max. IF40 loss 35% for Hard.<130 N
Humid Ageing	The percentage loss in CFD hardness after ageing in an autoclave under pressure, heat & steam (5h @ 120 °C).	• Same as Hardness • Accuracy ± 5% CFD	Max. 30%
Dry Heat Ageing	The percentage loss in ILD, Tensile and Elongation after ageing in an oven set at 140 °C for 22 hours.	• Same as Hardness & Tensile/ Elongation • Accuracy ± 5%	Max. 30%
Ash content	The percentage weight of inorganic residue left after burning off foam. This is indicative of the percentage of filler added.	• Balance • Operator (ash is easily displaced) • Accuracy ± 0.5 %	Max. 2 %
Cell count	The number of cells across a 1 '' line on a sample of foam. This indicates how coarse or fine the foam cell structure	• Operator • Accuracy ± 10	N/a

PROPERTY RELATIONSHIPS

	Increase Density	Increase Hardness	Decrease Porosity	Increase Filler (Ash)	Increase Cell count (Finer cells)
Density	-	-	-	Increase	-
Hardness	-	-	Increase	Increase	
Sag / Indentation Factor	Increase	Increase	Decrease	Increase	Increase
Resilience	Increase	Decrease	Decrease	-	-
Compression Set	Decrease	Increase	Increase	Increase	-
Tensile Strength	Increase	Decrease	-	Decrease	-
Elongation	Increase	Decrease	-	Decrease	-
Tear Resistance	Increase	-	-	Decrease	-
Roller Shear Fatigue (ASTM – Fixed load)	Decrease	Decrease	Increase	Increase	Decrease
Roller Shear Fatigue (AS 2282 –fixed height)	Decrease	Decrease	Increase	Increase	Decrease
Humid Ageing	Decrease	Decrease	-	Increase	-
Dry Heat Ageing	Decrease	Decrease	-	Increase	-

NOTE : The above tables are based on Dunlop Foams Lab results and observations in selected grades

DUNLOP FOAMS
COMFORT THAT LASTS

Foam Terminology in Specifications

The terms listed below are the most common used when testing foam to identify quality and durability. All flexible polyurethane foams manufactured by Dunlop are identified with industry coding which have a series of letters and numbers.

An example of this coding and its meaning is:

EN36-100

Foam Type

Dunlop Foams manufacture many different grades of foam for a multitude of furniture, bedding and industrial uses. The foams referred to on this sheet are those used in furniture and bedding, but the same terminology is used when describing most foam types. Foam types commonly used in furniture and bedding are:

AA – Utility grades of foam that do not fall into any of the brands/types listed below.
HS – Hypersoft foams that are generally suited for use as overlays, soft backs, quilting etc.
MA – Marathon ™ part of a branded range of foams with densities exceeding 23kg/m3
ST – Stamina ™ part of a "premium" branded range with a 10 year Guarantee.
EN – Enduro ™ are our top of the range branded foams which carry a Lifetime Guarantee in domestic applications.

Density

This is a measure of mass/volume and is expressed in kilograms per cubic metre. Density is the best indicator to the foams durability and is NOT related to hardness. Foams with higher density will not suffer the same loss of hardness or height/thickness loss as can be experienced with lower density foams. Higher density foams will therefore be more expensive due to the higher levels of raw materials required in their manufacture.

Hardness

In most instances foam is selected based on its hardness, which is closely related to perceived consumer preference. All foams manufactured by Dunlop are tested for hardness (load bearing) and must be within a set range of tolerance. To find the hardness we compress a standard sized piece of foam to 40% and measure the amount of force required to obtain that compression which is expressed in newtons. As mentioned above foams with lower density can lose a large percentage of the original hardness where hardness loss in higher density foams would be very low.

Indentation Factor

Also sometimes called the "Sac Factor" or "Comfort Factor", it is the ratio of a compression reading at 65% / 25%. A high ratio indicates a wider difference between the two readings and this would indicate initial softness and deep down support/firmness.

Resilience

This is a measure of the foams "bounce" and it is widely agreed that foams with higher resilience are the best suited for seating. The resilience of foam is found by dropping a steel ball onto a standard sized test piece and measuring the distance that the ball bounces back. The height the ball was dropped from divided by the bounce height equals the foams resilience percentage. An example of superior resilience foam is Dunlop Enduro ™.

Specifications are offered as a guide to suitability, and the best grade for any given application is dependant on other factors such as the suspension system being used etc.
Our Sales Representatives will be delighted to assist you in making a fully informed choice.

Appendix C

PSA Firmware

Generated by Doxygen 1.9.2
The latest Firmware for the king single PSAM is built off of the queen firmware.
This code documentation is not a comprehensive list of all the code use in the PSA's firmware. However, it does contain many functions with code snippets for each.

1 Pressure Sensor Array documentation

1.1 Structure

The code has driver modules for each of the extenal IC's

- PSA_Q_1_MCP32S17.h - 16 channel GPIO via SPI
- PSA_Q_1_Line_selector - module to select the right MCP and c
- PSA_Q_1_Pins.h - All external GPIO pins

It has modules to setup and control periferals inside the NRF52840

- PSA_Q_1_USB - USB functionality
- PSA_Q_1_PPI - Programable peripheral interconnect
- PSA_Q_1_Global_Clock - makes use of timer 2

Lastly, it has modules for assembling and sending a pressure image

- main
- PSA_Q_1_Pressure_Array

1.2 Do List

1.2.1 Bluetooth Communication

Data is currently only being transmitted via a wired usb connection. A bluetooth LE connection is yet to be implimented as the primary communication connection

1.2.2 Code formatting

From conversation with Watthew Vander Werff the "barr c coding standard 2018" been reccomended as the code standard to abide by. Intial readings of the document are under way and will be used from hence forth

2 Todo List

Class error_usb_details_t
This need to be implimented properly

3 Module Index

3.1 Modules

Here is a list of all modules:

4 Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

5 Module Documentation

5.1 Cap Sense Driver

Capacitive to Digital Sensor (Pcap04) Driver.

Classes

- struct spi_context_t

  Structure that is passed by pointer to the SPI event callback function. It is how data is transfers from the call back function.

Functions

- void cs_init ()

  Initalisation of the cap sensing chip (pcap04).
- void spi_event_handler (nrfx_spim_evt_t const ∗p_event, void ∗p_context)

  Spi event handler - dealing with events and where they origionate from.
  It's main purpose is to place raw CDC data into the appropriate memory location of the pressure image.
- void cs_spi_setup ()

  Initalise and setup SPI 3 for communication with PCAP04.
- void cs_request_data ()

  Setup spi to have repeated read requests but hold until PPI event for spi read occurs.
- void cs_request_conversion ()

  A request to the PCAP04 cap sensing chip to start a capacitance to digital conversion. A successful read will have the PCAP04 return a signal on PG5 Pin.
- void cs_print_reading ()

  A request to the PCAP04 cap sensing chip to start a capacitance to digital conversion. A high signal is sent over PIN_PCAP04_PG3 to start the conversion.
- uint32_t cs_read ()

  General purpose read of the PCAP04 IC.

5.1.1 Detailed Description

Capacitive to Digital Sensor (Pcap04) Driver.

5.1.2 Function Documentation

5.1.2.1 cs_init() void cs_init ( )

Initalisation of the cap sensing chip (pcap04).

The SPI Pins are set in PSA_Q_1_pins.h

```
129 {
130     spi_xfered_f = false;
131     APP_ERROR_CHECK(nrfx_spim_xfer(&spi, &td_PCAP04_POR, 0));   // power on reset
132     spi_xfer_wait();
133 
134     spi_xfered_f = false;
135     APP_ERROR_CHECK(nrfx_spim_xfer(&spi, &td_PCAP04_SPI_test, 0));
136     spi_xfer_wait();
137     if (td_PCAP04_SPI_test.p_rx_buffer[1] == SPI_TEST_SUCCESS)  // check to see if the SPI test is a
    success
138     {
139         NRF_LOG_RAW_INFO("PCAP04 Test : Success\n\r");
140     }
141     else
142     {
143         NRF_LOG_RAW_INFO("PCAP04 Test : Fail\n\r");
144     }
145 
146     spi_xfered_f = false;
147     APP_ERROR_CHECK(nrfx_spim_xfer(&spi, &td_PCAP04_write_nvmem, 0)); //write non volitlie memory
148     spi_xfer_wait();
149 
150     spi_xfered_f = false;
151     APP_ERROR_CHECK(nrfx_spim_xfer(&spi, &td_PCAP04_read_nvmem, 0)); //read non volitle memory. Was used
    for debugging the nvmem
152     spi_xfer_wait();
153 
154     spi_xfered_f = false;
155     APP_ERROR_CHECK(nrfx_spim_xfer(&spi, &td_PCAP04_start_conversion, 0));  //begin an inital conversion

156     spi_xfer_wait();
157 
158     spi_xfered_f = false;
159 }
```

5.1.2.2 cs_print_reading() void cs_print_reading ( )

A request to the PCAP04 cap sensing chip to start a capacitance to digital conversion. A high signal is sent over PIN_PCAP04_PG3 to start the conversion.

```
188 {
189     NRF_LOG_RAW_INFO("CDC Reading: %u\n\r", m_cap_value);
190 }
```

5.1.2.3 cs_read() uint32_t cs_read ( )

General purpose read of the PCAP04 IC.

Returns

(uint32_t) Returns the read value.

```
193 {
194     spi_xfered_f = false;
195     APP_ERROR_CHECK(nrfx_spim_xfer(&spi, &td_PCAP04_CDC_Data, 0)); // general purpose instant read
196     return m_cap_value;
197 }
```

5.1.2.4 cs_request_conversion() void cs_request_conversion ( )

A request to the PCAP04 cap sensing chip to start a capacitance to digital conversion. A successful read will have the PCAP04 return a signal on PG5 Pin.

```
183 {
184     nrfx_gpiote_out_task_force(PIN_PCAP04_PG3, NRF_GPIOTE_INITIAL_VALUE_HIGH);
185 }
```

5.1.2.5 cs_request_data() void cs_request_data ( )

Setup spi to have repeated read requests but hold until PPI event for spi read occurs.

```
178 {
179     spi_xfered_f = false;
180     APP_ERROR_CHECK(nrfx_spim_xfer(&spi, &td_PCAP04_CDC_Data, NRFX_SPIM_FLAG_HOLD_XFER |
        NRFX_SPIM_FLAG_REPEATED_XFER));
181 }
```

5.1.2.6 cs_spi_setup() void cs_spi_setup ( )

Initalise and setup SPI 3 for communication with PCAP04.

```
162 {
163     m_spi_config.frequency = NRF_SPIM_FREQ_1M;
164     m_spi_config.ss_pin = PIN_PCAP04_CS;
165     m_spi_config.miso_pin = PIN_PCAP04_MISO;
166     m_spi_config.mosi_pin = PIN_PCAP04_MOSI;
167     m_spi_config.sck_pin = PIN_PCAP04_SCK;
168     m_spi_config.use_hw_ss = true;
169     m_spi_config.ss_active_high = false;        // we want it to be low active
170     m_spi_config.orc = 0xFF;                    // overflow recieve character - send on the tx line when
        still wanting to recieve data
171     m_spi_config.mode = NRF_SPIM_MODE_1;
172     m_spi_config.ss_duration = 16;                                                    // wait 16 -
        16Mhz cycles after SS is selected (needed for MCP or PCAP..)
173     APP_ERROR_CHECK(nrfx_spim_init(&spi, &m_spi_config, spi_event_handler, &spi_context)); // initalise
        SPI
174     spi_Cap_sense_task_addr = nrfx_spim_start_task_get(&spi);                        // get the
        spim start task address for PPi
175 }
```

5.1.2.7 spi_event_handler() void spi_event_handler (nrfx_spim_evt_t const * *p_event*, void * *p_context*)

Spi event handler - dealing with events and where they origionate from.
It's main purpose is to place raw CDC data into the appropriate memory location of the pressure image.

Parameters

p_event	- Constant pointer to what transfer event set off this handler
P_contex	- Pointer that was given at the spi event setup (that we passed).

5.2 Error

Error reporting library.

Classes

- struct error_usb_details_t

  USB errors.

Enumerations

- enum error {
  success = 0 , **error_failor** , **error_failor_null** , **error_failor_mem_alocation** ,
  error_failor_mem_aloc_pressure_img , **error_failor_mem_aloc_pressure_array_capture_pattern** ,
  error_spi_3_xfer_wait_timeout , **error_out_of_range** }

  Enum for error reporting.

5.2.1 Detailed Description

Error reporting library.

5.2.2 Enumeration Type Documentation

5.2.2.1 error enum error

Enum for error reporting.

```
42     {
43         success = 0,
44         error_failor,
45         error_failor_null,
46         error_failor_mem_alocation,
47         error_failor_mem_aloc_pressure_img,
48         error_failor_mem_aloc_pressure_array_capture_pattern,
49         error_spi_3_xfer_wait_timeout,
50         error_out_of_range
51     } error;
```

5.3 Global Clock

The global clock used to measure accurate delta time. It works by recording the global clock at one time, then again at another time, and then subtracting the second from the first to get the delta time. The fuctions in this file are seperated in to two groups.

Functions

- void global_clock_setup ()

  Set the up global clock object.
- void global_clock_time_start ()

  Capture the current clock time to be the start time. The time is stored within the object/file.
- void global_clock_time_stop ()

  Capture the current clock time to be the finishing time. The time is stored within the object/file.
- uint64_t global_clock_time_capture ()

  Get a snapshot of the global time in (us) time = 0 is at startup.
- bool global_clock_time_passed_check (uint32_t delta_time)

  Check to see if the amount of time (microseconds) has pass.
- void global_clock_time_stop_and_print ()

  A clock capture is performed and a difference from the start time is then printed.
- uint64_t global_clock_time_dif_ext (uint64_t start_time, uint64_t end_time)

  Calculates the difference between two captures - a starting and fininshing.
- bool global_clock_time_passed_check_ext (uint64_t delta_time, uint64_t time_caputure)

  Check to see if the amount of time (microseconds) in the "time" variable has pass from the "time_capture" variable.
- void global_clock_print_delta_time_ext (uint64_t start_time, uint64_t end_time)

  print the time difference of two global clock captures.
- void global_clock_test ()

  A quick one second global clock test and print.

5.3.1 Detailed Description

The global clock used to measure accurate delta time. It works by recording the global clock at one time, then again at another time, and then subtracting the second from the first to get the delta time. The fuctions in this file are seperated in to two groups.

- External variables (from this file) keep track of time with function calls to process those variables. Such functions have the "ext" prefix
- Functions alter internal variables to track time. The Global clock uses timer instance 2.

5.3.2 Function Documentation

5.3.2.1 global_clock_print_delta_time_ext()

```
void global_clock_print_delta_time_ext (
        uint64_t start_time,
        uint64_t end_time )
```

print the time difference of two global clock captures.

