UNIVERSITÉ DU LUXEMBOURG

PhD-FSTC-2019-67
The Faculty of Sciences, Technology and Communication

DISSERTATION

Defence held on 31/10/2019 in Esch-sur-Alzette

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Gilles NEYENS
Born on 26 July 1991 in Luxembourg (Luxembourg)

CONFIDENCE-BASED DECISION-MAKING SUPPORT FOR MULTI-SENSOR SYSTEMS

Dissertation defence committee

Dr Denis Zampunieris, dissertation supervisor
Professor, Université du Luxembourg

Dr Jens Weber
Professor, University of Victoria, Canada

Dr Leon Van der Torre, Chairman
Professor, Université du Luxembourg

Dr Anthony Cleve
Professor, Université de Namur, Belgique

Dr Nicolas Navet
Professor, Université du Luxembourg

Abstract

We live in a world where computer systems are omnipresent and are connected to more and more sensors. Ranging from small individual electronic assistants like smartphones to complex autonomous robots, from personal wearable health devices to professional eHealth frameworks, all these systems use the sensors' data in order to make appropriate decisions according to the context they measure.

However, in addition to complete failures leading to the lack of data delivery, these sensors can also send bad data due to influences from the environment which can sometimes be hard to detect by the computer system when checking each sensor individually. The computer system should be able to use its set of sensors as a whole in order to mitigate the influence of malfunctioning sensors, to overcome the absence of data coming from broken sensors, and to handle possible conflicting information coming from several sensors.

In this thesis, we propose a computational model based on a two layer software architecture to overcome this challenge.

In a first layer, classification algorithms will check for malfunctioning sensors and attribute a confidence value to each sensor. In the second layer, a rule-based proactive engine will then build a representation of the context of the system and use it along some empirical knowledge about the weaknesses of the different sensors to further tweak this confidence value.

Furthermore, the system will then check for conflicting data between sensors. This can be done by having several sensors that measure the same parameters or by having multiple sensors that can be used together to calculate an estimation of a parameter given by another sensor. A confidence value will be calculated for this estimation as well, based on the confidence values of the related sensors.

The successive design refinement steps of our model are shown over the course of three experiments. The first two experiments, located in the eHealth domain, have been used to better identify the challenges of such multi-sensor systems, while the third experiment, which consists of a virtual robot simulation, acts as a proof of concept for the semi-generic model proposed in this thesis.

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor, Professor Dr. Denis Zampunieris, for having given me the opportunity to participate in his research and teaching team during the last years and for his advice and guidance during countless meetings.

I would also like to thank the members of my CET, Leon van der Torre and Philippe Lalanda, for their valuable advice and guidance during the last few years.

Furthermore, I would like to thank the professors Leon van der Torre, Nicolas Navet, Anthony Cleve and Jens Weber for agreeing to be part of my thesis committee.

I also express my thanks to Sandro Reis for his technical and moral support during my PhD.

Finally, I would like to thank my parents, Christian Grévisse, Remus Dobrican and my other friends for their moral support during the last years.

Contents

List of Figures

List of Tables

1 Introduction

Contents

1.1 Context of the thesis

Over the last years, sensors have become omnipresent. They are used in robots but also in the ever-growing market of mobile phones and wearable devices[2] as well as every day devices like coffee machines. Not only are sensors omnipresent, they also become more and more precise[3]. However, this improvement in precision only affects the base precision in normal circumstances. Situations in which the accuracy of the sensors get affected by external sources still present a challenge.

Consider the example of a system with a single sensor to monitor a given parameter. If there is an external influence that affects the accuracy of the sensor, the whole system becomes unreliable and, contrary to the case of a completely failing sensor, the situation cannot be helped by adding more sensors of the same type as they would all be malfunctioning due to the external influences.

A solution could be to use sensors of different types with different weaknesses to external influences and to detect the contextual situation of the system in order to decide which of the sensors to use. In some cases it may even be possible to calculate an estimation of the parameter given by a sensor based on the data of several other sensors.

To give a more concrete example, consider a robot navigating through an urban environment that has to follow a series of checkpoints using its GPS. As high buildings may influence the accuracy of a Global Positioning System (GPS) by up to 100m as shown in the study conducted in Chicago presented in [4], this may result in the robot being unable to correctly navigate through the city. In this case, the base accuracy of the sensor is irrelevant. However, the accuracy of the current position of the robot can be improved by detecting the contextual situation the robot is in and by calculating an estimation of the current position based on past valid data and other sensors like the inertial unit and accelerometer.

With these examples in mind, we define our research questions in the next section.

1.2 Research questions and objectives

Nowadays, systems generally use data coming from multiple sensors in order to make decisions and rarely rely on the use of a single sensor. Analysing the reasons the systems do not use a single sensor to make decisions anymore could help to further improve the decision-making. Therefore, our first research question reads as follows:

RQ1: What challenges arise when only using a single sensor for decision-making and how could they be solved?

The answer to this question does not seem to be that complicated at first. Using more sensors gives access to more data which in turn will lead to better decisions, so if the only sensor in the system fails, the system receives no data and thus cannot make good decisions. Additionally, using a single sensor limits the amount of information at the disposal of the system to make an informed decision. However, if you analyse the behaviour of the system in more detail, it becomes clear that the use of a single sensor makes handling noise tricky.
Using multiple sensors can help to overcome some of these problems, but is not necessary enough as shown by some eHealth systems that use multiple sensors but make a diagnosis based on each sensor individually [5]. Initial experiments that were done with an early version of our system in "Using hidden markov models and rule-based sensor mediation on wearable ehealth devices" [6] showed some of the limitations of using a single sensor. These initial experiments are presented in Sections 3 and 4. The topics discussed lead us to the next question.

RQ2: How can the use of multiple sensors help to improve decision-making?

Sensor fusion methods can be used to combine data coming from multiple sensors. The fusion methods can help handle noise in the data and partly compensate for failing sensors. This leads to better decisions. However, the existing techniques still have some drawbacks, as they have difficulties to handle conflicting information coming from different sensors [7]. These drawbacks and challenges will be discussed in more detail in Section 2.5 and Section 4 and we formulate the next question as follows:

RQ3: How can conflicting information between sensors be identified and how can resolving these conflicts help to improve the decision-making process?

This question was explored in the papers "Conflict handling for autonomic systems" [8] and "A Rule-Based Approach for Self- Optimisation in Autonomic EHealth Systems" [9]. The idea is to have a system architecture with multiple layers. In a first layer, machine-learning techniques analyse the data from each sensor in order to assign confidence values to a sensor, where a high confidence indicates that the sensor is working correctly and a low confidence indicates that the sensor is failing or malfunctioning.
However, this step will only detect the more obvious cases of malfunctioning sensors. Therefore, in a second layer, we will use the

rule-based proactive engine developed by Prof. Zampunieris in order to run expert scenarios based on contextual information gathered from the sensors (See Section 2.3 for more information on the proactive engine). In the second layer, the confidence values of the sensors are adapted continually and dynamically based on the contextual information, conflicts are detected and resolved. In addition, the final confidence level for the sensors along with potentially improved data is passed to the final step in order to make a decision.
It would also be interesting to see if it is possible to standardize a part of the second layer to simplify the development of systems with different sensors. Therefore, the final research question is:

RQ4: To what extent can the proposed solution, i.e. attributing confidence values to sensor data based on the system context, be standardised?

While the first layer will certainly be very domain and sensor specific, it could be possible to standardise a part of the second layer in such a way that an expert introducing a set of parameters would be enough to adapt the second layer for a new system.
The last two questions will be discussed in Sections 5 and 6 and were the topic of two publications ("Proactive Middleware for Fault Detection and Advanced Conflict Handling in Sensor Fusion" [10] and "Proactive Model for Handling Conflicts in Sensor Data Fusion Applied to Robotic Systems" [11]).

1.3 Thesis structure

In this section, we will outline the structure of this thesis and when applicable, cite the papers about this work that got accepted by the scientific community. First in Chapter 2 we will give the needed background information about the state-of-the-art and related work concerning this thesis. The topics presented include expert systems, context-based reasoning, autonomic computing, proactive computing, sensor fusion and conflict handling in sensor fusion.

In Chapter 3 we will then study a single sensor system in the eHealth domain and identify the drawbacks using a single sensor has for the decision making process of a system. The content of this chapter was published as the paper "Using hidden markov models and rule-based sensor mediation on wearable eHealth devices" [6].

Chapter 4 is similar to the previous chapter, only that the single sensor system will be switched out for a multi-sensor system. The advantages of the multi-sensor system over the single sensor system will be discussed, as well as the challenges that arise with using multiple sensors. The topics of

this chapter were discussed and published in "A Rule-Based Approach for Self-Optimisation in Autonomic EHealth Systems" [9].

The challenges identified will then be used in Chapter 5 in order to guide the development of the architecture of our system. The evolution of the system structure can be followed in the publications "A Rule-Based Approach for Self-Optimisation in Autonomic EHealth Systems"[9], "Conflict handling for autonomic systems" [8] and "Proactive Middleware for Fault Detection and Advanced Conflict Handling in Sensor Fusion" [10].

In Chapter 6, we will then apply our system to a robot in a simulation environment and show a proof a concept application that shows that our system helps the robot to navigate better through an environment that affects the accuracy of its sensors. The related paper is "Proactive Model for Handling Conflicts in Sensor Data Fusion Applied to Robotic Systems" [11].

Finally, in Chapter 7 we will take a look back at the achievements of this thesis and provide some ideas for future work.

2 State of the art

Contents

In this chapter, we are going over the state of the art of the concepts which are relevant for the work done in this thesis. In the first section, we will talk about reasoning, how expert systems and agents reason, and about context-based reasoning. In the subsequent sections we will then introduce the notions of autonomic computing and proactive computing and outline the differences between the two. Finally, we will introduce different sensor fusion methods and take a look at their conflict handling capabilities.

2.1 Reasoning in computer science

Reasoning in computer science is omnipresent. The type of reasoning can range from very basic "if else" statements to more complex strategies of choosing the right decision. These decisions are obtained by applying reasoning strategies and algorithms to knowledge that a system has. According to Mylopoulos and Levesque, this knowledge can be represented in 4 different ways: logical representation, procedural representation, networked representation and structural representation[12, 13]. The most relevant knowledge representation for the work in this thesis is the procedural representation, which includes rule-based production systems that can be used to implement expert systems presented in the next section.

2.1.1 Expert systems

Before the recent achievements of Google's DeepMind AlphaGo beating grandmaster Lee Sedol 4-1 after already defeating the european Go champion by 5-0 [14], expert systems took the spotlight in these kind of challenges. In 1997 IBM's Deep Blue won against the acting world champion Gary Kasparov[15], where IBM gave credit to expert systems for the opening moves of the system[16]. In 2011, IBM developed another expert system called Watson that was able to defeat two of the best Jeopardy! players in the world [17].

These expert system are rule-based and use rules in order to capture the knowledge from human experts. Some authors do not differentiate between production systems, rule-based systems and expert systems [18]. However, some differences between the aforementioned types of systems are sometimes considered. Both rule-based systems and production systems can provide a framework to create an expert system, but they are not necessarily expert systems. An example of a production system that is not an expert system is an email client that allows the user to add rules to manage his emails.

On the other hand expert systems do not necessarily have to be implemented using rules. Other approaches listed in [19] are case-based systems, neural networks or a combination of the different approaches. In order for

a rule-based system to be considered an expert system, it has to have received some domain specific knowledge from a human expert. The final goal of an expert system is to be able to replace an expert in the domain it was designed for[20]. For the rest of this section, we are focusing on rule-based expert systems.

A rule-based expert system generally consists of a collection of rules also called knowledgebase, a collection of facts also called working memory and an inference engine[21]. In some cases, they might only be composed of a collection of rules and an inference engine[22]. The collection of rules can be considered to be long term knowledge which can be used to change the short term knowledge contained in the collection of facts[13].

In order to ease the development of expert systems, Expert system shells like the well-known Java Expert System Shell (Jess) [23] were created. They provide an user interface in addition to the inference engine and allow users the easy creation of rules in order to develop their expert system. Additional functionality includes the choice between different knowledge representations and between different reasoning techniques.

In Section 2.3 we will present our own implementation of a proactive rule-based engine that we will use for the work done in this thesis.

2.1.2 Agents

Agent is a general term which covers multiple domains. The term of agent has been defined in different ways by many different authors[24][25][26]. It can be seen as a system that is able to replicate human intelligence which is why authors sometimes talk about intelligent agents. However, a common definition of the term has not been agreed upon.

One such definition is given by Woolridge in [25], who defines agents as systems that are autonomous, social, reactive and proactive. In Section 2.3, we will discuss how this definition of agent and its proactive property compares to the concept of proactive computing.

In [24], the authors provide a very general definition of the term agent, i.e. an agent is an entity that uses sensors to perceive the environment around it and uses effectors to act upon said environment. Additionally, some authors make further distinctions between normal agents and autonomous agents[27]. But here again, the definition of autonomous can vary from author to author. For the authors in [28], autonomous means that the agent can perform its actions without implicit intervention from a user or other program. In [29], the authors define autonomous as the ability to fill out the gaps caused by partial or incorrect prior knowledge. Yet another definition says that an autonomous agent can decide to perform a task without being affected by outside influences[30]. Coming back to the definition of an agent, the authors in [26] consider there to be two main properties that computer systems need to satisfy in order to be considered

an agent. They are the perception of the environment and temporal continuity.

As the definitions given until now would allow anything or anyone that is able to take a decision on its own to be an agent, the author in [31] proposed to separate agents into three categories: physical or tangible agents which include agents like robots, natural agents like humans or animals and software agents. To distinguish software agents from simple software programs the authors in [29] listed other characteristics that a software agent needs to have like autonomous control, perception of the environment, adaptiveness and persistence over long periods of time.

To be able to properly satisfy the property of environment perception, the system needs to be aware of the context it currently is in and needs to know how to reason about it, which we will cover in the next section.

2.1.3 Context-based reasoning

Expert systems and especially agents can be used in a wide range of different environments. As these environments are evolving over the life time of a system, the systems have to be able to react to changes in the environment and to do this they have to be aware of the current context they find themselves in.

The term context-awareness was first coined by Schilit and Theimer in 1994[32] in their work about mobile distributed computing. Schilit and his co-author considered the context to be the location and identities of nearby people and objects and changes to these objects. In other works context has been simply defined as the environment or situation of the system[33, 34, 35]. In another paper, Schilit listed three important aspects of context: where you are, who you are with, and what resources are nearby[36]. He considered context to be about more than just the location of the user or system as there are other things of interest around the system that are also evolving and changing. In the case of his work, context would thus also encompass lighting, noise level, network connectivity, communication costs, communication bandwidth, and the social situation the user finds himself in.

In his work in 2001[37], Dey gave another definition of context "Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

Dey also provided a definition for context-awareness of a system [37]: "A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task."

In this thesis, whenever we talk about context, we mean Dey's defini-

tions with the clarification that a user does not have to be a human user but can also represent another system.

The term context-based reasoning goes back to Gonzalez and Ahlers[38]. Context-based reasoning is a reasoning paradigm that allows for intelligent agents to be modeled for use in a variety of environments and scenarios where specific expertise is necessary[39]. The main idea behind this paradigm is that people tend to only use a fraction of their knowledge at any given time[38].

This brings us to the proactive computing paradigm and the proactive engine (PE), which is a rule-based system, used in this work. In this system, different contextual situations are represented by different scenarios, a scenario being a set of rules. This allows the system to reduce the amount of knowledge it has to use simultaneously, meaning that the set of rules that can get activated at a given time depends on the current context of the system. We will explain the detailed functioning of the proactive engine in section 2.3.

2.2 Autonomic computing

In 2001, forecasts by researchers from IBM predicted that computing devices would continuously increase in complexity[40]. This complexity does not encompass only the complexity of creation and development but also the complexity of operation, maintenance and management. Up to this point, these tasks were carried out solely by highly skilled and trained humans, however the rate of complexity increase pushed researchers from IBM to propose a new approach to address the problem: Autonomic computing [41].

In short, an autonomic system (AS) is a system that will manage itself based on the current context[42], while the task of the human in charge is reduced to setting general directives and policies to guide the system. Note that autonomic is not the same as autonomous, which describes systems that reach a certain goal without user intervention.

The requirements that a system needs to meet in order to be considered autonomic will be discussed in the next section.

2.2.1 Self-* properties

In their manifesto, the researchers from IBM listed eight characteristics that an autonomic system needs to have, which they later summarised by four fundamental properties also called self-* properties: [41, 43]:

- Self-configuration
  The system is capable of adapting its internal parameters to react to dynamic environmental changes while still meeting initial deployment requirements.

- Self-healing
  The system is able to identify failed components as well as repairing them in order to ensure maximum availability.
- Self-optimisation
  The system continuously tries to optimise its operation with respect to predefined goals. These goals may vary drastically depending on the system, ranging from optimising energy consumption to optimising response speed.
- Self-protection
  The systems needs to preserve its security and integrity by identifying threats and neutralizing them. It also has to anticipate threats.

Over the years this list of properties has been extended with several optional properties. Following is a selected list of these properties[43].

- Self-adapting
  The system needs to modify itself and/or its internal parameters to react to changes in the current external context in order to ensure that its objectives are still being met despite these external changes.
- Self-anticipating
  The system needs to be able to predict future events. An anticipating system should be able to manage itself proactively.
- Self-aware
  The system needs to possess knowledge of its internal elements, their current status, history, capacity and connections to external elements or systems. This can also include knowledge of the actions the system may perform.
- Self-monitoring
  The system needs to be able to retrieve information on its internal state and behaviour, be it globally, or for any of its elements.

2.2.2 Structure of autonomic systems

An AS consists of one or several autonomic elements that get information from the usage (persons or other systems interacting with the AS) and computing context (resources of the AS) and are guided by high-level policies set by human administrators. They are also required to provide understandable feedback to its administrator. An autonomic element is structured according to the architecture proposed by IBM [44, 43] and shown in Figure 2.1. It has 2 components: The managed resources or artefacts and the autonomic manager.

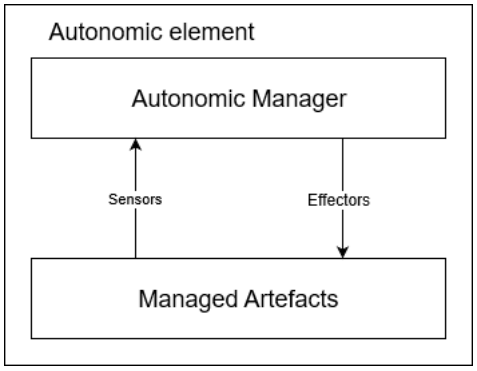

Figure 2.1: Architecture of an autonomic element

The autonomic manager is in charge of managing the resources allocated to it. Sensors provide the autonomic manager information about the resources while effectors allow the autonomic manager to act on the resources. The autonomic manager implements the MAPE-K loop in order to transform the high-level policies set by the administrators to direct actions to be performed on the managed resources [45]:

1. Monitor
   During monitoring, the system collects details from the resources and tries to detect any irregularities that may need to be analyzed.
2. Analyze
   During the analysis step, the system reasons about the irregularities that were detected during the monitoring step. If it determines that changes in the system are required, the necessary request is passed to the planning step.
3. Plan
   During the planning, the system determines and organizes the actions needed to address the issues discovered previously while still allowing itself to achieve its goals and objectives.
4. Execute
   The execute step executes the actions that were organized in the previous step.
5. Knowledge
   Knowledge designates the data shared among the monitor, analyze, plan and execute steps as well as additional information about the system such as historical logs, metrics and policies.

While autonomic systems, like proactive systems, originated around the same time and have some similarities, they also have some differences that we will cover in the next section after the presentation of the concept of proactive computing.

2.3 Proactive computing

The notion of proactive computing, has initially been introduced by Tennenhouse [46] in 2000. In proactive computing, the human user is no longer the center of the interaction between humans and computers but takes the role of a supervisor, which watches over the actions executed by a proactive system (PS). A PS thus does not rely on explicit user input and can act on its own initiative [46] to react to other events or proactively interpret the lack of user input. To achieve this, PSs need to be aware of their current context, extract the relevant information for their tasks from it and then react accordingly [47]. The idea of proactive computing lead to the development of a rule-based proactive engine (PE) by professor Zampunieris and his team, which was used in a multitude of projects at the University of Luxembourg, mainly in the domains of E-Learning, cognitive science and eHealth[48, 49, 50, 51].

