Czech Technical University
Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's thesis

Activity analysis from smart bed strain gauge data

Nguyen Diem Huong

Supervisor: Ing. Macaš Martin, Ph.D

Study Programme: Open Informatics
Field of Study: Computer and Informatic Science

May 2019

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Nguyen Diem Huong** Personal ID number: **456150**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Branch of study: **Computer and Information Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Activity Analysis from Smart Bed Strain Gauge Data

Bachelor's thesis title in Czech:

Analýza aktivit z tenzometrických dat chytrého lůžka

Guidelines:

1. Define typical classes of activities that can be performed by patient in a hospital bed.
2. Propose a pattern recognition system for activity detection and implement it.
3. Analyze experimentally which activities can be successfully detected and which cannot.
4. Implement a demonstrative example of online detection of selected activities including a proper visualization.

Bibliography / sources:

[1] Manuál k nemocničnímu lůžku LINET.
[2] Sánchez, D., Tentori, M., & Favela, J. (2008). Activity recognition for the smart hospital. IEEE intelligent systems, 23(2).
[3] Wong, S., & Tan, C. S. (2013). Smart hospital bed.
[4] Nguyen, H. H. (2016). Advanced assistive control strategies for smart hospital beds (Doctoral dissertation).

Name and workplace of bachelor's thesis supervisor:

Ing. Martin Macaš, Ph.D., Cognitive Neurosciences, CIIRC

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **11.01.2019** Deadline for bachelor thesis submission: **24.05.2019**

Assignment valid until: **30.09.2020**

Ing. Martin Macaš, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. | Date of assignment receipt

Student's signature

Acknowledgement

My deepest gratitude belongs to my mother, a person who has never given up no matter what circumstances to achieve better life for me and my family. I am also very much obliged to everyone who has participated in the project and mainly my supervisor Ing. Macaš Martin, Ph.D. for giving me a chance to contribute to the research and for his continuous support.

My sincere gratitude also belongs to Patrik Kopecký who has always encouraged me through my studies and took his time to proofread my first attempt on a serious official project.

Author statement

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date ....

Abstrakt

Užití automatizace a dat je nedílnou součástí v současném průmyslu, včetně toho zdravotnického. Vzrůstající popularita umělé inteligence umožnila vytváření pomocných nástrojů, které zvyšují kvalitu diagnóz a péče o pacienty. Jedno z využívaných „chytrých“ zařízení je nemocniční lůžko poskytující data o pacientovi pro vyhodnocování různých statistik. Tato práce se zaměřuje na vizualizaci a detekci poloh v reálném čase použitím čtyř tenzometrů vestavěných v konstrukci postele. Pro tyto účely byl vytvořen vlastní software na extrakci a zpracování dat. Experiment pro detekci poloh s modelem algoritmu SVM ukázal uspokojivé výsledky i při učení klasifikačního modelu pouze na jednom subjektu. Model byl schopen v reálném čase rozpoznat polohy čtyř cizích subjektů různých vah a konstitucí. Experimenty byly uskutečněny v laboratoři v CIIRCu a jejich průběh byl zaznamenán na video přiložené k práci.

Klíčová slova: inteligentní lůžko, online vizualizace dat, detekce aktivit, online detekce poloh

Abstract

Use of automation and data appears in most industries and branches including healthcare. The rising popularity of AI paved the way for creation of tools that help improving the quality of diagnosis and care. One of these "smart" gadgets is a hospital bed that provides data for evaluation of the patient's statistics. This thesis focuses on real-time visualization and posture detection using four strain gauges built within the bed's construction. For this purpose, it was necessary to implement a respective data processing software for data extraction. A conducted experiment with the SVM-trained model showed that despite being trained on only one subject, the model was able to sufficiently detect postures of four foreign subjects of different weights and constructions. All the experiments were held and recorded in the laboratory of CIIRC and corresponding demonstrative video can be found on CD attached to this thesis.

Keywords: smart hospital bed, online data visualization, activity detection, online posture detection

Contents

List of Figures

List of Tables

Acronyms

1-NN 1-nearest neighbour. 26, 31–33

5-NN 5-nearest neighbours. 30

CSV comma separated value. 19, 30

DT decision tree. 24, 26, 27, 31

FBG fiber bragg grating. 3

FSR force sensing resistor. 4

k-NN k-nearest neighbours. 24, 26, 32

KfCV k-fold cross validation. 25, 26

MAP maximum a posteriori. 23

NB naive Bayes. 23, 26, 31

OvO one-vs.-one. 23

OvR one-vs.-rest. 23

PU pressure ulcers. 3

RBF radial-basis function. 24

SkCV stratified k-fold cross validation. 26

SVM support vector machine. 24, 26, 31–33

Chapter 1

Introduction

1.1 Motivation

Recent trend of simultaneous growth of life expectancy and decrease in birth rate resulted in an aging population that requires assistance. The modernization of healthcare is necessary to reduce the cost of much needed resources and to make up for the concerning absence of hospital and care taking employees. The situation calls for support of technology to increase effectivity of the staff through automatic information evaluation or estimation of user activities within the hospital [1].

One of the demonstrations of healthcare automation is the smart hospital bed. Providing and evaluating statistics about patients such as movement patterns or position improves the quality of health service while simultaneously lifts responsibilities off the hospital staff. Moreover, monitoring sensitive activities like breathing and heart rate can be used to inform the personnel, leading to a reduction of unnecessary routine checkups on patients in good condition. Aside from monitoring activities, smart beds have also been developed with different intentions such as improving patient transportation [2] to reduce physical demand for nurses.

1.2 Goal description

This thesis intends to build bases for smart bed monitoring tool development. This goal has been deprecated for the time being due to insufficient information foundation about the bed's communication protocol. Beside its original goals, this thesis presents a solution for acquisition of the bed's data structure through reverse engineering, which allows real-time data visualization and analysis.

The ultimate task of the thesis is to implement an online posture detection system, including an online visualization tool. The real-time nature of the visualization requires constant flow of data, making the reconstruction of data acquisition software necessary. Such data is used for proposing a posture recognition model that is integrated into the visualization, creating an online detection system.

1.3 Structure

The thesis is divided according to the three extensive topics – description, visualization and pattern recognition. The description can be found in chapters 2 and 3 which contain the subject of recovering the data structure and introducing the equipment available for the project, respectively. The visualization is described in chapter 4 which focuses on transformation of the obtained data into an easily understandable format and its real-time display. The offline pattern recognition is described in chapter 5 that analyzes classifiers for posture detection. Finally, chapter 6 proposes an online posture detection system.

1.4 State of the art

Various technologies and non-intrusive methods have been proposed for a smart bed monitoring system, including **load cell sensors**. The authors of [3] suggested using sensors for **ballistocardiography** to measure signals of **heartbeat** and **respiration**, while [4] used it for **movement detection**. Although both papers were focused on relatively different aspects, both mentioned that the method offers the possibility for long term home monitoring without a medical supervisor, making it a non-stressful method of treating sleep disorders and similar issues.

The monitoring of physiological factors was the main focus of [5] that described implementation using **multimode fibre optic sensors**. The results showed that it was an effective low cost and comfortable method. However, it was mentioned that it is not a replacement for the standard methods due to the need of signal processing. Research [6] also proposed usage of an optical sensing technology – 12 wavelength-based FBG pressure sensors. The error rate of $\pm$ 1 breaths per minute was reached, in contrast to the manual computing. The research also proposed a real-time pressure distribution, which is meant to alert the staff to prevent pressure ulcers and with an improved behavior tracking even fall of the patient off the bed.

A different approach was presented in [7], utilizing an idea of detecting posture based on a sensor designed for a different task. The research focused on **sensory substitution** used **acoustic** and **temperature sensors**, reaching a result of detecting 90% of position changes in a controlled environment. The research showed that 'smart' monitoring does not have to rely solely on pressure sensors as the alternate sensors can be used for a satisfactory result as well.

Moving on from the smart bed technologies to posture recognition, a research focused on prevention of pressure ulcers (PU) [8] used posture recognition to correctly shift patients into a new position. Using support vector machines, the risk of developing PU is predicted, changing the patient's position when needed. The position detection was implemented with **k-nearest neighbors classifier**, reaching 97.7% accuracy. Solving the problem of extended stays due to PU would save hospital resources [9] while the automatic posture shift would reduce the physical demand.

The problem with PU was also a subject of [10] despite the absence of automatic position shifting. The research focused on detecting bed postures using **mutual capacitance sensing** to inform a care taker of patient's previous position to prevent bed sores. **Decision tree** was chosen to recognize sitting, lying on the sides and lying on the back or stomach. The classifier reached the accuracy of 93.8% with an experiment done on 8 people weighting 80-95 kg. The classifier performance showed to be greatly affected by the body construction

of the testing subjects, achieving worse results of 80.76% on early experiments with 14 people of varying weight and height.

Posture recognition has also been utilized in [11] for sleep patterns recognition purposes. The authors decided to record data for a particular period of time and work with vectors that represent values of pressure sensors for each time stamp. Vectors were then handled as frames of a video and sensors as pixel of an image. Due to that, the posture classification can be managed as a classification of static frames using **Hidden Markov Model**. Models were trained with Baum-Welch algorithm and the prediction was done by Viterbi algorithm. The accuracy of the classification was 90.4%.

A proposition of a posture detection method using **Bayesian classification** was made in 2008 by [12], using 16 FSR sensors. This approach was based on **kurtosis** and **skewness**, depicting the shape of pressure contour. For classification and modeling, it was assumed that the prior probabilities are uniformly distributed, selecting the **Gaussian distribution**. The experiment was done for 3 laying positions – supine, right log and left log – reaching the average precision rate of 78.7% for all classes, the most problematic detection being supine. This research showed that despite the lower number of pressure sensors, a satisfactory result can also be reached.

A **Gaussian Mixture Model-based clustering** method was applied for a posture classification and limb identification in [13]. Experiments were done on 9 subjects and 13 different sleeping positions. The method was able to recognize three main stable sleeping postures, showing the accuracy to be 98.4%. The identification of limbs also showed good results – accuracy of 91.6%, recognizing 8 limbs in supine and 5 limbs in left or right side postures.