Parameters

start_time	- The first clock capture.
end_time	- The second clock capture.

```
80                                                                                                                  {
81      uint64_t time = _global_end_time - _global_start_time;
82      NRF_LOG_RAW_INFO("Time difference = us|%lu  ms|"NRF_LOG_FLOAT_MARKER" \n\r",time,
        NRF_LOG_FLOAT(((double)time) / 1000));
83  }
```

5.3.2.2 global_clock_setup() void global_clock_setup ( )

Set the up global clock object.

```
29                                  {                                   // Setup Timer
30      nrfx_timer_init(&timer_clock, &timer_clock_cfg, GlobalClockIntHandler);
31      nrfx_timer_extended_compare(&timer_clock, NRF_TIMER_CC_CHANNEL0, 0xFFFFFFFF,
        NRF_TIMER_SHORT_COMPARE0_CLEAR_MASK, true);            //to set off the inturrupt
32      nrfx_timer_extended_compare(&timer_clock, NRF_TIMER_CC_CHANNEL1, 0xFFFFFFFF,
        NRF_TIMER_SHORT_COMPARE1_CLEAR_MASK, false);           //to read the value and not diturb the
        operation(just works this way)
33      nrfx_timer_enable(&timer_clock);
34  }
```

5.3.2.3 global_clock_test() void global_clock_test ( )

A quick one second global clock test and print.

```
85                          {
86      global_clock_time_start();
87      nrf_delay_us(1000);
88      global_clock_time_stop_and_print();
89  }
```

5.3.2.4 global_clock_time_capture() uint64_t global_clock_time_capture ( )

Get a snapshot of the global time in (us) time = 0 is at startup.

Returns

(uint64_t) return global time.

```
47  {
48      return global_clock_time | nrfx_timer_capture(&timer_clock, NRF_TIMER_CC_CHANNEL1);
49  }
```

5.3.2.5 global_clock_time_dif_ext() uint64_t global_clock_time_dif_ext (uint64_t *start_time*, uint64_t *end_time*)

Calculates the difference between two captures - a starting and fininshing.

Parameters

in	*start_time*	The first capture in the comparason.
in	*end_time*	The second capture in the comparason.

Returns

(uint64_t) The difference in time.

5.3.2.6 global_clock_time_passed_check() `bool global_clock_time_passed_check ( uint32_t delta_time )`

Check to see if the amount of time (microseconds) has pass.

Warning

global_clock_time_start() has had been called first at some point. If it hasn't then the comparasons will be with 0 and will most likely return true.

Parameters

time	- Amount of time to check has passed.

Returns

true - The "time" amount has passed.

false - The "time" amount hasn't passed.

```
52 {
53     if(((global_clock_time | nrfx_timer_capture(&timer_clock, NRF_TIMER_CC_CHANNEL1)) -
       global_time_capture_1) >= delta_time)
54     {
55         return true;
56     }
57     else
58     {
59         return false;
60     }
61 }
```

5.3.2.7 global_clock_time_passed_check_ext() `bool global_clock_time_passed_check_ext ( uint64_t delta_time, uint64_t time_caputure )`

Check to see if the amount of time (microseconds) in the "time" variable has pass from the "time_capture" variable.

Parameters

time	- Amount of time to check has passed.
time_caputure	- the reference or starting time.

Returns

true - The "time" amount has passed.

false - The "time" amount hasn't passed.

```
70                                                                                                  {
71      if((global_clock_time | nrfx_timer_capture(&timer_clock, NRF_TIMER_CC_CHANNEL1)) - time_caputure >=
        delta_time) return true;
72      return false;
73  }
```

5.3.2.8 global_clock_time_start() void global_clock_time_start ( )

Capture the current clock time to be the start time. The time is stored within the object/file.

```
37  {
38      global_time_capture_1 = global_clock_time | nrfx_timer_capture(&timer_clock, NRF_TIMER_CC_CHANNEL1);
39  }
```

5.3.2.9 global_clock_time_stop() void global_clock_time_stop ( )

Capture the current clock time to be the finishing time. The time is stored within the object/file.

```
42  {
43      global_time_capture_2 = global_clock_time | nrfx_timer_capture(&timer_clock, NRF_TIMER_CC_CHANNEL1);
44  }
```

5.3.2.10 global_clock_time_stop_and_print() void global_clock_time_stop_and_print ( )

A clock capture is performed and a difference from the start time is then printed.

```
64  {
65      global_time_capture_2 = global_clock_time | nrfx_timer_capture(&timer_clock, NRF_TIMER_CC_CHANNEL1);
        // Capture time 2
66      uint64_t time = global_time_capture_2 - global_time_capture_1 ;              // calculate difference
        from time 1 to time two.
67      NRF_LOG_RAW_INFO("Time difference = us|%lu  ms|"NRF_LOG_FLOAT_MARKER" \n\r",time,
        NRF_LOG_FLOAT(((double)time) / 1000)); // print time
68  }
```

5.4 Line Selection

Line selection (row or column) for Capacitive Measurement.

Macros

- #define **MCP_NUM_OF_CHANNELS** 16
- #define **MCP_NUM_OF_IC_ROWS** ((NUM_OF_ROWS/MCP_NUM_OF_CHANNELS)+(NUM_OF_ROWS%MCP↩ _NUM_OF_CHANNELS?1:0))
- #define **MCP_NUM_OF_IC_COLUMNS** ((NUM_OF_COLUMNS/1)+(NUM_OF_COLUMNS%MCP_NUM↩ _OF_CHANNELS?1:0))
- #define **SHIELD_STATE** 0x00

Functions

- void ls_init ()

  Line Selection initialisation.
- uint16_t ls_read_reg (uint8_t IC_address, uint8_t reg)

  Read a register of an MCP32S17.
- void ls_set_reg (uint8_t IC_address, uint8_t reg, uint16_t data)

  Set a register to a desired value. See MCP32S17 datasheet for more Register details.
- void ls_row (uint8_t row)

  Select the desired row.
- void ls_col (uint8_t column)

  Select the desired column.
- void ls_set_all_lines_shield ()

  this sets all lines to the shield signal it is used for debugging purposes.

5.4.1 Detailed Description

Line selection (row or column) for Capacitive Measurement.

5.4.2 Function Documentation

5.4.2.1 ls_col() `void ls_col ( uint8_t column )`

Select the desired column.

Parameters

column	- Column number from 1 to the number of columns defined.

```
242 {
243     if (column)
244     {
245         return;
246     }
247     column = column - 1; // to turn into 0 indexing
248     m_tx_MCP32S17_IO_set[1] = MCP_REGISTER_OLATA_ADDR;
249     for (int i = 0; i < MCP_NUM_OF_IC_COLUMNS; i++)
250     {
251         m_tx_MCP32S17_IO_set[0] = mcp_column_chip_write_address[i];
252         if (((mcp_column_packets[column].chip_and_port_index) / 2) == i)
253         {
254             if (mcp_column_packets[column].chip_and_port_index % 2)
255             {
256                 m_tx_MCP32S17_IO_set[2] = SHIELD_STATE;
257                 m_tx_MCP32S17_IO_set[3] = mcp_column_packets[column].value;
258             }
259             else
260             {
261                 m_tx_MCP32S17_IO_set[2] = mcp_column_packets[column].value;
262                 m_tx_MCP32S17_IO_set[3] = SHIELD_STATE;
263             }
264         }
265         else
266         {
267             m_tx_MCP32S17_IO_set[2] = SHIELD_STATE;
268             m_tx_MCP32S17_IO_set[3] = SHIELD_STATE;
269         }
270         m_spi_xfered_flag = false;
```

```
271         APP_ERROR_CHECK(nrfx_spim_xfer(&spi, &td_MCP_IO_set, 0));
272         spi_xfer_wait();
273     }
274 }
```

5.4.2.2 ls_init() `void ls_init ( )`

Line Selection initialisation.

```
174 {
175     spi_setup();
176     set_all_lines_shield();
177 }
```

5.4.2.3 ls_read_reg() `uint16_t ls_read_reg ( uint8_t IC_address, uint8_t reg )`

Read a register of an MCP32S17.

Parameters

IC_address	- The address of a specific MCP32S17.
reg	- The register in the MCP3217 IC.

Returns

- (uint16_t) The value of the register.

```
180 {
181     m_tx_MCP32S17_IO_read[0] = IC_address;
182     m_tx_MCP32S17_IO_read[1] = reg;
183     m_spi_xfered_flag = false;
184     APP_ERROR_CHECK(nrfx_spim_xfer(&spi, &td_MCP_IO_read, 0));
185     spi_xfer_wait();
186     uint16_t temp = ((m_rx_MCP32S17_IO_read[2] « 8) | m_rx_MCP32S17_IO_read[3]);
187     return temp;
188 }
```

5.4.2.4 ls_row() `void ls_row ( uint8_t row )`

Select the desired row.

Parameters

row	- Row number from 1 to the number of rows defined.

When setting a row or colum (line), we are setting the Analog SPDT switchs (through the MCP32S17 GPIO expander) to have all the lines bar the one of interest be in a shield state. That line of interest is connected to the PCAP04() - for row lines .... and column lines .....

```
206 {
207     if (row > NUM_OF_ROWS)
208     {
209         return;
210     }
211     row = row - 1; // to turn into 0 indexing
212     m_tx_MCP32S17_IO_set[1] = MCP_REGISTER_OLATA_ADDR;
213     for (int i = 0; i < MCP_NUM_OF_IC_ROWS; i++)
214     {
215         m_tx_MCP32S17_IO_set[0] = mcp_row_chip_write_address[i];
216         if (((mcp_row_packets[row].chip_and_port_index) / 2) == i)
217         {
218             if (mcp_row_packets[row].chip_and_port_index % 2)
219             {
220 
221                 m_tx_MCP32S17_IO_set[2] = SHIELD_STATE;
222                 m_tx_MCP32S17_IO_set[3] = mcp_row_packets[row].value;
223             }
224             else
225             {
226                 m_tx_MCP32S17_IO_set[2] = mcp_row_packets[row].value;
227                 m_tx_MCP32S17_IO_set[3] = SHIELD_STATE;
228             }
229         }
230         else
231         {
232             m_tx_MCP32S17_IO_set[2] = SHIELD_STATE;
233             m_tx_MCP32S17_IO_set[3] = SHIELD_STATE;
234         }
235         m_spi_xfered_flag = false;
236         APP_ERROR_CHECK(nrfx_spim_xfer(&spi, &td_MCP_IO_set, 0));
237         spi_xfer_wait();
238     }
239 }
```

5.4.2.5 ls_set_all_lines_shield() `void ls_set_all_lines_shield ( )`

this sets all lines to the shield signal it is used for debugging purposes.

```
277 {
278     for (int i = 0; i < MCP_NUM_OF_IC_ROWS; i++)
279     {
280         m_tx_MCP32S17_IO_set[1] = MCP_REGISTER_OLATA_ADDR;
281         m_tx_MCP32S17_IO_set[0] = mcp_row_chip_write_address[i];
282         m_tx_MCP32S17_IO_set[2] = SHIELD_STATE;
283         m_tx_MCP32S17_IO_set[3] = SHIELD_STATE;
284         m_spi_xfered_flag = false;
285         APP_ERROR_CHECK(nrfx_spim_xfer(&spi, &td_MCP_IO_set, 0));
286         spi_xfer_wait();
287     }
288     for (int i = 0; i < MCP_NUM_OF_IC_COLUMNS; i++)
289     {
290         m_tx_MCP32S17_IO_set[1] = MCP_REGISTER_OLATA_ADDR;
291         m_tx_MCP32S17_IO_set[0] = mcp_column_chip_write_address[i];
292         m_tx_MCP32S17_IO_set[2] = SHIELD_STATE;
293         m_tx_MCP32S17_IO_set[3] = SHIELD_STATE;
294         m_spi_xfered_flag = false;
295         APP_ERROR_CHECK(nrfx_spim_xfer(&spi, &td_MCP_IO_set, 0));
296         spi_xfer_wait();
297     }
298 }
```

5.4.2.6 ls_set_reg() `void ls_set_reg (`

`uint8_t` *IC_address*,
`uint8_t` *reg*,
`uint16_t` *data*)

Set a register to a desired value. See MCP32S17 datasheet for more Register details.

Parameters

IC_address	- The address of a specific MCP32S17.
reg	- The register in the MCP3217 IC.
data	- Data to set in the register.

```
191 {
192     m_tx_MCP32S17_IO_set[0] = IC_address;
193     m_tx_MCP32S17_IO_set[1] = reg;
194     m_tx_MCP32S17_IO_set[2] = data;
195     m_tx_MCP32S17_IO_set[3] = data » 8;
196     m_spi_xfered_flag = false;
197     APP_ERROR_CHECK(nrfx_spim_xfer(&spi, &td_MCP_IO_set, 0));
198     spi_xfer_wait();
199 }
```

5.5 MCP32S17

MCP32S17 16 channel GPI Driver for the queen pressure sensor array. It connects a MCP IC address and GPIO address to a specific row or column.

Classes

- struct mcp_packet_t

Macros

- #define **BASE_ADDRESS_WRITE** 0x40
- #define **BASE_ADDRESS_READ** 0x41
- #define **MCP32S17_1_ADDRESS_W** BASE_ADDRESS_WRITE|0x00
- #define **MCP32S17_2_ADDRESS_W** BASE_ADDRESS_WRITE|0x02
- #define **MCP32S17_3_ADDRESS_W** BASE_ADDRESS_WRITE|0x04
- #define **MCP32S17_4_ADDRESS_W** BASE_ADDRESS_WRITE|0x06
- #define **MCP32S17_5_ADDRESS_W** BASE_ADDRESS_WRITE|0x08
- #define **MCP32S17_6_ADDRESS_W** BASE_ADDRESS_WRITE|0x0A
- #define **MCP32S17_7_ADDRESS_W** BASE_ADDRESS_WRITE|0x0C
- #define **MCP32S17_8_ADDRESS_W** BASE_ADDRESS_WRITE|0x0E
- #define **MCP32S17_1_ADDRESS_R** BASE_ADDRESS_READ|0x00
- #define **MCP32S17_2_ADDRESS_R** BASE_ADDRESS_READ|0x02
- #define **MCP32S17_3_ADDRESS_R** BASE_ADDRESS_READ|0x04
- #define **MCP32S17_4_ADDRESS_R** BASE_ADDRESS_READ|0x06
- #define **MCP32S17_5_ADDRESS_R** BASE_ADDRESS_READ|0x08
- #define **MCP32S17_6_ADDRESS_R** BASE_ADDRESS_READ|0x0A
- #define **MCP32S17_7_ADDRESS_R** BASE_ADDRESS_READ|0x0C
- #define **MCP32S17_8_ADDRESS_R** BASE_ADDRESS_READ|0x0E
- #define **MCP_ROW_IC_1_ADDRESS_W** MCP32S17_1_ADDRESS_W
- #define **MCP_ROW_IC_2_ADDRESS_W** MCP32S17_2_ADDRESS_W
- #define **MCP_ROW_IC_3_ADDRESS_W** MCP32S17_3_ADDRESS_W
- #define **MCP_ROW_IC_1_ADDRESS_R** MCP32S17_1_ADDRESS_R
- #define **MCP_ROW_IC_2_ADDRESS_R** MCP32S17_2_ADDRESS_R
- #define **MCP_ROW_IC_3_ADDRESS_R** MCP32S17_3_ADDRESS_R
- #define **MCP_COL_IC_1_ADDRESS_W** MCP32S17_4_ADDRESS_W
- #define **MCP_COL_IC_2_ADDRESS_W** MCP32S17_5_ADDRESS_W
- #define **MCP_COL_IC_1_ADDRESS_R** MCP32S17_4_ADDRESS_R
- #define **MCP_COL_IC_2_ADDRESS_R** MCP32S17_5_ADDRESS_R
- #define **MCP_REGISTER_PORT_ADDR** 0x12;
- #define **MCP_REGISTER_OLATA_ADDR** 0x14;
- #define MCP_PACKET_ROW_INIT(chip_and_port_index_, value_)
- #define MCP_PACKET_COLUMN_INIT(chip_and_port_index_, value_)
- #define **ROW_1** MCP_PACKET_ROW_INIT(1,0x10)