2.3.1 The proactive engine

The proactive engine is the implementation of a rule-based proactive system that is used in the work of this thesis. It regroups both the strength of Object-oriented principles and the strength of rule-based systems. The PE is a middleware system that can be attached to other systems either directly or through the use of a common database. The proactive engine consists of a rule-engine, a database and rules.

2.3.1.1 Rule-engine and rule structure

The rule-engine is responsible for executing the proactive rules, which is done in iterations. This rule-engine is composed of two First in First out (FIFO) queues called currentQueue and nextQueue. The currentQueue contains the rules that need to be executed at the current iteration of the rule-engine, while the nextQueue contains the rules that were generated during the current iteration. At the end of each iteration the rules from the nextQueue will be added to the currentQueue and the next Queue will be emptied.
A rule consists of any number of input parameters and five execution steps [52]. These five steps have each a different role in the execution of the rule.

1. Data acquisition
   During the data acquisition step, the rule gathers data that is important for its subsequent steps. This data is provided by the context manager of the proactive engine, which can obtain this data from different sources such as sensors or a simple database.

```
1.  repeat for each data acquisition request DA
  a.  perform DA
  b.  if error then
      raise exception on system manager console and go to step 7
      else
      create new local variable and initialize it with the
          result of DA
2.  create new local Boolean variable "activated" initialized to
     false
3.  repeat for each activation guard test AG
  a. evaluate AG
  b. if result == false then go to step 6
     else if AG == last activation guard test
     then activated = true
4.  repeat for each conditions test C
  a. evaluate C
  b. if result == false then go to step 6
5.  repeat for each action instruction A
  a. perform A
  b. if error then raise exception on system manager console and
       go to step 7
6.  repeat for each rule generation R
  a. perform R
  b. insert newly generated rule as the last rule of the system
7.  delete all local variables
8.  discard rule from the system
```

Figure 2.2: The algorithm to run a rule

2. Activation guards
   The activation guards will perform checks based on the context information whether or not the conditions and actions part of the rule should be executed. If the checks are true, the activated variable of this rule will be set to true.
3. Conditions
   The objective of the conditions is to evaluate the context in greater detail than the activation guards. If all the conditions are met as well, the Actions part of the rule is unlocked.
4. Actions
   This part consists of a list of instructions that will be performed if the activation guards and condition tests are passed.
5. Rule generation
   The rule generation part will be executed independently whether the activation guards and condition checks were passed or not. In this section the rule creates other rules in the engine or in some cases just clones itself.

During an iteration of the rule-engine, each rule is executed in turn. The

algorithm to execute a rule is presented in Figure 2.2. The data acquisition part of the rule is run first. If the data acquisition fails none of the other parts of the rule is executed. Upon successful data acquisition, the activation guards part is executed and evaluated. If the tests pass, the conditions part is executed. If again all the tests of the conditions part pass the actions part of the rule is executed. Finally, the rule generation part of the rule is executed independent of whether the activation guards or conditions tests were passed.

2.3.1.2 Scenarios, Rules and Meta-Rules

The rules described in the previous section can be regrouped into scenarios. A Scenario is a set of rules that will be executed for a given event. Generally, a scenario conceptually regroups rules that are necessary to react to or proactively act in a specific situation. The starting point of a scenario is called a meta-rule, which is a context-aware continuous never-ending rule[53] that decides when the rest of the scenario should get activated.

2.3.1.3 The database

The database is used to save the state of the proactive system and in some cases to communicate with a third party system the proactive system is attached to. It contains all the information necessary to recover the last state of the proactive system in case of a crash, contains the historical data from sensors and the results obtained by executing the rules that were running on the proactive engine.

2.3.2 Proactive computing vs Autonomic computing

The two paradigms of proactive computing and autonomic computing have been compared by their creators at Intel Roy Want, Trevor Pering and David Tennenhouse [54]. The main difference highlighted is that an autonomic system seems to be more oriented towards self-management, whereas a proactive system is more dedicated to user-related context activity. According to Want et al., the two computing approaches can complete each other, if integrated within one framework. In the case of proactive systems, this user can also be another computer system that uses the services provided by the proactive system.

2.3.3 Proactive computing vs Agents

A proactive system might be seen as a software agent in some cases as both share some properties like context-awareness and adaptiveness. However, a proactive system missing some properties listed in Section 2.1.2 like long-time persistence and autonomy as the proactive system is specific to the

environment it is executed in and does not necessarily react well to unforeseen situations as it executes the actions that are incorporated in its rules[13].

2.4 Sensor fusion

In the past years, several sensor fusion methods have been used to aggregate data coming from different sensors. Depending on the level of fusion, different techniques have been used. In robotics the most popular method for low-level data is the Kalman filter which was developed in 1960[55] along with its variants the extended Kalman filter or the unscented Kalman filter who are often used in navigation systems[56, 57], but also other methods like the particle filter. On a decision level other methods like the Dempster Shafer theory[58, 59] are used. But what actually is sensor fusion?

Before we discuss sensor fusion itself we will first introduce the notion of uncertainty that comes with sensors. Uncertain data is data that contains noise. The origin of this noise can vary. It can occur due to malfunctioning or failing sensors, external attacks or due to parameters in the environment that influence the readings on the sensors. For this work we will not cover external attacks but rather focus on malfunctioning sensors, in particular if this malfunction is caused by the environment the system resides in.

When it comes to fusion system, like often in computer science, there is some confusion regarding the terminology. Depending on the scientist, the terms sensor fusion, data fusion,information fusion and multi-sensor data fusion are used interchangeably[60][61]. Some scientists consider data fusion to be the overall term for fusion systems and define data fusion as a formal framework that comprises means tools for the alliance of data originating from different sources [60][62]. Others proposed to use information fusion as the overall term for fusion systems[63].

Here we are going to use the definition given in [60] for both the information fusion and the sensor fusion terms:

"Information Fusion encompasses theory, techniques and tools conceived and employed for exploiting the synergy in the information acquired from multiple sources (sensor, databases, information gathered by human, etc.) such that the resulting decision or action is in some sense better (qualitatively or quantitatively, in terms of accuracy, robustness, etc.) than would be possible if any of these sources were used individually without such synergy exploitation."

"Sensor Fusion is the combining of sensory data or data derived from sensory data such that the resulting information is in some sense better than would be possible when these sources were used individually."

Furthermore, in sensor fusion, the data sources for a fusion process do

not need to come from homogeneous sensors. A difference is made between direct and indirect fusion. In direct fusion the data from homogeneous or heterogeneous sensors and past sensor data are used in the fusion process while in indirect fusion a priori knowledge about the environment and human input are also used[60].

When discussing about the number of sensors that should be used in a fusion system, some scientists argue that adding a sensor only helps the fusion system if the data of that sensor is of good quality[60] while others try to prove that more sensors means better fusion results[64].

Several fusion methods and techniques exist. One of them is the Kalman filter named after the Hungarian Rudolf E. Kálmán who created it in 1960[55]. One of the main difficulties in implementing the Kalman filter is the estimation of the noise covariance matrices Q_k and R_k. So not only have these matrices either be manually adjusted or be trained somehow, they are also not suited in variable noise environments. Extensions to the Kalman filter that allow for variable noise covariance matrices have been developed recently[65], but they still retain the same difficulties of actually obtaining acceptable noise covariance matrices.

Nevertheless, Kalman filters see a lot of applications to improve the estimation of the position of a robot or human as for example in [66] where the authors were fusing data from wifi, smartphone sensors and landmarks in order to improve indoor localisation. In [67], Kalman filters were used to slightly improve the GPS navigation in urban environments.

Another popular fusion technique due to its flexibility is the particle filter. It is a recursive implementation of the Sequential Monte Carlo algorithm [68][69] and have recently been used for pose estimation in capsule robots[70], assisting visually impaired people using image and radar data to detect obstacles [71] or in heart rate monitoring [72] to reduce artefacts introduced by motion of the patient. However, this flexibility comes at the price of being computationally more intensive then the Kalman filter[73].

A higher level fusion technique is the Dempster-Shafer theory of evidence was first introduced by Arthur P. Dempster[58, 74] and later extended by Glenn Shafer into a more general framework [59]. It has been applied to landslide detection using airborne laser scanning data[75], to the detection of traffic incidents[76] or to the detection of DDoS attacks[77].

These different techniques can be used to improve data in noisy environments, but in order to be able to handle conflicting data and to avoid counter-intuitive results additional steps need to be taken[73].

2.5 Conflict handling for sensor fusion

As seen in the previous section, sensor fusion methods can combine data from multiple sensors. However, information coming from different sensor

sources can be contradictory. If the data coming from different sources is contradictory, we will talk about conflicts or conflicting information for the remainder of this work. Conflicts in the context of this work are not high-level conflicting goals or policies of the system, but rather refer to inconsistent and contradictory information sources.

While the fusion methods described in the previous section perform very well in very good conditions, they can have problems when the information coming from different sensors is conflicting. In this section we will therefore take a closer look at the conflict handling capabilities of these different methods.

Kalman filter and particle filter can generally handle some noise and uncertainty in the data decently well, however they are very sensitive to outliers in the data that result in conflicting information[73]. A recent study partly addressed these issues by using a particle filter method in combination with recurrent neural networks[70]. This solution managed to correctly identify situations in which sensors were failing. In [78] the authors propose a fault tolerant architecture for sensor fusion using Kalman filters for mobile robot localization. The detection rate of the faults injected was 100%, however the diagnosis and recovery rate is lower at 60%. The study in [79] proposed two new extensions to the kalman filter, the Fuzzy Adaptive Iterated Extended Kalman Filter and the Fuzzy Adaptive Unscented Kalman Filter in order to make the fusion process more resistant to noise.

As it is possible to use Dempster's rule of combination to combine two independent sets of probability mass assignments, one would assume that they can be used very well for the fusion of multiple sensors. However, it is known for a long time that Dempster-Shafer theory can give unintuitive, illogical or contradictory results in cases of high and even sometimes low conflict [80, 81, 82].

In [81], Zadeh gives the following example to show Dempster's rule of combination giving counter-intuitive results in a case of high conflict:

A patient P is examined by two doctors A and B. Both doctors agree that the probability that P has a brain tumour is 0.01. However, doctor A thinks that P has meningitis with 0.99 probability while B thinks that P has a concussion with 0.99 probability.

Using Dempster's rule of combination would yield the result that patient P has a 100% chance of having a brain tumour, which both doctors thought was very likely to not be the case.

While Dempster-Shafer theory is widely used in the domains of sensor fusion/information fusion, and work has been done to try to overcome these counter intuitive results in cases of conflicting information [83, 84, 85], it still remains an open problem[86].

There are two different main approaches. The first approach is to modify the combination rule itself like done by Smet[87], Dubois[88] and Yager

[89]. While they generally performed better for conflicting information, the changes done broke other properties of the rule. The second approach is to pre-treat the information before applying the combination rule. One of the more recent works used a weighted combination method for conflicting information in multi-sensor data fusion by assigning credibility or trust values to sources of information[86]. These weights were then taken into account for the combination process. The authors showed that their method is more confident in its results than other methods.

One model proposed for conflict handling between input sources was by Uwe Mönks [90]. He proposes a two-layer conflict resolution model based on Dempster-Shafer set theory and Fuzzy set theory and inspired by a similar model from Li and Lohweg [91], in order to better estimate the state or health of the system.

3 eHealth systems with a single sensor

Contents

3.1 Introduction

In this first experiment, we explore single sensor systems and more precisely single sensor systems in eHealth. There exist several systems, that use a single sensor to make a diagnosis of a patient or to monitor a patients behaviour. Examples of such systems include the system described in [92] where the authors concentrate on electrocardiogram (ECG) data in order to detect cardiovascular diseases. Another similar approach in order to detect pulse loss based on blood pressure data is presented in [93]. In both of these examples, the systems concentrate on the data coming from a single electrocardiogram (ECG) or other health related sensor in order to make a diagnosis for the patient.

So, in this chapter we analyze how well a single sensor can be used under normal conditions for deciding in which state a patient is in. We present our system, the results obtained and discuss possible drawbacks of using only one sensor and make some observations which will help us in improving the system by using multiple sensors.

We will first present some related work that include single sensor eHealth systems. We will continue with giving a small introduction to Hidden Markov Models (HMMs). Then, we will present our system, the tests that were carried out and the results that were obtained and finish with a discussion.

3.2 Related work

The first experimental field that was used to test the system or parts of the system was the domain of eHealth. These eHealth systems have taken advantage of the advances in sensor technology in order to provide better services to patients. In this part, we present some existing eHealth systems that use only a single sensor.

One eHealth system that uses a single sensor is HeartToGo[94]. It can continuously monitor and analyse an ECG in real time in order to detect cardiovascular diseases. Another system that uses the ECG to detect cardiovascular diseases is described in [92], where the authors proposed a two smartphone-based wearable device to overcome the limitations of both Holter monitors and resting ECG machines. They underline the importance of being able to monitor the patient at home as the conditions may not be present during the hospital visits as they can show up only periodically. In [93], the authors present a system with similar objectives based on the data from a blood pressure sensor.In [95] and [96] an accelerometer sensor is used in order to monitor how well patients are recovering after a stroke.

Finally, the work in [51] was based on the proactive engine described

in Section 2.3. It allowed patients to be monitored constantly at home in a non-invasive way during a cardiac rehabilitation process[51].

3.3 Our single sensor eHealth system

3.3.1 Architecture

The system developed for this experiment [6] uses data coming from a single ECG sensor and try to detect 'bad' heartbeats. This is done by using a Hidden Markov Model (HMM) with the topology shown in Figure 3.1 [97]. The middle chain of states represents the normal cardiac cycle. The different states represent the P, PR, R, S , ST and T phases of the cardiac cycle (See Figure 3.2[98]). The other two sequences of states represent two types of faulty heartbeats, supraventricular and ventricular. The two states connecting these state sequences represent the phase in between heartbeats. As every state of a heartbeat does not just consist of a single data point or sensor reading, the states can also loop on themselves. Finally, at the end of the heartbeat, the last state of the model has a transition to loop back to it's initial state to handle the next heartbeat.

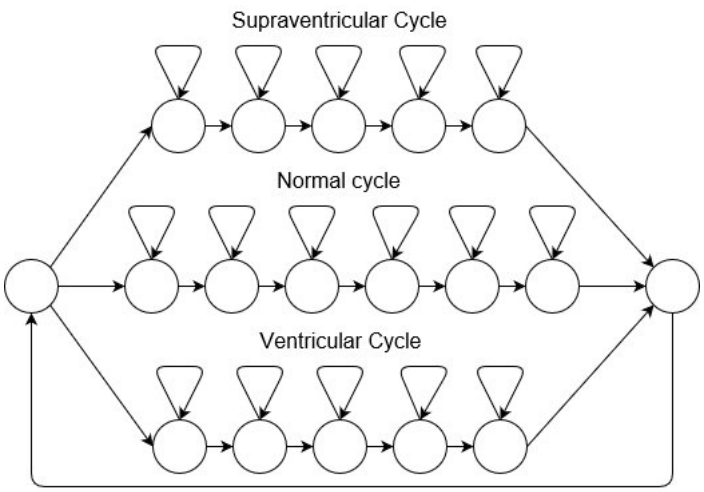

Figure 3.1: Topology of the HMM

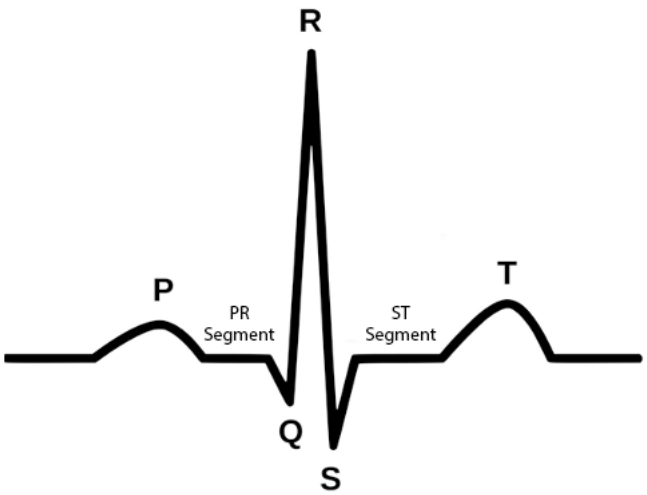

Figure 3.2: Normal heartbeat

3.3.2 Theoretical overview of Hidden Markov Models

Hidden Markov Models (HMMs) have been successfully used in many fields, be it for speech recognition [99][100][101], failure detection [102] or complex action recognitions [103].Different studies also used them for ECG [97][104][105] and respiration analysis [106]. In this section we will give a theoretical overview of HMMs.

An HMM models stochastic sequences as Markov chains where the states are hidden. HMMs consist of five parts [101] :

1. The number of states N in the model. Even though the states are hidden, they generally have a physical meaning. In the case of a patient, they can mean that the patient is in a low, medium, high or no risk state. We denote the individual states as S= $\{S_1, S_2, ..., S_N\}$ and the state at time t as q_t.
2. The number of distinct observation symbols. We denote the individual symbols as V= $\{v_1, v_2, ..., v_M\}$.
3. The state transition probability distribution A = $\{a_i j(k)\}$ where $\{a_i j(k)\} = P[q_{t+1} = S_j | q_t = S_j], \quad 1 \leq i, j, \leq N$.
4. The observation probability distribution B = $\{b_j(k)\}$ for every state j where $\{b_j(k)\} = P[v_k \text{ at } t | q_t = S_j], \quad 1 \leq j \leq N, 1 \leq k \leq M$.
5. The initial state distribution $\pi = \{\pi_i\}$ where $\pi_i = P[q_1 = S_i], \quad 1 \leq i \leq N$.

There are three fundamental problems for HMMS:

1. Given an observation sequence O= $O_1, O_2 ... O_T$ and a model $\lambda = (A, B, \pi)$, how do we compute $P(O|\lambda)$?

2. Given an observation sequence O= $O_1, O_2 ... O_T$ and a model $\lambda = (A, B, \pi)$, how do we find the most likely state sequence $Q = q_1, q_2, ..., q_T$)?
3. How do we optimise the model parameters A, B and π of the model λ in order to maximize $P(O|\lambda)$?

The solutions to the first and second problems can be both used for classification. With the solution to the first problem, we can calculate the probability that an observation belongs to a specific model. By doing this for different models, we can choose the model that yielded the highest probability to classify the observation sequence. Example:

We have an observation sequence O= $O_1, O_2 ... O_T$ and three trained models X,Y and Z representing three different classes, such that $\forall o \in O$, $o \in V_X \wedge o \in V_Y \wedge o \in V_Z$. Using the solution to the first problem we calculate the probability that O belongs to X, Y or Z: $P(O|X) = 0.4$, $P(O|Y) = 0.2$ and $P(O|Z) = 0.8$. As $P(O|Z) > P(O|X)$ and $P(O|Z) > P(O|Y)$ we can conclude that O most likely belongs to the class represented by model Z.

The second problem can be solved easily by trying every possible state sequence and taking the one with the highest probability. As this method increases exponentially with the length of the observation sequence a more effective solution was developed: the Viterbi decoding algorithm [107] [108]. The Viterbi algorithm calculates the state sequence that has the highest probability to have generated a given observation sequence by only doing subsequent calculations for the partial path with the best probability, thus the complexity only increases linearly with the observation sequence length.

The third problem consists of training the model in such a way that, given a training observation sequence O, the parameters of the model $\lambda = (A, B, \pi)$ are adapted in order to maximise the probability of O given lambda. There does not exist an optimal solution for this problem, but there are several solutions to find local maxima for $P(O|\lambda)$ including the expectation maximisation algorithm [109], the segmental K-means algorithm [110] and the Baum-Welch algorithm [111].