Chapter 2

Smart hospital bed

To put the data analysis into perspective, it is necessary to introduce the reader to the equipment used for data acquisition. The chapter provides information about the bed for better understanding of proficient parts of the thesis.

2.1 Basic description

The smart hospital bed prototype unit was provided by a major European hospital bed manufacturer Linet, known mainly for development of intelligent bed systems [14]. The prototype used for this particular research uses pressure measuring metal sensors called **strain gauges** (gages). The bed has exactly four such sensors built in its construction, labeled by the value of their position in the sent packet (142, 143, 144 and 145).

Any external disturbance that causes stress or strain generates changes in electrical resistance, creating a dimensionless value, called a 'factor' of the strain gauge. Then **gauge factor** is a constant of proportionality between strain and relative change of resistance [15].

Let us define the **gauge factor** as:

$$\frac{\Delta R/R_G}{\epsilon}, \tag{2.1}$$

where ΔR is resistance caused by strain, R_G is resistance of gauge at rest and ϵ is strain [16].

The measurement system offers to measure many other variables such as electric current, values indicating an all-over state of the bed, etc. These values are not necessarily relevant to the thesis and will not be explained further. Besides the gauge factors, the smart bed provides two other useful attributes for the current projects – **weight** and **centers of mass**.

2.2 Communication

The communication with the bed occurs via USB port connected to a computer.

Previously, the data extraction for the research was done through an application provided by Linet written in `C#`. To correctly understand the data extraction process, it was necessary to reverse-engineer the application and carefully examine the acquired data. A decompiling tool for `.NET` applications `dotPeek` was utilized for this matter [17].

Unfortunately, the documentation of the code was also somewhat incomplete, and so unresolved parts remain. However, they are mostly not relevant for this thesis.

To communicate with the bed, the port needs to be set with various parameters inherited from Linet's original application. The backend is provided by Python's `pySerial` [18], a module that assures a port communication.

The following code carries out the communication:

```
1
2  import serial
3
4  COM_PORT = 'COM3'
5  PORT_TIMEOUT = 10
6
7  try:
8    port = serial.Serial(COM_PORT, baudrate=38400, bytesize=serial.
   EIGHTBITS,
9              parity=serial.PARITY_NONE, stopbits=serial.STOPBITS_ONE,
10             timeout=PORT_TIMEOUT)
11
12 # If port is already used/wrong port.
13
14 except serial.serialutil.SerialException:
15   print('Specified COM port is already in use by another application or
   doesn\'t exist.')
16   exit(-1)
17
18
```

Chapter 3

Data

All data for the research had been previously acquired solely through Linet's application due to insufficient information about both communication protocol and data structure. Since a satisfactory way of data collection existed and the research was adapted to it, it hasn't been top priority to resolve the unknown data structure. However, the part of this particular thesis is an implementation of an online visualization, where working with raw data is unavoidable. To be able to correctly convey the data, the whole procedure of the data acquisition needs to be redone, even if it means having to deal with complications due to insufficient information. That, of course, requires a clear image of the raw data and its structure that is described in this chapter. Aside from reverse-engineering, the bed's documentation [19] was also used as the source of information for this chapter.

3.1 Structure description

Working with the obtained data in its raw binary form can be uncomfortable – especially when the knowledge about the structure is unknown. For that reason, the acquired data is converted into a **hexadecimal form**, utilizing its ability to **contain a byte in exactly two hexadecimal digits**. The form is also **easier to process** and read by a human which is beneficial when manual inspection is unavoidable. Due to its **compactness** and **readability** (for a human), it was decided to be a more suitable choice to work with. Hexadecimal form will also be preferred when mentioned in plain text for the reader's comfort.

The information about the structure was obtained from the reverse-engineered application, documentation and observation of data flow. The description is following:

```
Opening delimiter of message
7E - BEGIN

Message header
13 - protocol version and message type
17 - message ID
00 90 - message size
01 - data type

Data block begins here.
Each data packet consists of an ID and value/values.

Value of el.current, state of bed, etc.
01 00
02 00
03 00
04 00
05 00
06 AB
...

From ID 80 the values are sent in two bytes
More state of bed values
80 00 00
81 00 00
82 00 00
83 02 F9
84 02 A5
85 03 66
86 03 5F
87 00 B1
88 00 00
89 00 05
8A 00 04

8B 05 E8 - absolute weight
8C 00 23 - center of gravity X
8D 27 F8 - center of gravity Y

Strain gauge values (Gauge factor)
8E 83 22
8F 83 7D 1A
90 84 AF
91 85 56

Some other values, currently not interesting
97 18 88
```

```
48
49  Used for verifying data integrity after data transfers,
50  not used but potentially interesting
51  89 6B - CRC checksum
52
53  Ending delimiter of message
54  7E - END
```

Listing 3.1: Data structure

It should be noted that the message structure is very specific. Observation also revealed an occasional exception in the number of arriving bytes as is shown in the listing 3.1 section **Strain gauge values (Gauge factor)**. The occasion also always seemed to include a presence of **byte** `7D`. Ultimately, additional research revealed that the reasoning of this specific structure and additional bytes in the packet lies within **Point-to-point protocol (PPP)**.

According to **RFC1662** [20], the Point-to-Point Protocol (PPP) provides a standard method for transporting multi-protocol datagrams over point-to-point links. The protocol enforces the following:

- Flag Sequence `0x7E` implying the beginning or end of the frame
- Escaping byte `0x7D` (byte stuffing)
- Whenever an escape byte or flag is encountered in the message, it is escaped by `0x7D` and a bitwise XOR is performed on the byte and `0x20`.

It was clear that if the data packet arrived in such manner, the implementation of the receiver had to adapt as well. The PPP pointed out **byte stuffing** by `0x7D`, that clearly needs to be de-stuffed by our receiver. That is, of course, not the only issue that needs to be discussed further as two more issues regarding data had to be resolved – **endianness** and **signed number representation**. The following section addresses these topics more extensively.

3.2 Data processing

This section focuses on additional work with the newly acquired data and discusses the previously mentioned issues. The procedure of processing and filtering uninteresting parts of the message structure is also addressed, as well as the correct representation of the data flow, a crucial part to any kind of advancement in the thesis task.

The following code manages receiving of data:

```
1
2  global receiving
3  receiving = true
4
5  while receiving:
6    message = port.read_until(b'\x7e\x7e').hex()
```

The usage of global variables is most of the time considered risky. It is rightfully criticized for side-effects, illegibility and having destructive properties in the debugging stage. In this case, the variable needed a considerable scope for the present threads to see, while having the ability to be changed from within. Since the number of threads is small and the script is not extensive, it was decided not to complicate the implementation further. Clearly, there is no doubt that in the future, a better object design will be created.

The piece of code utilizes the already known information about the data structure – it reads the incoming message until an ending separator is found. The data structure 3.1 explained that `7E` was a **beginning** and **ending flag**. The conversion to hexadecimal form was also done at this level by calling function `hex()` right as the data was obtained. After that, one could easily filter the desired values through regular expressions.

Section 3.1 familiarized us with the data structure and hinted on three particular issues that are discussed in the following subsections.

3.2.1 Endianness

Endianness is a method for storing values that exceed the capacity of one byte. There are two conflicting basic formats based on the order of most and least significant bytes.

The **big-endian** format stores the most significant byte first and the least significant byte last. The big-endian format is frequently dubbed as **network (byte) order** due to the agreement of the network community. **Little-endian** uses the opposite approach, storing the least significant byte first and the most significant last. The format is particularly utilized in microprocessors [21].

The research experimentally established that the smart bed uses the **big-endian** format, which was later confirmed after the decompilation of the Linet's original application.

In the proposed script, converting values into the big endian format is a part of a function `to_16bits(byte1, byte2)` that is responsible for transforming byte streams into actual values for the respective attributes.

The following code handles the endianness:

```
1
2  if BYTE_IS_BIG_ENDIAN:
3    result = 256 * byte1 + byte2
4  else:
5    result = 256 * byte2 + byte1
```

Let us put bytes `byte1` and `byte2` alongside each other – the most significant byte's lowest bit would start at the value of 28. Knowing this, the significant byte in its decimal form is multiplied with value for the lowest bit of the byte (256 in decimal). Ultimately, the other byte in its decimal form is added to this value.

If the condition is fulfilled, the result is produced by multiplying variable `byte1` first, clearly because it was evaluated as the significant byte. Else, `byte2` is set to be the significant byte.

3.2.2 Signed number representation

Since numbers in computers are portrayed as sequences of bits, a signed number representation is means of interpreting a negative sign within the sequence. There are numerous methods of representation, from which computing devices use twos' complement in most cases.

The research experimentally established that bytes were signed, which was later confirmed with the Linet's original application. The code written below is again a part of a function `to_16bits(byte1, byte2)`. The following part handles the signed number representation section:

```
1
2  if BYTE_IS_SIGNED:
3    if result > 32767:
4      result -= 65536
5    return result
```

The representation of the biggest number in 16 bits would logically hold the first bit as a sign and the next 15 bits as ones (32767 in decimals). Any value higher than that must have accounted the 16th bit (the sign) as part of the value itself. Such value must be therefore inverted, done by subtraction of 16 bits of all ones (65636 in decimal).

The fact whether the byte is signed or unsigned plays a significant role in the final result. The table 3.1 shows an attempt to convert bytes `7E 7C` into a decimal value with both representations.

	7E 7C
Unsigned	32380
Signed	-33156

Table 3.1: Signed and unsigned bytes 7E 7C

It should be noted, that both endianness and signed number representation are **dependent** on each other. The priority is to correctly convert the binary representation of the value into a single decimal value and only after that, the program can consider the value's sign.

Bytes from the hospital bed are then obtained in a **big-endian** format, and they are **signed**.

3.2.3 Byte stuffing

Byte stuffing is utilized in situations of maintaining values of delimiters as part of the message. Data packets can be sent in varying sizes, where keeping a **message delimiter** or a **message flag** becomes necessary. However, there are situations where it is desired to keep the flag as a part of the message packet. The delimiter therefore needs its own escape token, which is exactly what byte stuffing implements.

Byte stuffing is then a method to differentiate between data and a message delimiter (flag). A stuffing byte is inserted in front of the problematic part, causing the receiver to treat the following as data and not as a flag.