- #define **ROW_2** MCP_PACKET_ROW_INIT(1,0x40)
- #define **ROW_3** MCP_PACKET_ROW_INIT(1,0x04)
- #define **ROW_4** MCP_PACKET_ROW_INIT(1,0x08)
- #define **ROW_5** MCP_PACKET_ROW_INIT(1,0x02)
- #define **ROW_6** MCP_PACKET_ROW_INIT(1,0x01)
- #define **ROW_7** MCP_PACKET_ROW_INIT(0,0x80)
- #define **ROW_8** MCP_PACKET_ROW_INIT(0,0x20)
- #define **ROW_9** MCP_PACKET_ROW_INIT(0,0x40)
- #define **ROW_10** MCP_PACKET_ROW_INIT(0,0x02)
- #define **ROW_11** MCP_PACKET_ROW_INIT(0,0x10)
- #define **ROW_12** MCP_PACKET_ROW_INIT(0,0x04)
- #define **ROW_13** MCP_PACKET_ROW_INIT(0,0x08)
- #define **ROW_14** MCP_PACKET_ROW_INIT(3,0x20)
- #define **ROW_15** MCP_PACKET_ROW_INIT(0,0x01)
- #define **ROW_16** MCP_PACKET_ROW_INIT(3,0x40)
- #define **ROW_17** MCP_PACKET_ROW_INIT(3,0x10)
- #define **ROW_18** MCP_PACKET_ROW_INIT(3,0x08)
- #define **ROW_19** MCP_PACKET_ROW_INIT(3,0x04)
- #define **ROW_20** MCP_PACKET_ROW_INIT(2,0x80)
- #define **ROW_21** MCP_PACKET_ROW_INIT(3,0x02)
- #define **ROW_22** MCP_PACKET_ROW_INIT(3,0x01)
- #define **ROW_23** MCP_PACKET_ROW_INIT(2,0x40)
- #define **ROW_24** MCP_PACKET_ROW_INIT(2,0x20)
- #define **ROW_25** MCP_PACKET_ROW_INIT(2,0x10)
- #define **ROW_26** MCP_PACKET_ROW_INIT(2,0x02)
- #define **ROW_27** MCP_PACKET_ROW_INIT(2,0x08)
- #define **ROW_28** MCP_PACKET_ROW_INIT(2,0x04)
- #define **ROW_29** MCP_PACKET_ROW_INIT(2,0x01)
- #define **ROW_30** MCP_PACKET_ROW_INIT(5,0x04)
- #define **ROW_31** MCP_PACKET_ROW_INIT(5,0x02)
- #define **ROW_32** MCP_PACKET_ROW_INIT(5,0x08)
- #define **ROW_33** MCP_PACKET_ROW_INIT(4,0x80)
- #define **ROW_34** MCP_PACKET_ROW_INIT(5,0x01)
- #define **ROW_35** MCP_PACKET_ROW_INIT(4,0x40)
- #define **ROW_36** MCP_PACKET_ROW_INIT(4,0x10)
- #define **ROW_37** MCP_PACKET_ROW_INIT(4,0x08)
- #define **ROW_38** MCP_PACKET_ROW_INIT(4,0x20)
- #define **ROW_39** MCP_PACKET_ROW_INIT(4,0x02)
- #define **ROW_40** MCP_PACKET_ROW_INIT(4,0x04)
- #define **ROW_41** MCP_PACKET_ROW_INIT(4,0x01)
- #define **COL_1** MCP_PACKET_COLUMN_INIT(0, 0x01)
- #define **COL_2** MCP_PACKET_COLUMN_INIT(0, 0x08)
- #define **COL_3** MCP_PACKET_COLUMN_INIT(0, 0x10)
- #define **COL_4** MCP_PACKET_COLUMN_INIT(0, 0x40)
- #define **COL_5** MCP_PACKET_COLUMN_INIT(1, 0x02)
- #define **COL_6** MCP_PACKET_COLUMN_INIT(1, 0x04)
- #define **COL_7** MCP_PACKET_COLUMN_INIT(1, 0x10)
- #define **COL_8** MCP_PACKET_COLUMN_INIT(2, 0x01)
- #define **COL_9** MCP_PACKET_COLUMN_INIT(2, 0x08)
- #define **COL_10** MCP_PACKET_COLUMN_INIT(2, 0x10)
- #define **COL_11** MCP_PACKET_COLUMN_INIT(2, 0x40)
- #define **COL_12** MCP_PACKET_COLUMN_INIT(2, 0x80)
- #define **COL_13** MCP_PACKET_COLUMN_INIT(3, 0x02)
- #define **COL_14** MCP_PACKET_COLUMN_INIT(3, 0x14)
- #define **COL_15** MCP_PACKET_COLUMN_INIT(3, 0x10)

- #define **COL_16** MCP_PACKET_COLUMN_INIT(3, 0x20)
- #define **COL_17** MCP_PACKET_COLUMN_INIT(3, 0x40)
- #define **COL_18** MCP_PACKET_COLUMN_INIT(3, 0x08)
- #define **COL_19** MCP_PACKET_COLUMN_INIT(3, 0x01)
- #define **COL_20** MCP_PACKET_COLUMN_INIT(2, 0x20)
- #define **COL_21** MCP_PACKET_COLUMN_INIT(2, 0x02)
- #define **COL_22** MCP_PACKET_COLUMN_INIT(2, 0x04)
- #define **COL_23** MCP_PACKET_COLUMN_INIT(1, 0x20)
- #define **COL_24** MCP_PACKET_COLUMN_INIT(1, 0x40)
- #define **COL_25** MCP_PACKET_COLUMN_INIT(1, 0x08)
- #define **COL_26** MCP_PACKET_COLUMN_INIT(0, 0x80)
- #define **COL_27** MCP_PACKET_COLUMN_INIT(1, 0x01)
- #define **COL_28** MCP_PACKET_COLUMN_INIT(0, 0x20)
- #define **COL_29** MCP_PACKET_COLUMN_INIT(0, 0x02)
- #define **COL_30** MCP_PACKET_COLUMN_INIT(0, 0x04)

Typedefs

- typedef struct mcp_packet_t **mcp_packet_t**

Variables

- uint8_t **mcp_packet_t::chip_and_port_index**
- uint8_t **mcp_packet_t::value**

5.5.1 Detailed Description

MCP32S17 16 channel GPI Driver for the queen pressure sensor array. It connects a MCP IC address and GPIO address to a specific row or column.

5.5.2 Macro Definition Documentation

5.5.2.1 MCP_PACKET_COLUMN_INIT `#define MCP_PACKET_COLUMN_INIT(`

```
chip_and_port_index_,
value_ )
```

Value:

```
    {\
    .chip_and_port_index = (chip_and_port_index_),\
    .value = (value_),\
}
```

5.5.2.2 MCP_PACKET_ROW_INIT `#define MCP_PACKET_ROW_INIT(`

```
chip_and_port_index_,
value_ )
```

Value:

```
    {\
    .chip_and_port_index = (chip_and_port_index_),\
    .value = (value_),\
}
```

5.6 Pins

Pins for queen sized pressure sensor array.

Macros

- #define **NUM_OF_ROWS** 41
  Number of rows in the sensor.
- #define **NUM_OF_COLUMNS** 30
  Number of columns in the sensor.
- #define **PIN_MCP_MISO** 36
  Section define for standard MCP board.
- #define **PIN_MCP_MOSI** 5
- #define **PIN_MCP_SCK** 38
- #define **PIN_MCP_CS** 39
- #define **PIN_MCP_RESET** 26
- #define **PIN_SN74LV_INH** 37
- #define **PIN_TXD** 45
- #define **PIN_OSC_TRIGGER** 45
- #define **PIN_RXD** 47
- #define **PIN_BME_MISO** 46
- #define **PIN_BME_MOSI** 3
- #define **PIN_BME_SCK** 2
- #define **PIN_BME_CS** 28
- #define **PIN_SW1** 30
- #define **PIN_BATTERY_V** 29
- #define **PIN_CAP_SIG_SEL** 27
- #define **PIN_SHIELD_SIG_IN** 4
- #define **PIN_PCAP04_SCK** 7
- #define **PIN_PCAP04_MOSI** 8
- #define **PIN_PCAP04_MISO** 41
- #define **PIN_PCAP04_CS** 11
- #define **PIN_PG3** 6
- #define **PIN_PG5** 40
- #define **PIN_MISO_PCAP05** 9
- #define **PIN_SHLD_SIG_OUT_SEL** 12
- #define **PIN_RESET_NRF** 18
- #define **PIN_QSPI_DATA_0** 17
- #define **PIN_QSPI_DATA_1** 23
- #define **PIN_QSPI_DATA_2** 22
- #define **PIN_QSPI_DATA_3** 21
- #define **PIN_QSPI_CS** 20
- #define **PIN_QSPI_SCK** 19

5.6.1 Detailed Description

Pins for queen sized pressure sensor array.

5.7 PPI

Setup of the programable peripheral interface, GPIOTE, an internal comparator and timer 1.

Functions

- void gpiote_setup ()

  Setting the following pins with the gpiote.
- void gpiote_enable_channels ()

  Enable the GPIOTE channels.
- void ppi_setup ()

  Make the connections between tasks and events.
- void lpcomp_init ()

  Initialize and setup the low power comparator for 1.23v on a rising edge.
- void timer_1_setup ()

  Setup timer 1. The purpose of the timer is to be started when the comparater detects a rising edge when the capacitive element being measured gets charged up. After a set period of time it triggers an event for PIN_CAP_SIG_SEL to select the other capacitive channel.
- void timer_1_update ()

  Update the values in the timer.

Variables

- uint32_t **m_spi_MCP_task_addr**
- uint32_t **spi_Cap_sense_task_addr**
- uint32_t **timer_1_task_addr**
- uint32_t **PG3_out_task_addr**
- uint32_t **SPDT_sel_task_set_addr**
- uint32_t **SPDT_sel_task_clr_addr**
- uint32_t **LPCOM_event_addr**
- uint32_t **PG5_in_event_addr**
- uint32_t **timer_1_event_cc0_addr**
- uint32_t **timer_1_event_cc1_addr**

5.7.1 Detailed Description

Setup of the programable peripheral interface, GPIOTE, an internal comparator and timer 1.

5.7.2 Function Documentation

5.7.2.1 gpiote_enable_channels() `void gpiote_enable_channels ( )`

Enable the GPIOTE channels.

```
55 {
56     nrfx_ppi_channel_enable(ppi_channel_PG5_to_SPIM);
57     nrfx_ppi_channel_enable(ppi_channel_opamp_sig_to_timer_1);
58     nrfx_ppi_channel_enable(ppi_channel_timer_1_to_shield_sel_lo);
59     nrfx_ppi_channel_enable(ppi_channel_timer_1_to_shield_sel_hi);
60 }
```

5.7.2.2 gpiote_setup() void gpiote_setup ( )

Setting the following pins with the gpiote.

- PG5 -> input event (signals when data from the PCAP04 is ready).
- PG3 -> output event sends a start conversion signal to the PCAP04 (low to high transition).
- PIN_CAP_SIG_SEL -> output event selects which capacitive measuring line to sample for the shield signal.

```
29 {
30     nrfx_gpiote_init();
31 
32     /*setup PG5 pin as an input. This pin recieves a signal to say when data fom the PCAP04 is ready to
       be read*/
33     nrfx_gpiote_in_config_t PG5_in_GPIOTE_config = NRFX_GPIOTE_CONFIG_IN_SENSE_HITOLO(true); // gpiote in
       configuration - data ready be read on a falling edge. (edge set high again by PCAP)
34     nrfx_gpiote_in_init(PIN_PCAP04_PG5, &PG5_in_GPIOTE_config, PG5_in_event_handler);        // initalise
       gpiote event w/ no  handler
35     PG5_in_event_addr = nrfx_gpiote_in_event_addr_get(PIN_PCAP04_PG5);                        // get the
       event address
36     nrfx_gpiote_in_event_enable(PIN_PCAP04_PG5, false);                                       // enable
       the event with no function handler
37 
38     /*GPIOTE OUT PG3. This pin sends a start conversion signal to the PCAP04 (low to high transition,
       when config of PCAP is in pin triggered) */
39     nrfx_gpiote_out_config_t PG3_out_task_config = NRFX_GPIOTE_CONFIG_OUT_TASK_LOW; // GPIOTE out -
       config to be set high on event (actualy set and clear driven anyway)
40     PG3_out_task_config.init_state = 0;                                                  // Set the inital
       ouput state to low
41     nrfx_gpiote_out_init(PIN_PCAP04_PG3, &PG3_out_task_config);                          // Initalise GPIOTE
       out task
42     PG3_out_task_addr = nrfx_gpiote_clr_task_addr_get(PIN_PCAP04_PG3);                   // Get the task
       address
43     nrfx_gpiote_out_task_enable(PIN_PCAP04_PG3);                                         // Enable the task
44 
45     /*GPIOTE OUT SPDT. This output signal selects which Measuring signal to select */
46     nrfx_gpiote_out_config_t SPDT_sel_out_task_config = NRFX_GPIOTE_CONFIG_OUT_TASK_HIGH; // GPIOTE out
       Config
47     SPDT_sel_out_task_config.init_state = 0;                                                 // Set the
       inital ouput state to low
48     nrfx_gpiote_out_init(PIN_CAP_SIG_SEL, &SPDT_sel_out_task_config);                        // Initalise
       GPIOTE out task
49     SPDT_sel_task_set_addr = nrfx_gpiote_set_task_addr_get(PIN_CAP_SIG_SEL);                 // Get the task
       set (set high) address
50     SPDT_sel_task_clr_addr = nrfx_gpiote_clr_task_addr_get(PIN_CAP_SIG_SEL);                 // Get the tack
       clear(set low) address
51     nrfx_gpiote_out_task_enable(PIN_CAP_SIG_SEL);                                            // Enable the
       Task
52 }
```

5.7.2.3 lpcomp_init() void lpcomp_init ( )

Initialize and setup the low power comparator for 1.23v on a rising edge.

```
82 {
83     uint32_t err_code;
84     nrf_drv_lpcomp_config_t config = NRF_DRV_LPCOMP_DEFAULT_CONFIG;
85     config.input = PIN_SHIELD_SIG_IN;
86     config.hal.detection = NRF_LPCOMP_DETECT_UP;                    // detect a crossing from a rising
       direction
87     config.hal.reference = NRF_LPCOMP_REF_SUPPLY_3_8;               //(3.3/8)*3 = 1.2375V
88     err_code = nrf_drv_lpcomp_init(&config, lpcomp_event_handler); // initialize LPCOMP driver, event
       handler will be executed when defined action is detected
89 
90     LPCOM_event_addr = (uint32_t)nrf_lpcomp_event_address_get(NRF_LPCOMP_EVENT_UP); // get event address
       for PPI
91     APP_ERROR_CHECK(err_code);
92     nrf_drv_lpcomp_enable(); // from this point LPCOMP will be active and provided
93 }
```

5.7.2.4 ppi_setup() void ppi_setup ( )

Make the connections between tasks and events.