3.3.3 Training

To train this model, we used data from the free and openly accessible Massachusetts General Hospital/Marquette Foundation (MGH/MF) Waveform Database [112, 113]. This data set contains three ECG leads (I, II, and V) and was sampled at a rate of 360 readings per second. It also contains a separate file in which expert cardiologists have identified and annotated every heartbeat, meaning that they provide the time stamp of the start of a new heartbeat and classify the heartbeat in the three categories Normal, Ventricular or Supraventricular. However, part of the annotations are incomplete, as the medical experts themselves were not sure which category

Table 3.1: Classification results

File	TP	FP	FN	TN	FP rate	Recall	Specificity
mgh001	11	824	1	1747	32.05%	91.67 %	67.95 %
mgh002	333	817	98	4320	15.90 %	77.26 %	84.10%
mgh003	7	1532	0	5449	21.95 %	100 %	78.05%

a heartbeat belongs to. These records have been ignored for the training of the HMM. The database is separated in different files for each patient and are named following the pattern mgh001 - mgh00N, where N is the total number of patients in the database.

To train HMMs we need good initial parameters to get satisfactory classification results [114]. In fact, the starting parameters have to be within one standard deviation [115] from the actual parameters of the system in order for the training algorithm to converge on the right values. Therefore the initial starting parameters have been calculated manually by averaging the values of ten normal, ventricular and supraventricular heartbeats.

Additionally, the HMMs are trained individually for every type of cardiac cycle (normal, ventricular and supraventricular), and once they are regrouped, they get merged into the final HMM shown in Figure 3.1. The training was done using the mgh001 data set with the Baum-Welch algorithm [111].

For the test sets we used records from 3 patients, mgh001 which was also used for training, mgh002 and mgh003. To classify the heartbeats in these test sets we used the Viterbi algorithm[107] [108] to find the most likely state sequence of the observations and checked in the topology of Figure 3.1 which type of cardiac cycle it matches.

3.4 Performance of the tested system

The classification results are shown in Table 6.1. The table contains information about correctly determined abnormal heartbeats (TP), correctly determined normal heartbeats (TN), false alarms (FP) and not detected abnormalities (FN). Furthermore, the recall (percentage of abnormal heartbeats that were actually detected) and specificity (percentage of normal heartbeats that were correctly labeled as normal). As we can see, the recall for the data sets mgh001 and mgh003 is quite high with 91.67% and 100%, which might be also partly due to the small total number of actual abnormal heartbeats (11 and 7). For the mgh002 data set the recall is lower with 77.26%. For the specificity, all 3 data sets show similar results. They all have a quite large number of false positives, meaning that the HMM

wrongly classified normal heartbeats as abnormal.

3.5 Discussion

These results are more or less in line with the results obtained in [97] which presents also a quite high rate of false positives. The results themselves can be interpreted in a few ways depending on the situation the system is used:

- Hospital:
  The system could help to quicken the workflow of the doctors. While it does not detect every single abnormal heartbeat, it is not really necessary as the doctors just want to know if the patient has an arrhythmia and is interested in the abnormalities found which leads us also to the drawback of the system: the high rate of false positives. This results in the doctor having to filter through a substantial amount of heartbeats manually only to find out that most of them or even all of them (in the case of a healthy patient) are normal.
- At home:
  One of the main goals of eHealth systems is to be able to monitor patients at home. If the system detects that something is wrong with the patient, it could decide to notify the patient, notify the doctor or even call emergency services. In this situation, the current system with a single ECG sensor that is analysed with a HMM could notify the patient that there is a problem and that he should go and see a doctor. The system, while not being able to identify every abnormal heartbeat, could still be useful. However, the large number of false positives could also annoy the patient with notifications to the point he just does not use the system anymore.

A system with a single sensor like the one shown here is not perfect. While the classification results certainly can still be improved by adapting the system or even completely changing the classification algorithm, a single sensor is just not enough to deal with all the uncertainties that such a system has to deal with.

These uncertainties, in the case of this system, include muscle noise related to the electrical signal of the muscles interfering with the ECG signal and power line interference [97, 116]. There is work to reduce the noise [116, 117], however the techniques will not be able to completely remove the influence of the noise.

Another drawback of a single sensor system is, that in case of a malfunction or complete failure of a sensor, the system itself becomes unable to fulfill its tasks and thus useless. So, how can these issues been dealt with?

In the medical domain you generally treat the patient on two fronts: you treat the symptoms in order to reduce any pain or discomfort on the

patients side but you also try to treat the source of the problems the patient is experiencing. This method could provide useful for sensor system as well, by not only trying to reduce the noise from the sensors but by also detecting the cause for the malfunctioning sensors and acting on it.

3.6 Conclusion

In this section, we explored existing single sensor eHealth systems and presented our version of a single sensor system. We presented the results of the performance of the system and discussed potential drawbacks of using only a single sensor.

If we further analyze the issues described earlier, we can make a few observations:

1. A lot of the time, it is known when a sensor is sensitive to external influences.
2. Again, a lot of the time, it is also known to which noise sources the sensors are sensitive, meaning that it could be potentially identified if the current situation a system operates in could influence it's sensor.
3. Advances in sensor technology gave birth to a large array of sensors. Some parameters can be measured using multiple distinct sensors that use different technologies to achieve the same goal and therefore could also react differently to noise sources.

With these observations in mind, we extend the single sensor system to a multi-sensor system in the next chapter. There are different versions of multi-sensor systems that will be presented. The advantages and potential challenges of these systems will also be discussed.

4 eHealth systems with multiple sensors

Contents

4.1 Introduction

The drawbacks described for a single sensor system and the observations made in the previous chapter indicate that a system that possesses more than just a single sensor will be likely to perform better than a system with just a single sensor. In this chapter we present different iterations of multi-sensor systems, discuss the advantages to the previous versions and analyse the disadvantages in order to further improve the architecture of the system. We first start by taking a look at existing multi-sensor systems in eHealth and present an extension of the system described in Chapter 3 where we add multiple sensors to the system and discuss the advantages this can be bring compared to the previous version of the system and what possible disadvantages or new challenges may appear. We then further refine the system in a second version, take a look at existing similar systems and again analyse the advantages and possible arising challenges which will lead us to our final model presented in the next chapter.

4.2 Multi-sensor System: first version

4.2.1 Related work

In this section we will take a look at eHealth systems that use multiple sensors but still use them in an individual manner, meaning that each sensor is used to make its own diagnosis, but the results are never combined.

In [5], the authors presented an approach with multiple sensors where the system not only uses an ECG sensor but also an Electroencephalogram (EEG) in order to measure brain activity and an Electrogastrogram (EGG), which records the electrical signals of the muscles in the stomach. However, while it uses multiple sensors, the diagnosis is done based on data from individual sensors. This means that if for example a sensor detects a problem, the system will alert the patients/doctors without considering the results of the analysis from other sensors. This can lead to false decisions taken if the data was for example influenced by an external source.

Similarly, in [118] the LifeGuard system is described, which is capable of measuring ECG, the respiration rate, the blood oxygen saturation, the skin temperature, the heart rate, the blood pressure and body movement. However, again every sensor is just used to make a diagnosis based on its own data, but the data is never combined to make a decision.

4.2.2 System and discussion

As described above, there exist eHealth systems that use multiple sensors in order to make a better diagnosis for the patient [5]. However, they use

them in an individual manner which still does not tackle the issues addressed in the previous chapter. To showcase this, we will extend our system with a few sensors in order to improve the diagnosis made.

In Figure 4.1, the extended version of the system is depicted. The system includes a layer of filtering and preprocessing of the data. The Electrocardiogram (ECG) used previously only for arrhythmia detection is also used for respiration analysis [119, 120]. The system additionally contains a blood pressure sensor and a module that analyses the blood pressure information. The system could be further extended by a number of sensors that are all used to detect a specific anomaly in the patient.

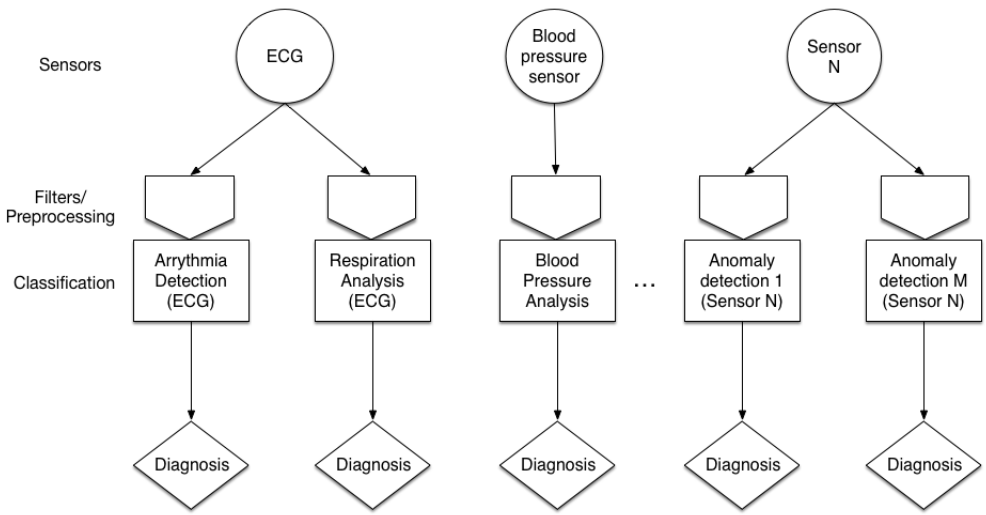

Figure 4.1: Multiple sensor eHealth system (sensors used individually)

Finally, these classification results are used to make a diagnosis of the patient. However, they all do so individually and thus the system still has pretty much the same drawbacks that it had before using only one sensor. The points in the system that have been improved are firstly that the system is able to detect a wider range of health conditions of the patient and secondly, that in the case a sensor is failing, it is not completely useless, but is still able to detect the other anomalies in the patient's health condition.

Concerning the uncertainties and noise that can influence the result of the classification, the system is in the exact same state that it was in previously. It still tries to fix the symptoms instead of trying to eliminate the sources that cause these symptoms. Therefore, some systems went a step further as we will show in the next section.

4.3 Multi-sensor System: second version

4.3.1 Related work

In this section we will take a look at related eHealth systems that use multiple sensors and try to combine the data from these sensors in order to make better decisions. The authors in [121] developed the alert portable telemedical monitor (AMON) and introduced reliability values for the different sensors to facilitate the combination of data from different sensors. The autonomic management of ubiquitous e-Health systems (AMUSE) presented in [122] adopted an autonomic approach based on self-managed cell in order to be able to react to device failures. Each cell manages a set of related devices.

The authors in [123] proposed a multi-tier hierarchy that uses data from multiple sensors in combination with machine learning methods for disease recognition. Another eHealth system uses data fusion methods in order to aggregate data coming from different sensors and other sources like social network feeds[124]. Both approaches allow for small optimisations for the decisions taken. In [125] the authors use an expert model to do the classification and showed that adding an expert model helped to improve the classification results of epilepsy detection in comparison to a standalone neural-network model.

4.3.2 System and discussion

Sensor fusion techniques, like described in Section 2.4 can be used to combine data coming from different sensors. The architecture of a version of the system that includes sensor fusion techniques is shown in Figure 4.2. Instead of trying to detect abnormalities in the patients health condition directly in the data of a single sensor, the data is first preprocessed and then data from multiple sensors that can be used to detect the same anomaly is sent to the sensor fusion layer in which it gets combined. This version of the system takes advantage of the redundant information present in the different sensors. As some sensors measure the same parameters, the fusion algorithms can use that information in order to reduce the uncertainty as much as possible and in the end a classification and diagnosis is made based on this improved the data. Depending on the system, a set of homogeneous sensors is used in order to compensate for potential failures or malfunctions, however this is generally mostly useful for individual hardware failures. In the case of external factors that could create noise in the data, it is very likely that every sensor of the set of homogeneous sensors will produce noisy data.

Therefore, another solution for these systems is to use heterogeneous sensor. These heterogeneous sensors can help in different ways:

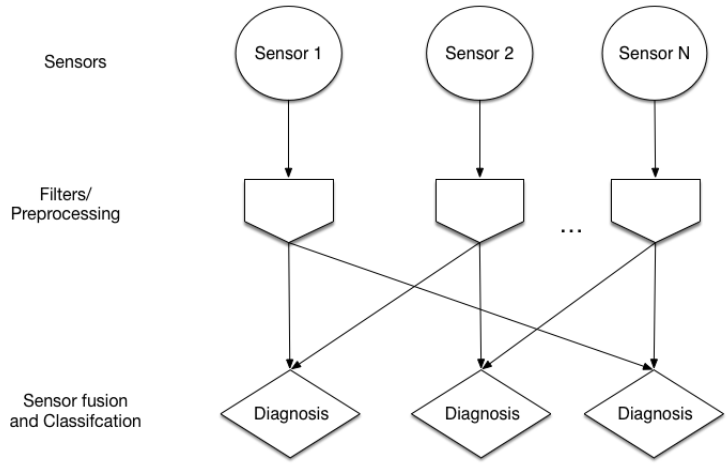

Figure 4.2: Multiple sensor eHealth system with sensor fusion

1. The sensors measure the same parameter.
   Contrary to a set of homogeneous sensors, in this case the sensors used take advantage of different technologies in order to measure the same parameters. For example in eHealth systems, the ECG and the photoplethysmogram (PPG) can be used in order to detect arrhythmias [126]. This can be useful as they use different methods to achieve the same goal. The ECG uses electrodes placed on the patient (which is why the ECG measurements are sensitive to power lines) while the PPG uses a light-based technology, which makes it react differently depending on the light level [127]. This means, that depending on the situation, one of the sensors is more reliable than the other and will thus likely provide better data.
2. The sensors measure different parameters.
   In this case some sensors measure different parameters, that can be used together to calculate or estimate a parameter that one of the sensors is measuring. An example of such a parameter would be the position that will generally be given by the GPS. However in case of a failure in the GPS, the current position could be estimated based on the last known position and data from an accelerometer (movement), gyroscope (orientation) and magnetometer (direction of movement). This is already used to some extend in IMU enabled GPS devices [128].

The system now possesses the right tools in order to make better and more reliable decisions. However, there are still some challenges left that need to be addressed.

1. The first one is to identify if the information from different sensors is conflicting, meaning that the system has to know that two differ-

ent sensors measure the same parameter and has to detect that the information coming from them is contradicting.

2. The second challenge is then to handle this conflicting data and to decide which one is the most trustworthy and should be used for the decision making process. For this to be possible, the system has to know which sensors can currently be trusted. Thus, the system has to know which sensors are likely to behave in a bad way if they find themselves in specific situations.
3. The third challenge then consists of the system correctly identifying the situation it finds itself in, so that it can then decide which sensors it can trust.

If the system manages to address all of these challenges, it will be able to make more informed and thus better decisions. In the next chapter, we will describe the model of our system that was designed to overcome the challenges presented in this chapter.

5 Proposed model

Contents

5.1 Introduction

To address the challenges presented in Chapter 4, we developed a multi-layer architecture for the second version of our multi-sensor system, which we will present in this chapter, starting with the overall architecture of the system and continuing with a detailed description of every layer. Finally, we will finish with a discussion about the objectives of the proposed model, of what it is supposed to be and what not, compare it to other possible approaches and identify the parts of the system that can be more or less generic and could be largely reused independent of the application domain.

5.2 Motivation

In the two previous chapters (3 and 4), we made observations about single and multi-sensor systems and identified challenges that arise when building such systems that need to be addressed. In this section we will first recapitulate the most important points and describe what could be possibly done to address them which will finally lead us to the next section in which we will present the architecture of our model. Let us list the observations made previously:

1. A lot of the time, it is known when a sensor is sensitive to external influences;
2. Again, a lot of the time, it is also known to which noise sources the sensors are sensitive, meaning that it could be potentially identified if the current situation a system operates in could influence it's sensor;
3. Advances in sensor technology gave birth to a large array of sensors. Some parameters can be measured using multiple distinct sensors that use different technologies to achieve the same goal and therefore could also react differently to noise sources.

Using these observations we formulate requirements that a system overcoming the previously described challenges needs to have. They are as follows:

1. to have multiple sensors. Subsets of these sensors should be able to be used to measure or estimate the same parameter;
2. to be aware of the current situation it operates in;
3. to have knowledge about what factors from the surrounding environment are causing noise for a given sensor type.

The first requirement can be considered more of a precondition or hypothesis for the system. A multi-sensor system that does not fulfill this precondition will be limited in its actions to overcome the challenges described in the previous chapters. Therefore, even though we present a par-

tially generic architecture for a multi-sensor system in the next section, the choice of sensors will be important and will have to be done for every situation in which the system will be deployed.

The second requirement can be addressed by adding a context module to the system. This means that the system has to constantly analyse the data coming from some sensors in order to build the context the system operates in. This can be done with a hybrid approach of machine learning classifiers and a rule-based engine. The job of the classifiers will be to check whether sensors are likely failing and provide a confidence values for them. They will also classify part of the data required to build the context. A rule-based engine could then be used to further refine the context by using some pre-defined expertise coded in the rules.

For the third requirement, the use of a rule-based engine and a knowledge base jumps to mind. The knowledge will be used to reason about the trustworthiness of certain sensors and to decide which sensors are the most reliable in the current situation. For this step, machine learning techniques are less useful as they would be extremely hard to train for this use case, be less flexible and also pose more challenges when it comes to following, understanding and improving the behaviour of the system. With these potential solutions in mind we will present the two-layer architecture of our system in the next section.

5.3 Overall Architecture

To address the challenges outlined in the previous section we developed a two-layer architecture that can act as a middleware between the sensors and the decision making of an existing system. To be able to address these issues, the system has to be composed of a set of heterogeneous sensors. As discussed previously, heterogeneous sensors provide the best possible opportunity to identify situations in which sensors are most likely to fail or malfunction. Additionally, the set of heterogeneous sensors has to be interrelated in some ways in order for the system to be able to decide which sensors are malfunctioning and to propose an alternative solution. In some cases, having multiple sensors of the same type can be a plus.

Additionally, the system also has to be able to correctly identify the context it currently operates in. When talking about this system, context describes the current situation the system finds itself in that could potentially influence the accuracy of its sensors. Context does not relate to any situation or information that would be necessary for the system to attain its objectives. Consider the overall architecture of the application shown in Figure 5.1.

The sensors and decision making parts belong to an already existing system. Our system is set in between the sensors and the decision making.

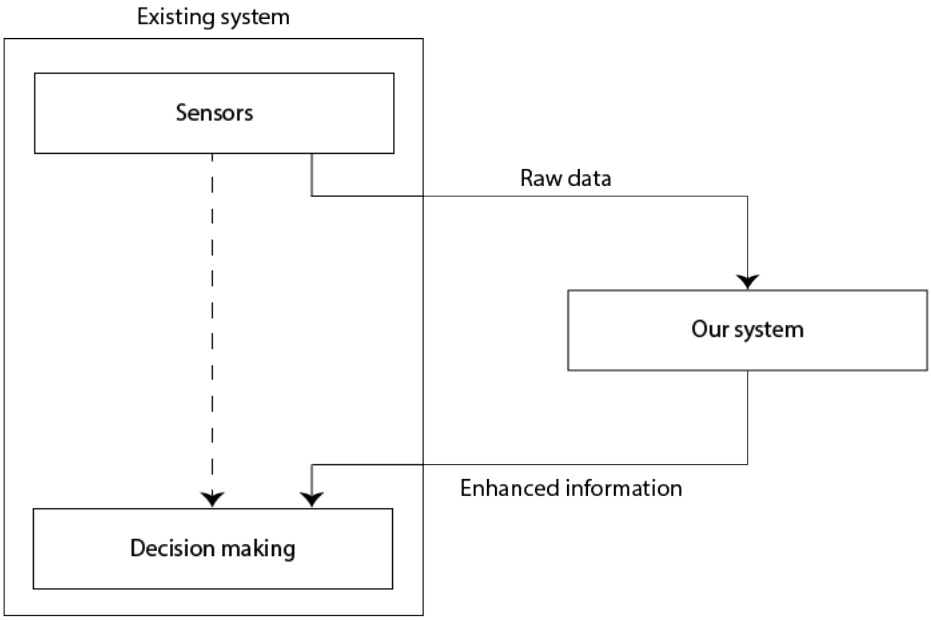

Figure 5.1: Overall architecture

It receives the raw data from the sensors of the existing system and relays enhanced information to the decision making module.