In section data structure 3.1, the PPP protocol has been introduced. It behaved in such way that whenever an escape byte or a flag is encountered in the message, it is escaped by `0x7D` and a bitwise XOR is performed on the byte and `0x20`.

On the receiving end, escaping byte is destuffed, and bitwise XOR is performed on the following byte and `0x20`. A following example can be observed:

```
1  Hex   Bin
2  0x5E  1011110
3  0x20  0100000
4
5  XOR   1111110 -> 0x7E
```

The product of the bitwise XOR is again the value `0x7E`.

Chapter 4

Data visualization

Chapter 2 has introduced strain gauges and their dimensionless values **gauge factors**. Even after establishing a formal equation for them, it is still quite difficult to process their meaning as they have no dimension and so important conclusions cannot be drafted from observing individual values by themselves. Strain gauges **effectively preserve information about physical aspects** of the subject placed on the bed if all four of them are considered. The set of values is mostly different depending on external stimuli – a sitting position would generate a different set of values than a lying position. For effective projection of the information, the research uses visualization tools for better understanding.

The goal of data visualization in this research is to ease recognition of interesting aspects in data. Graphs can be visualized either **offline** – a method of visualizing a precollected set of data, or **online** – real-time visualization with updating values. In this thesis, we will focus on an implementation of a real-time (online) visualization, bringing a new tool for the hospital bed research in form of two types of graphs:

Line plots represent each gauge factor itself in its absolute value. All four graphs are plotted into a single figure, allowing the perception of each sensor in real-time. Since the graph presents all the absolute values at once in current time it is computationally demanding. Its advantage is simplicity, demonstrating the gauge's functionality well without the need for further explanation. Compared to scatter plots, it only holds information about the gauge factor's absolute value.

Scatter plots are more expressive, and their features are heavily utilized in this project. They have proven to be beneficial in **finding correlation** between plotted variables, making it easier to notice characteristic attributes in data. They transfer a lot of information in a short amount of time, and for this project, they represent the best visualization option.

Since it is naturally more challenging to follow them at first, a bar plot has been created in addition for a signal feedback check.

4.1 Tools

Python provides many libraries for data visualization, the most popular one being `matplotlib`, an open-source 2D plotting library that is designed to bear resemblance to the environment of MATLAB [22]. It was chosen for being well supported, stable and powerful. It also works well with other libraries for data processing like `pandas`.

This implementation utilizes `matplotlib's` similarity to MATLAB with module `Pyplot`. One of their common features is an ability to **access a current figure** and **axes of figure**. Compared to the object-oriented version of the library, its strength lies within **automation of interaction** with the graph and its **unnecessity to hold references** to graph objects since the module does it internally.

`Pyplot` offers many useful operations and ones that were used in this implementation are following:

- creating a figure,
- adding multiple graphs to a figure,
- labeling individual graphs and their axes,
- plotting various types of graphs,
- handling user events (clicking, typing, etc.),
- updating data in a graph,
- styling,
- user interaction control elements (such as buttons),
- setting a view range,
- setting ax scales (logarithmic, linear, etc.).

4.2 Implementation

This section describes implementation parts of the visualization that were somehow intriguing, either because of their custom-built nature or tricky character. It is not an objective of this thesis to serve as `Pyplot` or `matplotlib` documentation. Moreover, the

basic functions used to plot the graph are mentioned in the previous section 4.1. Therefore, trivial parts will not be commented further. Scripts can be later found attached to the thesis.

Implementation was done in two different scripts, the line plot serving as an introductory prototype to the scatter plot. The priority of the project was always the implementation of an online scatter plot which holds an important purpose with intentions to use it for pattern recognition. Nevertheless, both graphs are valuable for different purposes and have their own advantages. Implementations have a set of parameters listed in the beginning of the script for easier editing.

The common parameters for both graphs are following:

```

COM_PORT = 'COM3'
PORT_TIMEOUT
BYTE_IS_BIG_ENDIAN
BYTE_IS_SIGNED
LOG_FILE  # text file that stores data sent from the bed
TENSO_LABEL  # labels for strain gauges

# Graph view
LOWER_LIMIT/SCATTER_LOWER_LIMIT
UPPER_LIMIT/SCATTER_UPPER_LIMIT
```

The parameters for line plot are:

```

LINE_COLOR
```

The parameters for scatter plot are:

```

MARKER_ALPHA  # transparency of a marker
MARKER_SIZE
MARKER_COLOR
MARKER_EDGE_COLOR
MARKER_HIGHLIGHT_COLOR
MARKER_LINE_WIDTH
MAX_NUM_OF_MARKERS_IN_GRAPH

MODEL_NAME  # pattern recognition model

EMPTY_BED_VALUE  # approximate values for gray cross plotting [LH, PH,
 LD, PD]
```

The following subsections about particular graphs might refer to these parameters.

4.2.1 Line plot

Line plot visualization did not contain any parts that were particularly tricky or challenging. The objective was to plot the representation of gauge factors as signals as it is shown in Figure 4.a.

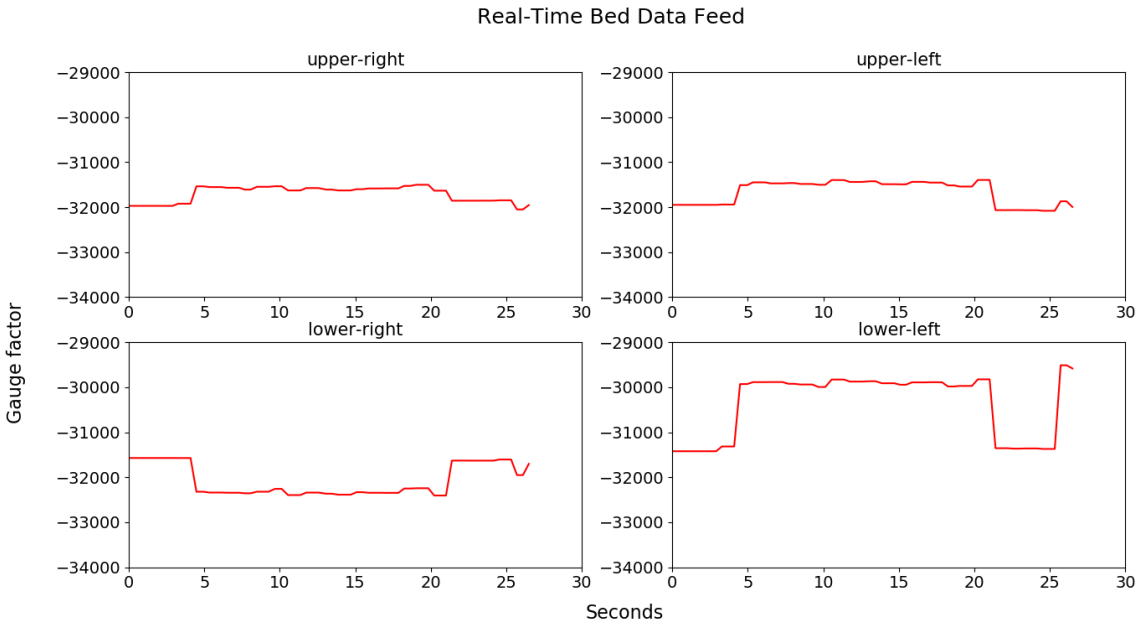

Figure 4.a: Line plot

Its particular purpose is **presentation of signal development throughout the time**. The plot's x-axis represents the continuous time in seconds with view initially set to 30 seconds. The graph adapts whenever the view does not suffice by readjusting x-axes. The line then reacts to changes in gauge values by motioning upwards or downwards on y-ax, while progressing in time on the x-axis.

The time information is preserved through variables of initial time `start_time` and its difference with current time `delta`. The x-axis shift is performed in the function `update_graph()` by a trivial condition:

```
1
2   if delta > 30:
3     axes[i].set_xlim(delta - 30, delta)
```

where `axes[i]` is one of the subplots in the figure.

The function that is responsible for line plotting is `upgrade_graph()` which is called within `matplotlib's` class `FuncAnimation` object that is used to set animation for a whole figure. Its parameters consist of function object to call, the figure to animate and other optional parameters such as interval, representing a function period in milliseconds.

```
1
2  animation = FuncAnimation(fig, update_graph, interval=200)
```

The function interval was set to 200 ms. A shorter span would have resulted in smoother progress, however, it is redundant to do so as the graph would have to plot more frequently, handicapping slower computers. Furthermore, new data packets might not have even arrived in time if the interval was too small.

4.2.2 Scatter plot

Scatter plot represents a graph of correlation between two strain gauges, with each axis responding to one of the four sensors chosen arbitrarily by the user. Compared to its offline counterpart, online scatter plot **allows real-time monitoring of data paths**. That enables easier perception of interesting attributes between two gauges, bringing new kinds of information to the research.

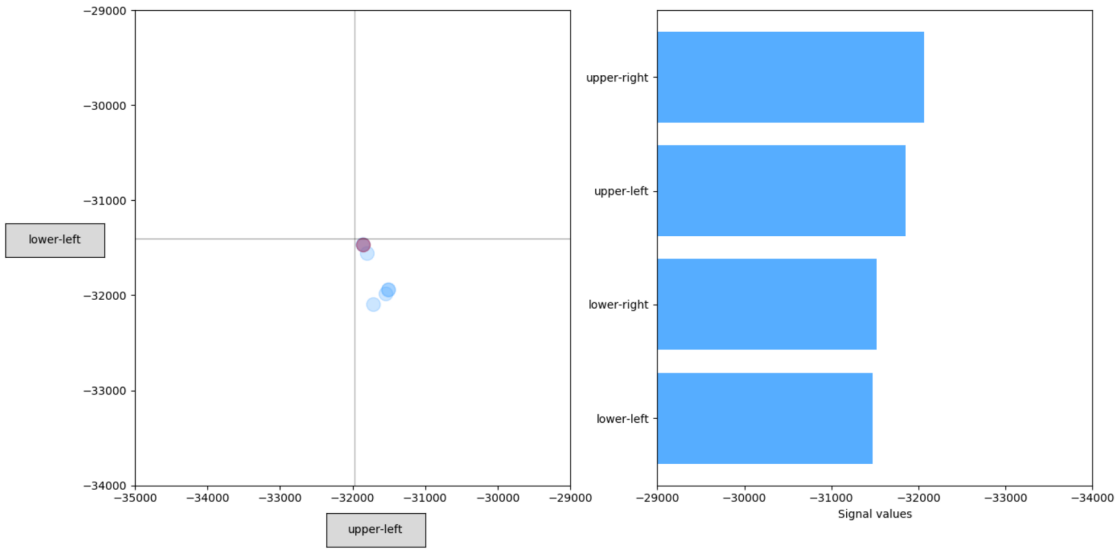

Figure 4.b: Scatter plot

To make the observation more comfortable, the newest point is always red to distinguish the newly added value. Points also have a set transparency through a parameter `alpha=MARKER_ALPHA` to deal with overlapping, allowing the user to see the point outlines instead of an incoherent clutter. If points remain at the same place, each update increases saturation, so these areas are easier to notice. These features can be viewed in figure 4.c.