Warning

All the task and event addresses need to be had before setting this up.

```
63 {
64     nrfx_ppi_channel_alloc(&ppi_channel_PG5_to_SPIM);
65     nrfx_ppi_channel_assign(ppi_channel_PG5_to_SPIM, PG5_in_event_addr, spi_Cap_sense_task_addr);
66     nrfx_ppi_channel_fork_assign(ppi_channel_PG5_to_SPIM, PG3_out_task_addr);
67 
68     nrfx_ppi_channel_alloc(&ppi_channel_opamp_sig_to_timer_1);
69     nrfx_ppi_channel_assign(ppi_channel_opamp_sig_to_timer_1, LPCOM_event_addr, timer_1_task_addr);
70 
71     nrfx_ppi_channel_alloc(&ppi_channel_timer_1_to_shield_sel_hi);
72     nrfx_ppi_channel_assign(ppi_channel_timer_1_to_shield_sel_hi, timer_1_event_cc0_addr,
       SPDT_sel_task_set_addr);
73 
74     nrfx_ppi_channel_alloc(&ppi_channel_timer_1_to_shield_sel_lo);
75     nrfx_ppi_channel_assign(ppi_channel_timer_1_to_shield_sel_lo, timer_1_event_cc1_addr,
       SPDT_sel_task_clr_addr);
77 }
```

5.7.2.5 timer_1_setup() void timer_1_setup ( )

Setup timer 1. The purpose of the timer is to be started when the comparater detects a rising edge when the capacitive element being measured gets charged up. After a set period of time it triggers an event for PIN_CAP_↩ SIG_SEL to select the other capacitive channel.

```
106 {
107     nrfx_timer_init(&timer_1, &timer_1_cfg, timer_1_event_handler);
108     nrfx_timer_compare(&timer_1, NRF_TIMER_CC_CHANNEL0, 0x0000000E, false);
109     nrfx_timer_extended_compare(&timer_1, NRF_TIMER_CC_CHANNEL1, 0x00000036,
       NRF_TIMER_SHORT_COMPARE1_STOP_MASK, false); // the order of the next two lines matter
110     nrfx_timer_extended_compare(&timer_1, NRF_TIMER_CC_CHANNEL2, 0x00000036,
       NRF_TIMER_SHORT_COMPARE1_CLEAR_MASK, false);
111 
112     // get task addresses
113     timer_1_task_addr = nrfx_timer_task_address_get(&timer_1, NRF_TIMER_TASK_START);
114     timer_1_event_cc0_addr = nrfx_timer_event_address_get(&timer_1, NRF_TIMER_EVENT_COMPARE0);
115     timer_1_event_cc1_addr = nrfx_timer_event_address_get(&timer_1, NRF_TIMER_EVENT_COMPARE1);
116 
117     nrfx_timer_enable(&timer_1);
118     timer_1.p_reg->TASKS_STOP = 1;  // dont need it to run just yet
119     timer_1.p_reg->TASKS_CLEAR = 1; // clear it too
120 }
```

5.7.2.6 timer_1_update() void timer_1_update ( )

Update the values in the timer.

```
125 {
126     uint16_t precharge_time = 0;
127     uint16_t fullcharge_time = 0;
128     if (pcap04_precharge_time == 0x3FF)
129         precharge_time = 0;
130     else
131         precharge_time = pcap04_precharge_time;
132     if (pcap04_fullcharge_time == 0x3FF)
133         fullcharge_time = 0;
134     else
135         fullcharge_time = pcap04_fullcharge_time;
136 
137     uint32_t new_time = (precharge_time + fullcharge_time + pcap04_discharge_time) / 2;
138     nrfx_timer_disable(&timer_1);
139     nrfx_timer_compare(&timer_1, NRF_TIMER_CC_CHANNEL0, new_time - TIMER_1_TIME_OFFSET_1, false);
140     nrfx_timer_extended_compare(&timer_1, NRF_TIMER_CC_CHANNEL1, (new_time * 3) + TIMER_1_TIME_OFFSET_2,
       NRF_TIMER_SHORT_COMPARE1_STOP_MASK, false); // the order of the next two lines matter
141     nrfx_timer_extended_compare(&timer_1, NRF_TIMER_CC_CHANNEL2, (new_time * 3) + TIMER_1_TIME_OFFSET_2,
       NRF_TIMER_SHORT_COMPARE1_CLEAR_MASK, false);
142     nrfx_timer_enable(&timer_1);
143 }
```

5.8 Pressure Array

The pressure array module communicates with and controls the external IC's to take in the raw data and assign it to a proper pixel in the pressure image.

Classes

- struct pressure_sensor_array_t

  Pressure sensor array (becuse this code is designed to be used ith many different sizes of arrays).
- struct pressure_image_t

  Pressure image meta data and data (becuse this code is designed to be used ith many different sizes of arrays).

Typedefs

- typedef struct pressure_sensor_array_t **pressure_sensor_array_t**

  Pressure sensor array (becuse this code is designed to be used ith many different sizes of arrays).
- typedef struct pressure_image_t **pressure_image_t**

  Pressure image meta data and data (becuse this code is designed to be used ith many different sizes of arrays).

Functions

- void pressure_array_setup (pressure_sensor_array_t ∗pressure_sensor_array, pressure_image_t ∗pressure↩ _image)

  Connects the pressure image to the pressure sensor.
- void pressure_array_image_get (pressure_image_t ∗pressure_image, pressure_sensor_array_t ∗pressure↩ _array)

  This is a loop that repeats for each pixel in the image until its done.
- error pres_img_setup (pressure_image_t ∗pressure_image, uint8_t width, uint8_t height)

  A pressure image has some parameters to be set. like the width and height of the image. This fuction also sets a default capture pattern. The capture pattern is what pressure points are being focused on in this frame. For now its all of them.
- error pres_img_set_capture_pattern (pressure_image_t ∗pressure_image, uint8_t ∗capture_pattern)

  The capture pattern is what pressure points would like to be focused on in this frame.
- error pres_img_print_caputre_pattern (pressure_image_t ∗pressure_image)

  Print the capture pattern that an image has. It presents it in an ASCII image of "1" and "0"'s representing the pixel to be recorded (1) and not be recorded (0).

Variables

- bool **pressure_sensor_array_t::get_pressure_image**
- spi_context_t ∗ **pressure_sensor_array_t::context**
- uint32_t ∗ **pressure_image_t::p_image**
- uint8_t **pressure_image_t::frame_width**
- uint8_t **pressure_image_t::frame_height**
- uint8_t ∗ **pressure_image_t::p_capture_pattern**
- uint8_t **pressure_image_t::capture_pattern_size**
- bool **pressure_image_t::image_updated**

5.8.1 Detailed Description

The pressure array module communicates with and controls the external IC's to take in the raw data and assign it to a proper pixel in the pressure image.

5.8.2 Function Documentation

5.8.2.1 pres_img_print_caputre_pattern() error pres_img_print_caputre_pattern (pressure_image_t * pressure_image)

Print the capture pattern that an image has. It presents it in an ASCII image of "1" and "0"'s representing the pixel to be recorded (1) and not be recorded (0).

Parameters

pressure_image	

Returns

error

```
163 {
164     if (pressure_image == NULL)
165         return error_failor_null;
166     int capture_pattern_byte = pressure_image->capture_pattern_size * pressure_image->frame_height;
167     NRF_LOG_INFO("number of bytes %d", capture_pattern_byte);
168     NRF_LOG_HEXDUMP_INFO(pressure_image->p_capture_pattern, capture_pattern_byte);
169     for (int i = 0; i < pressure_image->frame_height; i++)
170     {
171         for (int j = 0; j < pressure_image->capture_pattern_size; j++)
172         {
173             uint8_t index = ((i * pressure_image->capture_pattern_size) + j);
174             for (int k = 0; k < 8; k++)
175             {
176                 if (0x01 & (pressure_image->p_capture_pattern[index] » k))
177                 {
178                     NRF_LOG_RAW_INFO("1");
179                 }
180                 else
181                 {
182                     NRF_LOG_RAW_INFO("0");
183                 }
184             }
185         }
186         NRF_LOG_RAW_INFO("\n\r");
187     }
188     return 0;
189 }
```

5.8.2.2 pres_img_set_capture_pattern() error pres_img_set_capture_pattern (pressure_image_t * pressure_image, uint8_t * capture_pattern)

The capture pattern is what pressure points would like to be focused on in this frame.

Parameters

pressure_image	
capture_pattern	

Returns

error

```
154 {
155     uint16_t i_loop_range = pressure_image->capture_pattern_size * pressure_image->frame_height;
156     for (int i = 0; i < i_loop_range; i++)
157     {
158         pressure_image->p_capture_pattern[i] = capture_pattern[i];
159     }
160 }
```

5.8.2.3 pres_img_setup()

```
error pres_img_setup (
        pressure_image_t * pressure_image,
        uint8_t width,
        uint8_t height )
```

A pressure image has some parameters to be set. like the width and height of the image. This fuction also sets a default capture pattern. The capture pattern is what pressure points are being focused on in this frame. For now its all of them.

Parameters

pressure_image	- a pointer to the image object.
width	- the desired width of the image
height	- the the desired height of the image

Returns

error

```
121 {
122 
123     pressure_image->frame_width = width;
124     pressure_image->frame_height = height;
125     pressure_image->p_image = (uint32_t *)calloc((width * height), sizeof(uint32_t));
126     if (pressure_image->p_image == NULL)
127         return error_failor_mem_aloc_pressure_img; // return error
128     uint8_t capture_pattern_size_local = width % 8 ? ((width / 8) + 1) : (width / 8);
129     pressure_image->capture_pattern_size = capture_pattern_size_local;
130     pressure_image->p_capture_pattern = (uint8_t *)malloc((pressure_image->capture_pattern_size *
      height) * sizeof(uint8_t));
131     if (pressure_image->p_capture_pattern == NULL)
132         return error_failor_mem_aloc_pressure_array_capture_pattern; // return error
133     uint8_t frame_size_last_byte_remainder = width % 8;
134     for (int i = 0; i < height; i++)
135     {
136         for (int j = 0; j < capture_pattern_size_local; j++)
137         {
138             if (frame_size_last_byte_remainder && (j == (capture_pattern_size_local - 1)))
139             {
140                 // if there is a remainder on the last byte and the itteration is on the last byte
141                 pressure_image->p_capture_pattern[(i * capture_pattern_size_local) + j] = (0xFF >> (8 -
      frame_size_last_byte_remainder));
142             }
143             else
144             {
145                 pressure_image->p_capture_pattern[(i * capture_pattern_size_local) + j] = 0xFF;
146             }
```

```
147         }
148     }
149     return 0;
150     // pressure_image->p_image = (uint32_t *)malloc((frame_width * image_frame_height) *
        sizeof(uint32_t)); // to optimize instead of calloc
151 }
```

5.8.2.4 pressure_array_image_get() void pressure_array_image_get (pressure_image_t * *pressure_image*, pressure_sensor_array_t * *pressure_array*)

This is a loop that repeats for each pixel in the image until its done.

Parameters

pressure_sensor_array	- Reference to the setup array.
pressure_image	- The pressure image to store the data in.

```
23 {
24 //  this label is there for a recheck. A goto is for code efficiency (reduece if checks). goto strictly
       contained in this function and in one place.
25 //  all the conditional statements (ifs) make up a 3 sequence state machine (start -> getting ->end).
26 pressure_array_get_loop:
27 {
28     if ((getting_image) && (pressure_array->context->read_cycle_complette == true))
29     {
30         if ((column_number == pressure_image->frame_width) && (row_number ==
       pressure_image->frame_height))
31         {
32             getting_image = false;
33             start_sequence = false;
34             end_sequence = true;
35             goto pressure_array_get_loop; // go back to test the if statement
36         }
37         do
38         {
39             if (column_number >= pressure_image->frame_width) // when reaching the end of the column
40             {
41                 if (row_number >= pressure_image->frame_height)
42                 {
43                     break;
44                 }
45                 else
46                 {
47                     column_number = 1; // reset the column number to cycle through the row
48                 }
49                 bit_number = 0;
50                 byte_number = 0;
51                 row_number++;           // increment to the next row.
52                 row_number_changed = true;
53             }
54             else   // when the colum hasn't reached the end
55             {
56                 if (bit_number >= 7)
57                 {
58                     byte_number++;
59                     bit_number = 0;
60                 }
61                 else
62                 {
63                     bit_number++;
64                 }
65                 column_number++;
66             }
67         } while (!(pressure_image->p_capture_pattern[(row_number - 1) *
       pressure_image->capture_pattern_size + byte_number] & (1 << bit_number)));
68         if (row_number_changed == true)
69         {
70             ls_row(row_number);
71             //              NRF_LOG_INFO("Row Selected: %d ",row_number);
72             //              pressure_array_row_read();
73             row_number_changed = false;
74         }
75         ls_col(column_number);
76         // pressure_array_column_read();
```

```
77          if (column_number < 2)
78          {
79              // NRF_LOG_INFO("previous value for row %d - #d",column_number,pressure_array->context-> );
80          }
81          pressure_array->context->read_cycle_complette = false;
82          pressure_array->context->index = ((row_number - 1) * pressure_image->frame_width) +
        (column_number - 1);
83          //      nrf_delay_ms(10);
84          cs_request_data();
85          cs_request_conversion();
86      }
87      else if (start_sequence)
88      {
89          // nrf_gpio_pin_set(PIN_SHLD_SIG_OUT_SEL);
90          nrf_gpio_pin_set(PIN_OSC_TRIGGER);
91          nrf_delay_us(100);
92          nrf_gpio_pin_clear(PIN_OSC_TRIGGER);
93          global_clock_time_start();
94          start_sequence = false;
95          getting_image = true;
96          column_number = 0;
97          row_number = 1;
98          row_number_changed = true;
99          bit_number = -1;
100          byte_number = 0;
101          pressure_array->context->read_pcap04_CDC_data = true;
102          pressure_array->context->read_cycle_complette = true;
103          goto pressure_array_get_loop; // go back to test the if statement
104      }
105      else if (end_sequence)
106      {
107          // nrf_gpio_pin_clear(PIN_SHLD_SIG_OUT_SEL);
108          pressure_array->get_pressure_image = false;
109          pressure_image->image_updated = true;
110
111          end_sequence = false;
112          start_sequence = true;
113          global_clock_time_stop();
114      }
115  }
116
117      //  } while
        (!(pressure_image->p_capture_pattern[row_number*pressure_image->capture_pattern_byte_width +
        (column_number/8)] & (1 << column_number%8))) ;
118  }
```

5.8.2.5 pressure_array_setup() `void pressure_array_setup (`

`pressure_sensor_array_t * pressure_sensor_array,`
`pressure_image_t * pressure_image )`

Connects the pressure image to the pressure sensor.