Figure 5.2 shows the more detailed version of our system and how it tries to achieve its goal with a two-layer architecture that consists of a classification layer and a rule-based conflict-handling layer. The classification layer takes the raw data from the sensors and tries to detect whether they are functioning correctly or not. The output of the classification layer will be a confidence value that is attributed to each sensor, which will then be passed to the rule-based proactive engine (PE).

The second layer consists of the PE. It receives raw data from the sensors and the confidence values that were attributed by the classification layer to these sensors. With that information it first builds the context that the system finds itself in and using the context it adapts and refines the confidence values attributed to each sensor.

Finally, using these updated confidence values, the PE decides which sensors can be trusted and tries to circumvent failing and malfunctioning sensors by calculating estimates based on data from other trustworthy sensors where possible.

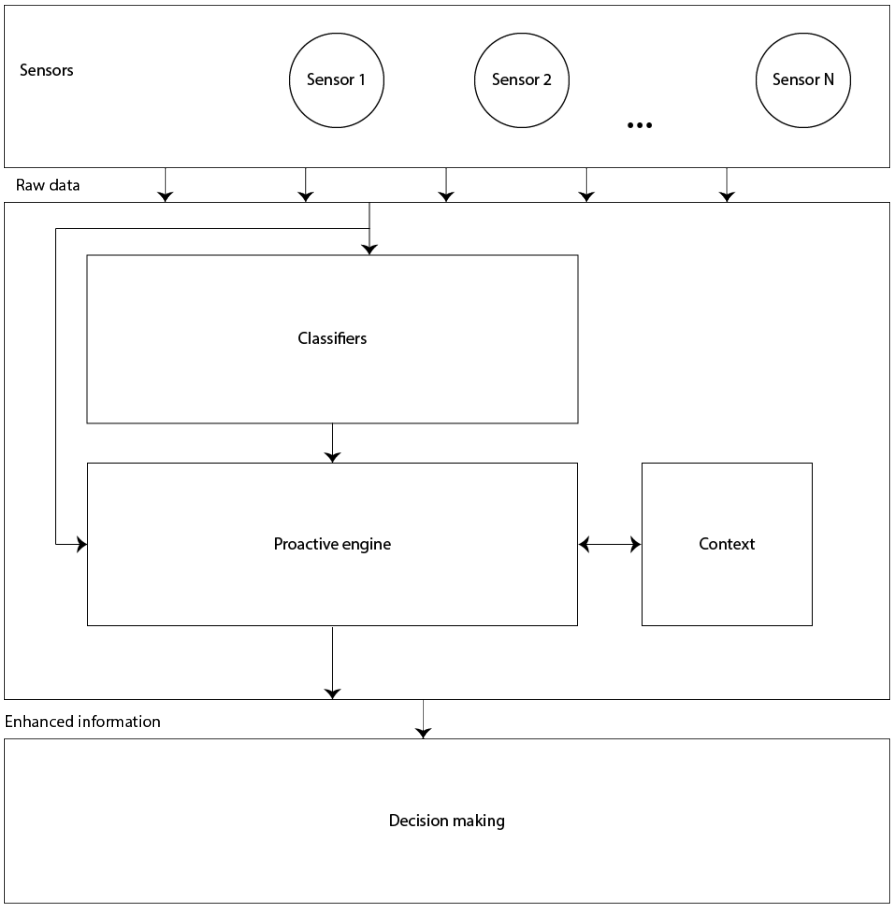

Figure 5.2: System architecture

5.4 First layer

The first layer is a classification layer that assigns an initial confidence value to every sensor. This confidence value represents the level of trust in the sensor. A low confidence value means that a sensor is likely malfunctioning. These classifiers have to be tailor made for every sensor and adapted to the appropriate environment as they have to get trained correctly before they can give appropriate confidence values.

The job of this layer is not to have a 100% accuracy in the confidence value or even in the classification result, but rather to give a rough estimate on whether or not a sensor is malfunctioning. The confidence value obtained is then sent along with the raw data to the second layer for further processing.

5.5 Second layer

The second layer consists of the PE on which scenarios will be run to build the current context, to further refine the decision on which sensors can be trusted and to decide whether to pass data directly from a sensor to the existing third party system or to calculate an estimate based on data from more reliable sensors.

A more detailed version of the second layer is shown in Figure 5.3. Conceptually, the processing of the data is divided into different steps:

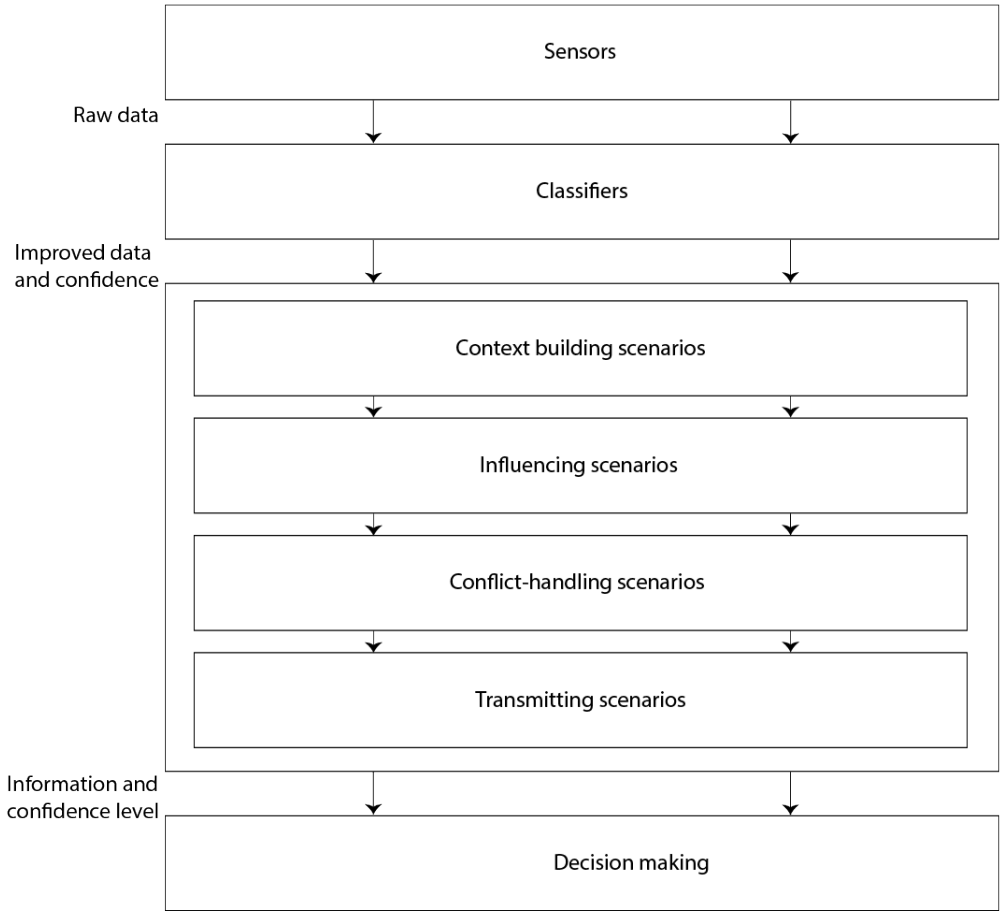

Figure 5.3: Scenario flow

1. Context-building scenarios
   The job of these scenarios is to build the current context the system operates in, so that it can be used by the following processing steps to decide which sensors to trust.

Sensor name	List of properties	Minimum confidence	Grace period	History Length
GPS	position	0.8	1s	1000
Accelerometer	acceleration	0.8	1s	800
Light	lightlevel	0.5	30s	50
...	...	...	...	...

Table 5.1: Sensor registration table

2. Influencing scenarios
   Influencing scenarios will adapt confidence values from sensors based on the current context and predefined knowledge about the sensors' weaknesses.
3. Conflict handling scenarios
   Conflict handling scenarios then check whether sensors measuring the same properties provide conflicting data. As the confidence values has already been updated by taking the current situation into account, these scenarios can then decide which sensors to trust and in some cases to calculate an estimate of the real data.
4. Transmitting scenarios
   The job of these scenarios is to get the results from the previous steps and to relay them to the third party system.

These different steps will be explained in more detail in the subsequent sections.

The second layer will also be connected to a database where domain specific knowledge about the sensors and the properties they are measuring will be stored. Consider Table 5.1. It contains every registered sensor that is part of the third party system and describes which properties they are measuring. Additionally it defines values for the minimum allowed confidence, a grace period and the kept history length, whose purpose will be explained in the appropriate section.

5.5.1 Context-building scenarios

The context-building scenarios will be, just like the classifier layer, very domain specific. The job of the context-building scenarios is to create a context that accurately represent the current environment of the robot and that is useful for the subsequent steps in the processing chain to reason about which sensor to trust.

The context parameters will be either generated by the classifiers or obtained by the proactive engine that deduces them from the data from multiple sensors. These context parameters will then also receive a con-

Sensor name	Context parameter	Value	Operation
GPS	high buildings	true	equals
Ultrasonic sensor	temperatureVariation	20	bigger
...	...	...	...

Table 5.2: Contextual influence of sensors

fidence value based on the classification results and the confidence of the sensor(s) they were obtained from.

5.5.2 Influencing scenarios

This processing step adapts the confidence values of the sensors that were obtained from the classifiers according to knowledge it has about the sensors sensitivity to contextual noise. This knowledge has to be defined in the database like shown in Table 5.2. For every sensor where it is applicable, one or more contextual parameters that can affect the accuracy of the data provided, are defined. The value field defines the actual number, the range or the limit for which the context parameter starts influencing the accuracy of its related sensor. Note that for conciseness this is represented here as a single table, but in reality it will be split in several tables for the different representations of the values.

The rule generation part of the meta influencing scenario will look as the pseudo code shown in Figure 5.4. The meta scenario starts a specific scenario for every contextual parameter that can influence the accuracy of a sensor. The underlying specific scenarios then vary slightly based on whether the accepted values for a contextual parameter is a range, a limit or a specific value.

```
for sensor in sensorList{
  contextParameterList=GetListOfContextParameters(sensor, DB);
  for contextParameter in contextParameterList{
    startSpecificInfluencingScenario(sensor, contextParameter);
  }
}
```

Figure 5.4: Starting influencing scenarios

5.5.3 Conflict handling scenarios

The conflict handling scenarios handle the cases where multiple sensors provide different contradicting data. We distinguish between two cases,

the first case where two sensors measure the same parameter and the second case where one sensor measures a parameter that can be calculated or estimated by combining the data coming from several other sensors.

In the first case, the scenario will simply chose the sensor with the highest confidence and trigger the related transmitting scenario with this sensor as parameter. This is possible as at this stage in the execution process, the contextual parameters that may potentially influence the accuracy and thus confidence put into the different sensors, have already been taken into account. Thus, the sensor with the higher confidence will be the one that provides the more accurate data in the current contextual situation.

In the second, more complex case, the scenario will first check the confidence of a sensor. If the confidence is lower than the allowed minimum confidence for the sensor, the next rule that will calculate an estimation for the parameter given by the distrusted sensor will get executed. If this estimated value diverges from the value given by the sensor by a predefined margin, the next rule will get executed.

This rule will take the average confidence of the sensors used to calculate the estimation, check if every sensor meets the minimum confidence requirements and notifies the transmitting scenarios about whether the connected system should use the data from the original sensor or the calculated estimate.

5.5.4 Transmitting scenarios

The transmitting scenarios communicate the recommendations of our system to the connected system. This is done through the database and includes the sensors, parameters, related confidences, when applicable confidence of the estimates and a recommendation which one to use.

The final decision on whether to use this recommendation or to ignore it, relies in the connected system, which is possible as it has access to both the real value from the sensors as well as the calculated estimates through the database.

5.6 Discussion

In this section we are going to discuss the objectives of the proposed model and which previously described challenges it addresses, what is not part of the objectives, its advantages and possible limitations.

The objective of the system is to make sure that a third party system gets the most reliable data possible to make its decisions. The decisions of our system are limited to the sensor and data management side. There are no decisions taken to achieve a goal other than providing trustworthy information to a third party system.

To see if our proposed model can address the existing challenges let us first recapitulate what these challenges actually are:

1. The system can correctly identify its current context so that it can then decide which sensors it can trust.
2. The system has to know which sensors are likely to behave in a bad way if they find themselves in specific situations, in order to decide which sensors can be trusted.
3. The system has to identify whether information from different sensors is conflicting, has to handle this conflicting data and has to decide which one is the most trustworthy and should be used for the decision making process.

Note that we changed the order of the challenges described in Chapter 4 in order to better fit the processing steps in our model. The first item on the list, identifying the current context, is addressed by the first processing step in the proactive engine, the context-building scenarios. The second challenge is addressed by the influencing scenarios and the knowledge about the different sensors that is saved in the database. Finally, the third one is addressed by the conflict handling scenarios.

The two layer approach presented that consists of a layer of classifiers and a layer consisting of a rule-based system, has some advantages over a two layer approach consisting of a second layer of neural networks regrouping the information from the first layer to make decisions about the trustworthiness of sensors.

Firstly, the neural networks would need to be trained, which is already challenging for the first layer and would be even more challenging for the second layer. Secondly, it is easier to understand, follow and fix the reasoning in a rule-based system than it is in neural networks. From this follows, that our model could potentially justify its decisions to a human user or could be able to share the reasoning behind its decisions with other systems. For neural networks it is very difficult to know why they do not give the expected results and where things went wrong.

Concerning the limitations of our system, like with every rule-based system, the larger it grows and the more complex it becomes the more difficult it becomes to foresee every possible interaction between sensors and different properties.

To answer our last research question on which parts of the system can be designed in a generic way, so that they can be applied to different domains, it has to be noted that having a generic solution for this type of problem is very difficult if not impossible. The first layer of our proposed model certainly has to consist of classifiers specifically trained for a given domain. The second layer also needs domain specific knowledge so that it can reason about which sensors to trust and detect and handle potential conflicts between the data of different sensors.

However, if this knowledge is encoded like in our system, the influencing scenarios and a part of the conflict handling scenarios (the first case where two sensors measure the same parameter) can be kept generic to a point that the number of scenario types is independent from the domain and from the number and types of sensors in the system as the system can manage to carry out these processing steps with differently parametrized instances of the existing scenarios.

This allows a developer or expert to simply add the required knowledge to the database and the system would automatically know how to use it without needing any changes to the source code. The additional knowledge can even be taken into account at runtime, meaning that adaptations can be done without the need of restarting or redeploying the system.

In the next chapter we will look at how the system and scenarios look like when applied to a specific application (in this case a robot in a simulation environment) and we will show a proof of concept of the system in action to show that it can improve the decision making of the robot.

6 Robotics Proof of concept

Contents

6.1 Introduction

In this chapter we are presenting a possible application for our model introduced in the previous chapter, in the domain of robotics. For this purpose, we use the robotics simulation software Webots[129]. We provide a robot with the functionality of our system and test its effectiveness. We will first start with a section about related work in robotics systems and continue by explaining the chosen example situation the robot has to deal with in the proof of concept. Moreover we give an overview about the Webots simulation environment, describe the implementation of the different layers and provide some test results. Finally, we finish with a discussion about the current system implementation.

6.2 Proof of concept

6.2.1 Webots

Webots [129] is a robotics simulation software that recently has become open-source. It is developed since 1996 in cooperation with the Swiss Federal Institute of Technology in Lausanne and is widely used in industry, education and research.

Webots provides a large collection of sensors,actuators, robots and objects that are frequently used in robotics. It includes an API for several programming languages including Java, C, C++ and Python. Another feature of the software is the recording of simulation movies.

However, as with every simulator there is a gap between reality and the simulation[130]. Consequently, while the test results obtained in a simulator are certainly an interesting indicator of the performance of a system, they still have to be taken with a grain of salt.

6.2.2 Our robot and its environment

To test our model we will deploy it on a robot in Webots. While Webots provides a large set of simulated real robots, we chose to use a custom robot that will only have the sensors needed for our example application, chosen in the set of predefined sensors that is already provided by Webots. Before we describe the sensors used, we will first describe the environment the robot will operate in.

The environment is shown in Figure 6.1. It is an urban environment with several skyscrapers and normal houses. As pointed out in [131, 132], urban environments with high buildings negatively affect GPS accuracy. A study conducted in Chicago even showed that the GPS can give wrong positions that are up to 100m away from the real position[4] and for more than half of the time, the positional error was above 20m.

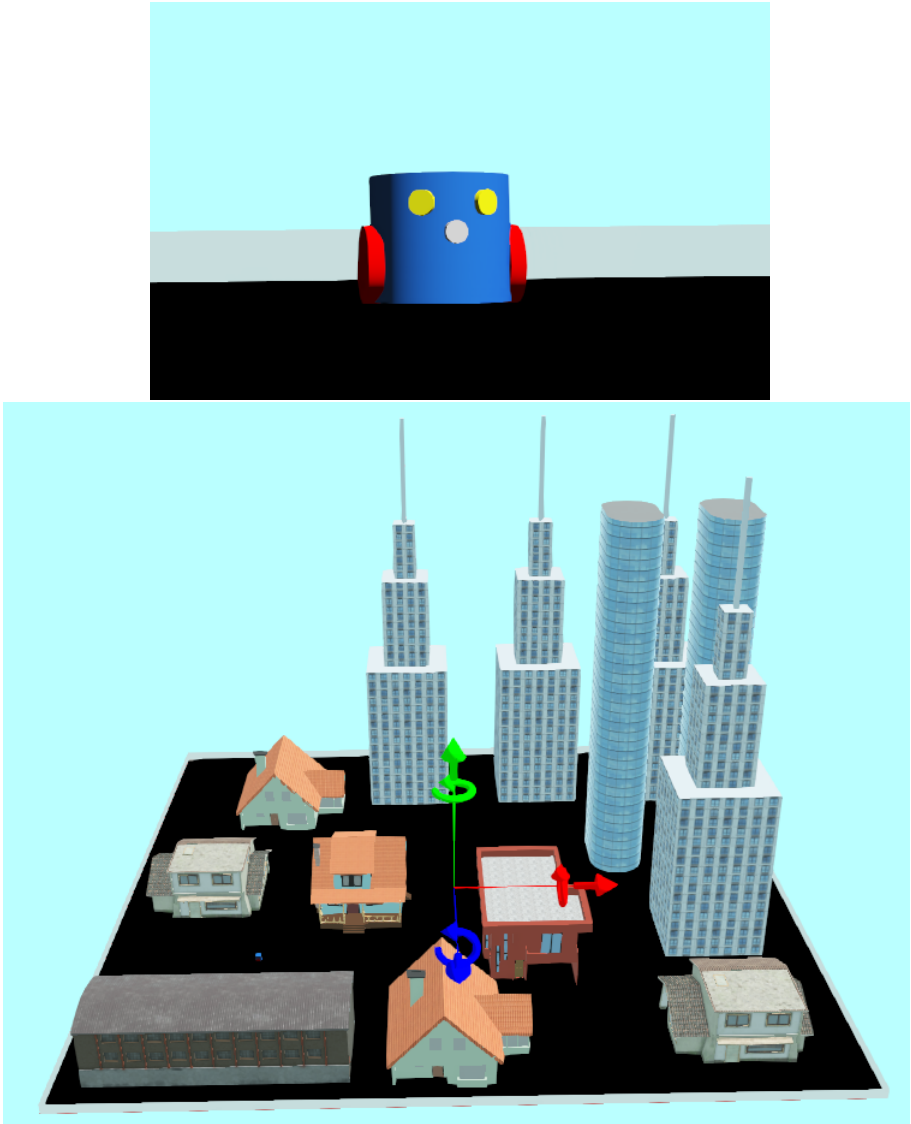

Figure 6.1: Robot and application environment

With this knowledge in mind, we simulate the situation through our environment in Webots. The sensors we will be using for this application are the following:

- GPS
  The GPS provides the position of the robot as well as its speed. It will be the main focus of the application as a failure or malfunctioning of the GPS sensor should trigger the mechanisms set in place by our system.
- Accelerometer

The Accelerometer provides the acceleration of the robot along all 3 possibles axes.