Another feature to ease the plot's readability is **highlighting areas of resting gauges**. That is marked in the graph by a gray cross which can be observed both in 4.b as in 4.c. Since values for the empty bed are different for each gauge (interestingly), it is necessary to re-plot the gray cross with every change of the strain gauge (in user induced events).

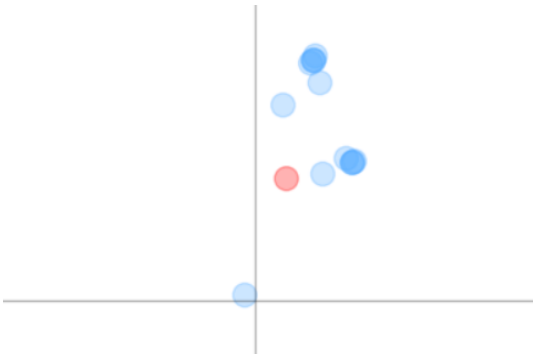

Figure 4.c: Saturation representing density of points

That is done through a script function `change_axempty()` that calls `matplotlib` functions `axvline()` to plot a vertical line and `axhline()` for a horizontal one. Since the functions don't automatically access the original lines and make a new line whenever they are called, it was decided that removal of the originals before re-plotting new ones was the easiest approach.

It might seem strange that empty values sometimes move around and are not absolute. That is due to numerous external inputs such as the imperfection of strain gauges, environment, mattress shift, etc. The online visualization also revealed that the sensors undergo hysteresis of some kind and are never completely at rest. This phenomenon can be observed by using a zooming tool at a plotted point.

Which two of the gauges will be plotted is chosen through a click event. The click event is assigned to a particular button by `matplotlib's` function `on_clicked(arg)`. It should be noted that the expected argument is a function and not a function call. The latter would evaluate the function right away, returning its result which wouldn't be callable from the inside of the on-click function. This issue is resolved by using lambda expression's lazy evaluation in the argument:

```
1
2  x_button.on_clicked(lambda x: change_axempty(x_button, data_selector,
    True))
3  y_button.on_clicked(lambda y: change_axempty(y_button, data_selector,
    False))
```

The range of values to display on the graph is set manually using parameters `SCATTER_LOWER_LIMIT` and `SCATTER_UPPER_LIMIT` based on empirical experiments. The automatic range that `matplotlib` deduced proved to be problematic due to insufficient number of data instances during the initialization. Limits are originally set closely around the very first data point, and any significant change would be plotted outside of the scope. It was considered to recalculate the limits dynamically, however, that would result in visual shift of all currently visible points. That would decrease the graph's legibility dramatically

because of its real-time nature. Should the observer deem change of the scope necessary, the parameters can be edited easily.

Data acquired from the bed is stored in a CSV format file for later use, however, it is not desired for the graph to be overcrowded with values as it would decrease its legibility. For this reason, a parameter `MAX_NUM_OF_MARKERS_IN_GRAPH` has been added to set a maximum number of points that can be seen at time.

```
1
2       MAX_NUM_OF_MARKERS_IN_GRAPH = 15
3       # ...
4       if len(data) == MAX_NUM_OF_MARKERS_IN_GRAPH:
5         data.pop(0)
6
7       data.append(data_packet[3:])
8
```

Update of the scatter plot is done every time a data packet arrives through the script's function `update_graph()`. At first, the whole `Axes` object was reinitialized, discarding user's interaction such as zooming in on the scatter plot. A lot of redundant operations were also performed because of that, resulting in computational heaviness. This approach has been replaced by changing the offsets of the points by `matplolib's` function `set_offset()` called on the scatter plot's data path. This way, the user interaction remains unchanged, graph is plotted only once, and only the path gets updated.

When deciding on the most suitable visualization composition for the scatter plot, many other possibilities were explored. The following section discusses a few of those visualization tools that were deemed interesting for this subject.

4.3 Discussion

4.3.1 Visualization in higher-dimensions

One of the most common concerns in data visualization is the representation of multi-dimensional data. While human perception processes up to three dimensions, visualization of more than two attributes begins to cause trouble due to the limitations of a computer screen. Useful off-the-shelf solutions exist such as feature selection, feature extraction and manifold learning. Some projects, as this one, need the data to be projected as it is. Although it might seem strange to discuss graph representations for sole 4 dimensions, there might be time in the future that the research may need to add more attributes. Before settling for the scatter plot, various methods were explored to find the most suitable possibility.

Paper [23] has suggested to express higher-dimensional values by **glyphs**. These graphical entities represent data through specific features such as shape and size. To a certain point, glyphs can be implemented in `matplotlib` through a previously mentioned graph attribute **marker**. Markers can be set in various shapes, sizes, colors, and transparencies. Glyphs would manage to present all four gauge factors at once, allowing to see correlations between all gauges. However, a question remains if it is always possible for a human to process the information given to them in such state in real time.

Representing multi-dimensional data through **bubble plots** holds a similar idea but simpler, shedding the shape attribute. The graph projects up to 6 dimensions, representing the higher dimensions as sizes and colors [24]. Even though such method does not seem to hinder the comprehension too much, the trial version showed that any kind of graph that has more than 2 attributes is not very suitable for a real-time updating visualization tool. Example of the bubble plot visualization where strain gauge 91 (145 in decimal) was represented as size can be seen in figure 4.d.

A scatter matrix was the last option considered before settling with the current composition. It projects each attribute against each, creating six graphs and their inversions into one figure. Since it uniformly holds 12 graphs, scatter matrices are computationally demanding due to the real-time update of all 12 subplots. The small size of each subplot was also uncomfortable to handle, even after utilizing the zooming tool. Visualization of the scatter matrix can be seen in figure 4.e

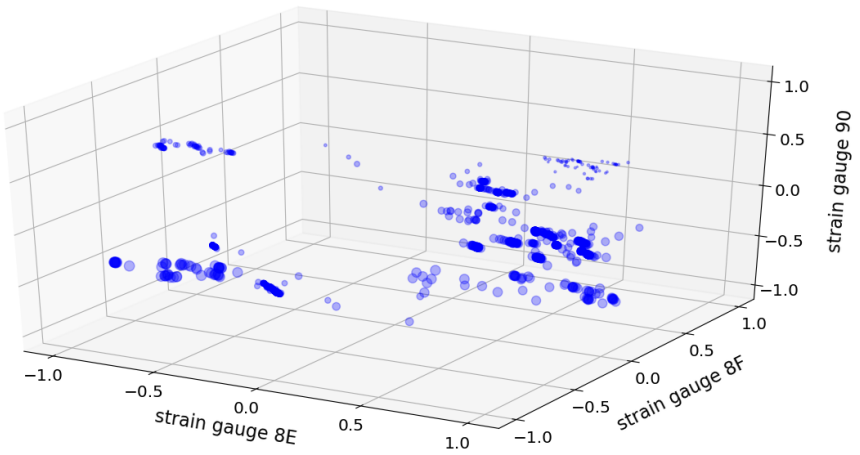

Figure 4.d: 4D bubble plot example

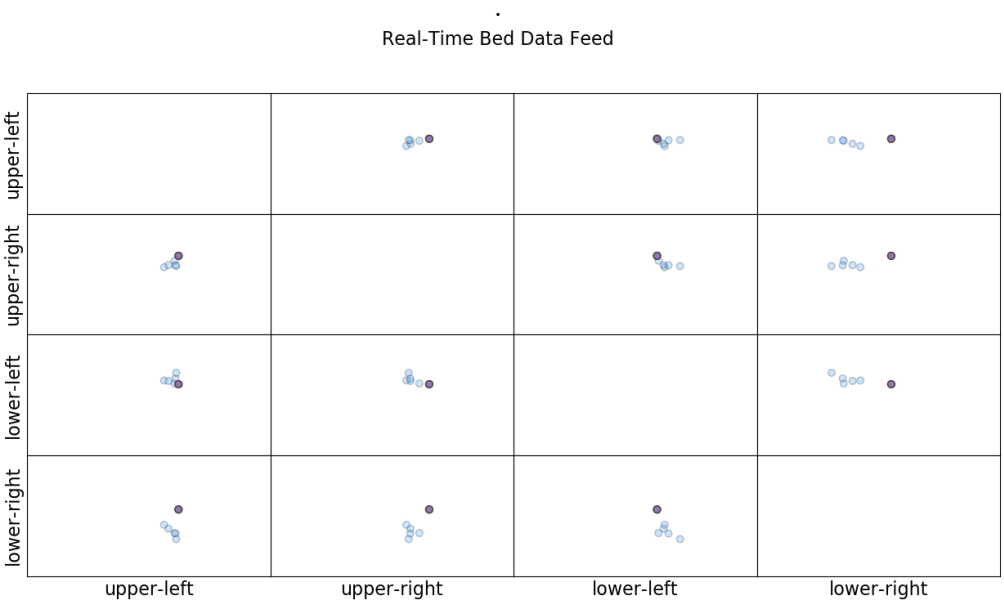

Figure 4.e: Scatter matrix example

Chapter 5

Activity detection

Chapter 4 visualized gauge factors for easier understanding of their functionality. As the reader has a better idea of how gauges react to stress and strain, the fact that different strain distribution results in a different set of values should be now apparent. This chapter concentrates on utilizing machine learning algorithms to classify states on the smart hospital bed. The ultimate goal is to recognize various states in real time.