Parameters

pressure_sensor_array	- The pressure sensor array.
pressure_image	- The image to record data to.

```
16  {
17      pressure_sensor_array->get_pressure_image = false;
18      pressure_sensor_array->context = &spi_context;
19      pressure_sensor_array->context->data_buffer = pressure_image->p_image;
20  }
```

5.9 USB

USB module to setup and send data out via usb.

Classes

- struct commands

Typedefs

- typedef struct commands **command**

Enumerations

- enum **command_type_t** {
  cmd_error = 0x00000000 , **new_command** = 0x646D636E , **select_row** = 0x00776F72 , **select_column** = 0x006C6F63 ,
  select_index = 0x78646E69 , **cmd_read** = 0x64616572 }

Functions

- void usb_init ()

  Initialize the USB peripheral. Usb much of the setup code is copied out of nordics USB example.
- void usb_write (char ∗message)

  Write a message.
- void usb_print_pressure_image (uint32_t ∗p_image, uint32_t size)

  Print out via USB a pressure image.
- command analyse_sentance (const char ∗sentence, uint32_t lenght)

  Analyse a line of characters to see if it matches an appropriate command.
- void usb_transcode_command (const char ∗message)

  Used for creating a new command to add to the command type enum.
- char ∗ usb_command_to_string (command_type_t command_type)

  Turns a command type into a text string.
- void usb_print_command (command cmd)

Variables

- command **g_usb_mesage**
- bool **usb_new_message_f**
- bool **read_psa_bool**

5.9.1 Detailed Description

USB module to setup and send data out via usb.

5.9.2 Enumeration Type Documentation

5.9.2.1 command_type_t enum command_type_t

```
33 {
34     cmd_error = 0x00000000,
35     new_command = 0x646D636E,   // ncmd backwards
36     select_row = 0x00776F72,    // row backwards
37     select_column = 0x006C6F63, // col backwards
38     select_index = 0x78646E69,  // indx backwards
39     cmd_read = 0x64616572,      // cmd read
40 } command_type_t;
```

5.9.3 Function Documentation

5.9.3.1 analyse_sentance() command analyse_sentance (const char * sentence, uint32_t lenght)

Analyse a line of characters to see if it matches an appropriate command.

Parameters

sentence	
lenght	

Returns

command

```
227 {
228     command r_value;
229     uint8_t command_size = 0;
230     uint32_t command = 0;
231     bool success = false;
232     for (int i = 0; i <= 5; i++)
233     {
234         if (sentence[i] >= 'a' && sentence[i] <= 'z' && i < 4)
235         {
236             command |= (int32_t)sentence[i] « i * 8;
237             command_size++;
238         }
239         else if (sentence[i] == ' ')
240         {
241             if (command == 0)
242             {
243                 r_value.ctype = cmd_error;  // return an error message
244                 r_value.value = 0x00646d63; // the message : cmd
245                 return r_value;
246             }
247             r_value.ctype = command;
248             break;
249         }
250         else
251         {
252             r_value.ctype = cmd_error;  // return an error message
253             r_value.value = 0x00646d63; // the message : cmd
254             return r_value;
255         }
256     }
257
258     bool potential_error = true;
259     bool potential_leading_zeros = true;
260     uint32_t new_value = 0;
261     for (int i = (command_size + 1); i < 9 + (command_size + 1) && i < lenght; i++)
262     { // 9+(1 extra from command size) is the digit size of a 2^32 number
263         if (r_value.ctype == cmd_read)
264         {
265             r_value.value = 0;
266             return r_value;
267         }
268         else if (r_value.ctype == new_command)
```

```
269         {
270             while (sentence[i] != '\n')
271             {
272                 new_value |= (int32_t)sentence[i] « (i - 5) * 8;
273                 i++;
274             }
275             r_value.value = new_value;
276             return r_value;
277         }
278         else if (sentence[i] >= '0' && sentence[i] <= '9')
279         {
280             if (potential_leading_zeros)
281             {
282                 if (sentence[i] == '0')
283                 {
284                     continue;
285                 }
286                 else
287                 {
288                     potential_leading_zeros = false;
289                 }
290             }
291             potential_error = false;
292             new_value = new_value * 10 + (sentence[i] - 48);
293         }
294         else
295         {
296             if (potential_error)
297             {
298                 r_value.ctype = cmd_error;  // return an error message
299                 r_value.value = 0x006c6176; // the message : val
300                 return r_value;
301             }
302             else
303             {
304                 r_value.value = new_value;
305                 return r_value;
306             }
307         }
308     }
309     r_value.value = new_value;
310     return r_value;
311 }
```

5.9.3.2 usb_command_to_string() `char * usb_command_to_string ( command_type_t command_type )`

Turns a command type into a text string.

Parameters

command_type	

Returns

char*

```
339 {
340     uint32_t cmdtype = (uint32_t)command_type;
341     int index = 0;
342     char temp_val = 0;
343     for (int i = 0; i < sizeof(int); i++)
344     {
345         temp_val = (char)(cmdtype » i * 8);
346         if (temp_val == 0)
347             continue;
348         m_command_string[index] = temp_val;
349         index++;
350     }
351     m_command_string[index] = '\0';
352     return m_command_string;
353 }
```

5.9.3.3 usb_init() void usb_init ( )

Initialize the USB peripheral. Usb much of the setup code is copied out of nordics USB example.

```
170 {
171     m_usb_port_open = false;
172 
173     ret_code_t ret;
174     static const app_usbd_config_t usbd_config = {
175         .ev_state_proc = usbd_user_ev_handler};
176 
177     ret = NRF_LOG_INIT(NULL);
178     APP_ERROR_CHECK(ret);
179 
180     ret = nrf_drv_clock_init();
181     APP_ERROR_CHECK(ret);
182 
183     nrf_drv_clock_lfclk_request(NULL);
184 
185     while (!nrf_drv_clock_lfclk_is_running())
186     { /* Just waiting */
187     }
188 
189     ret = app_timer_init();
190     APP_ERROR_CHECK(ret);
191 
192     app_usbd_serial_num_generate();
193 
194     ret = app_usbd_init(&usbd_config);
195     APP_ERROR_CHECK(ret);
196     NRF_LOG_INFO("USBD CDC ACM example started.");
197 
198     class_cdc_acm = app_usbd_cdc_acm_class_inst_get(&m_app_cdc_acm);
199     ret = app_usbd_class_append(class_cdc_acm);
200     APP_ERROR_CHECK(ret);
201 
202     if (USBD_POWER_DETECTION)
203     {
204         ret = app_usbd_power_events_enable();
205         APP_ERROR_CHECK(ret);
206     }
207     else
208     {
209         NRF_LOG_INFO("No USB power detection enabled\r\nStarting USB now");
210         app_usbd_enable();
211         app_usbd_start();
212     }
213     m_usb_is_setup = true;
214 }
```

5.9.3.4 usb_print_command() void usb_print_command (command *cmd*)

```
314 {
315     char letter;
316     if (cmd.ctype == 0)
317     {
318         NRF_LOG_RAW_INFO("Error : ");
319         char const *error_message = usb_command_to_string(cmd.value);
320         NRF_LOG_RAW_INFO("%s\n\r", error_message);
321     }
322     else
323     {
324         for (int i = 0; i < sizeof(int); i++)
325         {
326             letter = (char)cmd.ctype;
327             cmd.ctype >>= 8;
328             if (letter != 0)
329                 NRF_LOG_RAW_INFO("%c", letter);
330         }
331         NRF_LOG_RAW_INFO(" : %u\r\n", cmd.value);
332         // NRF_LOG_RAW_INFO("\r\n");
333     }
334 }
```

5.9.3.5 usb_print_pressure_image()

```
void usb_print_pressure_image (
        uint32_t * p_image,
        uint32_t size )
```

Print out via USB a pressure image.

Parameters

p_image	
size	

```
376 {
377     size_t usb_buffer_size;
378     usb_buffer_size = sprintf(m_tx_buffer, "<p_image>");
379     size = size - 1;
380     app_usbd_cdc_acm_write(&m_app_cdc_acm, m_tx_buffer, usb_buffer_size);
381     for (int i = 0; i < size; i++)
382     {
383         usb_buffer_size = sprintf(m_tx_buffer, "%u|%u,", i, p_image[i]);
384         app_usbd_cdc_acm_write(&m_app_cdc_acm, m_tx_buffer, usb_buffer_size);
385         nrf_delay_us(250);
386     }
387 
388     usb_buffer_size = sprintf(m_tx_buffer, "%u|%u</p_image>", (size), p_image[size]);
389     app_usbd_cdc_acm_write(&m_app_cdc_acm, m_tx_buffer, usb_buffer_size);
390 }
```

5.9.3.6 usb_transcode_command() void usb_transcode_command (const char * *message*)

Used for creating a new command to add to the command type enum.

Parameters

message	

```
356 {
357     char new_command[4] = {0, 0, 0, 0};
358     for (int i = 0; i < 4 && message[i] != '\0'; i++)
359     {
360         if (message[i] >= 'a' && message[i] <= 'z')
361         {
362             new_command[3 - i] = message[i];
363         }
364         else
365         {
366             NRF_LOG_INFO("command charater input invalid %C", new_command[i]);
367             return;
368         }
369     }
370     NRF_LOG_RAW_INFO("new command = 0x");
371     NRF_LOG_RAW_HEXDUMP_INFO(new_command, 4);
372     NRF_LOG_RAW_INFO("\n\r");
373 }
```

5.9.3.7 usb_write() void usb_write (char * *message*)

Write a message.

Parameters

message	

```
217 {
218     if (m_usb_port_open && m_usb_is_setup)
219     {
220         size_t size = sprintf(m_tx_buffer, "%s", message);
221         app_usbd_cdc_acm_write(&m_app_cdc_acm, m_tx_buffer, size);
```

```
222     }
223 }
```

5.10 Main

Conduct system setups. Most of them are peripherials.

Functions

- void setups ()
- void command_to_run (command cmd)

  When a command is received this fuction handles its.
- int main (void)

  Main function.

5.10.1 Detailed Description

Conduct system setups. Most of them are peripherials.

5.10.2 Function Documentation

5.10.2.1 command_to_run() void command_to_run (command *cmd*)

When a command is received this fuction handles its.

Parameters

cmd	The command to run

```
87 {
88   switch (cmd.ctype)
89   {
90   case cmd_error:
91     NRF_LOG_INFO("(%s) error ", usb_command_to_string(cmd.ctype));
92     break;
93   case cmd_read:
94     PSA.get_pressure_image = true;
95     break;
96   case select_row:
97     NRF_LOG_INFO("%s %u selected", usb_command_to_string(cmd.ctype), cmd.value);
98     ls_row(cmd.value);
99     break;
100
101   case select_column:
102     NRF_LOG_INFO("%s %u selected", usb_command_to_string(cmd.ctype), cmd.value);
103     ls_col(cmd.value);
104     break;
105   case new_command:
106     usb_transcode_command(usb_command_to_string(cmd.value));
107     break;
108
109   default:
110     NRF_LOG_INFO("(%s)no matching command found ", usb_command_to_string(cmd.ctype));
111     break;
112   }
```

```
113 }
```

5.10.2.2 main() int main (void)

Main function.

```
120 {
121   setups(); // Group all of the setups
122   pres_img_setup(&img_std, NUM_OF_COLUMNS, NUM_OF_ROWS); // setup a
123   pressure_array_setup(&PSA, &img_std);
124   pres_img_print_caputre_pattern(&img_std);
125 
126   while (1)
127   {
128     if (usb_new_message_f)          //if a new message flag is set
129     {
130       usb_new_message_f = false;    // reset the flag
131       command_to_run(g_usb_mesage); // look for what command it is that needs to be run
132     }
133     if (PSA.get_pressure_image)     // if get an image
134     {
135       pressure_array_image_get(&img_std, &PSA); // run the loop to get the image
136     }
137     if (img_std.image_updated && (PSA.get_pressure_image == false))   // if the image is completed
138     {
139       img_std.image_updated = false;
140       usb_print_pressure_image(img_std.p_image, (img_std.frame_height * img_std.frame_width)); // send
      the image out via usb
141     }
142     while (app_usbd_event_queue_process())
143     {
144       /* Nothing to do */
145     }
146     UNUSED_RETURN_VALUE(NRF_LOG_PROCESS());
147   }
148 }
```

5.10.2.3 setups() void setups ( )

```
61 {
62   APP_ERROR_CHECK(NRF_LOG_INIT(NULL)); //setup logging
63   NRF_LOG_DEFAULT_BACKENDS_INIT();
64 
65   nrf_gpio_cfg_output(PIN_SN74LV_INH);
66   nrf_gpio_pin_clear(PIN_SN74LV_INH); // set SPDT analog switches inhibit pin low to activate them
67 
68   setup_global_clock();
69   usb_init();
70   lpcomp_init();       //low power comparater setup
71   gpiote_setup();      // GPIO Task and events setup
72   timer_1_setup();     // timer 1 -
73   cs_spi_setup();
74   ppi_setup();
75   ls_init();
76   cs_init();
77   gpiote_enable_channels();
78 }
```

6 Class Documentation

6.1 commands Struct Reference

Public Attributes

- command_type_t **ctype**
- uint32_t **value**

The documentation for this struct was generated from the following file:

- C:/Users/sh03911/Documents/Work/Projects/PSA/Queen/V1/code/code/header/PSA_Q_1_USB.h