- Gyroscope
  The Gyroscope provides the angular velocity, i.e. the speed with which the robot rotates around the three axes.
- Inertial Unit
  The Inertial unit (IMU) measures the current orientation of the robot. It return the roll, pitch and yaw angles of the robot. (See Figure 6.2 [129])
- Camera
  The camera will allow the robot to detect high buildings in the area. As image recognition is a challenging topic and beyond the scope of this thesis, Webots provides the possibility to add a Recognition node to the camera which can detect some predefined objects by cheating a bit. It allows you to set a recognitionColor of a solid node (object) in addition to the normal color that will be used for rendering, and the camera then uses this recognitionColor to detect which objects are currently in its view. It also allows you to set a distance at which objects can be recognised, meaning that the recognition node can only detect objects that are closer than the given distance.

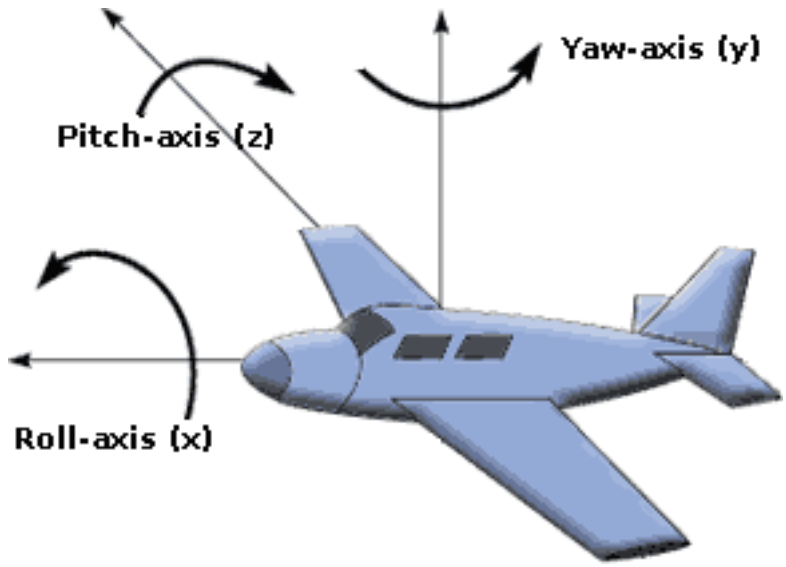

Figure 6.2: Roll, Pitch and Yaw angles

To simulate the inaccuracies of the GPS near skyscrapers, we will inject it with gaussian noise based on the proximity of the robot to one of the skyscrapers. The influence radius of the skyscrapers in the previously shown environment is represented by the red circles in Figure 6.3.

Figure 6.3: Influence radius of skyscrapers

The goal of the robot is to navigate through the city following a predefined route, while the objective of our system is to handle the conflicting information the robot will be receiving from its sensors due to the inaccuracies of the GPS and make sure that the robot receives enhanced GPS information in order to make the appropriate decisions about its navigation.

6.2.3 Software Architecture

As described previously, we will augment the robot with our system. The architecture of the final system is shown in Figure 6.4.

The robot possesses several sensors which we described previously: GPS, Accelerometer, Gyroscope, Inertial Unit and Camera. It also has a decision making module (controller in Webots terminology) where it will decide what actions it has to carry out in order to reach its goal.

The goal of the robot is to follow a predefined route through the city. The actions the robot can take are : TurnLeft, TurnRight, MoveForward and Stop. The robot will only continue its journey to the next checkpoint on the route if it is within a given radius of the previous checkpoint.

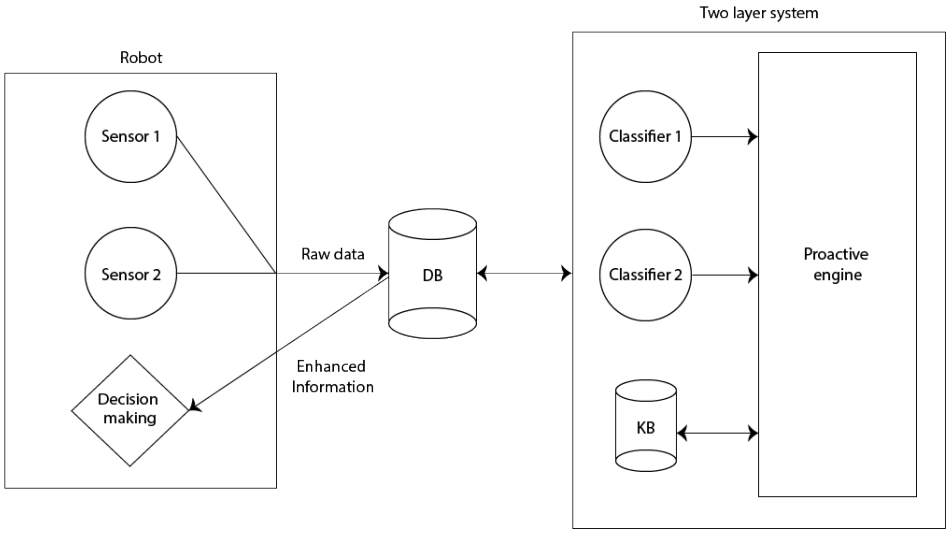

Figure 6.4: Robot architecture

After being augmented by our system, the robot will no longer use the data from the sensors directly in its decision making but will first pass the data to our system and then use the resulting data that comes from the computation done in our system.

In the part of our system, the data from the sensors will first get passed to the classification layer and then to the proactive engine along with the results from the classifications. The proactive engine will handle the possible conflicting information, enhance the data and pass it back to the robot along with a confidence value for the different sensors.

In the next sections, we will take a look a the implementation of these two layers for this example application.

6.3 Related work

6.3.1 Convolutional neural networks

Convolutional Neural Networks (CNNs) have seen increasing popularity over the last few years, and have had a huge success in image classification and object recognition tasks[133]. This success lead researchers to also try and use them for time series classification either by transforming the time series itself in an image [134] or by using a one dimensional version of a CNN[133]. They have recently be used successfully in single sensor data fault detection[135, 136, 137, 138, 139].

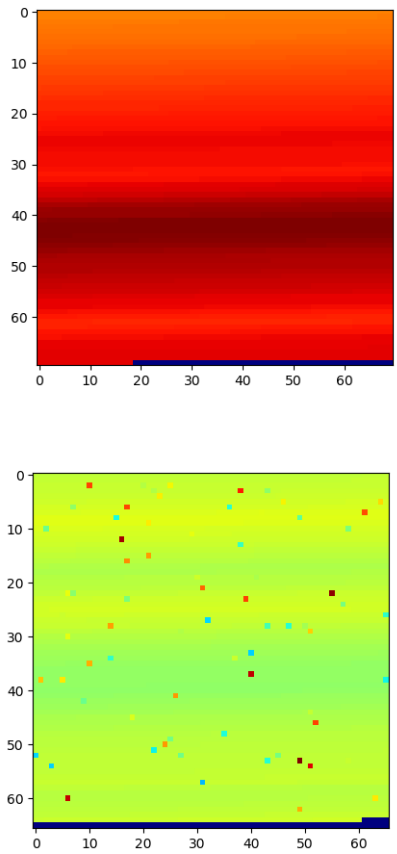

Figure 6.5: Example perfect GPS vs GPS injected with errors

This usage might seem strange at first as data coming from most sensors do not seem to have anything in common with images, however this strangeness comes rather from the human struggles of visualising time series rather than the inability of CNNs for time series classification.

In Figure 6.5 an example of data that is easier to visualise is presented. It consists of several consecutive sequences of the normalised latitude obtained from a GPS. The first image depicts measurements from a perfectly functioning GPS and in the second image wrong values have been introduced at random. In this visualisation it is easy to spot the outliers and an algorithm that was designed to recognise objects or patterns in an image should provide good results in detecting these anomalies. Of course the example shown in these pictures is an idealised example for visualisation purposes but the core idea is the same even if the GPS data would be a bit more noisy in the real world. One of the advantages of CNNs is that it can recognise these patterns in the images or data automatically[140].

In Figure 6.6 the architecture of a basic CNN for image processing is shown. The input is an image of X by Y pixels and consists of several channels which for images typically represent the RGB values, meaning that each channel represents the red, green and blue values of the pixels. The next layer is a convolutional layer which applies filters to the input by calculating the dot product of the input values with the filter values. Applying a filter to the input is called a convolution. A convolution can be seen like a sliding window as the filters slide all over the input image like shown in Figure 6.7. The steps by which the filter is repositioned is called stride. In the example shown the stride is equal to 1.

The resulting 'images' are called feature maps as they extract features from the image. They will be slightly reduced in dimensions compared to the input images based on the dimensions of the filter applied.

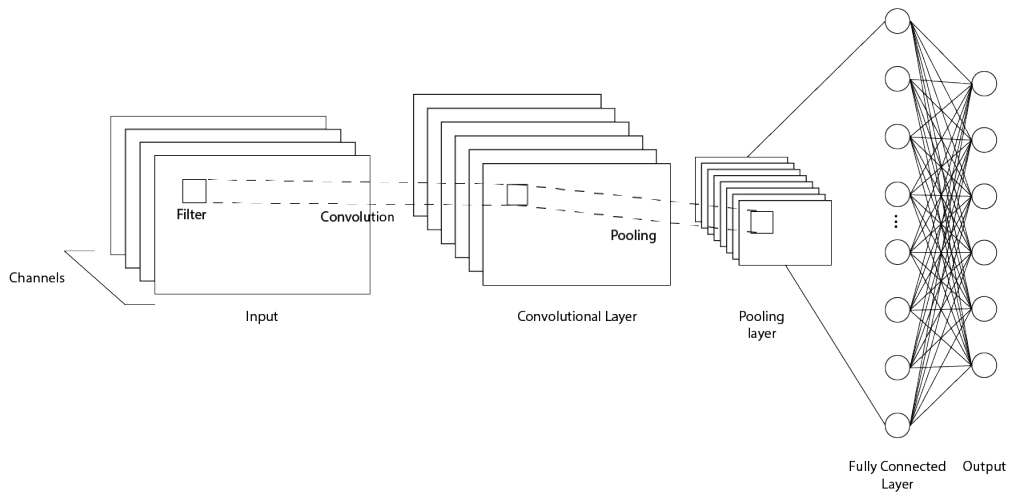

Figure 6.6: General architecture of a convolutional neural network

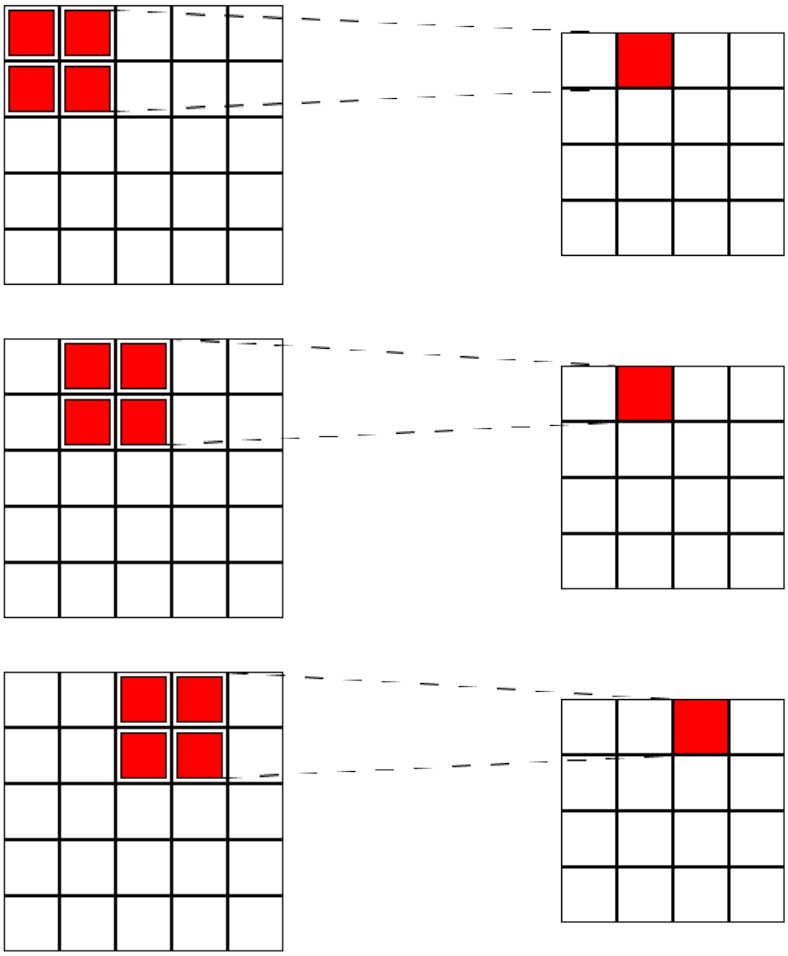

Figure 6.7: Example of a 2 by 2 filter applied to an image

The next layer is the pooling layer which is used to further reduce the dimensions of the image. There exist two main pooling techniques: max pooling and average pooling. Their names are indicative for the operations they apply to the image: max pooling takes the maximum value of a part of the image while average pooling calculates the average. An example of max pooling is shown in Figure 6.8.

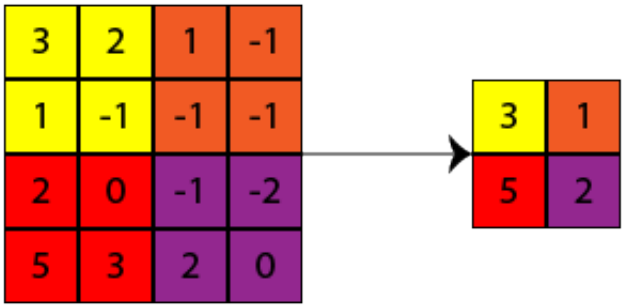

Figure 6.8: Example of max pooling

The two types of layers mentioned in the previous paragraphs can be repeated several times. At the end of the CNN the feature maps are flattened into a normal 1 dimensional dense layer of neurons. This layer then finally connects to the output neurons which will contain the classification or prediction results.

As CNNs can overfit quite easily, a Dropout layer is generally introduced after the pooling layers [141], which makes the network only use a part of the available neurons during training, scarifying some performance on the training set for better generalisation.

To use a CNN for the sensors in our application, we can use the structure shown in Figure 6.9. The concepts are the same, except that the input is one dimensional and as a result the filters are also one dimensional.

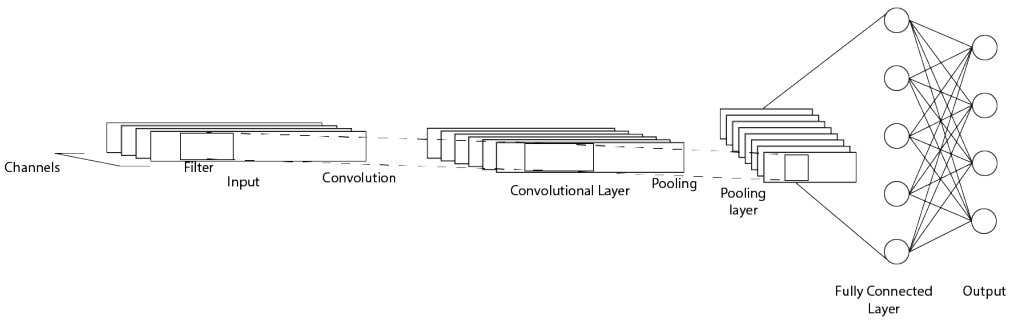

Figure 6.9: Architecture of a convolutional neural network for one dimensional time series

6.3.2 LSTM neural networks

Sensors provide a system with a stream of data. The data in this stream is often time dependent, meaning that the a value in the stream is dependent

of the previous values. The first neural networks that allowed for some sort of time dependency were the Recurrent Neural Networks (RNN). The architecture of an RNN and its unfolded form are shown in Figure 6.10. It consists of input and output nodes with hidden nodes in between. This represents a single time step. The output will be looped around to act as input for the next time step. When looked at in its unfolded form a RNN can be seen as a series of copies of traditional neural networks that pass the output from one network to another. However, it turned out that, due to the architecture of RNNs, they are only good at learning very short-term dependencies in data streams[142]. Nevertheless, they have been used successfully in sensor fault detection[143].

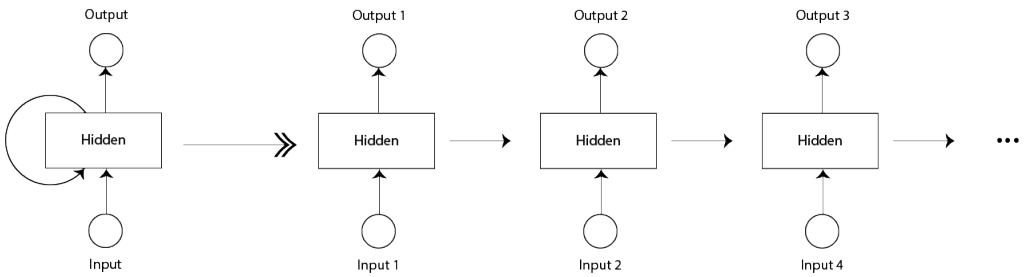

Figure 6.10: An RNN and its unfolded form

To overcome this short-term dependency problem and extend Recurrent Neural Networks in a way to be able to learn longer-term dependencies, Hochreiter and Schmidhuber developed Long Short Term Memory (LSTM) networks[144]. LSTM networks are a special kind of RNNs that replace the simple Hidden layer with an LSTM cell shown in Figure 6.11.

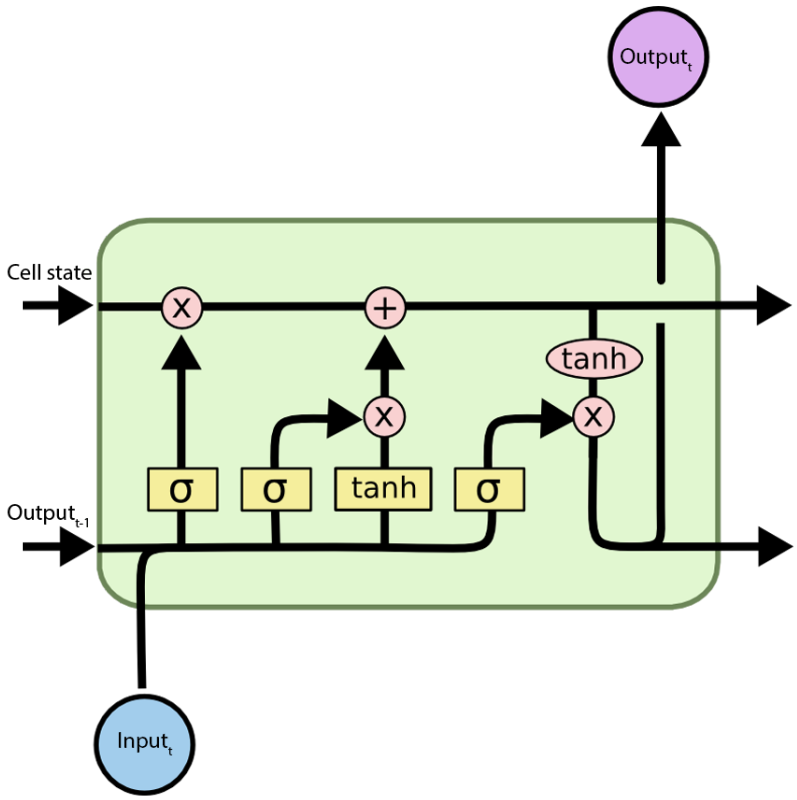

Figure 6.11: An LSTM cell [1]

Note that in this figure, rectangles represent neural network layers, while circles represent functions that are applied to the information, so the tanh inside the rectangle represents a neural network layer whose output is called to be between -1 and 1 by using a tanh function and the tanh inside the circle simply means that the tanh function is applied to the information that passes at that point in the execution of the cell.

The LSTM cell is composed of four different neural network layers that can influence the cell state and thus the output in different ways. Three of these layers are gates. They are composed of a sigmoid layer that returns values between 0 and 1, describing the amount of information that should get passed the gate (0 means nothing and 1 means everything), and a pointwise multiplication operation.

The first gate in the process is called the forget gate. It decides how much of the cell state from the previous time step should be forgotten. This is done by feeding the output from the previous time step and the input of the current time step to the sigmoid neural network layer and multiplying the result with the previous cell state.