5.1 Classes of activities

This section discusses activities that a patient can perform on the bed, from the typical activities to the complex ones. Let us divide the activities into two types.

Activities that are deemed typical are **lying**, **sitting** and **empty bed**. Lying is divided into **right log**, **left log** and **supine**. The respective positions are shown on figures 5.a to 5.d below, except for the empty bed.

Figure 5.a: Position sitting

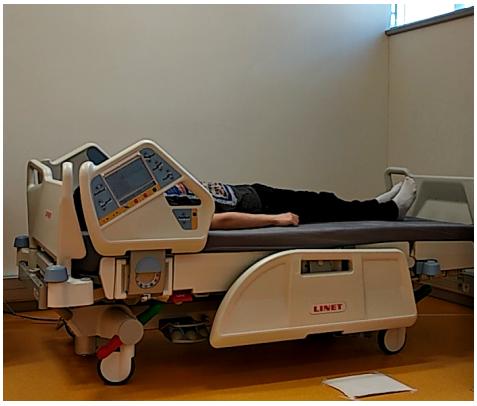

Figure 5.b: Position supine

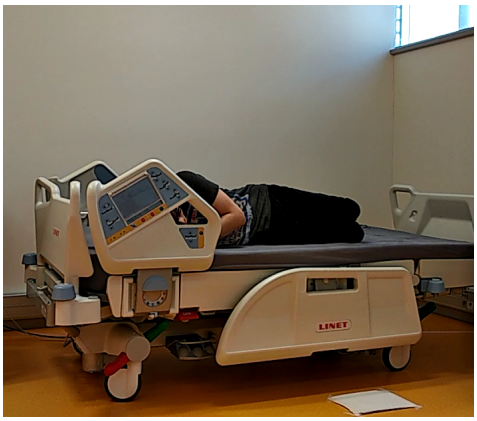

Figure 5.c: Position right log

Figure 5.d: Position left log

Activities that are deemed complex are **fall**, **sleep**, **seizure** and **violation of the work procedure**. During the research, these activities have been discussed but with the current available tools, it is not assumed that all the complex activities would be recognized with a satisfactory result. It would require measurements taken for a particular period of time to design a feature vector. Moreover, published researches that were done on such activities had used a different set of equipment and never just strain gauges by themselves. It is not a purpose of this thesis to resolve pattern recognition for these complex classes yet but surely, experimenting with these classes would prove beneficial to the research.

5.2 Classification

One of the well-known pattern recognition approaches is **classification** – a process of assigning data into classes based on its previous training on an already labeled data set. The training can be solved using **supervised learning** methods [25], [26].

The classification is performed for the **typical activities** listed in the section 5.1 which implies that it deals with a **multiclass classification**. It should be noted that classification of multiple classes is handled differently than its binary peer. Let us present a few approaches for a multiclass classification problem in the following paragraph.

Instead of letting an algorithm classify multiple classes at once, the problem is transformed into **several binary classification problems**. One of the strategies that is used for this matter is **one-vs.-rest (OvR)** where each class has its own trained classifier that deems that particular class 1 and the rest 0. Even though paper [27] wrote about its potential effectiveness, the method faces many troubles such as having to deal with unbalanced distribution of classes. Even if originally, the data set is balanced, the method will see many more 0 than 1. An alternative method is **one-vs.-one (OvO)** which trains $\frac{N(N-1)}{2}$ classifiers for each different pair of labels. OvO is less prone to unbalanced data sets but it is also much more computationally demanding [28].

Another method is to train a classifier that is naturally suitable for solving a multiclass classification problem. This thesis follows that approach, utilizing Python's well-known machine learning library `scikit` [29]. The considered algorithms are compared and discussed below:

Naive Bayes (NB)

A simple probabilistic classifier based on **Bayes' theorem** that always assumes independence between features, disregarding their relation. It uses **maximum a posteriori (MAP)** principle to predict association to a certain class. Despite its naive assumption, it is a valuable and effective machine learning algorithm as shown in Zhang's article [30] in 2004. Multiclass classification is naturally easy for NB as it calculates the probability of each class label and selects the one with the highest probability. [31]

Since the feature in our data is **continuous**, Gaussian distribution of likelihood is assumed.

K-nearest neighbours (k-NN)

The k-NN is one of the most trivial machine learning algorithms. Its prediction operates on assigning new data instances to the most common class between k most similar data instances. Typically, the similarity of two data instances is based on their distance. It falls under **instance-based** learning category for predicting new instances by comparison to the ones in the testing data set. That being said, the absence of learning is clear in k-NN, making it a **lazy learning algorithm**. The model representation is therefore the whole training set which proves troublesome with a growing data size. Its high memory requirement can be addressed by editing the training set as removing outliers and updating new values. [31]

Support vector machine (SVM)

A non-probabilistic typically linear classifier that is based on maximization of distance between classes (margin) creating an optimal hyperplane. The optimal hyperplane is represented by the following equation:

$$|\beta_0 + \beta^T x| = 1, \tag{5.1}$$

where β is a weight vector, β_0 is bias and x are instances closest to the hyperplane in training data set – referred to as **support vectors**.

This representation is called a **canonical hyperplane**. Computation of optimal hyperplanes presents a **dual problem** that is solvable through algorithms in quadratic programming [32] [31].

As kernel for the SVM model, the radial-basis function was chosen due to its flexibility and stable performance.

Decision tree (DT)

An intuitive machine learning algorithm that makes sequential decisions in order from the most important node, acquiring the class value of a new instance from leaves. Nodes and leaves can successfully keep information allowing to compute various class probabilities. DT is often used due to its interpretability and similarity to human decision making. Trees are naturally prone to **overfitting** and so several methods for better generalization are introduced, such as **pruning** or combining multiple DTs making a **decision forest** [31]. Utilizing how DT classifies data by information entropy, centers of mass were added to the set of features to improve the performance of all classifiers.

5.2.1 Data set acquisition

Training the model for online classification was done on a data set measured on subject H. The measurement procedure is following – subject is placed in a **certain position** on the bed for **60 seconds**, then the bed is left **empty** for **30 seconds**. The process is then **repeated 5 times**, getting a total of **5 minutes for each position**. It should be noted that the training data set contains various sitting positions such as sitting in the middle of the bed, on the sides, etc. For such reason, the data acquisition for sitting particularly needed to be altered which can be viewed in the appendix A.2.

Procedure	
Position	60s
Empty bed	30s
Position	60s
Empty bed	30s
Position	60s
Empty bed	30s
Position	60s
Empty bed	30s
Position	60s

Table 5.1: Data set acquisition procedure for one posture

The reason why the measurement procedure was designed this way is to achieve data set **robustness**. Since it is impossible for a person to place themselves in the same exact position as before, making the subject repeatedly position themselves within the same measurement provides the classifier with more possibilities of how a position might look like. If the recorded data is only made of a subject in one exact position for 5 minutes, the model used in online detection would probably not be able to recognize slight changes and shifts, making the model unusable. Classifiers would be basically testing data very similar to the training ones if not identical, leading to misleading accuracy.

5.2.2 Preprocessing

The attributes used to train the model are **4 strain gauge factors** and **center of mass on both axes**. It should be noted that their values differ immensely as values of strain gauges are always around -31 000, while centers of mass are around 100. It was decided that **normalizing the data to scale -1 to 1** would be used as it gave the best performance out of other normalization methods.

The data set also contains ambiguous parts where the subject is not yet in position. Since there is no point in trying to classify those kind of values into predetermined classes, they were removed from the data set.

5.2.3 Performance estimator

To measure a classifier's performance, a version of **k-fold cross validation** (KfCV) was used to estimate the accuracy of each classifier. The estimator parts the data set randomly into k disjunctive subsets with identical length, a model is trained using $k-1$ folds as training data and tested on the remaining fold. Compared to **hold-out** method that simply splits data into training and testing set, KfCV achieves less biased estimations and combats overfitting. It is also suitable to use on small data sets because of its unwasteful approach of using the whole data set for both testing and training. On the other hand, its price comes with its computational demands that is dificult to meet with bigger data sets.

Stratified k-fold cross validation (SkCV) is a version of KfCV that ensures that proportions of classes in particular folds are similar to the proportion of classes in the original data set. Since our data set is not perfectly balanced, SkCV represents a more suitable performance estimation method. It was decided to set the recommended $k = 10$.

5.3 Results

In terms of total accuracy, 1-NN showed the best performance with accuracy rate of 97%, while SVM followed closely behind with 96.3% and DT with 96.1%. NB had the worst results but still managed to get 92.9%. However, since the data is slightly unbalanced, the total accuracy can be misleading with the classifier preferring majority class. For that reason, the following tables also present results in a form of classifier errors for each class providing more detailed information about performance on each class (similarly to sensitivity and specificity in binary classification).

The cross validation estimates of class error rates for subject H can be found in table 5.2.

	Empty	Sitting	Supine	Right log	Left log
1-NN	0.0	0.13	0.02	0.02	0.0
GNB	0.05	0.22	0.03	0.03	0.03
DT	0.0	0.06	0.08	0.02	0.01
SVM	0.0	0.15	0.03	0.02	0.01

Table 5.2: Class error rates of each classifier on each position (subject H)

Algorithm 1-NN was able to recognize all instances of an empty bed and left log. Sitting caused quite a difficulty to detect, mainly because of the similarity between sitting in the middle of the bed and supine. Since other k-NN did not reach better results, they are not currently mentioned. Out of the compared classifiers, NB had the biggest problem with classification of the sitting position. It was able to detect left log with small mistakes,

surprisingly having the second highest error rate on class empty. Decision tree was able to perfectly recognize an empty bed while having problems with classes supine and sitting, with supine having the highest error rate. Support vector machines had the same performance with empty and left log like DT. Again, sitting had the highest error rate.

In conclusion, classes that are the easiest to recognize according to the error table are empty (except for NB), left log and right log. The classifiers often struggled with classes sitting and supine due to the similarity of these two positions in case of sitting in the middle of the bed. It is important to remind that the results are computed for one selected subject and inter-personal variability is ignored. The ability of classifiers to classify postures of subjects that it has never seen before is presented in chapter 6.