Appendix D

BoMI App Software

This appendix contains the GUI files

```xaml
<!-- MainWindow.xaml -->

<Window x:Class="BOMI_AI_GUI.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008" xmlns:port="clr-
namespace:System.IO.Ports;assembly=System"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes" mc:Ignorable="d"
Height="1050" Width="800" MaxHeight="1400" MinHeight="700" MaxWidth="1400" MinWidth="800"
ResizeMode="CanMinimize" BorderThickness="0" SnapsToDevicePixels="True">
  <Window.Resources>
    <ObjectDataProvider ObjectType="{x:Type port:SerialPort}" MethodName="GetPortNames"
x:Key="portNames" />
  </Window.Resources>
  <Border Padding="10">
    <Grid ShowGridLines="False">
      <materialDesign:DialogHost BorderBrush="{DynamicResource MaterialDesignDivider}">
        <materialDesign:DialogHost.DialogContent>
          <Grid Width="300" Height="150" HorizontalAlignment="Center">
            <StackPanel Orientation="Horizontal" Margin="15">
              <materialDesign:PackIcon Kind="Folder" Foreground="{StaticResource
PrimaryHueMidBrush}" Width="50" Height="50" />
              <TextBlock Foreground="Gray" Width="200" Margin="15 5" TextWrapping="Wrap">
</TextBlock>
            </StackPanel>
            <StackPanel Orientation="Horizontal" HorizontalAlignment="Right"
VerticalAlignment="Bottom" Margin="15">
              <Button Command="{x:Static materialDesign:DialogHost.CloseDialogCommand}"
Style="{DynamicResource MaterialDesignFlatButton}" Margin="4" VerticalAlignment="Center">
                                Deny
                              </Button>
              <Button Command="{x:Static materialDesign:DialogHost.CloseDialogCommand}"
Style="{DynamicResource MaterialDesignFlatButton}" Margin="4" VerticalAlignment="Center">
                                Allow
                              </Button>
            </StackPanel>
          </Grid>
        </materialDesign:DialogHost.DialogContent>

        <Grid>
          <Grid.ColumnDefinitions>
            <ColumnDefinition x:Name="Display_Grid" Width="400"></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
          </Grid.ColumnDefinitions>
          <StackPanel Grid.Row="0">
            <Canvas Grid.Column="0" x:Name="Canvas_Display" Focusable="true"
Grid.ColumnSpan="3" Width="380" Height="400" />


          </StackPanel>

          <StackPanel Grid.Row="0" Grid.Column="3" x:Name="sp">
            <Grid>
              <Grid.ColumnDefinitions>
                <ColumnDefinition Width="3*"></ColumnDefinition>
                <ColumnDefinition Width="*"></ColumnDefinition>
              </Grid.ColumnDefinitions>
              <TextBlock Text="Bomi AI Data Recording" FontSize="20" />
              <StackPanel Grid.Column="1" Orientation="Vertical">
                <Button x:Name="Connect" Height="30" Width="80" Content="Connect"
FontSize="12" Margin="0,0,0,0" HorizontalAlignment="Left" Click="ConnectClick" />
                <TextBlock x:Name="connection_status" Text="status" Margin="0,0,0,0" />
```

```
48              </StackPanel>
49            </Grid>
50
51            <Grid>
52              <Grid.ColumnDefinitions>
53                <ColumnDefinition Width="3*"></ColumnDefinition>
54                <ColumnDefinition Width="2*"></ColumnDefinition>
55              </Grid.ColumnDefinitions>
56
57              <TextBox Grid.Column="0" Text="output" TextWrapping="Wrap" x:Name="serialBox"
    Height="50" UseLayoutRounding="False" />
58              <ComboBox Grid.Column="1" Style="{StaticResource
    MaterialDesignFilledComboBox}" materialDesign:HintAssist.Hint="Select COM Port"
    x:Name="COM" ItemsSource="{Binding Source={StaticResource portNames}}"
    SelectionChanged="ComboBox_SelectionChanged" />
59            </Grid>
60
61            <StackPanel Orientation="Horizontal">
62              <ComboBox Style="{StaticResource MaterialDesignFilledComboBox}"
    materialDesign:HintAssist.Hint="Postures" x:Name="Shapes" ItemsSource="{Binding
    ShapeCollection}" SelectionChanged="Shapes_SelectionChanged" Width="180" SelectedIndex="-1"
    Cursor="Hand" IsEditable="True" HorizontalAlignment="Left" />
63            </StackPanel>
64
65            <Grid>
66              <Grid.ColumnDefinitions>
67                <ColumnDefinition Width="3*" />
68                <ColumnDefinition Width="2*" />
69                <ColumnDefinition Width="15*" />
70              </Grid.ColumnDefinitions>
71            </Grid>
72
73            <Grid>
74              <Grid.ColumnDefinitions>
75                <ColumnDefinition Width="5*" />
76                <ColumnDefinition Width="15*" />
77              </Grid.ColumnDefinitions>
78              <TextBox Grid.Column="0" Text="Detection %" TextWrapping="Wrap" Height="50"
    UseLayoutRounding="False" HorizontalAlignment="Left" VerticalAlignment="Center"
    VerticalContentAlignment="Center" />
79              <Slider Grid.Column="1" Minimum="0" Maximum="100" Style="{StaticResource
    MaterialDesignDiscreteSlider}" Value="5"
    ValueChanged="Accumulated_pressure_percentage_slider_changed" />
80            </Grid>
81
82            <Grid>
83              <Grid.ColumnDefinitions>
84                <ColumnDefinition Width="5*" />
85                <ColumnDefinition Width="15*" />
86              </Grid.ColumnDefinitions>
87              <TextBox Grid.Column="0" Text="Dynamic Range" TextWrapping="Wrap" Height="50"
    UseLayoutRounding="False" HorizontalAlignment="Left" VerticalAlignment="Center"
    VerticalContentAlignment="Center" />
88              <Slider Grid.Column="1" x:Name="Dynamic_resolution_slider" Minimum="1000"
    Maximum="10000" Style="{StaticResource MaterialDesignDiscreteSlider}" Value="7000"
    ValueChanged="Dynamic_range_silder_change" IsSnapToTickEnabled="True" TickFrequency="100"
    />
89            </Grid>
90
91            <Button Style="{StaticResource MaterialDesignFloatingActionMiniLightButton}"
    x:Name="object_detected_b" Margin="0,10" Background="#FFE2F2FF">
92              <materialDesign:PackIcon Kind="None" Height="24" Width="24" />
93            </Button>
94
```

```
            <Grid>
              <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
              </Grid.ColumnDefinitions>
              <Button Grid.Column="1" Height="30" Width="120" Content="Calibrate" FontSize="14" FontWeight="Bold" Click="Calibrate" Margin="0,10" />
              <Button Grid.Column="2" Height="30" Width="100" Content="Interpolate" FontSize="12" FontWeight="Bold" Click="Pressure_Array_Interpolation_Clicked" Margin="0,10" x:Name="Pressure_Array_Interpolation" />

            </Grid>
            <Button Style="{StaticResource MaterialDesignRaisedLightButton}" Height="30" Width="100" Content="Capture" FontSize="14" FontWeight="Bold" Click="Capture_Click" Foreground="#FF4C63AF" Margin="0,30,0,5" />
            <Button Style="{StaticResource MaterialDesignRaisedLightButton}" x:Name="continual_capture" Height="30" Width="180" Content="Contiual Capture off" FontSize="14" FontWeight="Bold" Foreground="#FF4C63AF" Click="Continual_Capture_Click" Margin="0,5,0,5" />
            <Button Style="{StaticResource MaterialDesignRaisedLightButton}" x:Name="Record_Image" Height="30" Width="150" Content="Record Image" FontSize="14" FontWeight="Bold" Foreground="#FF4C63AF" Click="Record_Image_Click" Margin="0,5,0,5" />
            <Button Style="{StaticResource MaterialDesignRaisedLightButton}" x:Name="continual_record" Height="30" Width="180" Content="Continual Record off" FontSize="14" FontWeight="Bold" Foreground="#FF4C63AF" Click="Continual_Record_Image_Click" Margin="0,5,0,5" />
            <Button Style="{StaticResource MaterialDesignRaisedLightButton}" x:Name="AI_Infer_Button" Height="30" Width="100" Content="AI Infer" FontSize="14" FontWeight="Bold" Foreground="#FF4C63AF" Click="Python_Code_click" Margin="0,5,0,5" />
            <TextBlock x:Name="AI_infer_results" Text="AI Results" HorizontalAlignment="Center" FontSize="32" />
          </StackPanel>
        </Grid>
      </materialDesign:DialogHost>
    </Grid>
  </Border>
</Window>
```

```
/*
-----------------------------------------------------------------------------------------
    MainWindow.xaml.cs
-----------------------------------------------------------------------------------------
*/

using System;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using System.IO.Ports;
using ZeroMQ;
using System.Diagnostics;

namespace BOMI_AI_GUI
{
    public partial class MainWindow : Window
    {
        const byte NUMBER_OF_ROWS = 41;
        const byte NUMBER_OF_COLUMNS = 16;
        private PressureDisplay display = new PressureDisplay(NUMBER_OF_ROWS,
NUMBER_OF_COLUMNS);
        private PressureDisplayImage display_image = new
PressureDisplayImage(NUMBER_OF_ROWS, NUMBER_OF_COLUMNS);
        private PressureImage pressure_image = new PressureImage(NUMBER_OF_ROWS,
NUMBER_OF_COLUMNS);
        private PressureImage pressure_image_calibration = new
PressureImage(NUMBER_OF_ROWS, NUMBER_OF_COLUMNS);
        private PressureImage pressure_image_normalized = new PressureImage(NUMBER_OF_ROWS,
NUMBER_OF_COLUMNS);

        private PressureImageMetaData PI_metadata = new PressureImageMetaData("unassigned",
"unassigned", "unassigned");

        private ComboBoxViewModel comboBoxViewModel = new ComboBoxViewModel();
        private int accumulated_pressure_percentage_value = 5;

        bool calibrate_b = true;
        bool continual_capture_b = false;
        bool continual_recording = false;
        static bool recieving_pressure_image = false;
        private byte pixel_noise_threshold = 0;
        int normalized_mean = 0;
        bool pixel_filter_b = false;


        SerialPort serialDevice = new SerialPort();

        public MainWindow()
        {
            InitializeComponent();
            DataContext = new ComboBoxViewModel();
            display.DrawDisplay(Canvas_Display, (int)Canvas_Display.Width);
        }
        string starting_token = "<p_image>";
        string ending_token = "</p_image>";
        string raw_pressure_image;
        private void OnDataReceived(object sender, SerialDataReceivedEventArgs e)
        {
            var serialDevice = sender as SerialPort;
            var buffer = new byte[serialDevice.BytesToRead];
```

```
57              serialDevice.Read(buffer, 0, buffer.Length);
58              string raw_data = System.Text.Encoding.Default.GetString(buffer);
59              if (!recieving_pressure_image)
60              {
61                  if (raw_data.Contains(starting_token.ToString()))
62                  {
63                      recieving_pressure_image = true;
64                      raw_pressure_image = "";
65                      raw_pressure_image +=
raw_data.Substring(raw_data.IndexOf(starting_token.ToString()) + starting_token.Length); //
get anything hiding behind the starting token
66                  }
67              }
68              else
69              {
70                  if (raw_data.Contains(ending_token.ToString()))
71                  {
72                      recieving_pressure_image = false;
73                      raw_pressure_image += raw_data.Substring(0,
raw_data.IndexOf(ending_token.ToString())); // get anything hiding behind the starting
token
74                      pressure_image.Update(raw_pressure_image);
75                      pressure_image.Mean = PressureImageAnalyser.Mean(pressure_image);
76                      if (calibrate_b)
77                      {
78                          pressure_image_calibration.Update(pressure_image);
79                      }
80                      display_image.Update(pressure_image, pressure_image_calibration);
81                      PressureImageAnalyser.Normalize(ref pressure_image_normalized,
pressure_image, pressure_image_calibration);
82                      pressure_image_normalized.Mean =
PressureImageAnalyser.Mean(pressure_image_normalized);
83                      if (calibrate_b)
84                      {
85                          normalized_mean = pressure_image_normalized.Mean;
86                          calibrate_b = false;
87                      }
88                      if (pixel_filter_b)
89                      {
90                          pixel_filter(display_image);
91                      }
92                      Application.Current.Dispatcher.Invoke(new Action(() =>
93                      {
94                          display.UpdateDisplayPixels(display_image);
95                          PressureImageFileSystem.RecordPressureImageJPEG(display_image,
PI_metadata, true);
96                          PythonCommunication();
97                      }));
98                      bool Item_detected = false;
99                      float mean_difference = (float)pressure_image_normalized.Mean /
(float)normalized_mean;
100                     Console.WriteLine("PI mean = "+ pressure_image_normalized.Mean + ",
cali mean = " + normalized_mean + ", mean diff = "+ mean_difference);
101                     if (mean_difference > (1.0f +
(float)accumulated_pressure_percentage_value/100))
102                     {
103                         Item_detected = true;
104                         Application.Current.Dispatcher.Invoke(new Action(() =>
105                         {
106                            object_detected_b.Background = new
SolidColorBrush(Color.FromArgb(0xFF, 0x48, 0xAD,0xFF));
107
108                         }));
```

```
                    }
                    else
                    {
                        Item_detected = false;
                        Application.Current.Dispatcher.Invoke(new Action(() =>
                        {
                            object_detected_b.Background = new
SolidColorBrush(Color.FromArgb(0xFF, 0xE2, 0xF2, 0xFF));
                        }));
                    }
                    if (continual_capture_b)    //resend the message
                    {
                        if (continual_recording)
                        {
                            PressureImageFileSystem.RecordPressureImageJPEG(display_image,
PI_metadata);
                        }
                        try
                        {
                            serialDevice.Write("read \n\r");
                        }
                        catch (Exception ex)
                        {
                            MessageBox.Show("Error opening/writing to serial port :: " +
ex.Message, "Error!");
                        }
                    }
                }
                else
                {
                    raw_pressure_image += raw_data;
                }
            }
        }

        private void ConnectClick(object sender, RoutedEventArgs e)
        {
            try
            {
                string portName = COM.SelectedItem as string;
                if (portName == null)
                {
                    portName = "COM18";
                    COM.SelectedIndex = COM.Items.IndexOf("COM18");
                }
                serialDevice.PortName = portName;
                serialDevice.BaudRate = 2000000;
                serialDevice.DataBits = 8;
                serialDevice.StopBits = StopBits.One;
                serialDevice.Parity = Parity.None;
                serialDevice.ReadBufferSize = 100000;
                serialDevice.DataReceived += new
SerialDataReceivedEventHandler(OnDataReceived);
                serialDevice.Open();
                serialDevice.DtrEnable = true;
                if (serialDevice.IsOpen)
                {
                    connection_status.Text = "Connected";
                    try
                    {
                        if (!(serialDevice.IsOpen)) serialDevice.Open();
                    }
```

```
                    catch (Exception ex)
                    {
                        MessageBox.Show("Error opening/writing to serial port :: " +
ex.Message, "Error!");
                    }
                }


            }
            catch (Exception err)
            {
                MessageBox.Show(err.Message, "test", MessageBoxButton.OK);
            }
        }

        private void Continual_Capture_Click(object sender, RoutedEventArgs e)
        {
            if (continual_capture_b)
            {
                continual_capture_b = false;
                Application.Current.Dispatcher.Invoke(new Action(() =>
                {
                    continual_capture.Content = "Continual Capture off";

                }));
            }
            else
            {
                continual_capture_b = true;
                try
                {
                    if (!(serialDevice.IsOpen)) serialDevice.Open();
                    serialDevice.Write("read \n\r");
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Error opening/writing to serial port :: " +
ex.Message, "Error!");
                }

                Application.Current.Dispatcher.Invoke(new Action(() =>
                {
                    continual_capture.Content = "Continual Capture on";
                }));
            }
        }

        private void Capture_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                if (!(serialDevice.IsOpen)) serialDevice.Open();

                serialDevice.Write("read \n\r");

            }
            catch (Exception ex)
            {
                MessageBox.Show("Error opening/writing to serial port :: " + ex.Message,
"Error!");
            }
        }
```

```
226
227         private void ComboBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
228         {
229             var selectedComboItem = sender as ComboBox;
230             string name = selectedComboItem.SelectedItem as string;
231         }
232
233         private void Size_SelectionChanged(object sender, SelectionChangedEventArgs e)
234         {
235             var selected_size = sender as ComboBox;
236             if (selected_size.SelectedItem != null)
237             {
238                 Console.WriteLine("size selected = " +
    selected_size.SelectedItem.ToString());
239                 PI_metadata.Size = selected_size.SelectedItem.ToString();
240             }
241         }
242
243         private void Shapes_SelectionChanged(object sender, SelectionChangedEventArgs e)
244         {
245             var selectedShape = sender as ComboBox;
246             if (selectedShape.SelectedItem.ToString().Contains("None"))
247             {
248                 PI_metadata.Shape = "None";
249             }
250             if (selectedShape.SelectedItem.ToString().Contains("Circle")) {
251                 PI_metadata.Shape = "Circle";
252             }
253             if (selectedShape.SelectedItem.ToString().Contains("Square"))
254             {
255                 PI_metadata.Shape = "Square";
256             }
257             if (selectedShape.SelectedItem.ToString().Contains("Triangle"))
258             {
259                 PI_metadata.Shape = "Triangle";
260             }
261             if (selectedShape.SelectedItem.ToString().Contains("RHS"))
262             {
263                 PI_metadata.Shape = "RHS";
264             }
265             if (selectedShape.SelectedItem.ToString().Contains("LHS"))
266             {
267                 PI_metadata.Shape = "LHS";
268
269             }
270             if (selectedShape.SelectedItem.ToString().Contains("Back"))
271             {
272                 PI_metadata.Shape = "Back";
273             }
274
275         }
276         private void Record_Image_Click(object sender, RoutedEventArgs e)
277         {
278             PressureImageFileSystem.RecordPressureImageJPEG(display_image, PI_metadata);
279         }
280         private void Continual_Record_Image_Click(object sender, RoutedEventArgs e)
281         {
282             if(continual_recording)
283             {
284                 continual_recording = false;
285                 Application.Current.Dispatcher.Invoke(new Action(() =>
```

```
286                 {
287                     continual_record.Content = "Continual Record Off";
288 
289                 }));
290             }
291             else
292             {
293                 continual_recording = true;
294                 Application.Current.Dispatcher.Invoke(new Action(() =>
295                 {
296                     continual_record.Content = "Continual Record On";
297                 }));
298             }
299         }
300 
301         private void Calibrate(object sender, RoutedEventArgs e)
302         {
303             calibrate_b = true;
304         }
305 
306         private void Accumulated_pressure_percentage_slider_changed(object sender,
    RoutedPropertyChangedEventArgs<double> e)
307         {
308             accumulated_pressure_percentage_value = (int)e.NewValue;
309         }
310 
311 
312         private void pixel_filter(PressureDisplayImage pi)
313         {
314 
315             for (int i = 0; i < pi.Rows; i++)   //loop through rows
316             {
317                 for (int j = 0; j < pi.Columns; j++)       //loop through columns
318                 {
319                     if (pi.Image[i,j] > pixel_noise_threshold)
320                     {
321                         if (i>0 && i < pi.Rows - 1)
322                         {
323                             if ( j>0 && j < pi.Columns - 1)
324                             {
325                                 if (pi.Image[i+1, j+1]> pixel_noise_threshold) break;
326                                 else if (pi.Image[i + 1, j-1] > pixel_noise_threshold)
    break;
327                                 else if (pi.Image[i-1, j + 1] > pixel_noise_threshold)
    break;
328                                 else if (pi.Image[i-1, j - 1] > pixel_noise_threshold)
    break;
329                                 pi.Image[i, j] = pi.Image[i + 1, j];
330                             }
331                         }
332                     }
333                 }
334             }
335         }
336 
337         private void Pixel_filter_threshold_value_changed(object sender,
    RoutedPropertyChangedEventArgs<double> e)
338         {
339             pixel_noise_threshold = (byte)e.NewValue;
340         }
341 
342         private void Pixel_filter_toggle_checked(object sender, RoutedEventArgs e)
```

```
{pixel_filter_b = true;}
        private void Pixel_filter_toggle_unchecked(object sender, RoutedEventArgs e)
{pixel_filter_b = false;}

        private bool AI_infer_b = false;
        private void Python_Code_click(object sender, RoutedEventArgs e)
        {
            if (AI_Infer_Button.Content.Equals("AI Infer ON")){
                AI_infer_b = false;
                AI_Infer_Button.Content = "AI Infer";
            }
            else
            {
                AI_infer_b = true;
                AI_Infer_Button.Content = "AI Infer ON";
            }
            AI_infer();
        }

        private async void AI_infer()
            {
            var errors = "";
            var results = "";
            try
                {
                    await Task.Run(() =>
                    {
                        var psi = new ProcessStartInfo();
                        psi.FileName =
"C:/Users/sh03911/Anaconda3/envs/tensorflow23/python.exe";

                        var script =
@"C:/Users/sh03911/Documents/Projects/PycharmProjects/shapes_inferrence/main.py";
                        var tf_image_location =
"C:/Users/sh03911/Documents/Projects/MachineLearing/Training_Data/16x16_test_data/img.bmp";
                        psi.Arguments = $"\"{script}\" -i";
                        psi.UseShellExecute = false;
                        psi.CreateNoWindow = true;
                        psi.RedirectStandardInput = true;
                        psi.RedirectStandardOutput = true;
                        psi.RedirectStandardError = true;

                        using (var process = Process.Start(psi))
                        {
                            errors = "";
                            results = "";
                            PythonCommunication();
                        }
                    });
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
        }

        void PythonCommunication()
        {
            if (AI_infer_b)
            {
                using (var requester = new ZSocket(ZSocketType.REQ))
                {
```

```
                    // Connect
                    requester.Connect("tcp://127.0.0.1:5552");

                    string requestText = "A";
                    Console.Write("Sending {0}...", requestText);

                    // Send
                    requester.Send(new ZFrame(requestText));

                    // Receive
                    using (ZFrame reply = requester.ReceiveFrame())
                    {
                        string input_data = reply.ReadString();
                        Console.WriteLine(" Received: {0} {1}!", requestText, input_data);
                        Application.Current.Dispatcher.Invoke(new Action(() => {
AI_infer_results.Text = input_data; }));
                    }
                }
            }
        }

        int Dynamic_resolution_scaler;
        private void Dynamic_range_silder_change(object sender,
RoutedPropertyChangedEventArgs<double> e)
        {
            display_image.Dynamic_Resolution_Scaler = (int)e.NewValue;
            if (display_image.Image[0, 0] != 0)
            {
                display_image.Update(pressure_image, pressure_image_calibration);
                display.UpdateDisplayPixels(display_image);
            }
        }

        private void Pressure_Array_Interpolation_Clicked(object sender, RoutedEventArgs e)
        {
            if (Pressure_Array_Interpolation.Content.ToString().Equals("Interpolate"))
            {
                Pressure_Array_Interpolation.Content = "Origional";
                display.BilinearInterpolation = true;
                display.AlterDisplay((UInt16)(display_image.Rows * 2 - 1), (UInt16)
(display_image.Columns * 2 - 1));
                display.DrawDisplay(Canvas_Display, (int)Canvas_Display.Width);

            }
            else if (Pressure_Array_Interpolation.Content.ToString().Equals("Origional"))
            {
                Pressure_Array_Interpolation.Content = "Interpolate";
                display.BilinearInterpolation = false;
                display.AlterDisplay((UInt16)(display_image.Rows), (UInt16)
(display_image.Columns));
                display.DrawDisplay(Canvas_Display, (int)Canvas_Display.Width);

            }
        }
    }
}
```

```
/*
-----------------------------------------------------------------------------------------
    PressureDisplay.cs
-----------------------------------------------------------------------------------------
*/

using System;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Shapes;

class PressureDisplay
{
    private Canvas m_my_canvas = null;          // reference to canvas that this display is drawn on.

    private bool m_display_values = false;      //overlay pixel values on pixels
    public bool BilinearInterpolation = false;

    private int m_number_of_rows;
    private int m_number_of_columns;

    private int m_display_element_pixel_size;
    public enum rotation : byte
    {
        rotation0,
        rotation90,
        rotation180,
        rotation270
    }
    rotation screen_rotation;

    public Rectangle[,] pressure_pixels;
    public TextBlock[,] pressure_pixels_values;


    public PressureDisplay(UInt16 number_of_rows, UInt16 number_of_columns, bool display_values = false)
    {

        AlterDisplay(number_of_rows, number_of_columns, display_values);
    }

    public void AlterDisplay(UInt16 number_of_rows, UInt16 number_of_columns, bool display_values = false)
    {
        pressure_pixels = new Rectangle[number_of_rows, number_of_columns];
        pressure_pixels_values = new TextBlock[number_of_rows, number_of_columns];

        m_number_of_columns = number_of_columns;
        m_number_of_rows = number_of_rows;


        screen_rotation = rotation.rotation0;
    }

    public void DrawDisplay(Canvas MyCanvas, int canvas_grid_width)
    {
        m_my_canvas = MyCanvas;