The second gate is called input gate. Again it is composed of a sigmoid layer that decides what values to update and by how much. The result is

multiplied with the output of a tanh layer whose task it is to propose new values to update the cell state. Thanks to the multiplication operation the result is a scaled version of the output of the tanh layer and is added to the cell state.

The third and final gate decides which parts of the cell state should be output. This is done by multiplying the output from the sigmoid layer with the current cell state that was scaled using the tanh function. The result represents the output of the current time step which will also be used along the current cell state to compute the output for the next time step.

LSTM networks also have been successfully applied to sensor fault detection [145].

6.4 Implementation

For the implementation we used the Python machine learning libraries Scikit-learn [146] and Tensorflow[147] as well as the high level deep neural network API Keras [148].

6.4.1 Data processing

The data necessary for the training has been gathered in the simulation environment Webots [129]. While gathering the data for every sensor described previously (except the camera), errors have been injected periodically and the data has been labelled accordingly.

As the data was gathered in a simulation environment and the dimensionality of the data for every sensor is quite limited, the amount of preprocessing that needs to be done for the training set is quite small:

1. Feature scaling
   Neural networks are sensitive to features that have different scales and train better when the data is normalised[141].
2. Split the data into sequences
   When using the NN, we will not be able to use really long data sequences in order to make a classification, but have to use rather small sequences. Therefore, during training the model has to be fed these same small sequences as well.
3. Data augmentation
   While we are gathering data from a simulation environment and could gather data at infinitum, in the real world we do not have this luxury. Data augmentation can help to have more data with which to train the networks[149]. In the case of the GPS we could for example apply different translations to the training set in order to get more data. Another possibility to get more data is to shift the sequences generated previously.

When using the trained NN in our application we have to pre-process the data as well:

1. Fill in missing values
2. Feature scaling
   The same scaling used for the training set has to be applied to the data from the sensors as well.
3. Split the data into sequences
   The data coming from the sensors has to be regrouped into sequences before it can be fed to the trained model.

In order to obtain satisfying results when using these classifiers, the training data has to be similar enough to the data at the time of execution. If it is not, we talk about a dataset shift or, if only the input distribution of the data changes, we talk about a covariate shift. The reasons for this shift can come from bias introduced when generating the training set or because it was not possible to reproduce the real execution conditions at training time[150]. A possible way to detect these distribution changes in the input data is kernel mean matching[151].

6.4.2 Classifiers

The classification layer for the sensor analysis and classification can be quite flexible. It potentially can contain any algorithm that is able to classify time series and provide the confidence it has in its classification. For this application we are using CNNs, that have seen increased use in time series classification after its success in image recognition[133], and LSTM recurrent neural networks. In the previous section we presented these two types of neural networks. In this section, we describe how we can apply a CNN to our system by training it on the values from the inertial unit.

Training The architecture of the CNN used for the inertial unit values consists of one convolutional layer (50 filters), one average pooling layer with dropout and a dense layer[135]. The first step of training consists of a grid search of hyperparameters in order to get an idea of which specific parameters give the best results. The hyperparameters tested include:

1. The activation function (softmax, softplus, softsign, relu, tanh, sigmoid, hard sigmoid, linear)
2. The weight constraint (1, 2, 3, 4, 5)
3. The dropout rate (0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9)
4. The number of epochs (5,10,20,50,100,200)
5. The batch size (4,8,16,32,64,128,256)
6. The optimization algorithm (SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam)
7. The learn rate (0.001, 0.01, 0.1, 0.2, 0.3)

8. The number of neurons (dense layer) (1, 5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90, 100)
9. The initializers for the initial random weights (uniform, lecun uniform, normal, zero, glorot normal, glorot uniform, he normal, he uniform)

To reduce the time complexity, only two different hyperparameters were used in the gridsearch at a time. The configuration that was finally used for the CNN based on the results from this gridsearch is shown below.

1. The activation function: relu for convolutional layer and softplus for dense layer. For the final output the sigmoid activation function is used as it is a binary classification task and we need values between 0 and 1 representing the probabilities of belonging to a class (softmax would have been a better option for multiple classes[152]).
2. The weight constraint: 3
3. The dropout rate: 0.5
4. The number of epochs: 100
5. The batch size: 64
6. The optimization algorithm: Adam
7. The learn rate: 0.001
8. The number of neurons (dense layer): 70
9. The initializers for the initial random weights: lecun uniform

With this configuration, the network has been trained on a dataset consisting of over 200000 entries and evaluated using a different dataset of roughly the same size. The goal of the classifier is to be able to differentiate between correct and incorrect data for a single sensor and to attribute a confidence value to a sensor that is higher the more likely the data is to contain no errors. The results of the evaluation will be shown in the next sections.

Evaluation As the data has been collected in a simulation environment and the errors have been introduced by ourselves, there is a risk that the evaluation results might be biased. To give a fair evaluation of the performance of the neural networks that were trained, we tried to create datasets that will allow us to better evaluate how generalisable the networks are. To make these datasets as different as possible from the training sets while still keeping the original information about the problem the networks are supposed to solve we changed the following parameters:

1. Seed of the random generator
   The random generator was used in 2 instances. First, for the robot to randomise its movement and second for when to inject errors.
2. Position and scaling of the area the robot is driving around in
   This means that we positioned the area in a different place in the coordinate system and increased the size of the area, so that the robot

has more freedom to drive around.

3. Error rate
   The goal of changing the error rate is to see how well the networks can handle different amounts of errors that are introduced.

To evaluate the performance of the classification we are using different metrics. (TP=true positives, TN=true negatives, FP=false positives, FN=false negatives):

- Precision
  $\frac{TP}{TP+FP}$
- Recall, sensitivity or True positive rate
  $\frac{TP}{TP+FN}$
- Specificty or True negative rate
  $\frac{TN}{TN+FP}$
- False positive rate
  $\frac{FP}{FP+TN}$
- F1 score
  $\frac{2*TP}{2*TP+FP+FN}$
- Balanced accuracy
  $\frac{Recall+Specificity}{2}$

As the datasets are highly imbalanced, accuracy is not a good measure to use. The balanced accuracy is preferred. Also the f1 score gives a good indication how well the classifier is doing in an unbalanced dataset.

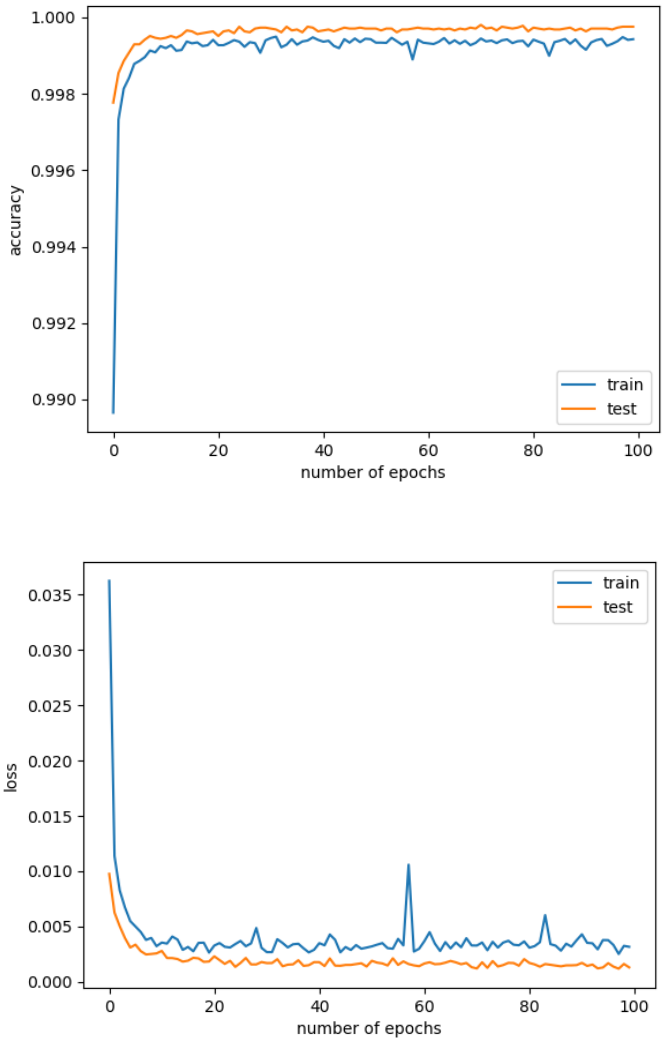

Figure 6.12: Accuracy and loss graphs for the inertial unit

Results The accuracy and loss graphs obtained during training is shown in Figure 6.12. Note that the train accuracy is lower than the test accuracy and the train loss is higher than the test loss which can be caused by the utilisation of a quite high dropout, which is used during training to prevent overfitting but is not used anymore when testing the network.

In Table 6.1 the confusion matrix of the result from the evaluation data set is shown. The network only raised 8 false alarms where it predicted a failure while there was none. Also the number of missed failures is quite low, which is shown by the listed metrics below.

Table 6.1: Confusion matrix

	Predicted No Failure	Predicted Failure
Actual No Failure	174954	8
Actual Failure	115	31627

Evaluation metrics:

- ROC AUC: 0.999903
- Accuracy: 0.999405
- Precision:0.9997477
- Recall: 0.996377
- F1 score: 0.998059
- False Positive Rate: 0.00004572421
- Specificity: 1-FPR=0.99996538689
- Balanced accuracy=0.99817119344

Discussion Note that the layer implemented here is not necessarily using the best possible solution for the failure detection of every sensor type, but a solid solution in order to test the effectiveness of our system. The topic of of sensor failure detection, classification and machine learning is huge and it is beyond the scope of this thesis to have an optimal classifier for every sensor.

6.4.3 Scenarios

In this example application there are several scenarios running on the proactive engine as shown in Figure 6.13. For the context building scenarios step and influencing scenarios step described in the previous chapter, there is one scenario each. The conflict-handling scenarios step and transmitting scenarios in this case are regrouped under a single scenario. In parallel, a clean-up scenario is running to periodically clean up the obsolete data from the database.

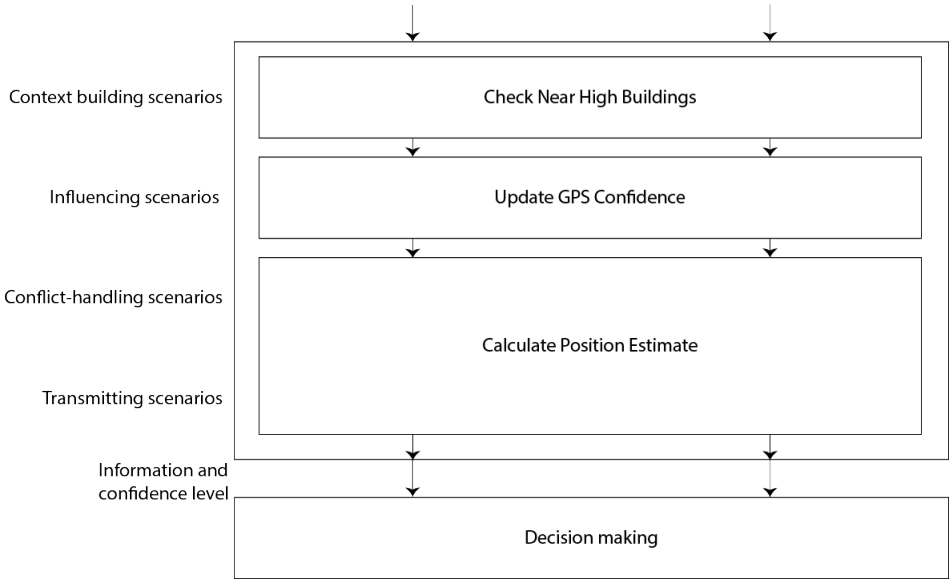

Figure 6.13: Scenarios present in the application

1. Context-building
   The context in this application consists of the knowledge of whether the robot is near a high building. The related scenario thus consists of a single rule that verifies this based on data from the recognition module of the camera and accordingly updates the knowledgebase to represent whether the robot is currently near a high building or not.
2. Influencing
   The only sensor in this example application whose accuracy can get influenced by the environment or contextual information is the GPS. Therefore this step consists of a single scenario with a single rule to update the confidence value of the GPS. This rule fetches the knowledge about the presence of near high buildings from the knowledgebase that was build during the previous step and in the case the robot really is next to some higher buildings, it will set the confidence of the GPS to 10%.
3. Conflict handling and Transmitting
   This scenario is composed of four rules. One META rule who is continuously running and checking if the right conditions are met and three normal rules who will get created by the META rule depending on the conditions that are met.
   The pseudo code for the META rule is shown in Figure 6.14. The rule keeps track of the confidence values for the GPS. In the dataAcquisition part of the rule it first fetches for the last known confidence value of the GPS. The rule is only activated if the fetched confidence

is lower than a predefined (or possibly dynamic) threshold. Whether the rule is activated or not, it always saves the confidence of the GPS and clones itself. In the case the rule was activated, meaning that the confidence for the GPS is lower than the threshold, it distinguishes between two cases. The first possibility is where the saved confidence from the previous execution of the rule is higher than the threshold. In this case the GPS was working fine previously. The rule will therefore create a StartEstimationRule. In the second case the estimation calculation was already started previously, therefore a ContinueEstimationRule is created. Finally, if the rule was not activated, a CleanEstimationRule is created.

```
GPSEstimationMetaRule:
double oldConfidence
dataAcquisition:
currentConfidence=database.getLastConfidence("gps")
activationGuards:
return currentConfidence<CONFIDENCE_THRESHOLD
conditions:
return true
actions:
None
rule generation:
if (getActivated()) {
if(lastConfidence>CONFIDENCE_THRESHOLD) {
createRule(new StartEstimationRule())
}else {
createRule(new ContinueEstimationRule())
}
} else {
createRule(new CleanEstimationRule())
}
lastConfidence=currentConfidence
createRule(this)
```

Figure 6.14: Pseudo Code of the GPSEstimationMetaRule

The pseudo code for the StartEstimationRule is shown in Figure 6.15. In the dataAcquisition part the rule gets the last valid GPS value (when the GPS was still working correctly) and the time this value was recorded. It then retrieves all the inertial unit values and speed values that were recorded since that time, sorted from old to new. The activationGuards and conditions for this rule are always true. In the actions part, the rule will then iterate over the inertial unit and speed values and calculates an estimation of the new GPS position with these values, the old GPS position and the time difference between the entries. Finally the estimated position is put into

the database, taking thus care of transmitting the estimated values to the robot. This rule does not create any other rules.
The ContinueEstimationRule is very similar to the StartEstimation-Rule, except that the calculations are based on the last estimated GPS position and not on the last valid GPS position.
The CleanEstimationRule removes all the estimations from the database.

```
GPSStartEstimationRule:
  dataAcquisition:
    lastGPSValue=database.getLastValidGPSValue()
    lastTime=lastGPSValue.getTime()
    iUnitValues=database.getIUnitValuesAfterTime(lastTime)
    speedValues=database.getSpeedValuesAfterTime(lastTime)
  activationGuards:
    return true
  conditions:
    return true
  actions:
    while(iUnitValues.next() and speedValues.next())
    {
      newTime=iUnitValues.getTime()
      timeDifference=newTime-lastTime
      newGPSValue=calculatePosition(lastGPSValue,iUnitValues,
          speedValues, timeDifference)
      lastTime=newTime
      lastGPSValue=newGPSValue
    }
    insertToDB(newGPSValue)
  rule generation:
    None
```

Figure 6.15: Pseudo Code of the GPSStartEstimationRule

6.5 Evaluation

6.5.1 Setup

To demonstrate that the system is working, we put our robot in an environment with normal buildings and some skyscrapers. The skyscrapers will influence the accuracy of the robot's GPS by injecting Gaussian noise once it enters a predefined radius. The goal of the robot is to follow a predefined list of checkpoints using its sensors. The robot can decide itself whether it has reached a checkpoint based on the position data from its GPS sensor.

This experiment is run first without our system and afterwards with our system.

6.5.2 Expected results

The experiment without our system should behave correctly as long as the robot does not enter an area near a skyscraper as its GPS should function normally during that time. Once it enters such an area, the GPS will get influenced by the nearby skyscrapers and the system will be unable to correctly get to the next checkpoint.

When the robot is equipped with our system, the behaviour of the robot will be similar to the one previously, but once the robot comes near a tall building, it will be able build the contextual data, know that it is near a tall building, ignore the data coming from the GPS, and calculate an estimated position based on the last known valid position. The resulting path taken should get closer to the checkpoints than the one taken by the previous version of the robot.

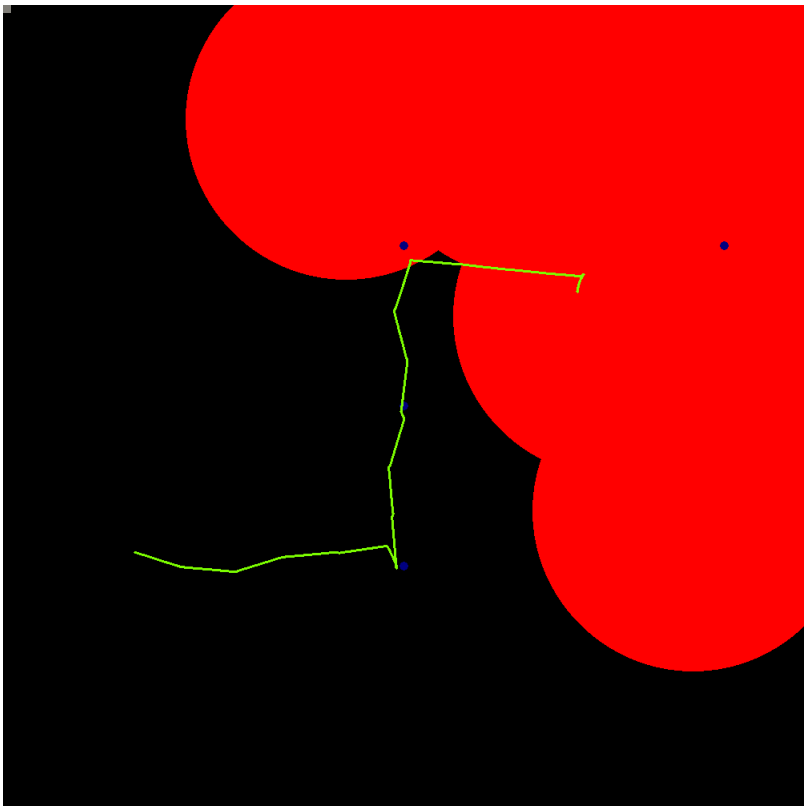

Figure 6.16: Robot path without our system

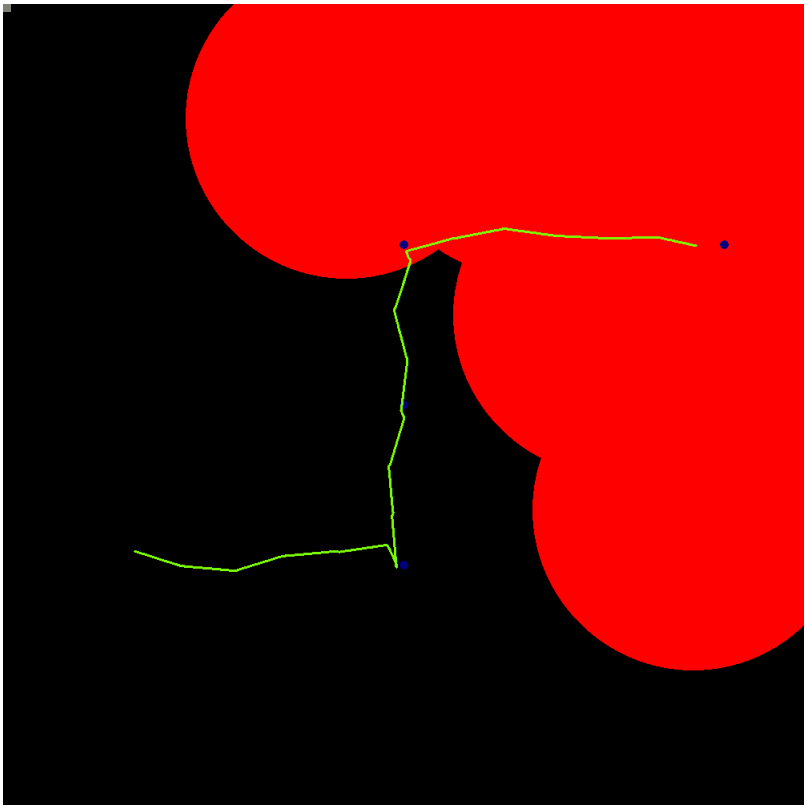

Figure 6.17: Robot path with our system

6.5.3 Results

The path of the robot without our system is shown in Figure 6.16. The robot is trying the objectives (blue dots) normally, until it reaches the area of influence of the skyscrapers (red circles). Because of the added noise it then has trouble following the predefined checkpoints and finally ends up next to a skyscraper and decides that it has reached the final objective.