5.3.1 Discussion

The results showed that all classifiers except for DT, had the highest error rate with class sitting. DT had the highest error rate with class supine, which is an analogous situation. Let us look at the features used for classification visualized in graphs:

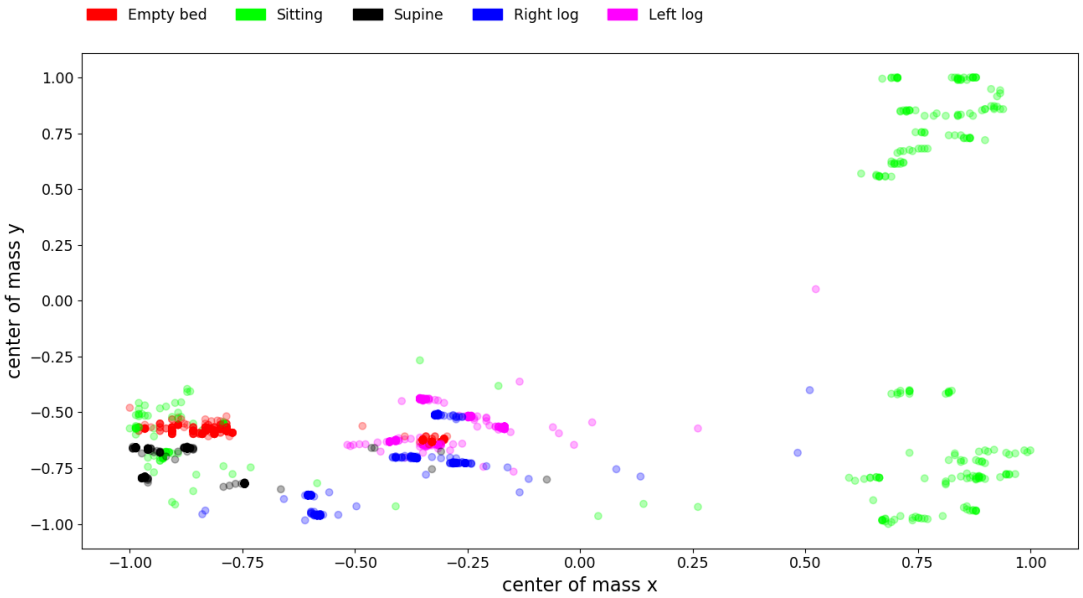

Figure 5.e: Centers of mass of subject H

As can be seen from graphs 5.e and 5.f, the data is not linearly separable. Instances where the subject sat on the edge of the bed (well separated green clusters), left the bed empty (red points) and was in position left log (pink clusters) were understandably the easiest to recognize. Zooming into the area of values for supine and sitting in figure 5.g shows that the data position is a quite peculiar but not entirely inseparable.

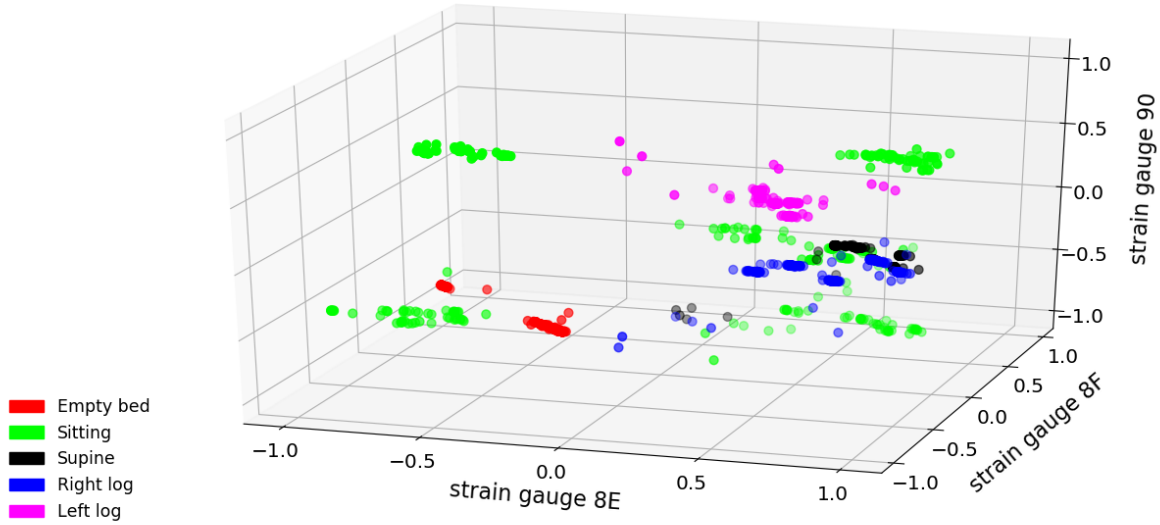

Figure 5.f: Gauge factors 8E, 8F and 90 of subject H (3D)

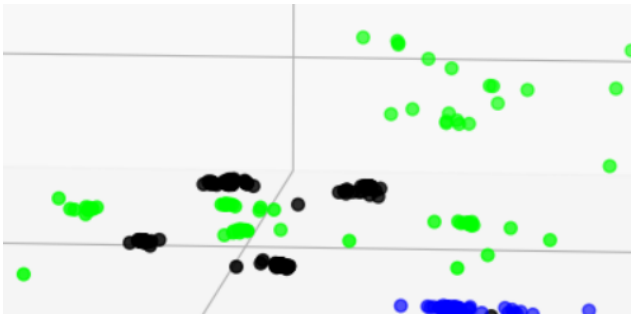

Figure 5.g: Detailed view of inseparable supine and sitting in figure 5.f

Figure 5.f omitted the strain gauge for clarity of the 3D graph. For a scatter matrix graph where all strain gauges are plotted against each other, see figure A.1 in appendix A.

Chapter 6

Online detection

Online detection is the primary goal of the pattern recognition part. Our intention is to propose and implement and online position recognition system. The result of such recognition should be reported to the user in the graphical user interface described above. Currently, it is sufficient to show a text string describing the posture. This will appear in the same vindow as the scatter plot and bar diagram visualization.

6.1 Implementation

Trained classification models are exported into a binary file included in the scatter plot visualization script. The model's function `predict()` returns its result as a number representing the predicted class corresponding to a position. The name of the position is printed in the top-right corner of the figure as showed in figure 6.a.

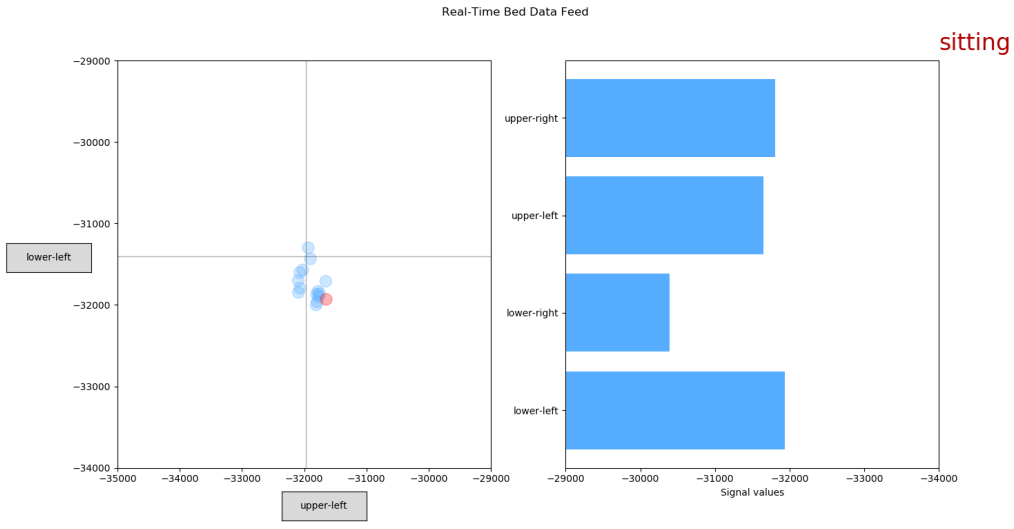

Figure 6.a: Online visualization with posture classification

```
classifier_result = model.predict(arg)

# Interpreting class values as position
if classifier_result == 0:
  label = 'empty'
elif classifier_result == 1:
  label = 'sitting'
  . . .
```

A parameter `MODEL_NAME` that was previously mentioned in the visualization chapter 4 was added for easier manipulation with the classification model. Module `pickle` [33] was utilized for import and export of Python objects.

It is also necessary to normalize the newly received data due to the previous normalization on the training data. Compared to the offline classification, the online detection does not have access to its training set – it only uses an imported classification model for predicting. The normalization; however, needs the training set's min and max values in order to scale the incoming data properly. Those values are obtained by importing an original library `normalize` and calling its function `init()` which loads maximal and minimal values of the training data set from a CSV file. The scatter plot visualization script calls `normalize.init()` to correctly set the minimal and maximal values before calling the function `predict`.

```
# Data normalization
normalize.init()

for i in range(0, 6):
  # normalize . . .
```

6.2 Classification model

Testing of the online detection system is done on 5 subjects of different weight and constitution. The intention is to experiment with a model trained on **one subject** and examine its functionality on the other 4 subjects. For that reason, a subject P with the average weight and height was chosen, measured according to the procedure 5.1 and the model was trained on data set of subject P. The all-over accuracy of the subject is listed in the appendix A.1. The error rate of 5-NN was also added for the online detection purposes.

Compared to the data set of subject H this data set does not seem to have problems with supine and sitting. Its visualization, graphs A.2 and A.3 show that the data set of subject P is more easily separable than the one listed in results.

	Empty	Sitting	Supine	Right log	Left log
1-NN	0.0	0.01	0.01	0.09	0.0
5-NN	0.0	0.15	0.02	0.02	0.01
NB	0.02	0.01	0.03	0.04	0.03
DT	0.01	0.13	0.03	0.03	0.02
SVM	0.0	0.02	0.03	0.03	0.01

Table 6.1: Class error rate of each classifier (subject P)

After some initial testing it was decided that classifiers NB and DT will not be considered for the online detection experiment. Despite NB showing better results with subject P, it did not seem to perform well in comparison with others. DT was not chosen for the same reason.