```

```
59          m_display_element_pixel_size = canvas_grid_width / m_number_of_columns;
60
61          for (int i = 0; i < pressure_pixels.GetLength(0); i++)   //loop through rows
62          {
63              for (int j = 0; j < pressure_pixels.GetLength(1); j++)       //loop through
columns
64              {
65                  Color color = HSVToRGB(255, 1, 1);
66                  pressure_pixels[i, j] = new Rectangle
67                  {
68                      Height = m_display_element_pixel_size,
69                      Width = m_display_element_pixel_size,
70                      Fill = new SolidColorBrush(color)
71                  };
72                  Canvas.SetLeft(pressure_pixels[i, j], j * (m_display_element_pixel_size));
73                  Canvas.SetTop(pressure_pixels[i, j], i * (m_display_element_pixel_size));
74                  m_my_canvas.Children.Add(pressure_pixels[i, j]);
75                  if (m_display_values)
76                  {
77                      pressure_pixels_values[i, j] = new TextBlock
78                      {
79                          Text = "0",
80                          FontSize = 8,
81                          Foreground = Brushes.White,
82                          Background = new SolidColorBrush(Color.FromArgb(128, 0, 0, 0))
83                      };
84                      Canvas.SetLeft(pressure_pixels_values[i, j], j *
(m_display_element_pixel_size));
85                      Canvas.SetTop(pressure_pixels_values[i, j], i *
(m_display_element_pixel_size));
86                      m_my_canvas.Children.Add(pressure_pixels_values[i, j]);
87                  }
88              }
89          }
90      }
91      public void UpdateDisplayLabels(in UInt32[] values, int lenght)
92      {
93          if (m_display_values)
94          {
95              if (pressure_pixels.GetLength(1) * pressure_pixels.GetLength(0) != lenght)
return;
96              int k = 0;
97              for (int i = 0; i < pressure_pixels.GetLength(0); i++)   //loop through rows
98              {
99                  for (int j = 0; j < pressure_pixels.GetLength(1); j++)       //loop through
columns
100                 {
101                     pressure_pixels_values[i, j].Text = k.ToString() + ")\n" +
values[k].ToString();
102                     k++;
103
104                 }
105             }
106         }
107     }
108     public void UpdateDisplayLabels(in PressureDisplayImage displayImage)
109     {
110         if (m_display_values)
111         {
112             int k = 0;
113             for (int i = 0; i < pressure_pixels.GetLength(0); i++)   //loop through rows
114             {
```

```
                for (int j = 0; j < pressure_pixels.GetLength(1); j++)       //loop through columns
                {
                    pressure_pixels_values[i, j].Text = k.ToString() + ")\n" + displayImage.Image[i, j].ToString();
                    k++;
                }
            }
        }
    }
    public void UpdateDisplayPixels(in Byte[] new_data, int lenght)
    {
        if (pressure_pixels.GetLength(0) * pressure_pixels.GetLength(1) != lenght)
        {
            Console.WriteLine("Pressure pixel lengh and this data's lenght is not the same.");
            return;
        }
        int k = 0;
        for (int i = 0; i < pressure_pixels.GetLength(0); i++)   //loop through rows
        {
            for (int j = 0; j < pressure_pixels.GetLength(1); j++)       //loop through columns
            {
                pressure_pixels[i, j].Fill = new SolidColorBrush(HSVToRGB(255 - new_data[k], 1, 1));
                k++;
            }
        }
    }
    public void UpdateDisplayPixels(in PressureDisplayImage displayImage)
    {
        if (BilinearInterpolation)
        {
            byte[,] new_image = new byte[displayImage.Rows * 2 - 1, displayImage.Columns * 2 - 1];

            for (int r = 0; r < new_image.GetLength(0); r++)   //loop through rows
            {
                for (int c = 0; c < new_image.GetLength(1); c++)       //loop through columns
                {
                    if (r % 2 == 0)
                    {
                        if (c % 2 == 0) new_image[r, c] = displayImage.Image[r / 2, c / 2];
                        else new_image[r, c] = (byte)((displayImage.Image[r / 2, c / 2] + displayImage.Image[r / 2, c / 2 + 1]) / 2);
                    }
                    else
                    {
                        if (c % 2 == 0) new_image[r, c] = (byte)((displayImage.Image[r / 2, c / 2] + displayImage.Image[r / 2 + 1, c / 2]) / 2);
                        else new_image[r, c] = (byte)((displayImage.Image[r / 2, c / 2] + displayImage.Image[r / 2, c / 2 + 1] + displayImage.Image[r / 2 + 1, c / 2] + displayImage.Image[r / 2 + 1, c / 2 + 1]) / 4);
                    }
                }
            }

            for (int i = 0; i < new_image.GetLength(0); i++)   //loop through rows
            {
                for (int j = 0; j < new_image.GetLength(1); j++)       //loop through columns
```

```
                {

                    pressure_pixels[i, j].Fill = new SolidColorBrush(HSVToRGB(255 - new_image[i, j], 1, 1));
                }
            }
        }
        else
        {
            for (int i = 0; i < pressure_pixels.GetLength(0); i++)   //loop through rows
            {
                for (int j = 0; j < pressure_pixels.GetLength(1); j++)       //loop through columns
                {
                    pressure_pixels[i, j].Fill = new SolidColorBrush(HSVToRGB(255 - displayImage.Image[i, j], 1, 1));
                }
            }
        }
    }

    public static Color HSVToRGB(double H, double S, double V)
    {
        double r = 0, g = 0, b = 0;

        if (S == 0)
        {
            r = V;
            g = V;
            b = V;
        }
        else
        {
            int i;
            double f, p, q, t;

            if (H == 360)
                H = 0;
            else
                H = H / 60;

            i = (int)Math.Truncate(H);
            f = H - i;

            p = V * (1.0 - S);
            q = V * (1.0 - (S * f));
            t = V * (1.0 - (S * (1.0 - f)));

            switch (i)
            {
                case 0:
                    r = V;
                    g = t;
                    b = p;
                    break;

                case 1:
                    r = q;
                    g = V;
                    b = p;
                    break;

```

```
                case 2:
                    r = p;
                    g = V;
                    b = t;
                    break;

                case 3:
                    r = p;
                    g = q;
                    b = V;
                    break;

                case 4:
                    r = t;
                    g = p;
                    b = V;
                    break;

                default:
                    r = V;
                    g = p;
                    b = q;
                    break;
            }

        }
        return Color.FromRgb((byte)(r * 255), (byte)(g * 255), (byte)(b * 255));
    }
}
```

```
/*
-----------------------------------------------------------------------------------------
    PressureDisplayImage.cs
-----------------------------------------------------------------------------------------
*/
using System;

public class PressureDisplayImage
{
    private byte[,] _displayImage;
    private byte _rows;
    private byte _columns;
    private int _dynamic_resolution_scaler;

    public byte[,] Image { get { return _displayImage; } }
    public byte Rows { get { return _rows; } }
    public byte Columns { get { return _columns; } }
    public int Dynamic_Resolution_Scaler { get { return _dynamic_resolution_scaler; } set { _dynamic_resolution_scaler = value; } }

    public PressureDisplayImage(byte NumberOfRows, byte NumberOfColumns)
    {
        _rows = NumberOfRows;
        _columns = NumberOfColumns;
        _displayImage = new byte[_rows, NumberOfColumns];
        _dynamic_resolution_scaler = 7000;
    }
    public void Update(in PressureImage pressureImage, in PressureImage clibrationImage)
    {
        _rows = pressureImage.Rows;
        _columns = pressureImage.Columns;
        _displayImage = new byte[Rows, Columns];
        for (int i = 0; i < _rows; i++)   //loop through rows
        {
            for (int j = 0; j < _columns; j++)       //loop through columns
            {
                int temp =0 ;
                temp = (int)(pressureImage.Image[i, j] - (clibrationImage.Image[i,j]-6000));
                if (temp < 1) temp = 1;

                temp = temp / Dynamic_Resolution_Scaler;
                if (temp < 1) temp = 1;
                if (temp > 254) temp = 254;
                _displayImage[i, j] = (byte)temp;

            }
        }
    }
    public void Print()
    {
        for (int i = 0; i < _displayImage.GetLength(0); i++)   //loop through rows
        {
            for (int j = 0; j < _displayImage.GetLength(1); j++)       //loop through columns
            {
                Console.WriteLine(i.ToString() + "," + j.ToString() + "  \t|" + _displayImage[i, j].ToString());
            }
        }
    }
```

```
58 }
```

```
/*
-------------------------------------------------------------------------------------------
    PressureImage.cs
-------------------------------------------------------------------------------------------
*/
using System;

public class PressureImage
{
    private uint[,] _image;
    public uint[,] Image { get { return _image; } }

    private byte _rows;
    private byte _columns;
    private int _mean;
    public int Mean;
    public byte Rows{ get { return _rows; }  }
    public byte Columns { get { return _columns; } }


    public PressureImage(byte NumberOfRows, byte NumberOfColumns)
    {
        _rows = NumberOfRows;
        _columns = NumberOfColumns;
        _image = new uint[_rows, NumberOfColumns];
    }
    public PressureImage(byte NumberOfRows, byte NumberOfColumns, String RawSerialString)
    {
        _rows = NumberOfRows;
        _columns = NumberOfColumns;
        _image = new uint[_rows, NumberOfColumns];


        Update(RawSerialString);

    }
    public PressureImage(in PressureImage pressureImage)    // deep copy
    {
        _rows = pressureImage.Rows;
        _columns = pressureImage.Columns;
        _image = new uint[_rows, _columns];
        for (int i = 0; i < pressureImage._image.GetLength(1); i++)   //loop through rows
        {
            for (int j = 0; j < pressureImage._image.GetLength(0); j++)       //loop
through columns
            {
                _image[i,j] = pressureImage._image[i, j];
            }
        }
    }
    public void Update(String RawSerialString)
    {
        string[] elements = RawSerialString.Split(',');
        foreach (var element in elements)
        {
            string[] numbers = element.Split('|');
            if (numbers.Length != 2)
            {
                Console.WriteLine("error in element splitting of rawserial string");
            }
            else
```

```
            {
                uint index = uint.Parse(numbers[0]);
                uint value = uint.Parse(numbers[1]);
                _image[index / Columns, index % Columns] = value;

            }
        }
    }
    public void Update(in PressureImage pressureImage)
    {
        if (_rows != pressureImage.Rows) return;
        if (_columns != pressureImage.Columns) return;
        for (int i = 0; i < pressureImage._image.GetLength(0); i++)   //loop through rows
        {
            for (int j = 0; j < pressureImage._image.GetLength(1); j++)       //loop through columns
            {
                _image[i, j] = pressureImage._image[i, j];
            }
        }
        Mean = pressureImage.Mean;
    }

    public void Print()
    {
        for (int i = 0; i < _image.GetLength(0); i++)   //loop through rows
        {
            for (int j = 0; j < _image.GetLength(1); j++)       //loop through columns
            {
                Console.WriteLine(i.ToString() + "," + j.ToString() + "  \t|" + _image[i, j].ToString());
            }
        }
    }
}
public class PressureImageMetaData
{
    string _shape;
    string _size;
    string _weight;

    public String Shape { get { return _shape; } set { _shape = value; } }
    public String Size { get { return _size; } set { _size = value; } }
    public String Weight { get { return _weight; } set { Weight = value; } }
    public PressureImageMetaData()
    {

    }
    public PressureImageMetaData(string shape, string size, string weight)
    {
        _shape = shape;
        _size = size;
        _weight = weight;
    }
}
```

```
/*
-----------------------------------------------------------------------------------------
    PressureImageAnalyser.cs
-----------------------------------------------------------------------------------------
*/
using System;

public class PressureImageAnalyser
{
    static uint offset = 30000;
    public PressureImageAnalyser()
    {

    }
    static public void Normalize(ref PressureImage newImage, in PressureImage image1, in
PressureImage Image2 )
    {

        for (int i = 0; i < newImage.Rows; i++)   //loop through rows
        {
            for (int j = 0; j < newImage.Columns; j++)       //loop through columns
            {
                int temp = (int)(image1.Image[i, j] - (Image2.Image[i, j] - offset));
                if (temp < 0) temp = 0;
                newImage.Image[i, j] = (uint)temp;
            }
        }
    }
    static public int Mean(in PressureImage image)
    {
        UInt64 SumofElements = 0;
        UInt32 numberOfElements = 0;
        foreach (var element in image.Image)
        {
            SumofElements += element;
            numberOfElements++;
        }

        return (int)(SumofElements / numberOfElements);

    }

}
```

```
/*
-----------------------------------------------------------------------------------------
    PressureImageFileSystem.cs
-----------------------------------------------------------------------------------------
*/
using System.Drawing;

using System.Runtime.Remoting.Contexts;
using System.Runtime.CompilerServices;
using System.Text.RegularExpressions;

public class PressureImageFileSystem
{
    public PressureImageFileSystem()
    {

    }
    static string previous_shape = "";
    static int file_index_num = 0;
    static string File_Root_Training_Data = "C:/Userlocation/Projects/MachineLearing/Training_Data/41x16";
    public static void RecordPressureImage(PressureImage PI, PressureImageMetaData PIMD)
    {

        if (PIMD.Size.Equals("unassigned") || PIMD.Shape.Equals("unassigned")) return;
        string path;
        if (!(PIMD.Shape + "/" + PIMD.Size).Equals(previous_shape)) {
            previous_shape = PIMD.Shape + "/" + PIMD.Size;
            file_index_num = 0;
        }
        if (PIMD.Shape.Equals("None")) path = System.IO.Path.Combine(File_Root_Training_Data, PIMD.Shape);
        else path = System.IO.Path.Combine(File_Root_Training_Data, PIMD.Shape + "/" + PIMD.Size);
        if (!System.IO.Directory.Exists(path))
        {
            System.IO.Directory.CreateDirectory(path);
        }


        string file_name = PIMD.Shape + "_" + PIMD.Size + "_" + file_index_num.ToString() + ".txt";
        string file_path = System.IO.Path.Combine(path, file_name);
        while (System.IO.File.Exists(file_path))
        {
            file_index_num++;
            file_name = PIMD.Shape + "_" + PIMD.Size + "_" + file_index_num.ToString() + ".txt";
            file_path = System.IO.Path.Combine(path, file_name);

        }
        string P_Image = "Shape = " + PIMD.Shape + "\nSize = " + PIMD.Size + "\nImage = ";
        foreach (var element in PI.Image)
        {
            P_Image += element.ToString() + ",";
        }
        System.IO.File.WriteAllText(file_path, P_Image);
    }
    public static void RecordPressureImageJPEG(PressureDisplayImage PI, PressureImageMetaData PIMD, bool For_AI = false)
    {
        if (For_AI)
```

```
        {
            Bitmap bmp = new Bitmap(16, 41);
            for (int x = 0; x < bmp.Width; x++)
            {
                for (int y = 0; y < bmp.Height; y++)
                {
                    System.Drawing.Color newColor = System.Drawing.Color.FromArgb(PI.Image[y, x], 0, 0);
                    bmp.SetPixel(x, y, newColor); // Now greyscale
                }
            }
            bmp.Save("img.bmp", System.Drawing.Imaging.ImageFormat.Bmp);
        }
        else
        {
            if (PIMD.Size.Equals("unassigned") || PIMD.Shape.Equals("unassigned")) return;
            string path = System.IO.Path.Combine(File_Root_Training_Data, PIMD.Shape + "/" + PIMD.Size);
            if (!System.IO.Directory.Exists(path))
            {
                System.IO.Directory.CreateDirectory(path);
            }
            int counter = 0;

            string file_name = PIMD.Shape + "_" + PIMD.Size + "_" + counter.ToString() + ".bmp";
            string file_path = System.IO.Path.Combine(path, file_name);
            while (System.IO.File.Exists(file_path))
            {
                counter++;
                file_name = PIMD.Shape + "_" + PIMD.Size + "_" + counter.ToString() + ".bmp";
                file_path = System.IO.Path.Combine(path, file_name);
            }

            Bitmap bmp = new Bitmap(16, 41);
            for (int x = 0; x < bmp.Width; x++)
            {
                for (int y = 0; y < bmp.Height; y++)
                {

                    System.Drawing.Color newColor = System.Drawing.Color.FromArgb(PI.Image[y, x], 0, 0);
                    bmp.SetPixel(x, y, newColor); // Now greyscale
                }
            }
            bmp.Save(file_path, System.Drawing.Imaging.ImageFormat.Bmp);
        }
    }
}
```

```
/*
-----------------------------------------------------------------------------------------
    ComboBoxViewModel.cs
-----------------------------------------------------------------------------------------
*/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Media;


namespace BOMI_AI_GUI
{
    public class ComboBoxViewModel
    {
        public List<string> ShapeCollection { get; set; }

        public ComboBoxViewModel()
        {
            ShapeCollection = new List<string>()
            {
                "None",
                "RHS",
                "LHS",
                "Back"
            };
        }
    }
}
```

Appendix E

Artificial Neural Network Keras Code

This portion of the appendix contains the python code for training the network and getting inferences. The first two pages are for Training a model. The page after that contains the code for making inferences and sending it as message via a zmq socket.

```python
import os
import sys
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

#To make sure tensor flow gpu is working properly
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
config = tf.compat.v1.ConfigProto(gpu_options = tf.compat.v1.GPUOptions(per_process_gpu_memory_fraction=0.8))
config.gpu_options.allow_growth = True
session = tf.compat.v1.Session(config=config)
tf.compat.v1.keras.backend.set_session(session)

#Global variables
image_height = 41
image_width = 16
batch_size = 46     #training samples in one iteration

training_data_file_location = 'File_Location_of_training_set '

# method for converting pixel values to a float normalised between 0-1
def process(image, label):
    image = tf.cast(image/255.0, tf.float32)
    return image, label


# Training Dataset
ds_train = tf.keras.preprocessing.image_dataset_from_directory(
    training_data_file_location,
    labels='inferred',   # Labels generated from directory structure
    label_mode="int",    # The labels are encoded as integers
    color_mode='rgb',
    batch_size=batch_size,
    image_size=(image_height, image_width),
    shuffle=True,
    seed=5,
    validation_split=0.25, # Use 75% of the data set for training
    subset="training"      # This is the traing dataset.
)

# Validation Dataset
ds_validation = tf.keras.preprocessing.image_dataset_from_directory(
    training_data_file_location,
    labels='inferred',    # Labels generated the directory
```

```
44 structure.
45     label_mode="int",      # The labels are encoded as integers.
46     color_mode='rgb',
47     batch_size=batch_size,
48     image_size=(image_height, image_width),
49     shuffle=True,
50     seed=20,
51     validation_split=0.25, # use 25% of the data set for
   validation.
52     subset="validation",   # This is the validation dataset.
53 )
54
55 #remap pixel values to 0-1
56 ds_train = ds_train.map(process)
57 ds_validation = ds_validation.map(process)
58
59 #setup model parameters
60 model = keras.Sequential([
61     layers.Input((41, 16, 3), name="input"),
62     layers.Flatten(),
63     layers.Dense(64, activation='relu'),
64     layers.Dense(8, activation='relu'),
65     layers.Dense(4, activation='softmax')
66 ])
67
68 #model compilation
69 model.compile(
70     loss='sparse_categorical_crossentropy',
71     optimizer=keras.optimizers.Adam(lr=1e-4),
72     metrics=["accuracy"]
73 )
74
75 # train the model
76 model.fit(ds_train, epochs=15, batch_size=10, verbose=2)
77 model.evaluate(ds_validation, verbose=2)
78 model.save('model_location.h5')
79
80 sys.exit()
81
```

```python
import os
import tensorflow as tf
import numpy as np
import zmq

from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

config = tf.compat.v1.ConfigProto(gpu_options =
         tf.compat.v1.GPUOptions(
             per_process_gpu_memory_fraction=0.8))
config.gpu_options.allow_growth = True
session = tf.compat.v1.Session(config=config)
tf.compat.v1.keras.backend.set_session(session)

class_names = ['Back', 'LHS', 'None', 'RHS']

#open a ZMQ Socket
context = zmq.Context()
socket = context.socket(zmq.REP)
socket.bind("tcp://*:5552")

# load the pretrained model
model = load_model('model_location.h5')

while True:
    # Wait for next request from client
    message = socket.recv()

    # Load in or reload the image to be inferred
    testImage = load_img('path_to_image.bmp', color_mode='rgb')

    # convert image to numpy array
    img_array = img_to_array(testImage)
    img_array = tf.cast(img_array/255.0, tf.float32)
    img_batch = np.expand_dims(img_array, axis=0)

    # make an inference on the image
    result = model.predict(img_batch, batch_size=0, verbose=0)

    # get name of the prediction
    posture = class_names[np.argmax(result)].encode()

    # send the inferred shape as a string through the socket
    socket.send(posture)
```