In Figure 6.17, the path taken by the robot that has been augmented with our system is shown. In the black area, where there are no external influences applying on the GPS, the robot takes the same path as its version without our system, however once it enters the area of influence of the skyscrapers, the system detects with the help of the camera that the robot is too close to the skyscrapers for the GPS to be 100% reliable, reduces the confidence of the GPS and thus begins calculating an estimate based on the last known position with the highest confidence, the speed and the orientation of the robot. The results are then transmitted to the controller for the robot which can then decide whether to use the GPS values or the calculated estimate based on the confidence values transmitted. In this case it will use the estimated values to navigate through the red zones and,

even though the path taken is not perfect as the robot still stops a bit before the final checkpoint, it does a lot better job at navigating than the version without our system.

6.6 Conclusion

In this chapter we have presented a proof of concept for the application of the system presented in the previous section. The classification layer to compute initial confidences for the different sensors has been implemented using two different kinds of neural networks (CNNs and LSTMs) to show that multiple methods could be used for that layer without needing to change anything to the architecture of the system.
We also have shown how the scenarios can be implemented using an example application of a robot navigating through an area of high buildings and we have shown that the implemented scenarios can help the robot to use contextual data to improve its navigation through the city.
In the next and last chapter we will take a look back at the work done in this thesis, draw some conclusions and give some directions for possible future work.

7 Conclusion

Contents

In this thesis, we have seen that sensor malfunctions due to external influences can lead to bad decisions being taken. While this kind of malfunction is difficult to detect when only looking at sensors individually, we have proposed a semi-generic model that can remedy this kind of issue and handle conflicting information coming from different sensors. The model proposed is based on a two layer architecture, one using classification algorithms to check for sensor failures and attribute confidence values to each sensor, and a second one to use contextual data along with some knowledge about sensor weaknesses to detect and handle conflicting data. The system thus gives a recommendation on which sensor the system can currently trust and in some more complex cases provides an estimation calculation for the real values of the malfunctioning sensors. The evolution of the system has been illustrated over three experiments, two of which have been located in the eHealth domain and one in the robotics domain.

In this chapter we will look back at these achievements and relate them to the research questions stated in the introduction. Furthermore, in the last section we will propose some possible outlooks for future work to further enhance the current system.

7.1 Achievements

The research questions that have been stated in Section 1.2 have been answered over the course of this thesis. In this section, we will summarize the answers.

RQ1 was answered in Chapter 3. The results presented in the Chapter showed that, while the classification done by the system could still be improved, a single sensor is not enough to deal with the uncertainties that arise in such a system. The uncertainties regarding this system included muscle noise and power line interference. Furthermore, in the case of a sensor failure, the system becomes unable to carry out the tasks it was designed to do. The solution to overcome these challenges is to add multiple sensors to the system that can help to overcome the uncertainties and to keep the system running in case of a sensor failure.

RQ2 was discussed in Chapter 4. To describe how multiple sensors can help to improve decision making, we distinguished between 2 cases. In the first case, we have multiple sensors that measure the same parameter while using different technologies for doing so. The ECG and photoplethysmogram (PPG) are examples of such sensors in the eHealth domain. They can be both used to detect arrhythmias [126], but they both have different weak points. These weak points could be overcome by detecting the situation the system is currently in and by deciding to use the sensor that currently is functioning correctly.

The second case was about sensors that measure different parameters

but could be combined in order to calculate or estimate a parameter that is normally given by a another sensor but which has some weak points where it is not functioning correctly anymore. Again, the system could decide based on the context whether to use the data from the single sensor, or, if it the context suggests that this sensor is malfunctioning, it could use the data from the other sensors to calculate an estimate.

RQ3 and RQ4 were answered in Chapters 5 and 6. The model that was proposed has a two-layer architecture in order to detect and handle potential conflicting information coming from different sensors. The classification layer first provides a confidence values for the different sensors that will give a good estimate whether the sensors are malfunctioning in most situations. Then in the second layer, the rule-based proactive engine will build the context based on information coming from the sensors, decide based on predefined rules what contextual situation potentially affects the accuracy of some sensors and reduces the confidence for them, and finally, in case a sensor was tagged as too unreliable, will calculate an estimate for the data that can no longer be used directly from the sensor.

Over the course of these two chapters it was also discussed that, while the classification layer has to be trained for every domain, the overall structure of the system remains unchanged. Also, there is the potential to use a single predefined scenario for the influencing step, by reading the different parameters and effects from the database that can be changed without affecting the code.

7.2 Future work

In this section we are going to present different ideas on how the system presented in this thesis could potentially be extended in the future in order to further improve and optimise the decision making.

A possibility could be to let modules of the third party system decide the precision of the data they require. Our system would then potentially pass different data to the different modules. For example, while temperature affects ultrasonic sensors, the divergence from the real values is insignificant in some circumstances.

Another, quite challenging idea that is also recently emerging in related works[153], is to create a feedback loop from the conflict-handling part to the classification layer and use online machine learning techniques in order to update the classifiers and thus to improve the classification results. The end goal of this improvement could then be to have so reliable confidence values, that most of the steps in the second layer of the system could be skipped and that the entire application could directly decide whether it wants to use the data from a sensor or the calculated estimate.

The proposed model could also be used to find out relations between

data that are not yet known. A possible approach would be to test multiple scenarios in parallel and see if one of them provides better results. Additionally, association rule learning with algorithms like Eclat[154] and FP-Growth[155] can extract interesting relations between the data from the database.

Moreover, while for the proof of concept in the simulation environment the sensors are synchronized, this will not necessarily be the case in real-life real-time systems, especially if the sensors are part of a wireless sensor network like the sensors used to monitor the health status of machines [156]. It is thus necessary to extend the proposed model in order to solve this time synchronization problem of asynchronous sensor measurements. Several approaches exist, including the convex-hull based TICSync algorithm[157], a filter-based approach proposed by Nilsson and Händel[158] or the the flooding time synchronization protocol that uses periodic flooding of synchronisation messages[159] that can get a network-wide time synchronisation at micro seconds accuracy and is scalable to up to 100 nodes[160].Other algorithms are: Multihop flooding time synchronization protocol[161], Rapid Time Synchronization[162], Low overhead Time-sync[163], Power efficient time synchronisation protocol[164] and many more. In order to decide about the appropriate algorithm for our system, these different algorithms will need to get tested against each other.

Finally, as an important recent topic is Trustworthy Artificial Intelligence for which the High-Level Expert Group on Artificial Intelligence (HLEGAI) appointed by the European Union published a set of ethics guidelines in April 2019 [165], we could consider integrating these guidelines into our system. They put forward 7 key principles that trustworthy AI should have: Human agency and oversight, technical robustness and safety, privacy and data governance, transparency, diversity and non-discrimination, societal and environmental well-being and accountability. We will give possible improvements for the system regarding Human agency and oversight and accountability.

For the HLEGAI, AI systems should provide some sort of human agency and oversight. In our system, the decisions from the rule-based proactive engine that are logged could be displayed to a user that supervises the system. This could be structured in a way such that the user first gets an overview of the scenarios that were triggered and can then look at the detailed rule execution if he so chooses. In the case confidence values of multiple sensors are so low that our system would not know what to do, the supervising human could get prompted to make a decision. This decision would then be used by the system to carry out its tasks.

With regards to accountability, the HLEGAI requires AI systems to have mechanisms that allow for the assessment of the algorithms, data and design processes used in the system. Again, the log of the decisions taken by

the rules of the proactive engine could be helpful in order to retrace decision processes and evaluate whether the system is performing well enough.

List of my publications

[1] G. Neyens and D. Zampunieris, "Proactive middleware for fault detection and advanced conflict handling in sensor fusion," in *International Conference on Artificial Intelligence and Soft Computing*. Springer, 2019, pp. 643–653.

[2] G. Neyens and D. Zampunieris, "Proactive model for handling conflicts in sensor data fusion applied to robotic systems," in *Proceedings of the 14th International Conference on Software Technologies (ICSOFT), 2019 Prague, Czech Republic, 26-28 July, 2019*. SCITEPRESS, 2019, pp. 468–474.

[3] G. I. F. Neyens and D. Zampunieris, "A rule-based approach for self-optimisation in autonomic ehealth systems," in *ARCS Workshop 2018; 31th International Conference on Architecture of Computing Systems*. VDE, 2018, pp. 1–4.

[4] G. Neyens, "Conflict handling for autonomic systems," in *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE, 2017, pp. 369–370.

[5] G. Neyens and D. Zampunieris, "Using hidden markov models and rule-based sensor mediation on wearable ehealth devices," in *Proceedings of the 11th International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, Barcelona, Spain 12-16 November 2017*. IARIA, 2017.

[6] R.-A. Dobrican, G. Neyens, and D. Zampunieris, "A context-aware collaborative mobile application for silencing the smartphone during meetings or important events," *International Journal On Advances in Intelligent Systems*, vol. 9, no. 1&2, pp. 171–180, 2016.

[7] G. Neyens, R.-A. Dobrican, and D. Zampunieris, "Enhancing mobile devices with cooperative proactive computing," in *Proceedings of the 5th International Conference on Advanced Collaborative Networks, Systems and Applications (COLLA 2015)*. IARIA, 2015, pp. 1–9.

[8] R.-A. Dobrican, G. Neyens, and D. Zampunieris, "Silentmeet-a prototype mobile application for real-time automated group-based collaboration," in *Proceedings of the 5th International Conference on Advanced Collaborative Networks, Systems and Applications (COLLA 2015)*. IARIA, 2015, pp. 52–56.

Bibliography

[1] "Understanding lstm networks," https://colah.github.io/posts/2015-08-Understanding-LSTMs/, accessed: 2019-07-15.

[2] "Wearable devices forecasts," https://www.gartner.com/en/newsroom/press-releases/2018-11-29-gartner-says-worldwide-wearable-device-sales-to-grow-, accessed: 2019-09-05.

[3] S. Goel, X. Luo, A. Agrawal, and R. L. Reuben, "Diamond machining of silicon: a review of advances in molecular dynamics simulation," *International Journal of Machine Tools and Manufacture*, vol. 88, pp. 131–164, 2015.

[4] C. Bo, X.-Y. Li, T. Jung, X. Mao, Y. Tao, and L. Yao, "Smartloc: Push the limit of the inertial sensor based metropolitan localization using smartphone," in *Proceedings of the 19th annual international conference on Mobile computing & networking*. ACM, 2013, pp. 195–198.

[5] F.-S. Jaw, Y.-L. Tseng, and J.-K. Jang, "Modular design of a long-term portable recorder for physiological signals," *Measurement*, vol. 43, no. 10, pp. 1363–1368, 2010.

[6] G. Neyens and D. Zampunieris, "Using hidden markov models and rule-based sensor mediation on wearable ehealth devices," in *Proceedings of the 11th International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, Barcelona, Spain 12-16 November 2017*. IARIA, 2017.

[7] W. D. Blair and T. Bar-Shalom, "Tracking maneuvering targets with multiple sensors: Does more data always mean better estimates?" *IEEE Transactions on Aerospace and Electronic Systems*, vol. 32, no. 1, pp. 450–456, 1996.

[8] G. Neyens, "Conflict handling for autonomic systems," in *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE, 2017, pp. 369–370.

[9] G. I. F. Neyens and D. Zampunieris, "A rule-based approach for self-optimisation in autonomic ehealth systems," in *ARCS Workshop 2018; 31th International Conference on Architecture of Computing Systems*. VDE, 2018, pp. 1–4.

[10] G. Neyens and D. Zampunieris, "Proactive middleware for fault detection and advanced conflict handling in sensor fusion," in *In-*

ternational Conference on Artificial Intelligence and Soft Computing. Springer, 2019, pp. 643–653.

[11] G. Neyens and D. Zampunieris, "Proactive model for handling conflicts in sensor data fusion applied to robotic systems," in *Proceedings of the 14th International Conference on Software Technologies (ICSOFT), 2019 Prague, Czech Republic, 26-28 July, 2019*. SCITEPRESS, 2019, pp. 468–474.

[12] J. Mylopoulos and H. Levesque, "An overview of knowledge representation," in *GWAI-83*. Springer, 1983, pp. 143–157.

[13] R. Dobrican, *Collaborative Rule-Based Proactive Systems: Model, Information Sharing Strategy and Case Studies*, 2013.

[14] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016. [Online]. Available: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

[15] M. Campbell, A. J. Hoane, Jr., and F.-h. Hsu, "Deep blue," *Artif. Intell.*, vol. 134, no. 1-2, pp. 57–83, Jan. 2002. [Online]. Available: http://dx.doi.org/10.1016/S0004-3702(01)00129-1

[16] R. E. Korf, "Does deep blue use ai?"

[17] D. A. Ferrucci, "Introduction to "this is watson"," *IBM Journal of Research and Development*, vol. 56, no. 3.4, pp. 1–1, 2012.

[18] I. Management Association, *Machine Learning: Concepts, Methodologies, Tools and Applications: Concepts, Methodologies, Tools and Applications*, ser. Premier reference source. Information Science Reference, 2011. [Online]. Available: https://books.google.lu/books?id=1GWcHmCrl0QC

[19] J. Sokolowski and C. Banks, "Modeling and simulation for analyzing global events," *Modeling And Simulation For Analyzing Global Events*, 06 2009.

[20] E. A. Feigenbaum, "Expert systems: principles and practice," 1992.

[21] R. Davis and J. J. King, "The origin of rule-based systems in ai," *Rule-based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project*, 1984.

[22] G. Chryssolouris, *Manufacturing systems: theory and practice*. Springer Science & Business Media, 2013.

[23] E. Friedman-Hill, "Jess, the java expert system shell," *Biosystems*, 01 2003.

[24] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995.

[25] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *The knowledge engineering review*, vol. 10, no. 2, pp. 115–152, 1995.

[26] S. Franklin and A. Graesser, "Is it an agent, or just a program?: A taxonomy for autonomous agents," in *International Workshop on Agent Theories, Architectures, and Languages*. Springer, 1996, pp. 21–35.

[27] M. Luck, M. d'Inverno *et al.*, "A formal framework for agency and autonomy." in *ICMAS*, vol. 95, 1995, pp. 254–260.

[28] N. R. Jennings, K. Sycara, and M. Wooldridge, "A roadmap of agent research and development," *Autonomous agents and multi-agent systems*, vol. 1, no. 1, pp. 7–38, 1998.

[29] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards, *Artificial Intelligence: A Modern Approach*. Prentice hall Upper Saddle River, 2003, vol. 2.

[30] B. T. Clough, "Metrics, schmetrics! how the heck do you determine a uav's autonomy anyway," AIR FORCE RESEARCH LAB WRIGHT-PATTERSON AFB OH, Tech. Rep., 2002.

[31] T. Salamon, *Design of agent-based models*. Eva & Tomas Bruckner Publishing, 2011.

[32] B. N. Schilit and M. M. Theimer, "Disseminating active mop infonncition to mobile hosts," *IEEE network*, 1994.

[33] T. Rodden, K. Cheverst, K. Davies, and A. Dix, "Exploiting context in hci design for mobile systems," in *Workshop on human computer interaction with mobile devices*. Citeseer, 1998, pp. 21–22.

[34] A. Ward, A. Jones, and A. Hopper, "A new location technique for the active office," *IEEE Personal communications*, vol. 4, no. 5, pp. 42–47, 1997.

[35] P. J. Brown, "The stick-e document: a framework for creating context-aware applications," *Electronic Publishing-Chichester-*, vol. 8, pp. 259–272, 1995.

[36] B. N. Schilit, N. Adams, R. Want *et al.*, *Context-aware computing applications*. Xerox Corporation, Palo Alto Research Center, 1994.

[37] A. K. Dey, "Understanding and using context," *Personal and ubiquitous computing*, vol. 5, no. 1, pp. 4–7, 2001.

[38] A. J. Gonzalez and R. Ahlers, "Context-based representation of intelligent behavior in training simulations," *Transactions of the Society for Computer Simulation*, vol. 15, no. 4, pp. 153–166, 1998.

[39] B. S. Stensrud, G. C. Barrett, and A. J. Gonzalez, "Context-based reasoning: A revised specification." in *FLAIRS Conference*, 2004, pp. 603–610.

[40] P. Horn, "Autonomic computing: Ibm\'s perspective on the state of information technology," 2001.

[41] A. C. Manifesto, "IBMs perspective on the state of information technology," http://www.research.ibm.com/autonomic/manifesto/, 2001.

[42] K. Ahuja and H. Dangey, "Autonomic computing: An emerging perspective and issues," in *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*. IEEE, 2014, pp. 471–475.

[43] P. Lalanda, J. A. McCann, and A. Diaconescu, *Autonomic computing*. Springer, 2013.

[44] A. Computing *et al.*, "An architectural blueprint for autonomic computing," *IBM White Paper*, vol. 31, no. 2006, pp. 1–6, 2006.

[45] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[46] D. Tennenhouse, "Proactive computing," *Communications of the ACM*, vol. 43, no. 5, pp. 43–50, 2000.

[47] D. Shirnin, S. Reis, and D. Zampunieris, "Experimentation of proactive computing in context aware systems: Case study of human-computer interactions in e-learning environment," *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2013 IEEE International Multi-Disciplinary Conference on*, pp. 272–279, 26-28 February 2013.

[48] S. M. Dias, S. Reis, and D. Zampunieris, "Personalized, Adaptive and Intelligent Support for Online Assignments Based on Proactive Computing," in *2012 IEEE 12th International Conference on Advanced Learning Technologies*. Rome, Italy: IEEE, Jul. 2012, pp. 668–669.

[49] R. A. Dobrican, S. Reis, and D. Zampunieris, "Empirical Investigations on Community Building and Collaborative Work inside a LMS using Proactive Computing," in *Proceedings of E-Learn - World Conference on E-Learning 2013 Conference*. Theo Bastiaens and Gary Marks, 2013.

[50] D. Shirnin, "Formalising the twofold structure of a proactive system:proof of concept on deterministic and probabilistic levels," *University of Luxembourg*, 2014.

[51] R. A. Dobrican and D. Zampunieris, "A proactive solution, using wearable and mobile applications, for closing the gap between the rehabilitation team and cardiac patients," in *Healthcare Informatics (ICHI), 2016 IEEE International Conference on*. IEEE, 2016, pp. 146–155.

[52] D. Zampunieris, "Implementation of a Proactive Learning Management System," in *E-Learn World Conference on E-Learning in Corporate, Government, Healthcare and Higher Education*, Hawaii, 2006, pp. 3145–3151.

[53] S. Reis, D. Shirnin, and D. Zampunieris, "Design of proactive scenarios and rules for enhanced e-learning," in *CSEDU 2012 - Proceedings of the 4th International Conference on Computer Supported Education*. Porto, Portugal: SciTePress, 2012, pp. 253–258.

[54] R. Want, T. Pering, and D. Tennenhouse, "Comparing autonomic and proactive computing," *IBM Systems journal*, vol. 42, no. 1, pp. 129–135, 2003.

[55] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[56] C. Hide, T. Moore, and M. Smith, "Adaptive kalman filtering for low-cost ins/gps," *The Journal of Navigation*, vol. 56, no. 1, pp. 143–152, 2003.

[57] J. Sasiadek and Q. Wang, "Sensor fusion based on fuzzy kalman filtering for autonomous robot vehicle," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 4. IEEE, 1999, pp. 2970–2975.