The rest of the classifiers were trained and applied onto the visualization script. Prior to experiments, the following was expected:

H1 1-NN will be too sensitive to noise, might have to try i.e. $k = 5$,

H2 SVM will be the most reliable,

H3 classes 'supine' and 'sitting' (in the middle of the bed) might cause problems,

H4 small quantity of strain gauges will cause excessive reliance on position placement rather than position itself,

H5 more frequent misclassification of postures on foreign subjects can be expected,

H6 the missclassification rate will increase with bigger difference in constitution and weight.

6.3 Experiment

The classification model was trained on **subject P**. The experiment consists of testing the model's functionality on subjects of various weights and constitutions, particularly **subject G**, **subject H**, **subject D** and **subject M**. See information about subjects in the table 6.2.

Subject	**Weight**	**Height**
Subject G	40kg	150 cm
Subject H	55kg	165 cm
Subject P	65kg	173 cm
Subject D	78kg	176 cm
Subject M	109kg	190 cm

Table 6.2: Subject weights and heights

Through testing, it was confirmed that even if the model performed well in the results, it does not mean that it is automatically suitable for online detection. Although in an offline environment 1-NN performed well, its absence of learning and high memory requirement makes it inappropriate to use in practice. It is also slow, compared to other classifiers, which is understandably an undesirable trait in online detection. The assumption **H1** was confirmed to be true. Even though in offline environment, 1-NN performed better than other k-NN where $k > 1$ in the offline results, using only 1 neighbor is dangerous as the algorithm will be much more prone to noise. It was also prone to overfitting to positions on the bed, e.g. even small shifts caused confusion for the model.

By increasing $k = 5$, its performance in online detection improved but it was still lacking in comparison to SVM.

Hypothesis **H2** proved to be true as SVM was able to detect all postures from the subject it was trained on. Moreover, it was also able to effortlessly recognize all postures of subjects G and subject H.

On the other hand, assumption **H3** was proven false as all three classifiers were able to differentiate between sitting and supine without much trouble. However, all of the classifiers seemed to be somewhat too dependent on the position placement. The classes that caused trouble with some subjects were sitting and right log and the problem seemed to be more apparent with subject M, misclassifying logs as sitting and vice versa. SVM seemed to cope with the issue the best, which might be due to its generalizing properties.

That confirms the theory **H4**. A higher number of strain gauges would allow a better grasp of underlying features. The introductory section 1.4 mentioned researches utilizing pressure mats with more than 1000 sensors, even allowing pressure point visualizations.

Even though **H5** is a fact, the experiments with SVM-trained model surprisingly reached a better result than expected. The model was still able to somehow detect positions of subject D and even subject M that was 44 kg heavier than subject P (although having significantly more difficulties to do so). Subjects that were smaller and lighter were recognized correctly even in an unusual setting. Compared to subject H and G, subjects D and M did not have the freedom to move around in positions without it resulting in misclassification, confirming the theory **H6**. There was no observation of height somehow affecting the classifier's performance.

The experiment showing the functionality of the online detection system on all subjects is recorded on a video attached to the thesis.

Chapter 7

Conclusion

The goal of the thesis was to build bases for the smart bed monitoring tool development, implement a real-time visualization tool and to create an online posture detection system.

Reaquiring the information about the data structure and communication was done through reverse-engineering the bed's existing software of data acquisition and thorough examination of documentations. The knowledge was then applied into the implementation of real-time visualization tools in a form of a **line plot** and **scatter plot**. The line plot was created to examine the signal behavior, while the scatter plot was developed to examine relations between attributes. The chapter also explores various approaches for **visualization in higher dimensions**. The online detection system was integrated into the scatter plot, showing the prediction at the top right corner of the figure.

Firstly, the classifiers were evaluated on subject H with 1-NN reaching the best all-over result of 97% accuracy. Since the classes are not perfectly balanced, the all-over accuracy can be somewhat misleading and so error rates for each class were computed. The class error rates consistently showed, that the most problematic classes were sitting and supine. The testing was done on subject P as well, again showing the best results for algorithm 1-nearest neighbour (1-NN). Interestingly, frequent overlapping of classes sitting and supine did not occur on the data set of subject P.

To test the online classification system, an experiment consisting of 5 subjects was conducted. The classification model was trained on a subject of average height and weight, subject P (65 kg) and tested on the rest. The experiment showed that model trained on SVM was able to effortlessly recognize postures of subjects G (40 kg) and H (55 kg), while clearly having more difficulties with subject M (109 kg). Nevertheless, the model was still somehow able to recognize most of the postures.

7.1 Future research

The future plans for the research are to continue working on various activity detection problems, like the ones mentioned in the section 5.1. Due to the newly acquired independence from Linet in terms of data collection, the research can now easily implement different approaches that were previously not suitable. It might be even possible to finally utilize the bed's technology to also enforce an automatic response or trigger an alarm.

In terms of the thesis, the future intentions are to develop a better object-oriented design for the current scripts and to create a full-fledged user friendly software. That would hopefully help to create a tool for new projects that will one day contribute to the scientific community.

Bibliography

[1] D. Sánchez, M. Tentori, and J. Favela, "Activity recognition for the smart hospital", *IEEE intelligent systems*, vol. 23, no. 2, pp. 50–57, 2008.

[2] H. H. Nguyen, "Advanced assistive control strategies for smart hospital beds", PhD thesis, 2016.

[3] A. Sivanantham, "Measurement of heartbeat, respiration and movements detection using Smart Bed", in *2015 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*, IEEE, 2015, pp. 105–109.

[4] A. M. Adami, M. Pavel, T. L. Hayes, and C. M. Singer, "Detection of movement in bed using unobtrusive load cell sensors", *IEEE Transactions on Information Technology in Biomedicine*, vol. 14, no. 2, pp. 481–490, 2010.

[5] W. SpillmanJr, M. Mayer, J. Bennett, J. Gong, K. Meissner, B. Davis, R. Claus, A. MuelenaerJr, and X. Xu, "A 'smart'bed for non-intrusive monitoring of patient physiological factors", *Measurement Science and Technology*, vol. 15, no. 8, p. 1614, 2004.

[6] J. Hao, M. Jayachandran, P. L. Kng, S. F. Foo, P. W. A. Aung, and Z. Cai, "FBG-based smart bed system for healthcare applications", *Frontiers of Optoelectronics in China*, vol. 3, no. 1, pp. 78–83, 2010.

[7] L. Russell, R. Goubran, and F. Kwamena, "Posture detection using sounds and temperature: LMS-based approach to enable sensory substitution", *IEEE Transactions on Instrumentation and Measurement*, vol. 67, no. 7, pp. 1543–1554, 2018.

[8] R. Yousefi, S. Ostadabbas, M. Faezipour, M. Nourani, V. Ng, L. Tamil, A. Bowling, D. Behan, and M. Pompeo, "A smart bed platform for monitoring & ulcer prevention", in *2011 4th International Conference on Biomedical Engineering and Informatics (BMEI)*, IEEE, vol. 3, 2011, pp. 1362–1366.

[9] D. M. Smith, "Pressure ulcers in the nursing home", *Annals of internal medicine*, vol. 123, no. 6, pp. 433–438, 1995.

[10] S. Rus, T. Grosse-Puppendahl, and A. Kuijper, "Recognition of bed postures using mutual capacitance sensing", in *European Conference on Ambient Intelligence*, Springer, 2014, pp. 51–66.

[11] V. Metsis, G. Galatas, A. Papangelis, D. Kosmopoulos, and F. Makedon, "Recognition of sleep patterns using a bed pressure mat", in *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*, ACM, 2011, p. 9.

[12] 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.

[13] S. Ostadabbas, M. B. Pouyan, M. Nourani, and N. Kehtarnavaz, "In-bed posture classification and limb identification", in *2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings*, IEEE, 2014, pp. 133–136.

[14] Linet s.r.o. (2015). Company Profile, [Online]. Available: http://www.linet.com/en/about-us/company-profile (visited on 02/25/2019).

[15] Hannah, R.L. and Reed, S.E. and Society for Experimental Mechanics (U.S.), *Strain Gage Users' Handbook*. Elsevier Applied Science, 1992, ISBN: 9780912053363. [Online]. Available: https://books.google.cz/books?id=ig64QgAACAAJ.

[16] Omega Engineering, Inc., *The pressure strain and force Handbook*, ser. Transactions in measurements and control. Omega Engineering, Inc., 1999, vol. 3. [Online]. Available: https://www.omega.com/literature/transactions/volume3/trantocvol3.html.

[17] JetBrains s.r.o., *dotPeek: Free .NET Decompiler and Assembly Browser*, version 2018.3.3, Feb. 27, 2019. [Online]. Available: https://www.jetbrains.com/decompiler/.

[18] C. Liechti. (2017). pySerial 3.4 Documentation, [Online]. Available: https://pyserial.readthedocs.io/en/latest/pyserial.html (visited on 02/22/2019).

[19] LINET, *Manuál k nemocničnímu lůžku LINET*.

[20] W. Simpson. (Jul. 1994). PPP in HDLC-like Framing, [Online]. Available: https://tools.ietf.org/html/rfc1662#page-4 (visited on 02/22/2019).

[21] A. Tanenbaum and T. Austin, *Structured Computer Organization*. Pearson, 2013, ISBN: 9780132916523. [Online]. Available: https://books.google.cz/books?id=mOHHygAACAAJ.

[22] J. D. Hunter, "Matplotlib: A 2D graphics environment", *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[23] M. O. Ward, "A Taxonomy of Glyph Placement Strategies for Multidimensional Data Visualization", *Information Visualization*, vol. 1, no. 3-4, pp. 194–210, 2002. DOI: 10.1057/PALGRAVE.IVS.9500025. eprint: https://doi.org/10.1057/PALGRAVE.IVS.9500025. [Online]. Available: https://doi.org/10.1057/PALGRAVE.IVS.9500025.

[24] Y. Holtz. (© 2017). Bubble plot, [Online]. Available: https://python-graph-gallery.com/bubble-plot/ (visited on 03/09/2019).

[25] E. Alpaydin, *Introduction to Machine Learning*, ser. Adaptive Computation and Machine Learning series. MIT Press, 2014, ISBN: 9780262325752. [Online]. Available: https://books.google.cz/books?id=7f5bBAAAQBAJ.

[26] T. Ayodele, "Types of Machine Learning Algorithms", in. Feb. 2010, ISBN: 978-953-307-034-6. DOI: 10.5772/9385.

[27] R. Rifkin and A. Klautau, "In defense of one-vs-all classification", *Journal of machine learnnig research*, vol. 5, no. Jan, pp. 101–141, 2004.

[28] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006, pp. 338–339.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[30] H. Zhang, "The optimality of naive Bayes", *AA*, vol. 1, no. 2, p. 3, 2004.

[31] S. Shai and B. Shai, *Understanding machine learning from theory to algorithms*. Cambridge University Press, 2014. [Online]. Available: https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/understanding-machine-learning-theory-algorithms.pdf.

[32] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, 10. Springer series in statistics New York, 2001, vol. 1.

[33] P. S. Foundation. (2019). Pickle: Python object serialization, [Online]. Available: https://docs.python.org/3/library/pickle.html (visited on 04/20/2019).

Appendix A

Scatter plots and tables

Classifier	Accuracy
1-NN	99.3 %
5-NN	98.7 %
NB	97.4 %
DT	95.7 %
SVM	98.6 %

Table A.1: Total accuracy of classifiers on subject P

Procedure	
Upper-left edge	12s
Lower-left edge	12s
Lower-right edge	12s
Upper-right edge	12s
Middle	12s
Empty	30s

Table A.2: Altered data set acquisition procedure for sitting

By following the procedure A.2, 1 minute of the sitting posture can be acquired and 30 seconds of empty bed. This procedure needs to be repeated 5 times.

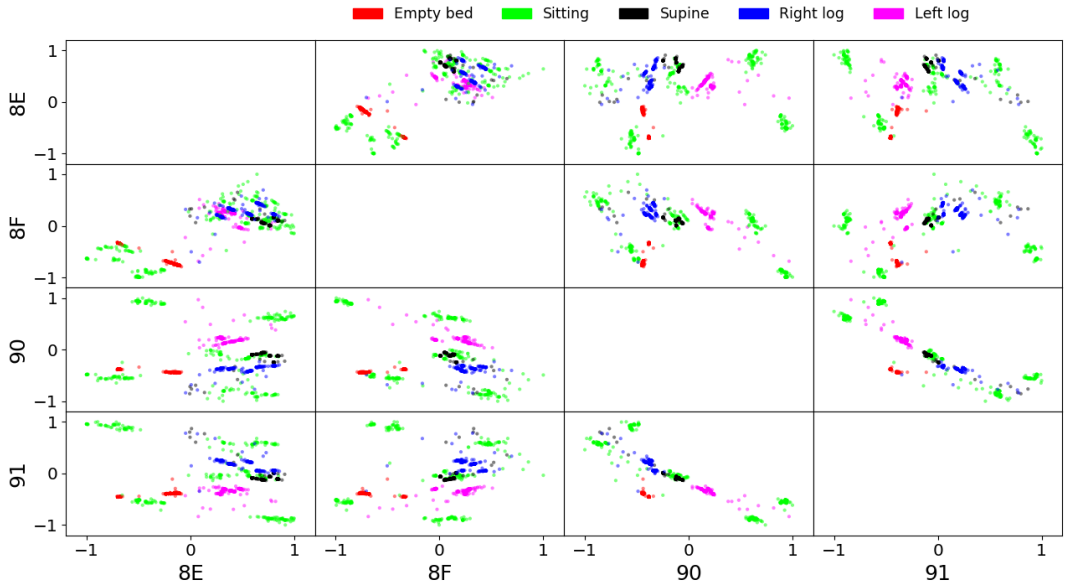

Figure A.1: Scatter matrix of strain gauges (subject H)

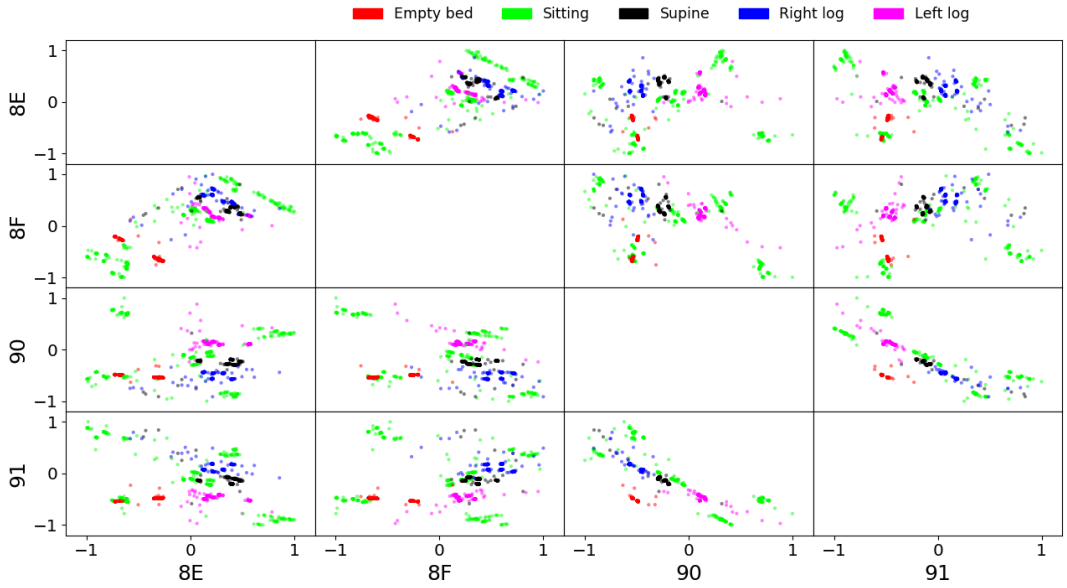

Figure A.2: Scatter matrix of strain gauges (subject P)

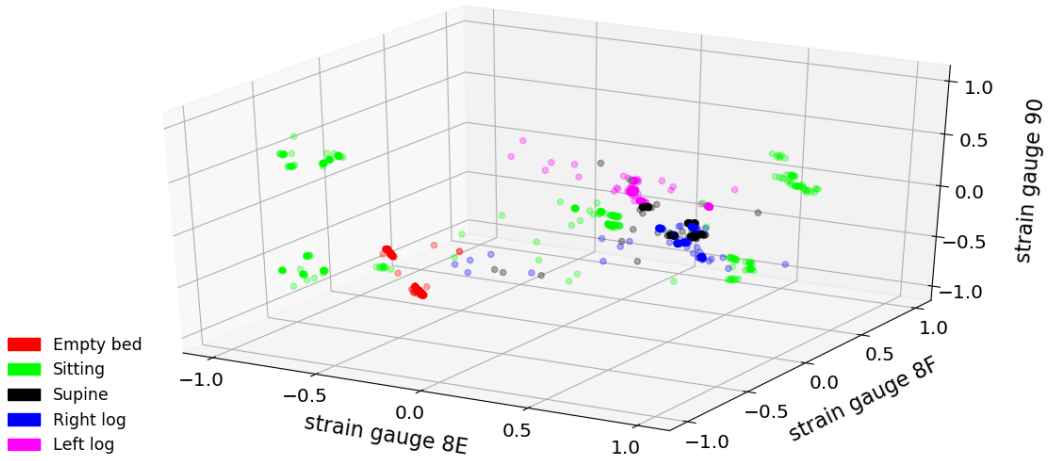

Figure A.3: 3D Scatter plot of strain gauges (subject P)

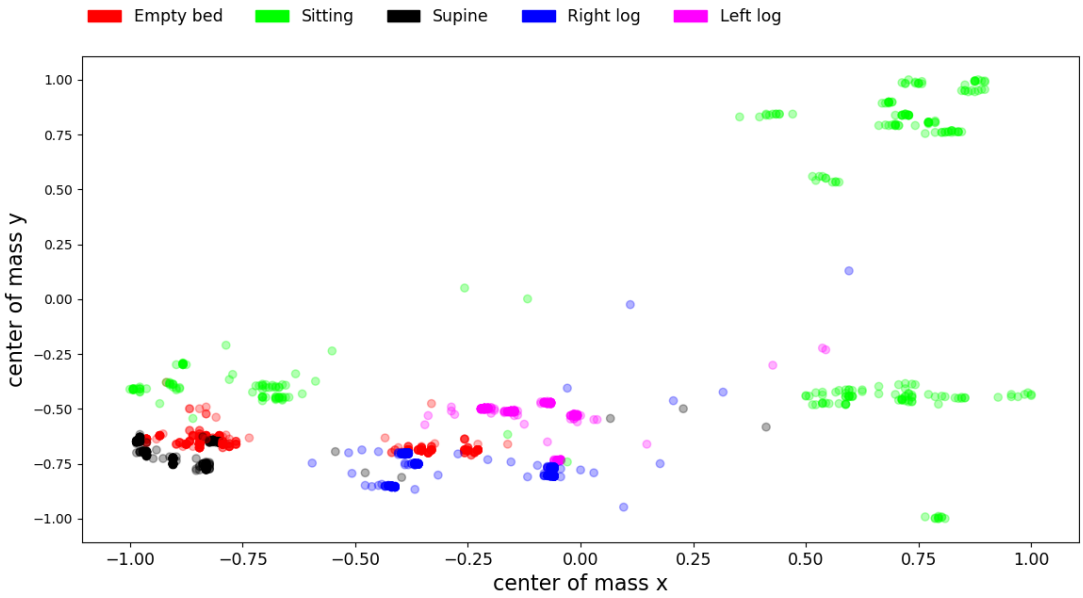

Figure A.4: Scatter plot centers of mass (subject P)

Appendix B

List of attachments

`experiment.mp4` – the recording of the experiment described in chapter 6

`scatter-plot.py` – the scatter plot implementation, contains prediction and classification model

`line-plot.py` – implementation of the line plot

`normalize.py` – normalization for classification purposes

`classification.py` – classifier model generation script.