[58] A. P. Dempster, "A generalization of bayesian inference," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 30, no. 2, pp. 205–247, 1968. [Online]. Available: http://www.jstor.org/stable/2984504

[59] G. Shafer, *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[60] W. Elmenreich, "An introduction to sensor fusion," *Vienna University of Technology, Austria*, vol. 502, 2002.

[61] P. L. Rothman and R. V. Denton, "Fusion or confusion: knowledge or nonsense?" in *Data Structures and Target Classification*, vol. 1470. International Society for Optics and Photonics, 1991, pp. 2–12.

[62] L. Wald, "A european proposal for terms of reference in data fusion," in *Commission VII Symposium" Resource and Environmental Monitoring"*, vol. 32, no. 7, 1998, pp. 651–654.

[63] B. V. Dasarathy, "Information fusion-what, where, why, when, and how?" *Information Fusion*, vol. 2, no. 2, pp. 75–76, 2001.

[64] P. J. Nahin and J. L. Pokoski, "Nctr plus sensor fusion equals iffn or can two plus two equal five?" *IEEE Transactions on Aerospace and Electronic Systems*, no. 3, pp. 320–337, 1980.

[65] T. Yoshioka, T. T. Phuong, K. Ohishi, T. Miyazaki, and Y. Yokokura, "Variable noise-covariance kalman filter based instantaneous state observer for industrial robot," in *2015 IEEE International Conference on Mechatronics (ICM)*, March 2015, pp. 100–105.

[66] Z. Chen, H. Zou, H. Jiang, Q. Zhu, Y. Soh, and L. Xie, "Fusion of wifi, smartphone sensors and landmarks using the kalman filter for indoor localization," *Sensors*, vol. 15, no. 1, pp. 715–732, 2015.

[67] Y. Liu, X. Fan, C. Lv, J. Wu, L. Li, and D. Ding, "An innovative information fusion method with adaptive kalman filter for integrated ins/gps navigation of autonomous vehicles," *Mechanical Systems and Signal Processing*, vol. 100, pp. 605–616, 2018.

[68] D. Crisan and A. Doucet, "A survey of convergence results on particle filtering methods for practitioners," *IEEE Transactions on signal processing*, vol. 50, no. 3, pp. 736–746, 2002.

[69] F. Caron, M. Davy, E. Duflos, and P. Vanheeghe, "Particle filtering for multisensor data fusion with switching observation models: Application to land vehicle positioning," *IEEE Transactions on Signal Processing*, vol. 55, no. 6, pp. 2703–2719, June 2007.

[70] M. Turan, Y. Almalioglu, H. Gilbert, H. Araujo, T. Cemgil, and M. Sitti, "Endosensorfusion: Particle filtering-based multi-sensory data fusion with switching state-space model for endoscopic capsule robots," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

[71] N. Long, K. Wang, R. Cheng, K. Yang, and J. Bai, "Fusion of millimeter wave radar and rgb-depth sensors for assisted navigation of the visually impaired," in *Millimetre Wave and Terahertz Sensors and Technology XI*, vol. 10800. International Society for Optics and Photonics, 2018, p. 1080006.

[72] V. Nathan and R. Jafari, "Particle filtering and sensor fusion for robust heart rate monitoring using wearable sensors," *IEEE journal of biomedical and health informatics*, vol. 22, no. 6, pp. 1834–1846, 2017.

[73] B. Khaleghi, A. Khamis, F. O. Karray, and S. N. Razavi, "Multisensor data fusion: A review of the state-of-the-art," *Information fusion*, vol. 14, no. 1, pp. 28–44, 2013.

[74] A. P. Dempster, "Upper and lower probabilities induced by a multivalued mapping," in *Classic Works of the Dempster-Shafer Theory of Belief Functions*. Springer, 2008, pp. 57–72.

[75] M. Mezaal, B. Pradhan, and H. Rizeei, "Improving landslide detection from airborne laser scanning data using optimized dempster–shafer," *Remote Sensing*, vol. 10, no. 7, p. 1029, 2018.

[76] P. Mehrannia, A. A. Moghadam, and O. A. Basir, "A dempster-shafer sensor fusion approach for traffic incident detection and localization," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 3911–3916.

[77] W. Guo, X. Tang, J. Cheng, J. Xu, C. Cai, and Y. Guo, "Ddos attack situation information fusion method based on dempster-shafer evidence theory," in *International Conference on Artificial Intelligence and Security*. Springer, 2019, pp. 396–407.

[78] K. Bader, B. Lussier, and W. Schön, "A fault tolerant architecture for data fusion: A real application of kalman filters for mobile robot localization," *Robotics and Autonomous Systems*, vol. 88, pp. 11 – 23, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889015302943

[79] S. Yazdkhasti and J. Z. Sasiadek, "Multi sensor fusion based on adaptive kalman filtering," in *Advances in Aerospace Guidance, Navigation and Control*. Cham: Springer International Publishing, 2018, pp. 317–333.

[80] L. A. Zadeh, *On the validity of Dempster's rule of combination of evidence*. Electronics Research Laboratory, College of Engineering, University of California, Berkeley, 1979.

[81] L. A. Zadeh, "Review of a mathematical theory of evidence," *AI magazine*, vol. 5, no. 3, pp. 81–81, 1984.

[82] L. A. Zadeh, "A simple view of the dempster-shafer theory of evidence and its implication for the rule of combination," *AI magazine*, vol. 7, no. 2, pp. 85–85, 1986.

[83] X. Deng and W. Jiang, "An evidential axiomatic design approach for decision making using the evaluation of belief structure satisfaction to uncertain target values," *International Journal of Intelligent Systems*, vol. 33, no. 1, pp. 15–32, 2018.

[84] W. Jiang and W. Hu, "An improved soft likelihood function for dempster–shafer belief structures," *International Journal of Intelligent Systems*, vol. 33, no. 6, pp. 1264–1282, 2018.

[85] E. Lefevre, O. Colot, and P. Vannoorenberghe, "Belief function combination and conflict management," *Information fusion*, vol. 3, no. 2, pp. 149–162, 2002.

[86] F. Xiao and B. Qin, "A weighted combination method for conflicting evidence in multi-sensor data fusion," *Sensors*, vol. 18, no. 5, p. 1487, 2018.

[87] P. Smets, "The combination of evidence in the transferable belief model," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 12, no. 5, pp. 447–458, 1990.

[88] D. Dubois and H. Prade, "Representation and combination of uncertainty with belief functions and possibility measures," *Computational intelligence*, vol. 4, no. 3, pp. 244–264, 1988.

[89] R. R. Yager, "On the dempster-shafer framework and new combination rules," *Information sciences*, vol. 41, no. 2, pp. 93–137, 1987.

[90] U. Mönks, *Information Fusion Under Consideration of Conflicting Input Signals*. Springer, 2017.

[91] R. Li and V. Lohweg, "A novel data fusion approach using two-layer conflict solving," 2008.

[92] J. J. Oresko, Z. Jin, J. Cheng, S. Huang, Y. Sun, H. Duschl, and A. C. Cheng, "A wearable smartphone-based platform for real-time cardiovascular disease detection via electrocardiogram processing," *IEEE Transactions on Information Technology in Biomedicine*, vol. 14, no. 3, pp. 734–740, 2010.

[93] J. Rickard, S. Ahmed, M. Baruch, B. Klocman, D. O. Martin, and V. Menon, "Utility of a novel watch-based pulse detection system to detect pulselessness in human subjects," *Heart Rhythm*, vol. 8, no. 12, pp. 1895–1899, 2011.

[94] Z. Jin, J. Oresko, S. Huang, and A. C. Cheng, "Hearttogo: a personalized medicine technology for cardiovascular disease prevention and detection," in *Life Science Systems and Applications Workshop, 2009. LiSSA 2009. IEEE/NIH*. IEEE, 2009, pp. 80–83.

[95] S. Patel, R. Hughes, T. Hester, J. Stein, M. Akay, J. G. Dy, and P. Bonato, "A novel approach to monitor rehabilitation outcomes in stroke survivors using wearable technology," *Proceedings of the IEEE*, vol. 98, no. 3, pp. 450–461, 2010.

[96] T. Hester, R. Hughes, D. M. Sherrill, B. Knorr, M. Akay, J. Stein, and P. Bonato, "Using wearable sensors to measure motor abilities following stroke," in *Wearable and Implantable Body Sensor Networks, 2006. BSN 2006. International Workshop on*. IEEE, 2006, pp. 4–pp.

[97] D. A. Coast, R. M. Stern, G. G. Cano, and S. A. Briller, "An approach to cardiac arrhythmia analysis using hidden markov models," *IEEE Transactions on biomedical Engineering*, vol. 37, no. 9, pp. 826–836, 1990.

[98] M. Srinivas, T. Basil, and C. K. Mohan, "Adaptive learning based heartbeat classification," *Bio-medical materials and engineering*, vol. 26, no. 1-2, pp. 49–55, 2015.

[99] X. D. Huang, Y. Ariki, and M. A. Jack, *Hidden Markov models for speech recognition*. Edinburgh university press Edinburgh, 1990, vol. 2004.

[100] M. Gales and S. Young, "The application of hidden markov models in speech recognition," *Foundations and trends in signal processing*, vol. 1, no. 3, pp. 195–304, 2008.

[101] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[102] F. Salfner and M. Malek, "Using hidden semi-markov models for effective online failure prediction," in *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*. IEEE, 2007, pp. 161–174.

[103] M. Brand, N. Oliver, and A. Pentland, "Coupled hidden markov models for complex action recognition," in *Computer vision and pattern recognition, 1997. proceedings., 1997 ieee computer society conference on*. IEEE, 1997, pp. 994–999.

[104] S. Hu, Z. Shao, and J. Tan, "A real-time cardiac arrhythmia classification system with wearable electrocardiogram," in *Body Sensor Networks (BSN), 2011 International Conference on*. IEEE, 2011, pp. 119–124.

[105] R. V. Andreão, B. Dorizzi, and J. Boudy, "ECG signal analysis through hidden markov models," *IEEE Transactions on Biomedical engineering*, vol. 53, no. 8, pp. 1541–1549, 2006.

[106] T. Al-Ani, Y. Hamam, R. Fodil, F. Lofaso, and D. Isabey, "Using hidden markov models for sleep disordered breathing identification," *Simulation Modelling Practice and Theory*, vol. 12, no. 2, pp. 117–128, 2004.

[107] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[108] G. D. Forney, "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[109] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38, 1977.

[110] B.-H. Juang and L. R. Rabiner, "The segmental k-means algorithm for estimating parameters of hidden markov models," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 9, pp. 1639–1641, 1990.

[111] L. E. Baum and J. A. Eagon, "An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology," *Bull. Amer. Math. Soc*, vol. 73, no. 3, pp. 360–363, 1967.

[112] J. Welch, P. Ford, R. Teplick, and R. Rubsamen, "The massachusetts general hospital-marquette foundation hemodynamic and electrocardiographic database–comprehensive collection of critical care waveforms," *Clinical Monitoring*, vol. 7, no. 1, pp. 96–97, 1991.

[113] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley,

"Physiobank, physiotoolkit, and physionet," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000.

[114] L. R. Rabiner, B.-H. Juang, S. Levinson, and M. Sondhi, "Some properties of continuous hidden markov model representations," *AT&T technical journal*, vol. 64, no. 6, pp. 1251–1270, 1985.

[115] D. A. Coast, *Cardiac arrhythmia analysis using hidden Markov models*. UMI, 1988.

[116] A. Hashemi, M. Rahimpour, and M. R. Merati, "Dynamic gaussian filter for muscle noise reduction in ecg signal," in *2015 23rd Iranian Conference on Electrical Engineering*. IEEE, 2015, pp. 120–124.

[117] M. Tomasini, S. Benatti, B. Milosevic, E. Farella, and L. Benini, "Power line interference removal for high-quality continuous biosignal monitoring with low-power wearable devices," *IEEE Sensors Journal*, vol. 16, no. 10, pp. 3887–3895, 2016.

[118] C. W. Mundt, K. N. Montgomery, U. E. Udoh, V. N. Barker, G. C. Thonier, A. M. Tellier, R. D. Ricks, R. B. Darling, Y. D. Cagle, N. A. Cabrol *et al.*, "A multiparameter wearable physiologic monitoring system for space and terrestrial applications," *IEEE Transactions on Information Technology in Biomedicine*, vol. 9, no. 3, pp. 382–391, 2005.

[119] M. Schmidt, A. Schumann, J. Müller, K.-J. Bär, and G. Rose, "Ecg derived respiration: comparison of time-domain approaches and application to altered breathing patterns of patients with schizophrenia," *Physiological measurement*, vol. 38, no. 4, p. 601, 2017.

[120] N. Sadr and P. de Chazal, "A fast principal component analysis method for calculating the ecg derived respiration," in *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2018, pp. 5294–5297.

[121] U. Anliker, J. A. Ward, P. Lukowicz, G. Troster, F. Dolveck, M. Baer, F. Keita, E. B. Schenker, F. Catarsi, L. Coluccini *et al.*, "AMON: a wearable multiparameter medical monitoring and alert system," *IEEE Transactions on information technology in Biomedicine*, vol. 8, no. 4, pp. 415–427, 2004.

[122] E. Lupu, N. Dulay, M. Sloman, J. Sventek, S. Heeps, S. Strowes, K. Twidle, S.-L. Keoh, and A. Schaeffer-Filho, "Amuse: autonomic management of ubiquitous e-health systems," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 3, pp. 277–295, 2008.

[123] Y. Hongxu and N. Jha, "A hierarchical health decision support system for disease diagnosis based on wearable medical sensors and machine learning ensembles," *IEEE Transactions on Multi-Scale Computing Systems*, 2017.

[124] G. Suciu, A. Vulpe, R. Craciunescu, C. Butca, and V. Suciu, "Big data fusion for ehealth and ambient assisted living cloud applications," in *Communications and Networking (BlackSeaCom), 2015 IEEE International Black Sea Conference on*. IEEE, 2015, pp. 102–106.

[125] A. Subasi, "EEG signal classification using wavelet feature extraction and a mixture of expert model," *Expert Systems with Applications*, vol. 32, no. 4, pp. 1084–1093, 2007.

[126] N. Paradkar and S. R. Chowdhury, "Cardiac arrhythmia detection using photoplethysmography," in *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2017, pp. 113–116.

[127] H. Liu, Y. Wang, and L. Wang, "The effect of light conditions on photoplethysmographic image acquisition using a commercial camera," *IEEE journal of translational engineering in health and medicine*, vol. 2, pp. 1–11, 2014.

[128] R. Bommi, V. Monika, R. Narmadha, K. Bhuvaneswari, and L. Aswini, "Imu-based indoor navigation system for gps-restricted areas," in *International Conference on Computer Networks and Communication Technologies*. Springer, 2019, pp. 157–166.

[129] Webots, "http://www.cyberbotics.com," commercial Mobile Robot Simulation Software. [Online]. Available: http://www.cyberbotics.com

[130] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *European Conference on Artificial Life*. Springer, 1995, pp. 704–720.

[131] Q. Chen, D. Ding, X. Wang, A. X. Liu, and A. Munir, "A speed hump sensing approach to global positioning in urban cities without gps signals," in *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2017, pp. 1–8.

[132] S. Deep, S. Raghavendra, and B. Bharath, "Gps snr prediction in urban environment," *The Egyptian Journal of Remote Sensing and Space Science*, vol. 21, no. 1, pp. 83–85, 2018.

[133] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[134] Z. Wang and T. Oates, "Encoding time series as images for visual inspection and classification using tiled convolutional neural networks," in *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[135] T. Ince, S. Kiranyaz, L. Eren, M. Askar, and M. Gabbouj, "Real-time motor fault detection by 1-d convolutional neural networks," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 11, pp. 7067–7075, 2016.

[136] L. Jing, T. Wang, M. Zhao, and P. Wang, "An adaptive multi-sensor data fusion method based on deep convolutional neural networks for fault diagnosis of planetary gearbox," *Sensors*, vol. 17, no. 2, p. 414, 2017.

[137] X. Guo, L. Chen, and C. Shen, "Hierarchical adaptive deep convolution neural network and its application to bearing fault diagnosis," *Measurement*, vol. 93, pp. 490–502, 2016.

[138] O. Abdeljaber, O. Avci, S. Kiranyaz, M. Gabbouj, and D. J. Inman, "Real-time vibration-based structural damage detection using one-dimensional convolutional neural networks," *Journal of Sound and Vibration*, vol. 388, pp. 154–170, 2017.

[139] O. Janssens, V. Slavkovikj, B. Vervisch, K. Stockman, M. Loccufier, S. Verstockt, R. Van de Walle, and S. Van Hoecke, "Convolutional neural network based fault detection for rotating machinery," *Journal of Sound and Vibration*, vol. 377, pp. 331–345, 2016.

[140] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu, "Convolutional neural networks for time series classification," *Journal of Systems Engineering and Electronics*, vol. 28, no. 1, pp. 162–169, Feb 2017.

[141] A. Gron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st ed. O'Reilly Media, Inc., 2017.

[142] Y. Bengio, P. Simard, P. Frasconi *et al.*, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[143] T. de Bruin, K. Verbert, and R. Babuška, "Railway track circuit fault diagnosis using recurrent neural networks," *IEEE transactions on*

neural networks and learning systems, vol. 28, no. 3, pp. 523–533, 2017.

[144] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[145] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proceedings*. Presses universitaires de Louvain, 2015, p. 89.

[146] 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.

[147] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[148] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[149] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *arXiv preprint arXiv:1712.04621*, 2017.

[150] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, *Dataset Shift in Machine Learning*. The MIT Press, 2009.

[151] A. Gretton, A. Smola, J. Huang, M. Schmittfull, K. Borgwardt, and B. Schölkopf, "Covariate shift by kernel mean matching," *Dataset shift in machine learning*, vol. 3, no. 4, p. 5, 2009.

[152] L. Lenc and P. Král, "Deep neural networks for czech multi-label document classification," in *International Conference on Intelligent Text Processing and Computational Linguistics*. Springer, 2016, pp. 460–471.

[153] J. Yanase and E. Triantaphyllou, "A systematic survey of computer-aided diagnosis in medicine: Past and present developments," *Expert Systems with Applications*, p. 112821, 2019.

[154] M. J. Zaki, "Scalable algorithms for association mining," *IEEE transactions on knowledge and data engineering*, vol. 12, no. 3, pp. 372–390, 2000.

[155] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *ACM sigmod record*, vol. 29, no. 2. ACM, 2000, pp. 1–12.

[156] L. Hou and N. Bergmann, "Novel industrial wireless sensor networks for machine condition monitoring and fault diagnosis," *Instrumentation and Measurement, IEEE Transactions on*, vol. 61, pp. 2787–2798, 10 2012.

[157] A. Harrison and P. Newman, "Ticsync: Knowing when things happened," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 356–363.

[158] J.-O. Nilsson and P. Händel, "Time synchronization and temporal ordering of asynchronous sensor measurements of a multi-sensor navigation system," in *IEEE/ION Position, Location and Navigation Symposium*. IEEE, 2010, pp. 897–902.

[159] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM, 2004, pp. 39–49.

[160] M. A. Sarvghadi and T.-C. Wan, "Message passing based time synchronization in wireless sensor networks: A survey," *International Journal of Distributed Sensor Networks*, vol. 12, no. 5, p. 1280904, 2016.

[161] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, "Robust multi-hop time synchronization in sensor networks." in *International Conference on Wireless Networks*, 2004, pp. 454–460.

[162] B. Kusy, P. Dutta, P. Levis, M. Maroti, A. Ledeczi, and D. Culler, "Elapsed time on arrival: a simple and versatile primitive for canonical time synchronisation services," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 1, no. 4, pp. 239–251, 2006.

[163] Z. Shang and H. Yu, "A low overhead multi-hop time-sync protocol for wireless sensor networks," in *Proceedings. 2005 IEEE Networking, Sensing and Control, 2005.* IEEE, 2005, pp. 54–59.

[164] C.-M. Chao and Y.-C. Chang, "A power-efficient timing synchronization protocol for wireless sensor networks," *Journal of information science and engineering*, vol. 23, no. 4, pp. 985–997, 2007.

[165] H.-L. E. G. on Artificial Intelligence, "Ethics guidelines for trustworthy ai." [Online]. Available: https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai