Reducing the need for communication in wireless sensor networks using machine learning and planning techniques

Stefan Hugtenburg

TU Delft

Reducing the need for communication in wireless sensor networks using machine learning and planning techniques

by

Stefan Hugtenburg

to obtain the degree of Master of Science
at Delft University of Technology,
to be defended publicly on Monday July 17, 2017 at 10:00 AM.

Thesis committee:	Dr. M. M. de Weerdt,	TU Delft, chair & supervisor
	Dr. M. T. J. Spaan,	TU Delft, co-supervisor
	Dr. P. Pawełczak	TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Abstract

Wireless sensor networks are commonly used to remotely and automatically monitor environments. One of the main challenges in wireless sensor networks is to use the limited available energy as efficiently as possible, to ensure longevity of the network. For such networks to survive their intended deployment period no energy may be wasted on inconsequential actions. As communication is one of the most energy-consuming tasks a sensor mote can perform, we propose a set of techniques that allow a base station to form an accurate environmental state estimation from a selected subset of measurements. In this thesis we present a novel methodology that combines three forms of intelligence. The sensor mote and base station both maintain a neural network-based predictor of the environmental state, which the sensor mote uses as input for different controllers (both handmade and based on Partially Observable Markov Decision Processes) that determine the actions performed by the sensor mote. Armed with the prediction mechanism, a model of the environment, the controller executed by the sensor mote, and the reported measurements, the base station performs computations akin to those commonly used with Hidden Markov Models to form an accurate environmental state estimation. We apply our techniques to real world data sets and reduce the required number of report operations by over 90% whilst incurring only minimal accuracy penalties.

Preface

Well here we are, it's done. Having spent one and a half year working on this thesis it feels great to finally be able to write that. As I know I can be quite verbose in writing, I hope you will forgive me for a rather lengthy preface. Through some reflection and meandering, we'll get to some words of thanks. Many prefaces I have read before are often in a similar formal style to the rest of the thesis. Although certainly possible, I wanted this preface to be a bit more personal and as a result slightly less formal.

Knowing that my master thesis took about 18 months is something that I still regret, but when placed in context I do not mind so much. Even though I could probably have finished my thesis much sooner, the time spent on other tasks is what has ultimately resulted in a job I am very much looking forward to. Student assistant jobs and a variety of other activities have given me the opportunity to work as a teacher at TU Delft for the next two years.

Unsurprisingly perhaps, there are many people that have helped me to get to the thesis you are now reading. Some have contributed in terms of content, some in terms of form, others by simply providing great topics for lunch-break discussions. Hopefully I have made the following list complete. Unfortunately a result of 18 months of thesis work means that I might be forgetting some people. Most importantly in terms of content, I would of course like to thank my "official" daily supervisors Mathijs and Matthijs. Another name that "unofficially" is a part of that list is that of Frits. Notwithstanding the fact that this thesis work has turned out unrelated to his PhD work, he still invested time and effort in providing clear and constructive feedback, for which I am very grateful. I would also like to thank the rest of the algorithmics department for their warm welcome into the group. Some of your comments and constructive feedback have helped shape the thesis you now have in front of you.

The second group of people that deserve some words of gratitude has played a role not only in my thesis, but also throughout the rest of my education. Having known Pim Otte for about 11 years now, I am still surprised at the ease with which we can work together on solving problems and studying new things. Although I have collected a new group of friends at TU Delft, I am very glad that he is even now a part of that group. Together with Jesse, Tim, Otto, Pim Veldhuisen, and Elvan, the many distractions in the form of movie nights, game sessions, and "interesting" news articles have helped to get me through my studies. Whereas I might now have given the impression that this group has provided mostly stress relief, I should stress that they have also provided very useful and constructive feedback on ideas and thesis drafts. Either way, this thesis would not have been the same without them.

For advice and moral support, there are two other groups of people I should thank. In the first place my family for their support throughout not just my thesis, but also the rest of my education. Gratitude seems too weak a word to express my feelings for all they have done, including the many occasions on which they provided the moral support I needed. Hobbies are also great stress-relievers of course and it is in that context that I would also like to mention the great people that I have met through judo. These few hours a week during which I could fully forget about study-related worries were very precious to me.

Finally then this preface can come to an end. Often times in the past, I have ended pieces of text or speech I deem important with a quote. Rarely have there been situations for which I did not feel some sort of quote could be appropriate. Old habits die hard it seems. There is the one quote that I normally used for the really important pieces of text and the careful reader will find it is also partially included in this preface. However to end with it seems another quote might be more appropriate:
"Ends are not bad things, they just mean that something else is about to begin." (by C. JoybBell C.)

Regards,
Stefan Hugtenburg

Contents

List of Figures

1 Introduction

Although mankind has come a long way in improving our living conditions and environments, it can not be denied that we have also negatively influenced many ecosystems. Through our own expansion and search for knowledge we have disrupted many preciously balanced systems with disastrous results. Even now our academic pursuits often require us to observe wildlife in its natural environment, which introduces huge risks through disruption of the local ecosystem. Not only does this result in measurements that can be considered dubious as the habitat and behaviour of the wildlife is disturbed, it can even lead to the death of the wildlife we wish to study. As research in Maine [7] suggests, a 15 minute visit to bird colonies can result in up to 20% mortality among eggs and chicks. Sending humans to observe an ecosystem carries risk and results in many negative side-effects.

With computational units rapidly becoming smaller and more intelligent, digital devices can be applied in a non-intrusive manner in many new situations. Small devices placed in or around nesting locations allow us to remotely monitor wildlife without disrupting the ecosystem. In other domains such small and portable devices are also gaining popularity. Whether it is home-automation in which small devices can be used to automate household-appliances like the coffee machine, or in military applications in which sensor nodes can be used to monitor an area for enemy activity, they all share common goals. With the ability to obtain and send sensor readings, they are well equipped to monitor environments and wildlife. As Mainwaring et al. [57] described in 2002, a wireless sensor network can perform the necessary measurements in the ecosystem of the Great Duck Island, whilst minimising the impact and invasiveness on the ecosystem. The network observes the behaviour of birds during the breeding season, studying bird presence patterns in the nesting burrows. By connecting the “sensor patches” to base stations through gateways the information can logged for future, remote study by for instance biologists.

As these sensor nodes (or “motes”) can be placed before the breeding season starts, human involvement can be limited to times outside of the critical seasons. However, these critical seasons can span multiple months or even years depending on the environment. Many other use cases like temperature and humidity monitoring in green houses or crop fields also demand long life cycles. To ensure the entirety of the critical season can be monitored, it is important that the mote uses its limited energy supply effectively.

Depending on the use case it is also important that the data is available in the base station during the monitoring, rather than only after the period of interest has passed. In the case of green houses for instance, the temperature is monitored so that heat lamps can be turned on if it drops beneath the acceptable level. In other applications, such as sensor networks that monitor forest fires, overflowing rivers, or illegal trappers hunting wildlife, the need for immediate action is obvious. For such use cases we want to ensure that the base station has an accurate estimation of the environmental state at all times, so that we can immediately take action to prevent the situation from escalating (e.g., by calling the fire brigade in case of a forest fire).

Limited	Sensor Mote	Base station
Power	Yes	No
Available memory	Yes	No
Processing power	Yes	No
Processing time	Yes	Yes

Table 1.1: The limitations imposed on the base station and sensor motes.

1.1. Problem statement

In wireless sensor networks there are two types of components. The first is a sensor mote (or “source”), which can perform and report measurements. Second is the base station (or “sink”), which aggregates the data reported by the sensor motes. Through cooperation of these components we aim to obtain an accurate state estimation in the base station and a long battery life in the sensor motes. There are several different limitations in the base station and sensor motes that result in conflicts for fulfilling the goal.

The difference in hardware between the base station and sensor mote is the most important reason for their different limitations. The sensor motes operate wirelessly and have a limited power supply in the form of batteries. One of the most expensive operations a sensor mote can perform in terms of battery life, is reporting its measurement to the base station. Much research has already been done to reduce the required power for communication. With new routing techniques [1, 40, 93] and adaptations of network protocols [25] battery life can be extended by large amounts of time already. Far less research has been done to investigate what changes can be made in the application layer of the network. In monitoring applications sensor motes often do only little to no local computation, but instead send all their data to a central base station where it is recorded for offline processing [43, 86]. As discussed by Anastasi et al. [6] in their survey of energy conservation in Wireless Sensor Networks (WSNs) communication is more expensive than computation in terms of power used. These computations should not take very long to complete, as the base station may require reports of important measurements in a timely fashion. Due to the limited and cheap hardware the motes consist of, the computations that they can perform are limited. They should not be overly complex, they should be quick, and with only several KB of memory, they should not depend on the availability of large data structures.

The base station often has fewer limitations. As the base station has a constant power supply and is often connected to the outside world via the Internet, it has much more processing power available. With the Internet connection it can off-load work to a processing cluster or cloud, if it can not process the information itself. The main limitation for the base station is a time-constraint. If it has to take actions based on reports by the sensor motes, it should be able to quickly process these reports. The limitations of the base station and sensor mote are summarised in Table 1.1.

These limitations also influence the goals of the individual components. Sensor motes want to ensure their power lasts as long as possible, that is they aim to “survive” the intended time period. As communication consumes a lot of power, one way to achieve this is to minimise the number of measurements they report. The base station on the other hand is tasked with maintaining an accurate estimate of the environmental state and producing a log containing an entry of the environmental state at every time step. To create an accurate log it requires information about the environmental state from the sensor motes, in other words it requires reports from the sensor motes. It can produce the most accurate log when the sensor motes send it all of the measurements. These conflicting goals form a trade-off between accuracy and battery life that we explore in this thesis.

1.2. Research questions

This trade-off between accuracy and battery life has been explored before. In their survey of energy conservation techniques, Anastasi et al. [6] formulate a taxonomy of such techniques. For this thesis work we focus on a data-driven approach, further specified as data reduction using data prediction.

Similar to other data prediction techniques described by Anastasi et al., our approach also assumes a model of the environmental state is shared between the sensor motes and the base station. Sensor motes compare their measurements to this model and report them if the measurement falls outside

of the allowed tolerance. This model is the predictor in the "data prediction" classifier. By updating the model in the sensor mote and base station at the same time, such shared knowledge can provide valuable information about the environment even when no measurements are actually reported by the sensor motes.

Previous work in this category is best represented by what Chu et al. [22] call the Ken solution. This solution substitutes the prediction for the state estimation if no measurements are received. By defining a maximum tolerated error, there are clear bounds on the accuracy, and up to 80% of the reports can be saved. There are two extensions on this work that we handle in this thesis. First, as the authors of the Ken solution point out, stochastic actions in the sensor motes can further help reduce the required number of report operations. Second, more advanced prediction mechanisms can help also reduce the number of report operations. It is these two extensions, or forms of "increased intelligence" that form the inspiration of this thesis work. Through them we set out to answer the following research question:

Research question. *What is the effect of adding more intelligence to sensor motes on the accuracy of the state estimation by the base station and the battery life of the sensor motes?*

We focus solely on adding more intelligence to the application layer of the sensor network. For the purposes of this thesis therefore we assume that the routing layer of the network handles issues such packets that are dropped and never make it to the base station. Similarly we do not consider in-depth other energy conservation techniques which involve sending more data after short delay as opposed to sending every measurement immediately. We discuss several of these techniques in Chapter 2, in which we discuss related work.

To answer the main research question, we define three sub research questions that focus on different parts of the intelligence that we add to the sensor motes and how we can reason about that intelligence.

To implement stochastic action selection in the sensor motes, we have them execute stochastic finite state controllers that operate on the measurements taken and predictions made. These controllers determine the behaviour of the sensor mote, in other words: do we report a measurement or not? To allow the base station to reason about *why* the sensor mote has taken a certain action, every report operation not only includes the measurement taken by the sensor mote, but also includes the current state in this finite state controller. Using the design of the controller and a model of the environment, the base station then has to produce an accurate environmental state estimation. The question we pose is:

Sub research question 1. *How can the base station use information about the state of the controller to estimate the environmental state, whilst it has access to only a subset of the measurements?*

As an improved prediction model we consider neural networks, which have been applied to predict time-based data before [28, 68]. In previous work the networks are usually very complex which makes them unfeasible to be applied in sensor motes directly. Additionally we consider updating the network as predictions differ too much. The variety of parameters (network training, update rate, maximum allowed error) allows for much freedom in its configuration. The question we ask is:

Sub research question 2. *By how much can the communication of sensor motes be reduced through the introduction of prediction mechanisms before the loss of accuracy in the base station is considered unacceptable?*

Finally we combine these two extensions in an experimental evaluation. In an attempt to find the optimal controllers for a given use case, we provide a systematic approach that can be applied for any use case in the domain of environment monitoring using wireless sensor motes. This approach models the use case as a Partially Observable Markov Decision Process (POMDP) which can be solved to a stochastic finite state controller. The question we pose is:

Sub research question 3. *Compared to handmade controllers based on rules of thumb, by how much can POMDP-based controllers decrease the report rate without affecting the accuracy?*

1.3. Contributions

Through the investigations of these research questions, this thesis puts forth three major contributions. The first is the base station logic that allows us to reason about the state of the environment using the state of the controller in the sensor mote as well as a subset of reported observations. The state estimation obtained by this logic is efficiently computable and the logic is sufficiently abstract to work with many different configurations of finite state controllers. As a result we can have fine-grained control over the trade-off between accuracy and report rate. Furthermore the inclusion of the controller allows us to obtain more accurate logs with the same number of reports.

Secondly we offer an analysis of different prediction mechanisms to predict the daily temperature trend inside an office building. Comparing them to two baseline (untrained) predictors that assume the temperature to be constant or linear, we analyse different neural network predictors. We analyse the different parameters that can be configured in these neural network configurations with respect to the trade-off between report rate and accuracy. Although the neural network predictors obtain similar performance to the baseline predictors, they are unable to consistently outperform them. This is likely due to differences between the training and evaluation data sets used in the evaluation which impact the neural network predictors, but do not influence the performance of the baseline (untrained) predictors.

Finally we perform an experimental evaluation of the systemic approach to finding the optimal controller for a given use case. We evaluate a variety of controllers that all balance the trade-off in slightly different ways. We also evaluate the POMDP-based controllers and compare their performance to handmade controllers based on rules of thumb. This evaluation provides great insight into what makes certain types of controllers work well for a given use case.

1.4. Structure of this thesis

Figure 1.1 presents a visual overview of the methodology developed in this thesis. The green items represent components that are present in “traditional” wireless sensor network deployments. There is an environment monitored by a sensor network, this network reports to a base station which logs the measurements for analysis. The blue components represent that additions we make to this “traditional” set-up. Each of these components also contains a link to the chapter in which we develop this addition. As input for these additions, we require the yellow components, which are (derivations from) data sets that are used to model the environment and the trade-off between accuracy and report rate. Finally the red components are methods we use in this thesis that have been previously explored in literature, but are not improved upon in this thesis.

First we investigate related work in Chapter 2 to determine what has been done before in the context of wireless sensor networks and what optimisation techniques we could apply to balance our trade-off. In Chapter 3 we design a mathematically grounded policy for the base station to reason about the environmental state given a subset of the measurements and a stochastic controller in the sensor mote. Chapter 4 continues with the notion of controllers, by considering the design of the controllers for the sensor motes that allow the base station to execute our policy. Chapters 5 and 6 focus on designing and evaluating prediction mechanisms based on neural networks. These chapters can be read separately from Chapters 3 and 4 as these focus on an exploration of different prediction mechanisms and do not consider the notion of a controller. Finally we combine the two forms of intelligence in an evaluation of different controllers that use different prediction mechanisms. This evaluation is described in Chapter 7, in which we evaluate a large variety of controllers that the sensor mote can execute in conjunction with the predictor to report only those measurements that inform the base station most.

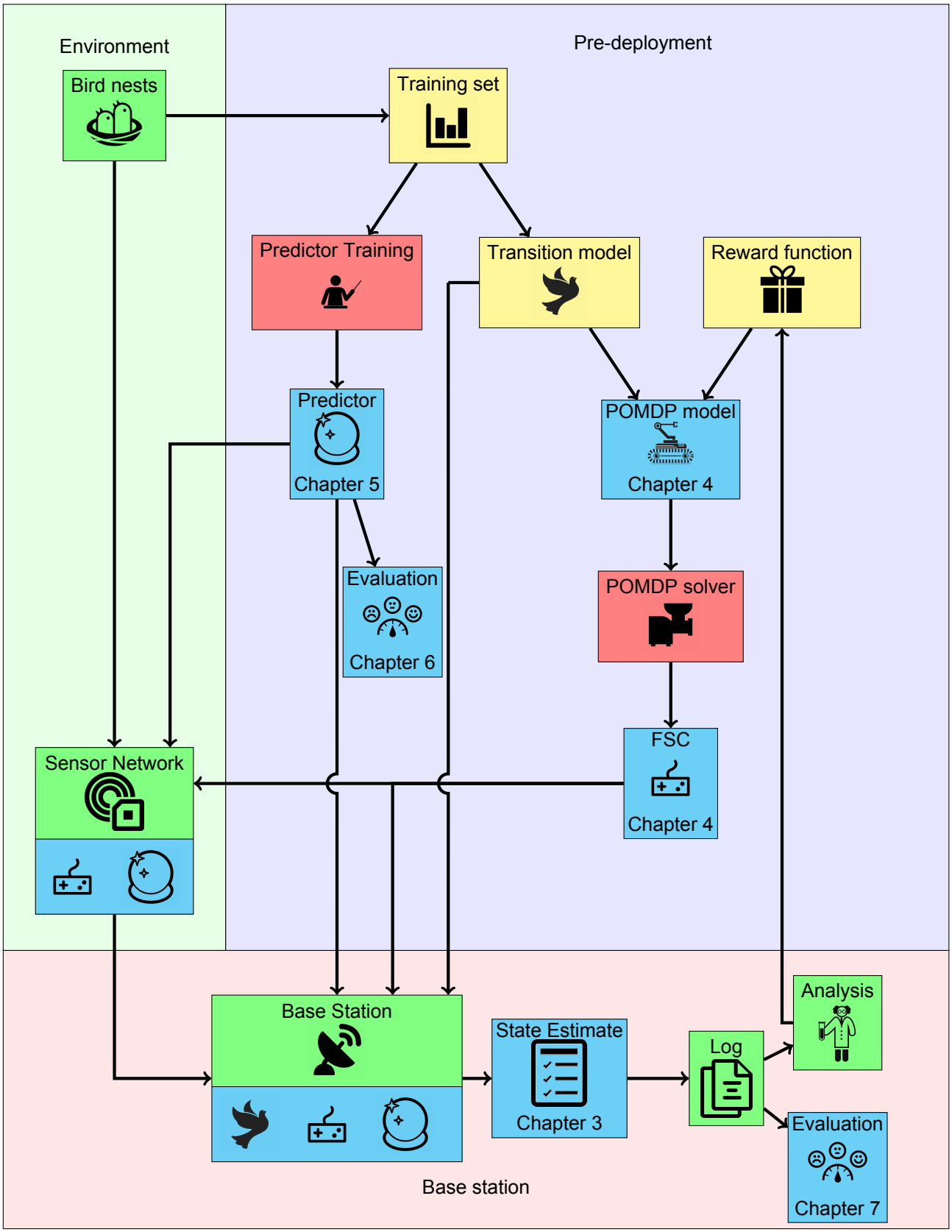


Figure 1.1: A visual overview of the different models and processes that this thesis combines. Green components are part of "traditional" WSNs, blue components are added by us and explored in this thesis. red and yellow components are existing techniques and data sets respectively that are needed for our additions.

2

Related Work

For our data-prediction energy conservation technique, we combine research from the field of wireless sensor networks and the theory of sequential decision making problems. In this chapter we present an overview of the state-of-the-art techniques, frameworks, and algorithms deployed in these fields. First we examine the contexts of wireless sensor networks in Section 2.1. We then look at the techniques for solving decision making problems in Section 2.2. In Section 2.3 we examine work that already combines these two fields in the form of Hidden Markov Models (HMMs).

2.1. Application domain: wireless sensor networks

Over the last decades advances in electrical engineering have led to computational units becoming increasingly cheaper and smaller. Whereas initially this led to computers becoming available to consumers, we have long surpassed that state and are now able to mass produce computational units that are no larger than a coin [67]. Although such tiny computational units have very limited processing capabilities, they can monitor and act within environments by equipping them with sensors and wireless communication to off-load the collected data. These "wireless sensor motes" can be deployed in a variety of circumstances leading to all kinds of new applications. Due to their small physical dimensions and low manufacturing costs, large networks of these sensors can be rapidly deployed to monitor environments in an unobtrusive manner.

We describe the requirements imposed on these sensor motes and networks in Section 2.1.1. With these requirements in mind we look at some application domains found in literature in Section 2.1.2. Finally in Section 2.1.3 we discuss the state-of-the-art routing protocols and algorithms that consider these requirements and are used in the identified application domains.

2.1.1. Requirements of WSNs

As with any technology, wireless sensor networks bring many challenges of their own. As weight, physical dimensions and cost are minimised they need to use whatever little energy they have efficiently and they lack the processing power and resources to perform extensive computations. As WSNs share many of their properties with ad-hoc networks [69], some of the challenges have already been extensively investigated. However, wireless sensor networks offer some new challenges as well. Akyildiz et al. [2] identify several such challenges in their survey on wireless sensor networks. We highlight the six relevant challenges here:

- WSNs can contain many nodes, orders of magnitude more nodes than an ad-hoc network, thus scalability takes a very prominent role. Communication protocols need to work well even with thousands of nodes.

- Sensor motes are densely deployed, thus there can be overlap in their measurements or interference from their communication. This allows communication protocols to consider aggregating data within the network, to avoid sending redundant messages or needlessly blocking the communication channel.

Sensor Mote	Processor	RAM (KB)	Program Memory (KB)	Battery	TinyOS	Contiki-OS
BTnode3	ATMega128	64	128	2xAA	✓	
Cricket	ATMega128	4	128	2xAA	✓	
micaz	ATMega128	4	128	2xAA	✓	✓
Tmote Sky	TI MSP430	10	48	2xAA	✓	✓
wismote	TI MSP430	128	256	1xAAA		✓

Table 2.1: An updated overview of several types of sensor motes, their capabilities, and operating system support, based on Winkler et al. [92].

- Sensor motes are prone to failure. Due to the usage of cheap hardware to ensure low prices, the quality of individual motes is often fairly low. Thus communication protocols should be able to handle node failures.
- The topology of the network is likely to change. Although not applicable for all use-cases, mobility of the motes should be taken into account. Combined with the common failure of motes, changes in routing strategies are needed as the network remains deployed for longer periods of time.
- Sensor motes typically use broadcast mechanisms for communication as opposed to the point-to-point communication commonly found in ad-hoc networks. The reason for this difference is easily explained when we consider the cost. Having a simple transfer medium that simply sends data hoping that some other mote or base station receives it, is much cheaper to produce than specialised directed communication hardware.
- Sensor motes feature limited processing capabilities, hardware and battery power. This effectively ensures that sensor motes can only execute rather low-complexity policies. As these sensor motes are deployed in an environment that can require real-time operation, lack of memory and lower CPU speed forces us to keep the processing algorithms simple. Table 2.1 shows the limited memory available on these motes.

As operating systems created for normal machines do not take into account the specific requirements (most importantly energy efficiency), special custom operating systems have been developed for these wireless sensor motes. Two examples of these operating systems are TinyOS[1] and Contiki-OS[2], both featuring an active open-source development. To give an impression of the kind of hardware and software used, Table 2.1 lists several of the most-commonly used sensor mote types, their hardware specifications, and operating system support.

2.1.2. Applications of WSNs

Due to their small size and low production costs, sensor networks can be deployed in many different situations. Both Alyildiz et al. [2] and Garcia-Hernández et al. [27] identify several different application domains, of which we now describe four: environmental applications, security/military applications, health applications, and home applications.

Environment Applications

To help prevent an endangered species going extinct, it is important we learn about its behaviour so that we can protect their environment. By planting sensor motes in burrows before the breeding season [57, 85] or by putting collars on zebras once (a one time interference) [43, 94], the animals can be studied remotely without further interference in their environment. The alternative in the form of sending humans to check up on these endangered species in the wild might be counterproductive as it could at best throw off the readings as the animal changes its behaviour or at worst cause large amounts of distress due to the presence of the human being. The small size of wireless sensor motes can help to prevent such issues. Using a variety of sensors we can learn about movement patterns, feeding times, group behaviour, and many other so-far unstudied properties of wildlife.

Mainwaring et al. [57] have successfully studies bird presence patterns during the nesting period on Great Duck Island using a wireless sensor network. By measuring the temperature using the so-called

[1] https://github.com/tinyos/tinyos-main
[2] http://www.contiki-os.org/

Mica Weatherboard, they can infer the presence of a bird inside a burrow. In the ZebraNet WSN [43, 94] the sensor motes regularly record the positions of the zebras outfitted with special collars and report this data back to the base station. This has lead to the first record of zebra movement at night, leading to new insights in the way zebras live.

Other applications in the environmental domain focus on flora rather than fauna and include the tracking of pollution in water and air flows. Khedo et al. [47] describe a deployment on the island of Mauritius that monitors air pollution on the island now that the industry is picking up on the island. Not only do Khedo et al. propose a novel strategy that helps reduce the amount of data a cluster head needs to communicate, the authors also provide an intuitive interface that the Mauritian government can use to determine if the population needs to be evacuated because of excessive air pollution levels.

Security/Military Applications
Due to their ability to monitor their environment and rapidly aggregate data in a centralised command post, wireless sensor networks offer many advantages to the military. From monitoring the vitals of soldiers, their ammunition and location [83] to classifying and tracking potential enemy aircraft [8]. Ding et al. [24] describe WSNs for the purpose of uncovering radioactive materials intended for terrorist attacks. The ease of a (temporary) cheap deployment of such a WSN before a meeting of political leaders makes this an attractive choice in terms of security. With their deployment the network remains operational for up to 20 hours, which may not be sufficient for all use cases.

In their survey of military applications of wireless sensor networks, Winkler et al. focus on the use case of surveillance around a command post in an area of active engagement [92]. In such cases data security and encryption are additional requirements. If hostile forces can hide themselves by misinforming the network, then the deployment of the network is meaningless. Additionally even the notion of communication can already be a risk. Hostile forces could intercept the communication and even without understanding the contents, become aware of your presence. Another requirement mentioned by Winkler et al. is that sensors might occasionally be airdropped or launched by a rocket launcher. This introduces requirements on the physical structure of the sensor that you will not see in other application domains. Lee et al. [54] describe how such an airdropped or artillery-delivered network can form a network structure that can reliably send information back to a base station.

Healthcare Applications
Although many different kind of measuring devices are already employed in hospitals around the world, wireless sensor networks allow for new types of information to reach the system. For instance in elderly homes a simple wristwatch can help to detect inhabitants suffering from dementia wandering off or inhabitants that have fallen [4]. Some systems, such as that described by Suryadevera et al. [84], even monitor electrical appliance usage and sleeping habits inside the home, leading to new information that can be used to give health advice to the elderly. An obvious concern for privacy and general acceptance seems not be an overly large concern. As an exploratory survey by Steele et al. [82] reveals, the elderly seem generally positive towards using WSN to improve their healthcare provided that the benefit of the WSN is made clear to them.

Home and Office Applications
Finally we look at the home and office applications for wireless sensor networks, which are closely related to some of the healthcare applications we have already seen. Smart homes which have sensor motes embedded in many household appliances can allow remote operation of household appliances (thermostats, television recorders etc.). By recognising activities performed by residents [46], such appliances can automatically be operated at the right time. In the Intel research lab [15] several aspects of an office environment have been measured, such as the temperature and the amount of light. We describe the resulting data set in more detail in later chapters of this thesis as we use it for experimental validation.

2.1.3. Routing protocols & algorithms in WSNs

In all applications discussed in the previous section, energy constraints help to shape the role and behaviour of the wireless sensor motes. Much research has already been done in policies for the network and transport-layers of the network [1, 21, 40]. Traditional techniques such as the TCP or UDP protocol on top of the IP protocol can be applied, but require a lot of modification to be feasible in this

environment [25]. As a result, many custom routing mechanisms have been devised for communication in wireless sensor networks, exploiting characteristics such as geographical closeness or data redundancy. These techniques are often unaware of the application running on top of the network and focus only on reducing the battery cost of communication which means they can be combined with the new application-layer technique for saving battery life outlined in this thesis. In this section we describe several of these mechanisms outlining their potential benefits and differences. These routing protocols are based on the classification and explanations presented by Al-Karaki et al. in their survey on routing techniques in WSNs [3].

Flat Routing

Without any type of hierarchy within the network, flat routing essentially focuses on flooding messages in a smart manner. All motes carry the same responsibilities and communicate in the same way. A popular routing protocol in this category is *directed diffusion* as introduced by Intonanagowiwat et al. [40]. Focusing on scalability and robustness, directed diffusion works in three phases. First, a so-called "sink" can signal that it has interest in certain types of events. This interest is forwarded throughout the network and every mote will remember through what mote the interest reached it. Second, if an event that matches the interest is observed, it will be sent to the mote through which the interest reached it. Finally when this event data reaches the sink, the sink will reinforce the link by requesting more frequent updates. After the event period is over, or if a better route is found to the event location, the requested frequency can be lowered again, thus downgrading the link. Although good for on-demand data delivery, it is not well-suited for continuous delivery of data, which is an essential requirement for the monitoring applications described in Section 2.1.2.

Hierarchical Routing

Unlike flat routing, hierarchical routing assumes or places an explicit hierarchy in the sensor network. Using clustering and appointing cluster-heads, a hierarchy can easily be established within the network. As pointed out by Al-Karaki et al. [3], many of the techniques in this category answer the question of who should send the information and at what time, rather than focusing solely on routing. A fundamental example from this category that forms the basis of many of the hierarchical routing strategies can be found in *LEACH* the Low-Energy Adaptive Clustering Hierarchy [38]. This protocol architecture, as Heinzelman et al. call it, is evaluated in terms of ease of deployment, system lifetime, latency, and quality. The algorithm consists of rounds of two phases. During the first phase the k cluster heads for this round are randomly selected. The optimal value for k can be analytically determined based on several factors including the number of motes and the available energy. Having determined the cluster heads, motes decide what cluster head they join for this round, resulting in k clusters. During the second phase every cluster head creates a schedule for its cluster, outlining when what mote is allowed to communicate its information. This should avoid collisions and maximise sleep time for the motes. The idea is to spend most time in the second phase, as this minimises the overhead of the protocol. At the same time, it is also important for the role of cluster head to rotate often. As being the cluster head will require more energy, shifting the role around will result in more uniform battery lifetimes. Although these techniques can help with continuous data delivery, it does not help to answer the question of what data is worth reporting.

Location-based Routing

As wireless sensor networks are often spread throughout a large geographical area, using the physical location of a mote in a routing protocol could also lead to new insights. One such insight is found in Staffetta [21]. Staffetta uses the distance from a mote to the base station to determine how often it should wake up and listen or transmit data. The rule of thumb is that motes closer to the base station wake up more often. As a result when a random mote wakes up, the probability that it finds an awake mote closer to the base station is larger than finding one in the wrong direction. Thus for data collection, data is naturally inclined to move towards the base station. Whereas the performance of this strategy is very promising, it can quickly lead to nonuniform battery levels. After all motes that are closer wake up more often and thus use more power.

2.2. Solution space: decision-making problems

All of the techniques discussed in Section 2.1.3 aim to save battery in the communication layer of the network. However these techniques do not use the contents of the messages in this process. They are not aware of the application that the network is running, they only handle the routing layer of the network. On top of them an intelligent policy in the application layer of the network can be deployed that answers the questions of what data is important enough to be communicated and when it should be communicated. These questions can be modelled well by techniques found in the field of decision-making problems. In fact many problems in all kinds of different contexts can be boiled down to decision-making problems. When there is a traffic jam, should we stay on the highway or travel over smaller country roads to bypass it? Given a limited budget what gifts should we buy for what person? As human beings it is tough to decide what answer to these questions are optimal. Unfortunately many of these problems are often equally hard for machines to decide. Nevertheless, there are several problem models and techniques that can help to find (approximate) solutions to these decision-making problems.

Based on the categorisation by Kochenderfer et al. [48] we examine five different methods for modelling decision problems. Each of these methods makes different assumptions about the problem that is being modelled, for instance about the amount of data available before modelling and whether or not some modelling can be done online, i.e. during the decision-making process.

Explicit Programming

Although arguably the most naïve method, explicit programming is the most direct method of solving a problem. It essentially considers all the different scenarios that could present itself to the agent and instructs the agent on what to do in these scenarios. The general idea of an agent-oriented programming approach was first described by Shoham et al. in 1993 [76]. They put forward many notions often found in agent-based frameworks, such as belief states, message based communication, and taking actions based on observations. These general notions are also found in the other methods for modelling decision-making problems described below.

Supervised Learning

If you show a child several examples of fishes, mammals, lizards, and insects, the child is likely to automatically look for a pattern or set of characteristics that defines each of these groups. If you then show them a new picture of an animal they have not seen before, but one that clearly exhibits insect-like characteristics, a child can probably identify it as an insect. Humans in general are rather good at such methods of pattern recognition [87]. If you show us several examples of how to solve a problem (be it a classification problem or something else) then soon we are able to apply a similar solution strategy to new problems. This is exactly what supervised learning also aims to do in decision-making problems.

Supervised learning features an offline learning phase during which we show an agent what to do in certain scenarios. From this, the agent can then derive a common strategy that it can apply to new scenarios. Of course the efficiency of such techniques is largely dependent on the correctness of the learning problems and solutions. Even with perfect solutions to these training problems however, it is unlikely that such techniques will outperform human designers that specialise in solving these problems [48]. As this machine learning technique specialises in pattern detection, applications can be found in many different areas. In biology, or more specifically genetics, it can be used to find certain genetic characteristics [75], and in wireless sensor networks, it can be used to estimate link quality [88]. In Chapter 5 we propose neural networks as a prediction mechanism in the sensor motes. These neural networks are also a supervised learning technique.

Optimisation

Unlike supervised learning, optimisation approaches do not rely on a training set, but rather a mathematical definition of the design space. By expressing constraints and performance measures in a mathematical framework, algorithms that traverse this solution space looking for optima can be designed. As the solution space increases in size, the algorithms have to become more intelligent. Simply trying all solutions will quickly become unfeasible. Several well-known optimisation strategies are evolutionary and genetic algorithms [9, 31], tabu search [29, 30, 39], and the mathematical mixed-integer programming model which was already used in the 1960s [55] for which different frameworks are still

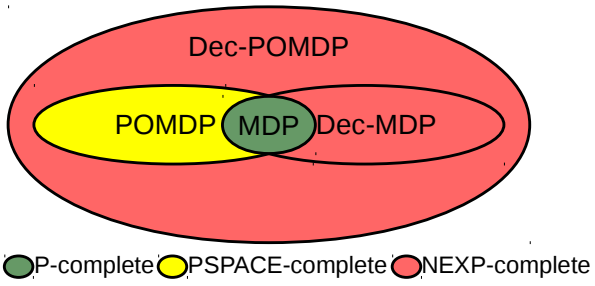


Figure 2.1: Superclasses of MDPs, based on [13, 48, 56], indicating the complexity class of solving the finite-horizon version of the problem class.

improving in efficiency. Gurobi[3]and CPLEX[4] are two such examples. The policy outlined in Chapter 3 is best qualified as an optimisation technique.

Planning

Whilst knowledge of the dynamics of the problem model might be known in general optimisation problems, planning problems are a form of optimisation in which this model is explicitly used in finding the right decision. These models do not necessarily have to be deterministic, which leads to planning under uncertainty. A commonly used model in this field is the Markov-Decision Process (MDP) model, which assumes that the next state of an agent depends only on its current state and its current action [10]. In other words, if we assume Markovian behaviour in the state transitions, the states form a Markov chain. This has many advantages as it means we do not have to keep track of a long history of states and actions, but can reason only about our current knowledge. Many variations of this problem model exist, including Partially-Observable MDPs (POMDPs) in which uncertainty over observations or local state can be introduced [77], and Decentralised (PO)MDPs in which we reason about multiple agents that either can or can not communicate, but lack a central command node that determines the actions for all agents [14, 73]. Figure 2.1 presents an overview of several of the different superclasses of MDPs. The systematic approach to create a controller for the sensor mote uses a POMDP model of the environment, as explained in Chapter 4.

Reinforcement Learning

Sometimes a problem domain is sufficiently complex so that no structured model of the interactions, possibilities and rewards within it exists (yet), rendering the approaches listed above unfeasible. For these situations reinforcement learning can help to find the right decision to make. The most famous example of reinforcement learning are the dogs of Pavlov that were trained to expect food when a bell rang. These dogs, who were unaware of the system before, learned in an online fashion, of the model in which bell ringing is followed by eating. Fortunately for these dogs, learning this model required little effort on their part, something that is not necessarily true for all situations. In general, agents will have to strike a balance between working towards their goal and exploring the domain. Consider for instance a transport problem, in which goods need to be transported from A to B. Once you have found a route from A to B, do you continually take that route, or do you try variations in hopes of improving on the route?

An example of a strategy in this area is Q-learning, which relates back to MDPs [90]. Q-learning focuses on learning the reward function based on the current state and the action it takes, which uses a learning rate parameter to balance the importance of new information with existing old information. For finite MDPs it has been proven that Q-learning can find the optimal solution [89], though for real-world problems it is often unknown whether or not the system can be modelled by a finite MDP.

[3] http://www.gurobi.com/
[4] http://www.ibm.com/software/commerce/optimization/cplex-optimizer/

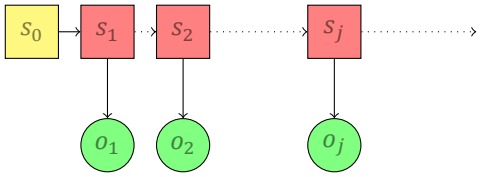


Figure 2.2: A visual representation of a Hidden Markov Model, where the known observations (depicted in green) and an a priori guess for the initial state (s_0 in yellow) help us to determine the a posteriori probability distribution over the states (in red).

2.3. Decision-making problem models in WSNs: Hidden Markov Models

In the past some planning techniques have been applied in the application layer of the wireless sensor networks to reason about the environment under observation. One such set of techniques often found in relation to the state uncertainty described by POMDP models, consists of techniques related to Hidden Markov Models (HMMs) [18, 37]. In the rest of this section we discuss the techniques commonly associated with HMMs, their applications in wireless sensor networks, and what potential drawbacks there are in these techniques.

A Hidden Markov Model is essentially a Markov chain of discrete states extended with observations at every time step. In this model the observations at every time step are known, but the states emitting these observations are not. Based on these observations we then aim to learn something about the states. Figure 2.2 shows a simple Hidden Markov Model, where green nodes indicate nodes whose values are known and red nodes indicate nodes whose values are not known. The representation used here is similar to that of Dynamic Bayesian Networks (DBNs) which is intentional as HMMs can easily be expressed as DBNs [48, 59].

Murphy et al. [59] list a number of variants of this HMM model, describing how they can expressed as DBNs. For instance HMMs in which the observation at time $t + 1$ directly depends on the observation at time t known as correlation or switching HMMs [32, 36], as well as Input-Output HMMs which take a known input that affects the unknown state that emits the known output (observation) [11]. In the context of wireless sensor networks, the HMM configurations Murphy et al. refer to as “coupled HMMs” are worth mentioning. These HMMs combine two or more HMMs to model a system (or state) that influences both of these HMMs independently [16, 41, 53]. In an application with many sensors monitoring the same system indirectly (e.g. many sensors measuring the water level of a river) all of the individual states can tell us something about the water level and whether or not action is required (e.g. to open the water locks).

Often the HMM techniques that are applied are applied not on the sensor node themselves, but rather on an external computer (or the base station). Sensor nodes simply forward all of their observations and the base station computes the probability distribution over the accompanying states [20, 72]. In other cases there is some preprocessing of the data on the sensor nodes themselves (e.g. [37]), but ultimately these systems rely on either total or at least regular communication to form their beliefs on the state.

The reason HMM techniques are often not applied directly on the sensor nodes is that the memory required for the computations can be an issue. In order to perform the techniques from HMMs on the wireless sensor node themselves is unfeasible. It requires the transition function and observation function, which, for a discretised state space, feature a quadratic space-complexity in terms of the number of observations. Depending on the discretisation used, this can lead to an unsatisfiable memory requirement.

An alternative approach featuring a continuous state space requires the usage of an extended or unscented Kalman Filter [19]. Such a filter keeps track of the probability density function for the states based on the observations. Julier et al. [44] suggest however that the extended version is often “hard to implement” and their description of the unscented version also seems too taxing for a wireless sensor node. For distributed systems, such as our sensor network, a distributed Kalman Filter exists. However the initial version requires $O(n^2)$ communication, with n being the number of sensors in the network [62]. An improvement manages to reduce this to only ”local” communication (with neighbours) [61, 63], but this method is then still non-trivial to operate.

As far as we can tell previous research has focused so far on using Hidden Markov Models or DBNs in combination with wireless sensor nodes only when all observations are reported. In this work however, we focus on strategies that rely on communicating only a subset of the observations, such that the base station can still formulate an accurate environmental state estimation.

3

Designing the base station logic: reasoning with limited information

In this chapter we answer the first research question (Sub research question 1) by designing logic for the base station logic that produces an environmental state estimation. This logic uses only the reported measurements and the reported state of the controller executed by the sensor mote in forming this estimation. Before we can formally introduce this logic however, we first introduce notation to describe our environmental monitoring use case in Section 3.1. Then we introduce a small toy example in Section 3.2 that allows us to demonstrate how the techniques we describe in this chapter work.

Inspired by Hidden Markov Models, we start with logic that uses only a transition model of the environmental state to estimate the current environmental state in Section 3.3. This logic applies Bayesian probability methods to calculate a posterior probability distribution given a prior (the transition model) and evidence (a new observation). Since the base station does not have information about uncommunicated observations, we enhance the reported observation with the state of a finite state controller executed by the sensor mote in Section 3.4. This controller can, if properly designed, provide insight into the uncommunicated measurements through reporting its current state.

Next we improve the sensor mote by allowing it to reason about if something is or is not worth reporting. Using a prediction model of some kind, we can give the FSC control over what actions the sensor mote performs. In Section 3.5 we explore the resulting logic. Finally we discuss the feasibility of this logic as well as alternative manners of evaluating it in Section 3.6.

3.1. Problem description

Based on the description given by Chu et al. [22], we formalise our problem as follows.

Problem statement. *We are given a WSN tasked with monitoring the environmental state s through indirect (potentially incorrect) measurements o. For a deployment period during which h measurements are recorded, find the smallest set of measurements $M \subseteq \{o^0, o^1, \dots, o^{h-1}\}$ that should be reported to the base station such that it has the most accurate state estimation z^t at all times $0 \leq t < h$.*

A solution X to this problem is a function that prescribes what measurements should be reported by the sensor mote and what state estimations should be formulated made by the base station as a result. To this end we have the sensor mote select an action a after every measurement, which is either *wait* or *report*.

A solution X thus takes the measurements $\{o^0, \dots, o^{h-1}\}$ and produces the set of actions $\{a^0, \dots, a^{h-1}\}$ and environmental state estimations $\{z^0, \dots, z^{h-1}\}$. In choosing action a^t for a time t, the solution X can only consider observations $o^{t'}$ with $t' \leq t$ as well as the time t itself. Similarly for selecting z^t at a time t, the solution X can only consider the actions $a^{t'}$ for $t' \leq t$, the subset of reported observations $\{o^{t'} | a^{t'} = \text{report}, 0 \leq t' \leq t\}$, and the time t itself.

We elaborate on two aspects introduced in this formalisation.

First we consider the measure of success of a potential solution X. The problem statements refers to

two goals that should be optimised for, a “smallest set of measurements” and a “most accurate state estimation”. These two goals represent the trade-off described in the introduction, the trade-off between battery life and estimation accuracy. To quantify these goals, we introduce two metrics to express the performance of a solution in terms of these goals.

With regards to the report rate, m_{sent} considers the number of messages sent. For m_{sent} we take the fraction of actions that are report actions. The resulting score is between zero and one, with a lower score indicating better performance:

$$m_{\text{sent}}(X) = \frac{|\{a^t = \text{report} | 0 \leq t < h\}|}{h}$$

The accuracy of the environmental state estimations, requires us to compare the estimation z^t made by the base station at time t with the real state s. Depending on the type of environment we monitor, such a comparison could be expressed on a numerical scale (for instance when monitoring temperature, a deviation in degrees Celsius can be used). For the m_{acc} metric we do not assume the environmental state to be comparable in such manner however. Our toy example introduced in the next section does not feature such a numeric state for instance. Instead we count the number of times the base station is expected to predict the correct environmental state. Any relaxations by allowing small deviations can be encoded in this metric by changing the comparison:

$$m_{\text{acc}}(X, \{s^0, \dots, s^{h-1}\}) = \frac{E\left[|\{s^t = z^t | 0 \leq t < h\}|\right]}{h}$$

In multi-objective problems such as this, we are likely to find a set of Pareto-optimal solutions. These solutions all outscore each other in one, but not both of these metrics. Depending on the use case different weights can be attributed to the different metrics and a solution can be selected based on those weights.

Second we consider the sets of environmental states and their associated observations. Similar to other work [18, 41, 46, 72], we model this environment as a Markov chain to allow for analysis of the system, which is very difficult without Markov assumptions [45]. For this model, we introduce a transition function T and an observation function O.

The transition function T describes how the environment is likely to change over time. In the discretised setting, we define S as the set of all possible environmental states. The stochastic transition function then assigns a probability to every state transition. Meaning we can denote T as:

$$T : S \times S \to [0, 1]$$

Additionally there is the observation function O that describe how likely we are to make an observation $o \in \Omega$ given that we are in state $s \in S$. This function can be denoted as:

$$O : \Omega \times S \to [0, 1]$$

3.2. Toy example: a simplified duck monitoring system

Throughout this chapter we introduce several different policies that the base station can execute based on the subset of measurements that the sensor mote reports. To illustrate the way each of these policies work, we apply each of them to a small toy example. This toy example is inspired by the bird monitoring scenario we refer to in the introduction of this thesis.

Consider a simplified version of the bird monitoring scenario in which we have only two possible states $s \in S$ describing the state of the nest: d and e for “duck” and “empty”. Additionally we have two possible observations $o \in \Omega$ of the temperature: h and l for “high” and “low” temperature respectively. Finally we have two actions $a \in A$ that we can perform: r and w for “report” and “wait”. In short:

$$S = \{d, e\}, \Omega = \{h, l\}, A = \{r, w\}$$

Transitions	Observations
$T(d\|d) = 0.9$	$O(l\|d) = 0.09$
$T(e\|d) = 0.1$	$O(h\|d) = 0.91$
$T(d\|e) = 0.2$	$O(l\|e) = 0.8$
$T(e\|e) = 0.8$	$O(h\|e) = 0.2$

Table 3.1: The transition probabilities $T(s'|s)$ and the observation probabilities $O(o|s)$ for the toy example.

From previous studies we know that temperature can be a good indicator of bird presence in our conditions (see for example [57]). For this example, assume that previous studies provide us with a trace of measurements, indicating both duck presence and temperature for consecutive measurements.

$$[(d,h),(d,h),(d,h),(e,h),(e,l),(e,l),(e,l),(e,l),(d,l),(d,h),(d,h),(d,h),(d,h),(d,h),(d,h)]$$

Given no other information, we can infer from these measurements that when the duck is present it is very likely to remain in the nest and we are likely to measure a high temperature. When the duck is absent it is likely to remain absent and we are likely to measure a low temperature. Based on this "training data", we define these probabilities as a state transition function $T(s'|s)$ and an observation function $O(o|s)$ as described in Table 3.1.

Unfortunately we do not know what state the nest is in when we start measuring. It seems likely that the nest was empty, as we would have seen the bird when placing the sensor, but perhaps it has returned since we have started measuring. This uncertainty over the initial state (at time $t = 0$) is described with the following "belief" b_0:

$$b_0(d) = 0.1, b_0(e) = 0.9$$

The policies in the remainder of this chapter all aim to answer a generalised form of the following question. If the sensor mote reports an observation of l at time $t = 2$, does this mean the duck is present at $t = 2$? And what about the states during which no measurements were reported, was the duck present at $t = 0$ and $t = 1$?

3.3. Base station knowledge: a transition model.

Consider the knowledge that we have in our toy example, that is we have an (uncertain) initial state of the environment and (stochastic) functions describing how the state changes over time and what we are likely to observe given a certain state. For a policy based on this information, we use the observation sent along by the sensor node at some point in time to more accurately determine what the state must have been during the periods when no communication happened. In essence this is a HMM with only one known observation, namely the most recent. By applying Bayesian probability methods we use the new observation ("evidence") to update the initial belief over the state to a so-called posterior (or "a posteriori") probability. In this section we describe the policy that the sensor node and base station execute, and we apply the policy to the toy example.

3.3.1. A high level description of the policy

In order for the base station to have "evidence" to reason about, we first consider the operation of the sensor mote. Although we can use any policy there that "occasionally" sends data to the base station, for the purpose of explanation we take a policy that sends its last observation regularly, at every X'th time step. This simple policy for the sensor mote is described in Algorithm 1.

At the receiving end, in the base station, we then use the information on this observation at the t'th time step to get a more accurate belief of what the state has been in the period from $t - X$ to t. In order to do so we use the transition and observation function from the model, and the new knowledge of the observation reported at time step t. To express the probabilities of interest, we define the following variables. Let $s_t \in S$ and $o_t \in \Omega$ be the state and observation at time t respectively. The probabilities we are after are now defined as:

$$P(s_j|o_t) \qquad \forall t - X \leq j \leq t$$

Algorithm 1 An agnostic policy for the sensor node that simply sends the last observation every X'th time step.

```
1: function onMeasurement(measurement)
2:     if time mod X == 0 then
3:         Report(measurement)
4:     else
5:         Wait()
6:     end if
7: end function
```

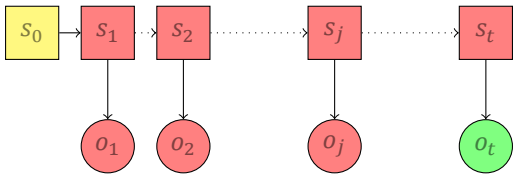


Figure 3.1: The dependencies of the variables s_t and o_t. Square variables are those we are interested in, whereas circular variables are potentially useful for computation but ultimately irrelevant. Green variables have a known realisation, yellow variables have a known prior distribution, whereas red variables only have a prior distribution that can be computed from their dependencies.

To clarify how these random variables depend on one another, we consider the transition and observation function. Figure 3.1 visualises the dependencies. Every directed edge from A to B indicates that B depends on A. As can be seen the states form a simple Markov chain, which generates observations at every time step. Thus the situation is very similar to the HMMs as introduced in Section 2.3. The main difference is that for our system we only know o_t instead of al all observations o_1 through o_t. What we now want to do is use this "evidence" introduced by o_t to learn more about the states s_0 through s_t.

In Appendix A.1 we give a mathematical derivation that allows us to compute $P(s_j|o_t)$ for all $t-X \leq j \leq t$ given an initial belief b_0 and an observation o_t. We describe how this can be implemented in Algorithm 8. We explicitly note which parts of the code can be parallelised, which we return to in the discussion section at the end of this chapter.

3.3.2. Applying the policy to our toy example

To further clarify the mechanics of this policy, let us turn to our toy example. Remember that at time $t = 0$ and $t = 1$ the sensor node had not sent, but at $t = 2$ it had notified the base station that its last observation was l. To visualise the procedure that is performed to compute s_2, consider Figure 3.2. This tree shows all possible paths from an initial state, with edges showing the probability of taking that path. The "impossible paths", that is those ending with an observation of h, are left out of the image.

To find the probability of $P(s_2 = d|o_2 = l)$ all we have to do is consider all paths that have d as the final state, the paths that are highlighted in orange in the figure. For each path we can compute the probability by simply multiplying the probabilities along the path and dividing this by the sum of all paths, i.e. normalise it. As an example consider the bold branch from the root through d, e, d, and l.

$$\begin{aligned} P(\text{bold branch}) &= \frac{b_0(d) \cdot T(e|d) \cdot T(d|e) \cdot O(l|d)}{\sum \text{all paths}} \\ &= \frac{0.1 \cdot 0.1 \cdot 0.2 \cdot 0.09}{\sum \text{all paths}} \\ &= \frac{1.8 \cdot 10^{-5}}{\sum \text{all paths}} \end{aligned}$$

For s_1 and s_0 we can apply the same logic based on this visualisation. We simply sum the probabilities of all paths through the required state at the required time. For instance for $P(s_1 = e|o_2 = l)$ we take all paths for which $s_1 = e$ holds.

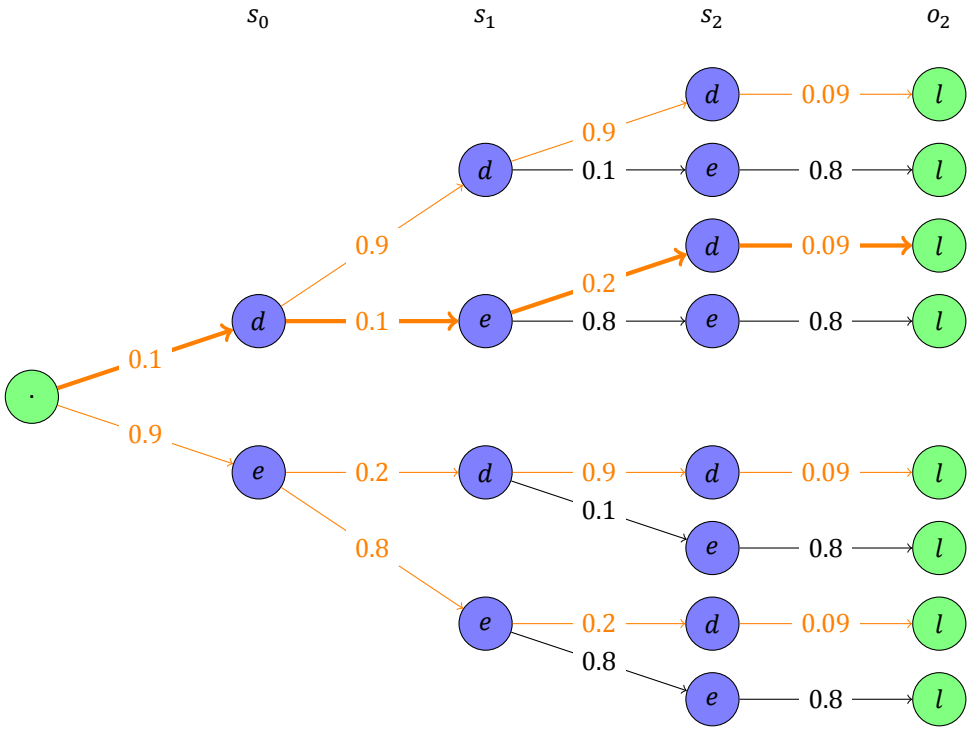


Figure 3.2: All possible ways in which the state can change over the three time steps, with their probabilities, including a final layer with the reported observation. Green nodes denote variables we know for certain, whereas blue nodes denote variables that are still uncertain.

For the toy example this results in the probabilities shown in Table 3.4 on page 25. As can be observed this changes the probabilities drastically for the second and especially the third time step. From not being sure which state we are in after two time steps (as we approach a 50/50 probability), we now know that state e is far more likely than state d. To test these probabilities, we have implemented a simple simulation that randomly transitions according to the model of the toy problem and also generates observations according to this model. After running this simulation selecting only the simulations results in which we observe e at $t = 2$, we get to the same distribution over the states at each time step as was computed here.

3.3.3. Discussion

Whereas this approach allows us to use the evidence introduced by the final observation to refine the probabilities (the effect of which is clearly observed for our toy example), this approach also features a very big downside. All unreported observations can only be reasoned about. This means that any low-probability fluctuations in observations are considered improbable by the base station, whether they occurred or not. Even with a fairly accurate model, low-probability events are not likely to be picked up unless they happen to be the X'th observation. In other words, whereas the addition of the transition model gives us some ground for analysis in the base station, we hypothesise that for increasing X the accuracy very quickly dwindles. We formulate this hypothesis as follows:

Hypothesis 1 (Transition-based policies require frequent communication)**.** *An increasing X quickly decreases the accuracy of a policy that sends observations every X'th time step, irregardless of the contents of the observation.*

To combat this pitfall of the policy, we need to provide more insight into what the observations could have been during the period of radio silence. In the next section we introduce a small controller that the sensor mote executes based on the observations. By examining the resulting state of this controller we can infer information about the uncommunicated observations.

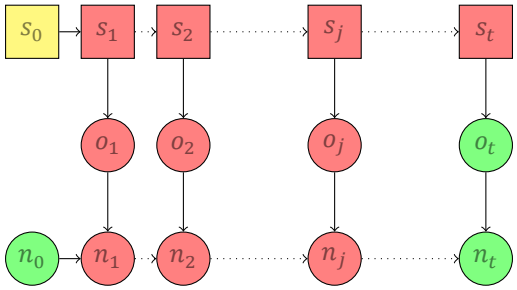


Figure 3.3: The dependencies of the variables s_t, o_t, and n_t. Square variables are those we are interested in, whereas circular variables are of little interest to us. Green variables have a known realisation, yellow variables have a simple known a priori distribution, whereas red variables only have an a priori distribution that can be computed from their dependencies.

3.4. Smarter sensor motes: adding FSC behaviour to the sensor mote

By taking only the transition data into account, we are wasting computational effort that could be done on the wireless sensor node. As the cost of computation is insignificant compared to communication, it can be worthwhile for the sensor node to do some computation if that reduces the need for (frequent) communication. The idea is that the result of the computations provides information about the uncommunicated observations. Thus this provides us more "evidence " for the base station to base its posterior probability on. The computations done by the sensor node can only be based on a limited amount of information, namely the performed measurements and potentially some data pre-loaded into the sensor node. A possible solution would be keeping track of a belief using the same belief updates commonly found in belief MDPs or POMDPs. This requires the sensor mote to have the full transition and observation models in memory however, but with a state space that can potentially grow arbitrarily large, there is simply not enough memory to keep track of the entire state space. So we turn to an alternative solution that we can scale more naturally.

The option that we explore in this thesis is the use of a finite state controller (FSC) in the sensor mote. This controller has transitions based on the observations made by the sensor mote. The sensor mote still reports at every Xth time step, but instead of only reporting the last observation, it also sends the current node[1] in the FSC.

These nodes in the FSC can serve several purposes that can provide the base station with more information. For instance the FSC can count the number of observations that are higher than some threshold and have the current node reflect the current count. When the base station is informed that the current count is 4, it knows that 4 out of the X observations were above this threshold, thus indirectly providing more information about the uncommunicated observations.

In the rest of this section we describe the properties of the FSC and how these can be used, how the probabilities over the states are now computed, and finally how this can be applied to our toy example.

3.4.1. Components of the FSC

As stated above, only the observations can influence the behaviour of the FSC, as that is all the information the wireless sensor mote has. Thus we define the FSC as the following 3-tuple: $F =< N, T_{\text{fsc}}, m >$, wherein N is the collection of nodes in the controller. The function T_{fsc} has the subscript "fsc" to differentiate it from the transition function T for the state of the environment. m denotes the initial node of the FSC.

The transition function T_{fsc} describes how we transition from one node in the finite state controller to another. This transition can be stochastic, but should only be based on values that are available in the wireless sensor mote. For instance the observation made by the sensor mote and the current node in the FSC, but not the state of the environment (duck or empty in the toy example). Thus we need a function $T_{\text{fsc}}(n'|n, o)$ to describe what the probability is of getting to node n' from node n, after observing o. More formally:

$$T_{\text{fsc}} : N \times N \times \Omega \rightarrow [0, 1]$$

[1] To avoid confusion with the states of the environment, we will call states in the finite state controller "nodes" instead and also refer to them as $n \in N$, leaving states and $s \in S$ for the states of the environment.

Transitions to x	Transitions to y
$T_{\text{fsc}}(x\|l,x) = 0.80$	$T_{\text{fsc}}(y\|l,x) = 0.20$
$T_{\text{fsc}}(x\|l,y) = 0.01$	$T_{\text{fsc}}(y\|l,y) = 0.99$
$T_{\text{fsc}}(x\|h,x) = 0.30$	$T_{\text{fsc}}(y\|h,x) = 0.70$
$T_{\text{fsc}}(x\|h,y) = 0.99$	$T_{\text{fsc}}(y\|h,y) = 0.01$

Table 3.2: The transition function for the FSC used in the toy example.

Now that the base station receives not only the last observation, but also the current node in the FSC, we have more evidence for the posterior probability distribution of the states. To this end we introduce a third variable n_t that denotes the node in the FSC we are in at time t. This additional dependency changes the way our dependency figure looks. Figure 3.3 shows this updated dependency figure. Notice how this is subtly different from a switching HMM [32] as we have indirect information on all observations at times before t through n_t, whereas switching HMMs assume either all observations or all nodes to be known. Given the extra information provided by n_t, we can indirectly gain more insight into the distributions of states s_0 through s_t.

3.4.2. Using the information provided by the FSC

Let us first consider the policy executed by the wireless sensor node. This policy evaluates the FSC, that is based on the node n, after observing o, it will move on to node n' with probability $T_{\text{fsc}}(n'|n,o)$. Algorithm 2 implements that policy.

Algorithm 2 The FSC behaviour in the sensor node. After every measurement (observation) a new node is selected and depending on the time this node and the observation are communicated.

```
1: function onMeasurement(measurement)
2:     n ← sample a node n' from N with probability T_fsc(n'|n, measurement)
3:     if time mod X == 0 then
4:         Report(measurement, n)
5:     else
6:         Wait()
7:     end if
8: end function
```

On the other side of the network, in the base station, we again have to determine the probability s_j given the latest observation o_t and node n_t in the FSC. This extra information changes the required probability slightly, we can now describe this as:

$$P(s_j|o_t,n_t) \qquad \forall t - X \leq j \leq t$$

In Appendix A.2 we provide a mathematical derivation of this probability in terms of known priors. We describe a parallelised implementation of this derivation in Algorithm 9.

3.4.3. Applying the policy to our toy example

Before we can apply this policy to the toy example, we need to extend it by introducing a FSC. To this end we introduce a small FSC with two states x and y. Additionally we introduce the transition function T_{fsc} as described in Table 3.2. Finally we have to introduce an initial node for the FSC as well as the node that is communicated alongside the observation l at time step 2. For the initial node we choose node x and for the final node that is communicated we choose y. This means that there is only uncertainty over what node the sensor was in at time $t = 1$.

To visualise the procedure that is performed to compute s_2, consider Figure 3.4. This tree shows some possible paths from an initial state/node combination, with edges showing the probability of taking that path. The “impossible paths”, that is those ending with an observation of h or in a node x, are left out of the image. Furthermore we have drawn only 2 possible (state, observation, node)-combinations for each initial state out of the 8 an initial state can transition to. The probability on an edge are now

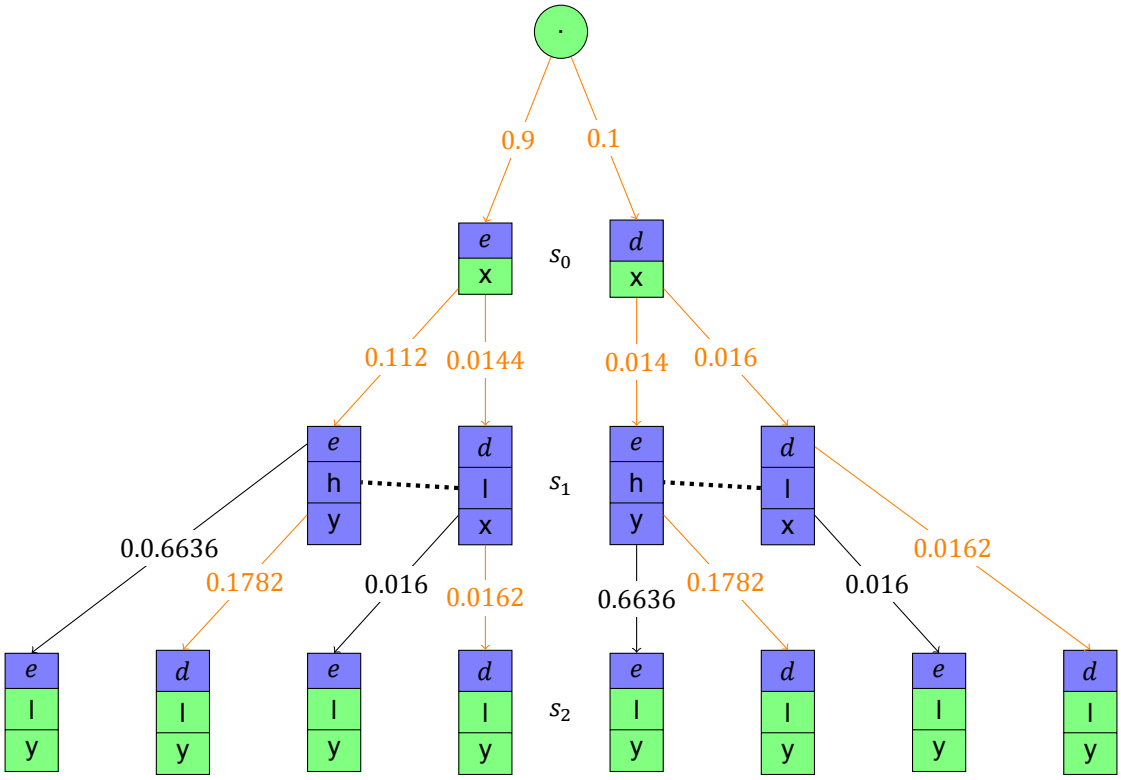


Figure 3.4: Possible ways in which the state and node can change over the three time steps, with their probabilities, including a final layer with the reported observation and node. For sake of the drawing only two out of eight possible transitions between s_0 and s_1 are left out. Green nodes denote variables we know for certain, whereas blue nodes denote variables that are still uncertain.

slightly more complex than before. As an example consider the edge between (e,h,y) and (e,l,y). This probability is now computed as:

$$\begin{aligned}\text{Probability} &= T(e|e) \cdot O(l|e) \cdot T_{\text{fsc}}(y|l,y) \\ &= 0.8 \cdot 0.8 \cdot 0.99 \\ &= 0.6636\end{aligned}$$

To find the probability of $P(s_2 = d|o_2 = l, n_2 = y)$ all we have to do is consider all paths that have d as the final state, the paths that are highlighted in orange in the figure. Again for each path we can compute the probability by simply multiplying the probabilities and normalising it.

By performing these operations we find the probabilities reported in Table 3.4 on page 25 in the column: “Observation + Node”. Again we have confirmed these probabilities through extensive simulation of the prescribed environment and the FSC, by selecting only the paths that end with an observation l in node y.

3.4.4. Discussion

Through the introduction of the controller, we can indirectly inform the base station about the effect of unreported observations. This alleviates the downside identified in the previous section. Unfortunately it is not fully resolved yet, as there is still no method to immediately react to changes in the environment. We are still bound by the “report at every X’th time step”-rule. This means that whereas we expect the FSC-based approach to offer improvements over the approach based only on the transition-function, it is still likely to lose accuracy as X increases. We formulate the following two hypotheses to summarise these expectations:

Hypothesis 2 (FSC-based dominates transition-based)**.** *The FSC-based approach outperforms the transition-based approach for an equal report interval X.*

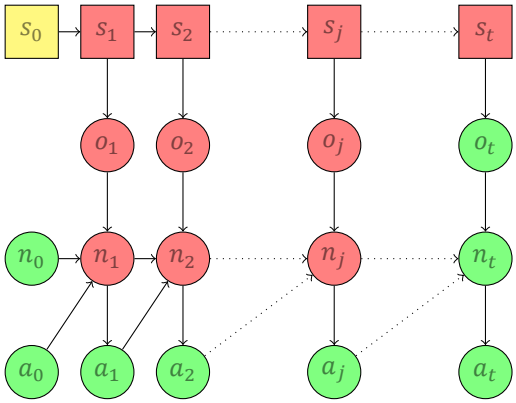


Figure 3.5: The dependencies of the random variables s_t, o_t, n_t, and a_t. Square variables are those we are interested in, whereas circular variables are of little interest to us. Green variables have a known realisation, yellow variables have a simple known a priori distribution, whereas red variables only have an a priori distribution that can be computed from their dependencies.

Hypothesis 3 (FSC-based still requires frequent communication)**.** *An increasing X decreases the accuracy of a controller-based policy that sends observations and nodes every X'th time step.*

As we still expect the accuracy to decrease for increasing X, we turn to an alternative approach. Rather than reporting every X'th time step, we instead report only when we deem an observation to be worth reporting. This allows us to potentially reduce the report rate further without decreasing the accuracy.

3.5. Informative reports: actions determined by the FSC

In the final extension of our model we let the sensor mote choose the action it performs based on the node in the FSC. This has two main advantages: it allows the sensor mote to react immediately to strange observations (rather than waiting till the interval X is over), and it can provide the base station information about *why* the sensor mote has (not) chosen to report a measurement. In this section we describe how the FSC is extended, what this extra information can do for us, and how this changes the outcome for our toy example. Additionally we describe how this policy can be used to model the previous policies from this chapter.

3.5.1. An extension of FSC functionality

To incorporate the notation of a node-dependent action into our FSC, we introduce a new function A_{fsc}. This A_{fsc} describes what action we should take in a certain node. To allow as much freedom in this model as possible, this A_{fsc} is also stochastic, meaning in every node we have a chance of taking either the report or the wait action available to the sensor mote. Thus we need a function $A_{\text{fsc}}(a|n)$, or more formally:

$$A_{\text{fsc}} : A \times N \to [0, 1]$$

Additionally we also change T_{fsc} slightly, by including the chosen action in this transition function. Doing so again opens up several more possibilities in the things we can model using this policy. For instance, this allows a counting FSC as mentioned before to be reset every time a "report" action is chosen. Thus T_{fsc} is now written as: $T_{\text{fsc}}(n'|n, a, o)$ or rather:

$$T_{\text{fsc}} : N \times N \times A \times \Omega \to [0, 1]$$

This introduction of A_{fsc} requires another variable to be introduced: a_t that denotes the chosen action at time t. As a_i depends only on n_i, and n_{i+1} only requires one extra dependency, our dependency figure changes little as is depicted in Figure 3.5. For consistency we again denote the time an observation is received as time t, though it should be noted that t is no longer necessarily a multiple of some regular interval X. Now we have fully deviated from the HMM models as the time at which we received the limited data is now also determined by the state, that is the state indirectly determines when a "report" action is chosen and thus when limited information to reason about it becomes available.

3.5.2. More information in the base station

The operations of the sensor node change slightly based on this extended FSC model, as an action now needs to be chosen and executed based on the FSC. Algorithm 3 demonstrates this.

Algorithm 3 The extended FSC behaviour in the sensor node. After every measurement (observation) a new node is selected and based on this new node an action is selected that determines whether or not we should report.

Require: n is the current node in the FSC.
Require: a_p is the previous action chosen by the FSC.

```
function onMeasurement(measurement)
    n ← sample a node from N with probability T_fsc(n'|n, measurement, a_p)
    a ← sample an action from A with probability A_fsc(a|n)
    if a == report then
        Report(measurement, n)
    else
        Wait()
    end if
    a_p ← a
end function
```

Unlike the addition of nodes to the dependency model, adding actions does not require many changes in the computation procedure. Although T_{fsc} depends on the action, we can also describe this as having two transition functions for the FSC, one that is used when we "report" and one that we use when we "wait". Call these T_{report} and T_{wait} respectively. By considering them as these two distinct function, we only need to slightly modify equations A.12, A.16 and A.18 to get to a correct expression for this new situation in which we also consider actions.

In each of these equations we find a usage of $T_{\text{fsc}}(n_{i+1}|n_i, o_i)$, which now has to be updated as it should also take the action into account. We do so by replacing it:

$$T_{\text{fsc}}(n_{i+1}|n_i, o_i) \rightarrow \begin{cases} T_{\text{report}}(n_{i+1}|n_i, o_i)A_{\text{fsc}}(\text{report}|n_i) & \text{if } a_i = \text{report} \\ T_{\text{wait}}(n_{i+1}|n_i, o_i)A_{\text{fsc}}(\text{wait}|n_i) & \text{if } a_i = \text{wait} \end{cases} \tag{3.1}$$

This modification encapsulate the two extra dependencies introduced by the extra action variable, namely the dependency of n_{i+1} on the action as well as the dependency of the action on the node. Since all actions chosen are known by the base station, this introduces no extra complexity compared to Algorithm 9.

3.5.3. Applying the policy to our toy example

We extend our toy example by also introducing an action function A_{fsc}. This function is shown in Table 3.3. As can be observed, the "wait" action is more likely in node x and the "report" action is more likely in node y. As we have full knowledge of the chosen actions, namely that they were "wait" at time $t = 0$, "wait" at time $t = 1$, and "report" at time $t = 2$, this makes it more likely that we were at node x at time $t = 1$. Additionally we change the T_{fsc} function slightly, by introducing a dependency on the action. For this toy example we use:

$$T_{\text{wait}}(n'|n, o) = T_{\text{fsc}}(n'|n, o)$$
$$T_{\text{report}}(n'|n, o) = 1 - T_{\text{fsc}}(n'|n, o)$$

For the purposes of this example we do not include the notion of a predictor in our system. By applying equations A.9 with the modifications of equation 3.1 to our toy example, we find the probabilities reported in the final column of Table 3.4. Again simulation confirms these results, this time selecting all simulations in which the actions also match the actions described above.

Wait actions	Report actions
$A_{\text{fsc}}(w\|x) = 0.8$	$A_{\text{fsc}}(r\|x) = 0.2$
$A_{\text{fsc}}(w\|y) = 0.1$	$A_{\text{fsc}}(r\|y) = 0.9$

Table 3.3: The action function for our toy example.

Probability	A Priori	Observation	Observation + Node	Observation + Node + Actions
$P(d)$ at $t = 0$	0.10	0.04	0.03	0.04
$P(e)$ at $t = 0$	0.90	0.96	0.97	0.96
$P(d)$ at $t = 1$	0.27	0.08	0.06	0.07
$P(e)$ at $t = 1$	0.73	0.92	0.94	0.93
$P(d)$ at $t = 2$	0.39	0.07	0.06	0.06
$P(e)$ at $t = 2$	0.61	0.93	0.94	0.94

Table 3.4: The probabilities of being in state d or e at times $t = 0$, $t = 1$, and $t = 2$, depending on what information we take into account.

3.5.4. Modelling the other approach using this policy

As indicated before, this policy is sufficiently generic to simulate the other policies described in this chapter. We briefly describe how we can model each of the other policies:

Report only the observation every X'th time step
Consider a finite state controller with exactly X nodes, such that in all nodes we always chose the "wait" action except in node n_X, in which we always select the "report" action. Now the transition function of the FSC simply has the FSC transition from node n_i to node n_{i+1}, ensuring that after X time steps it ends up in node n_X. This FSC implements exactly the policy that reports every X'th time step described in Section 3.3 as the node that the FSC is currently in provides no extra information about the state of the environment. After all the node transition is independent of the observations and the node will always be n_X at the time of sending.

Report observation and node every X'th time step
Consider a finite state controller with $N \cdot X$ nodes. That is, we can have any FSC, but we replace every node from this FSC by X nodes in our updated FSC. Every node from this new collection of nodes encodes a specific time t. We change our transition function so that for every original transition from a node a time t, we transition to the corresponding node in the set of nodes for time $t + 1$. As for the actions, all nodes that correspond to time X will always chose the "report" action, all other nodes always select the "wait" action. This FSC implements exactly the policy described in Section 3.4, including the notion that the chosen actions do not provide any extra information about the state of the environment, but the nodes do.

3.5.5. Discussion

By combining the shared knowledge of the prediction mechanism with the ability of the FSC to react to any deviations between prediction and measurement, we gain fine-grained control over the report rate versus accuracy trade-off. Furthermore since we can now report based on the node (and by extension the observation), the received reports can focus on the significant deviations from the predictor. Thus we formulate the following hypothesis:

Hypothesis 4 (FSC with varying report rates)**.** *Controllers that base their actions on the node in the FSC outperform controllers with fixed reported intervals in terms of report rate for a similar accuracy.*

Now that we offloaded much of the optimisation process to the finite state controller, a new question arises. What should such a controller look like for optimal performance? In other words how do we find the optimal controller for a use case? In the next chapter we introduce an optimisation framework that can help us find such an optimal controller.

Function	space-complexity	time-complexity
computeStateNodeProbabilities	$O(X \cdot \|S\| \cdot \|N\|)$	$O(X \cdot \|S\|^2 \cdot \|N\|^2 \cdot \|O\|)$
computeObsNodeGivenNodeStateProbabilities	$O(X \cdot \|S\| \cdot \|N\|)$	$O(X \cdot \|S\|^2 \cdot \|N\|^2 \cdot \|O\|)$
onReceivedMessage	$O(1)$	$O(X \cdot \|S\|^2 \cdot \|N\|)$

Table 3.5: The time and space complexity for computing and storing the required values for the base station logic.

3.6. Discussion

Now that we have introduced a model that can react to observations in a meaningful way, such that is the result of these reactions can provide extra information to the base station, we explore some of the benefits, drawbacks, and open issues with this model. As the model described in Section 3.5 can emulate all other policies outlined in this chapter we focus our discussion on that model. We group our items of discussion by two main topics: feasibility and evaluation.

3.6.1. Feasibility

In order for the technique to be feasible for our use case, there are two things we should consider. First the space and time complexity of the algorithm run on the sensor node, and second the space and time complexity of the computations done by the base station. For the sensor node it is important that the required amount of memory is relatively small to ensure a wide variety of sensor nodes can be used, but it is also important that computations are manageable as sensor nodes feature cheap (and slow) hardware that take (highly) frequent measurements. For the base station these constraints are slightly more relaxed, as we can potentially off-load work to a distributed setting here. However an exponential time or space complexity would still not be desirable, especially if a live analysis based on the received observations is required for the base station to act on.

The algorithm in the wireless sensor node features efficient time and memory complexity. Consider Algorithm 3. Assuming that the sampling can be implemented in $O(|N|)$ time, line 2 takes $O(|N|)$ time, but all other lines take constant time (remember that we already know that $|A| = 2$). Thus in terms of time complexity, we achieve $O(|N|)$ complexity.

As for the space complexity, the sensor node requires two matrices in memory: One for T_{fsc} and one for A_{fsc}. Thus the total memory required is $O(|N|^2 \cdot |A| \cdot |\Omega| + |N||A|) = O(|N|^2|\Omega|)$. As $|A|$ is constant (and small) for our use cases, and $|N|$ is a parameter we can tweak ourselves, that leaves just $|\Omega|$ as the main concern.

In Section 2.3 we reject a belief-based update on the sensor node as that would require $O(|S|^2 + |S| \cdot |\Omega|)$ memory. At first glance the policy outlined in this chapter has a similar memory requirement, but there are two big differences. First of all we can freely control $|N|$ as we decide how the FSC is built, thus we can make it sufficiently small to still fit in the wireless sensor node. Secondly for many use cases $|S| = |\Omega|$, as every state has one observation associated with it (see for example the office temperature use case introduced in Chapter 5), meaning the belief-based system would impose quadratic memory requirements as opposed to the linear requirement imposed by the controller-based policy.

The complexity of the computations in the base station are somewhat higher. Using dynamic programming techniques, we can fill up a number of caches corresponding to the different equations, as Algorithm 9 shows. Despite the fact that Figure 3.4 initially seems to show exponential growth of the possible options, the dynamic programming techniques can help ensure that we can still efficiently compute the probabilities. Table 3.5 gives the space and time complexities corresponding to the different functions. In the complexity analysis we use X to denote the time between two measurements and for the time-complexity analysis we assume that any other cached values can be retrieved in $O(1)$, that is we build the caches in such a way that all required cache values are already computed. The total complexity is then simply the sum of the listings in the table. This results in a time-complexity of $O(X \cdot |S|^2 \cdot |N|^2 \cdot |O|)$ and a space-complexity of $O(X \cdot |S| \cdot |N|)$.

Note that this is only for situations in which we compute the probabilities at the moment of a report operation. If we compute the output after a wait action, it is worthwhile to cache more values and the space- and time-complexity are increased by a factor $O(|O||N|)$. Either way the algorithm performed by the base station is also computationally efficient.

3.6.2. Evaluation

Although the report rate is clearly represented by m_{sent}, our solution method results in a probability distribution over the possible environmental states rather than a single state estimation z^t required for the m_{acc} metric. To resolve this we adapt the metric to instead evaluate these probability distributions.

There are many different methods with which we could measure the accuracy of our solution. For instance for some use cases we might prefer a normal distribution with the mean at the real environmental state over a Pareto distribution which is maximal for the real environmental state. This is very dependent on the use case. In this thesis we instead consider only the probability associated with the real environmental state and we ignore the rest of the probability distribution. The metric m_{acc} can then be written as:

$$m_{\text{acc}}(X, \{s^0, \dots, s^{h-1}\}) = \frac{E\left[\sum_{t=0}^{h-1} P(s^t = z^t)\right]}{h}$$

For the purposes of evaluation in this thesis this metric is sufficient as it allows us to compare different solutions. When designing controllers for real world deployments it is important to reconsider this metric and adapt it to represent the accuracy measure needed for that specific deployment.

4

Designing the sensor mote intelligence

For the evaluation of the hypotheses formulated in the previous chapter, we require a number of finite state controllers. In Section 4.1 we describe how we built the finite state controllers required for these hypotheses. Next we consider a systematic approach to finding the optimal controller for a use case. To this end we formulate the optimisation problem as a Partially Observable Markov Decision Process (POMDP), which we then solve to a finite state controller. In Section 4.2 we offer some background information on POMDPs, existing methods to solve them, as well as existing implementations of solvers. We go on to describe how we can model our optimisation problem as a POMDP in Section 4.3 and formulate hypotheses about the performance of the POMDP-based controllers in Section 4.4.

4.1. Handmade controllers

In order to evaluate the hypotheses introduced in Chapter 3 we design several controllers by hand that embody the aspects we want to test for in the hypotheses. In this section we describe how these controllers are designed. We focus on two types of controllers. Counting controllers that keep track of how many measurements of a certain kind have occurred, and deviation-based controllers that react strongly to the deviation between the last measurement and the last prediction.

4.1.1. A counting controller

To test Hypothesis 2 about the benefits of using a controller, we use a counting controller. We refer to this type of controller as CC for Counting Controller. The controller simply counts the number of observations that lower or higher than the "average" observation, where the average is simply taken as the average of the minimum and maximum temperatures. Say that we are measuring temperatures between zero and twenty degrees Celsius for instance. We lower the counter by one if we measure a temperature below ten degrees, keep it constant if we measure a temperature of ten degrees and increase the counter by one for higher temperature measurements. This controller reports the current counter every C'th time step. Let n_y denote the node that represents a count of y, then we can describe the behaviour as:

$$T_{\text{fsc}}(n_y, o, \text{report}) = n_0$$

$$T_{\text{fsc}}(n_y, o, \text{wait}) = \begin{cases} n_{y+1} & \text{if } o > \text{average} \\ n_y & \text{if } o = \text{average} \\ n_{y-1} & \text{else} \end{cases}$$

$$A_{\text{fsc}}(n_y, \text{report}) = \begin{cases} 1 & \text{if time} \mod C == 0 \\ 0 & \text{else} \end{cases}$$

$$A_{\text{fsc}}(n_y, \text{sleep}) = 1 - A_{\text{fsc}}(n_y, \text{report})$$

There are many improvements that can be made to this controller. For instance we could count both the number of observations above and below this average independently, we could explicitly count

every observation independently, etc. Unfortunately all of these methods would drastically increase the size of the controller. To implement this controller successfully, we already require $C \cdot (2C + 1)$ nodes in the controller (the first dimension to capture time, and the second to capture the count). For a relatively small $C = 5$, this already requires 55 nodes in the controller. With proper optimisations this type of controller can still be fit into a sensor mote, but due to the way we construct controllers for our generic framework, evaluation takes a very long time for $C >= 10$ (one hour for $C = 10$, on a multi-core high-end desktop computer).

Alternatively we could use the median temperature as the temperature that we compare the observations to. Depending on the use case this could divide the data in a more insightful manner. For the purposes of evaluating Hypothesis 2, these "optimisations" or alternatives are not required. We only aim to show that adding a controller can improve the accuracy of the base station environmental state estimates. If we can obtain an improvement even with such a simple, generic, controller, then it follows that better comparisons or more counters can potentially increase performance even more. Instead of exploring these alternative implementations of the same idea in-depth, we instead change the notion of a fixed comparison point for the next type of controller.

The predictor-based counting controller (PCC) uses the prediction mechanism that the sensor mote and base station share as the reference point for the comparison. Since the predictor should often match the measurement, counting deviations from the predictor is potentially much more insightful than comparing to a fixed point. This changes the transition function of the controller to be:

$$T_{\text{fsc}}(n_y, o, \text{report}) = n_0$$
$$T_{\text{fsc}}(n_y, o, \text{wait}) = \begin{cases} n_{y+1} & \text{if } o > \text{prediction} \\ n_y & \text{if } o = \text{prediction} \\ n_{y-1} & \text{else} \end{cases}$$

To allow us to test Hypothesis 4, we create a third third, and final, form of the counting controller, which chooses the action based on the current count. This stochastic prediction-based counting counter (SPCC) uses its current count to determine if it should or should not perform a report action. Given a higher count, the probability of reporting it becomes higher. This provides two main advantages. First, fewer reports are necessary, as counts of zero (i.e. data that matches the predictor) are reported less frequently. Second, given fluctuations in the measurements surrounding a predictor, we are increasingly more likely to report them if several fall on the same side of the predictor.

This controller is designed as follows, with n_y again denoting a count of y in one direction.

$$T_{\text{fsc}}(n_y, o, \text{report}) = n_0$$
$$T_{\text{fsc}}(n_y, o, \text{wait}) = \begin{cases} n_{y+1} & \text{if } o > \text{prediction} \\ n_y & \text{if } o = \text{prediction} \\ n_{y-1} & \text{else} \end{cases}$$
$$A_{\text{fsc}}(n_y, \text{report}) = \begin{cases} (1/2M)^2 & \text{if } y = 0 \\ (y/M)^2 & \text{else} \end{cases}$$
$$A_{\text{fsc}}(n_y, \text{sleep}) = 1 - A_{\text{fsc}}(n_y, \text{report})$$

There are of course many ways in which the probability function for A_{fsc} could be encoded. We have chosen a quadratic function here, but depending on the use case other functions may work better. For the purposes of evaluating Hypothesis 4, this quadratic function suffices.

4.1.2. A deviation-based controller

As a variation on the stochastic prediction-based counting controller, consider instead a controller with a maximum count of one. Essentially this controller would report any deviation from the predictor immediately. We refer to this type of controller as the deviation-based controller (DC). As the last observation already holds the required information about the deviation, we can encode the deterministic

version of such a controller in only two nodes. This deviation-based controller can be denoted as:

$$T_{\text{fsc}}(n, o, a) = \begin{cases} n_{\text{report}} & \text{if } |o - \text{predictor}| > X \\ n_{\text{wait}} & \text{else} \end{cases}$$
$$A_{\text{fsc}}(n_{\text{report}}, \text{report}) = 1$$
$$A_{\text{fsc}}(n_{\text{wait}}, \text{report}) = 0$$
$$A_{\text{fsc}}(n_y, \text{wait}) = 1 - A_{\text{fsc}}(n_y, \text{report})$$

Notice how this controller does not encode the information about unreported observations. It does not necessarily have to as the base station can already infer from the lack of report operations that the observations where less than X removed from the predictor.

We can also make a stochastic variation of this controller (SDC) in a similar way to the counting controllers. This time let n_y denote the node that represents a deviation of y from the predictor.

$$T_{\text{fsc}}(n_y, o, a) = n_z \quad \text{with } z = \min(|o - \text{predictor}|, D)$$
$$A_{\text{fsc}}(n_y, \text{report}) = (y/D)^2$$
$$A_{\text{fsc}}(n_y, \text{sleep}) = 1 - A_{\text{fsc}}(n_y, \text{report})$$

The comparison between the DC and SDC variants provide us with another set of controllers for which Hypothesis 4 can be evaluated.

4.2. Theoretical background: the POMDP model

To find the optimal controller we model our problem using a well-known optimisation framework, namely the Partially Observable Markov Decision Process (POMDP). A POMDP is defined as a tuple of six elements (based on [48]), that together describe a situation in which an agent reasons about their environment and actions to work towards a goal. We use the following nomenclature, which matches the previously introduced names in relation to the FSC model.

S the possible states for the agent to be in.

A the possible actions for the agent. This describes what an agent can do at every time step.

$T : S \times S \times A \to [0, 1]$ the transition function for the system that specifies the probability of transitioning from state s to s' when choosing action a.

$R : S \times A \to \mathbb{R}$ the reward function for the system that specifies the reward for being in state s and choosing an action a.

Ω the possible observations for an agent after performing an action.

$O : \Omega \times S \times A \to [0, 1]$ the observation function for the system that specifies the probability of observing o from state s when choosing action a.

h the horizon or time window for which we need a policy that the agents execute.

With the nomenclature established, we can define our problem as the tuple of seven elements for the POMDP model.

$$P = < S, A, T, R, \Omega, O, h >$$

We plan for a horizon of h time steps during which the rewards are obtained. The total system-wide reward is defined as: $\sum_{t=0}^{h-1} E[R(s_t, a_t)]$, where s_t and a_t represent the state and action that the agent is in/chooses to take at time t.

If $h = \infty$ then we call the problem an “infinite horizon” POMDP.In the case of an infinite horizon POMDP we use a decaying reward function that gives future rewards an increasingly smaller impact on the total rewards. To this end a discount factor $\gamma < 1$ is often used, which exponentially decreases future rewards.

4.2.1. Algorithms for solving POMDP problems

Although a variety of methods to solve POMDP problems exist, we focus on two in this section. The first method obtains a policy in the form of a set of α-vectors that describe what action should be taken for a certain belief state b. The second method takes a different route and tries instead to formulate the policy as a finite state controller that encapsulates the actions that should be taken when in a given state.

Point-based methods

There are many different methods that sample beliefs from the belief space reachable from the initial belief under certain actions and observations, for instance [79, 80]. These methods then compute what actions should be performed for these samples and during execution they map beliefs to these sampled points to determine the optimal action. As they only sample a subset of the (often infinitely large) belief space, this method results in an approximation of the optimal policy. Another method by the name of Sarsop [52] tries to reduce the belief space from which it samples by only considering the belief space reachable from the initial belief under an optimal policy. As the optimal policy is unknown it iteratively restricts this reachable space using an upper and lower bound on the optimal policy. The resulting reachable space is much smaller and thus the computation is more tractable whilst returning similar or better policies as a result.

Solving to a Finite State Controller

Another way to represent the policy produced by a POMDP is not by formulating it as a set of α-vectors, but rather as a Finite State Controller (FSC) [5, 33, 71]. Such a controller is a compact stochastic machine that prescribes what actions should be taken to maximise the expected reward. The main advantage of this representation of the policy is that we can scale the size solution very naturally. We can limit the number of nodes in the FSC to the desired size [5]. Such an FSC can be described by the tuple $< N, V, A_{\text{FSC}}, T_{\text{FSC}} >$, where:

N**:** is the set of nodes in the FSC.

$V : N \times S \to \mathbb{R}$ is the valuation function that represents the expected reward when executing the policy. This links back to the POMDP model that the FSC is based on. For the execution of the FSC however, this function is not required, it is only during the creation that we need it.

$A_{\textbf{fsc}} : A \times N \to [0, 1]$ is the stochastic function describing what action $a \in A$ should be taken when we are in node $n \in N$.

$T_{\textbf{fsc}} : N \times N \times A \times \Omega \to [0, 1]$ is the stochastic transition function describing what the probability is of transitioning to node $n' \in N$, after starting in node $n \in N$, performing action $a \in A$ and observing observation $o \in \Omega$.

One way to get to such an FSC is to use Bounded Policy Iteration (BPI) [5, 71]. This method focuses on iteratively solving an LP and a set of linear equations of the form $Ax = b$ to converge to an approximation of the A_{fsc} and T_{fsc} function listed above after updating the valuation function V based on the current state of the others. Given a random initialisation of these functions, the pseudocode for BPI is shown in Algorithm 4

4.2.2. Existing solvers for POMDP problems

Using such a well-known and studied framework also allows us to use existing solvers to obtain a policy that can be run in the individual nodes. Several solvers have been created over the last years, in this section we briefly describe several of them.

MADP-toolbox

Developed and maintained by Frans Oliehoek, Matthijs Spaan, and others, the MADP-toolbox offers a variety of algorithms to solve a variety of Multi-Agent Decision Problems (MADP) [78]. As an open-source project hosted on GitHub[1], extensions and modifications to this toolbox can be made if required. Among the planning problems that this solver can handle is also the (Dec-)POMDP framework, for

[1] `https://github.com/MADPToolbox/MADP`

Algorithm 4 The BPI algorithm that solves a partially observable Markov decision process to a finite state controller using iterations of a set of linear equations and an LP.

Require: n is the maximum number of nodes we are allowed to have in the FSC.
Require: FSC is an FSC with $|N| \leq n$.

function BPI(n, FSC)
  while not satisfied **do**
    Create system of lin. eq. using A_{fsc} and T_{fsc} ▷*The Bellman equation from 2.1 of [5]*
    Solve the system and update V.
    for every node $n \in N$ **do**
      Formulate LP, using V ▷*Table 1 of [5]*
      Solve LP to update A_{fsc} and T_{fsc}
    end for
    if Not enough improvement was made **then** ▷*Corollary 1 of [71]*
      success ← false
      if $|N| < n$ **then**
        success ← addNode() ▷*See below, based on section 5.2 of [71]*
      end if
      if not success **then**
        return FSC
      end if
    end if
  end while
  return FSC
end function

function addNode()
  get tangent beliefs from dual LPs.
  $B \leftarrow$ one-step look-ahead on all tangent beliefs.
  for all beliefs $b \in B$ **do**
    create node n' with value $\max_{\forall n \in N}(V(n, b))$.
    formulate LP using V and weighted using b ▷*Weighted version of Table 1 of [5]*
    solve LP and store improvement with n'.
  end for
  add node n' with the largest positive improvement to the FSC
end function

which it can use a number of exact algorithms. From a brute-force search to more advanced techniques such as the GMAA* algorithm [64], they cover a variety of exact solution methods. Unfortunately for instances with large horizons, these exact methods are often too slow or too tasking on the system's memory to be be used in this thesis.

APPL Software
In contrast to the MADP-toolbox, the Approximate POMDP Planning Software (APPL Software) focuses on finding an approximate solution to POMDP problems. Based on the SARSOP algorithm [52] with improvements based on the work on Mixed Observability MDPs (MOMDPs) by Ong et al. [65], its offline planner can generate policies for problems modelled in the POMDP framework. With an open-source solver and evaluator[2], custom evaluation of the generated policies is also possible. As instances with large state spaces and infinite horizon can be approximated by this solver it is a more promising candidate for this work.

Both solvers output α-vectors, whereas we require a finite state controller for our base station logic. There are methods to solve a set of α-vectors to a finite state controller [34, 50], but in this thesis we focus on solving directly to a controller using the BPI-algorithm. Possible improvements on this process by using other algorithms are left to future work.

4.3. Formulating the optimisation problem as a POMDP

The model we have used thus far of the environment under study, which includes the transition function T and the observation function O, can easily be extended to the POMDP model. The only property that is missing is the reward function R that maps being in a certain state and performing an action to a reward. The choice of this reward function is going to determine the behaviour of the FSC and thus the behaviour of our wireless sensor mote. In this section we give an overview of how we can translate our existing functions to functions compatible with a POMDP model, saving specific implementation details for the next section.

S the set of states is a 3D vector value, expressing three different sub states of the sensor node. The first is the current time of the phenomenon being studied. The second is the current prediction from the predictor we have of the phenomenon being studied. Finally the third is the state of the phenomenon being studied. This is the only value which is uncertain, as it can only be observed through imperfect observations. In other words:

$$S = \{(s_t, s_p, s_f) \mid s_t \in S_{\text{time}}; s_p \in S_{\text{prediction}}; s_f \in S_{\text{phenomenon}}\}$$

In many cases there will be one state in $S_{\text{prediction}}$ for every state in $S_{\text{phenomenon}}$, meaning we can write it as:

$$S = \{(s_t, s_p, s_f) \mid s_t \in S_{\text{time}}; s_p, s_f \in S_{\text{phenomenon}}\}$$

The resulting size of S, denoted as $|S|$, is given by:

$$|S| = |S_{\text{time}}| \cdot |S_{\text{phenomenon}}|^2$$

However as only $S_{\text{phenomenon}}$ is uncertain and the rest are assumed to be fully observable ("known"), the algorithms described in Chapter 3 are bound by $|S_{\text{phenomenon}}|$.

A the agent can take two actions regarding on what to do with the last observation, it can either *report* its last observation or it can *wait* until the next observation. Thus:

$$A = \{\text{report}, \text{wait}\}$$

T the transition probability function that specifies the probability of going from state s to s' when taking action a. This function is composed of three factorised transition probability functions that determine how each of the sub states of the sensor node change.

[2] http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/

$T_{\text{phenomenon}}$ describes how the phenomenon changes. Since we assume the phenomenon to exhibit periodic behaviour this will at the very least depend on the current $s_t \in S_{\text{time}}$. Furthermore it seems likely that the new state also depends on its current state $s_f \in S_{\text{phenomenon}}$. Thus $T_{\text{phenomenon}}$ can be described as:

$$T_{\text{phenomenon}} : S_{\text{phenomenon}} \times S_{\text{phenomenon}} \times S_{\text{time}} \to [0, 1]$$

T_{time} describes how the time changes. Assuming we discretise time in such a manner that there is one time state for every moment that the sensor node has to choose an action, the time transition function is non-stochastic. In addition it only depends on the current time. Thus in general the function can be described as:

$$T_{\text{time}} : S_{\text{time}}^2 \to [0, 1]$$

As this function is the same for all models in which we have one time state for every action moment, we can define the function as:

$$T_{\text{time}}(s_t' \mid s_t) = \begin{cases} 1 & s_t' \text{ follows } s_t \\ 0 & \text{else} \end{cases} \tag{4.1}$$

This leads to an explosion of states however. If we need one time state for every moment we can take an action, then for half-minute report intervals, we would need 2880 states to represent a single day. This is clearly unfeasible. What we can do instead is divide time into 48 different segments (half an hour each). Every time segment has its own temperature transition function. To ensure we get rid of the 2880 states and truly have only 48, we make the time transition function stochastic. We advance time by one with a probability of 1/60, which means that in expectation we advance every 60th time step, so 48 times in a day. We add an observation for the time to see what time period we are in. So although the time transition has become stochastic, it is still fully observable. Yet even a factor of forty-eight might be too large for a suitable state space. We return to this in the next section.

$T_{\text{prediction}}$ largely depends on the prediction model used for the POMDP. However it is clear that one thing is very likely to influence the prediction, that is the current phenomenon state $s_f \in S_{\text{phenomenon}}$ as we want our prediction to change in accordance with the current situation. Additionally the time can potentially be of influence here, depending on the sophistication of the prediction model used. We assume at least that our prediction model is deterministic. As a result this $T_{\text{prediction}}$ assigns all transitions either a probability of one or of zero. Furthermore a report action might allow us to update the predictor used (see Chapter 5), thus the action chosen also influences this transition. Thus one form of $T_{\text{prediction}}$ might be:

$$T_{\text{prediction}} : S_{\text{prediction}} \times S_{\text{phenomenon}} \times S_{\text{time}} \times A \to \{0, 1\}$$

Ω the observations that the sensor node can take are strongly correlated to the state space. In fact we assume time and the predictions to be fully observable, meaning that their transitions, whilst stochastic, are observable during policy execution. You can think of them as having observations with probability 1 for the current state. The result is that O only has one non-trivial factorised subspace, which is $O_{\text{phenomenon}}$. The observation space $\Omega_{\text{phenomenon}}$ is still straightforward however, it contains one observation for every possible phenomenon state, thus we get:

$$\Omega_{\text{phenomenon}} = \{o_f \mid s_f \in S\}$$

O the observation probability function is also composed of several factorised functions. As explained above, the only function that warrants explanation is $O_{\text{phenomenon}}$.

$O_{\text{phenomenon}}$ depends only on the current state of the phenomenon $s_f \in S_{\text{phenomenon}}$. Depending on the quality of the sensor that the sensor node is equipped with, we expect the observations to match the current phenomenon state. We can define the observation function as:

$$O_{\text{phenomenon}} : \Omega_{\text{phenomenon}} \times S_{\text{phenomenon}} \to [0, 1]$$

R the reward function is based on the action taken and the deviation between the actual phenomenon value and the predicted value. If the sensor node chooses to wait when there is a large deviation between the actual phenomenon and the predicted value, it should not be rewarded. Similarly it should be penalised for reporting. The first case would hurt the accuracy of the measurements recorded in a base station, whereas the second case would waste energy and thus shorten the lifetime of the sensor node. This means that R can be denoted as:

$$R : S_{\text{prediction}} \times S_{\text{phenomenon}} \times A \to \mathbb{R}$$

We can also say something about the shape of this function, based on what we know about when we do or do not want the sensor node to report.

$$\begin{aligned} R(s_t^i, s_f^j, \text{report}) &\leq R(s_t^i, s_f^j, \text{wait}) && \forall i, j \\ R(s_t^i, s_f^j, a) &\leq R(s_t^i, s_f^i, a) && \forall i \neq j, a \end{aligned}$$

Whereas there are infinite options to encode these criteria, we focus on one that clearly embodies the trade-off between accuracy and report rate:

$$R(s_t, s_f, \text{wait}) = \begin{cases} 0 & \text{if } s_t \neq s_f \\ 1 & \text{else} \end{cases}$$

$$R(s_t, s_f, \text{report}) = \begin{cases} -\alpha & \text{if } s_t \neq s_f \\ 1 - \alpha & \text{else} \end{cases}$$

The idea behind this reward function is simple. We want to maximise the number of times during which we have correctly predicted the behaviour of the environment. Therefore we get a reward of one for being in such state where the environment and predictor states match. On the other hand sending messages is expensive, so we always incur a penalty of $0 \leq \alpha$ for reporting a measurement. This means that the incentive for reporting is rather low, and thus reporting measurements is only really worth it when we expect an updated predictor to work better in the future. A future extension of this work could investigate the use of the POMDP-IR model [81] that rewards "low-uncertainty beliefs" rather than solely looking at states.

4.4. Discussion

By using a thoroughly studied optimisation model in the form of a POMDP-model, which can be solved to a finite state controller matching our requirements, we aim to find the optimal controller that balances the trade-off between accuracy and report rate to match our use case. The α-parameter in the reward function is likely to be the main factor influencing this trade-off, which means that for a set of α-values we expect to get a set of Pareto-optimal controllers. We summarise this in the form of the following hypothesis:

Hypothesis 5 (Pareto optimal POMDP-based controllers)**.** *The POMDP-based controllers form a set of Pareto-optimal controllers for a set of different α values.*

Additionally we expect these POMDP-based controllers to outperform handmade controllers, as they are the result of solving the optimisation problem instead of handmade solutions based on simple rules not tailored for one specific use case. Since they use all information available, in terms of transition and observation model, as well as knowledge about what actions are worth doing, they should be able to find the optimal balance in our trade-off. Thus we formulate our final hypothesis as:

Hypothesis 6 (POMDP-based controllers dominate)**.** *The POMDP-based controllers dominate the handmade controllers in terms of both report rate and accuracy.*

5

The design of predictors for environmental monitoring

Having qualified our approach as a data-driven prediction-based energy conservation technique, we require a prediction mechanism. Because accurate predictions allow us to reduce the number of sent messages required in a sensor network, we want to accurately predict complex real-world phenomena. To what extent these predictors can reduce the need for communication, is the core of sub research question 2. As we expect these environmental properties to exhibit behaviour that is not easily captured by a piece-wise linear function (let alone a piece-wise-constant one), a more complex and trained prediction model may well be needed. To this end we design several neural network configurations that can be trained for environmental monitoring. The question we set out to answer is: can a simple neural network outperform some untrained baseline predictors?

In this chapter we describe several prediction mechanisms. First in Section 5.1 we discuss two simplistic prediction mechanisms that serve as a baseline in our experimental evaluation. Next we focus on using neural networks (NNs) as a predictor. As described in Section 5.2 neural networks can be very powerful in approximating a variety of functions. We look at the features of the data that seem relevant in making predictions about the data, based on an established technique for predicting the behaviour of time-series data in Section 5.3. Unfortunately due to the limited capabilities of the sensor nodes that will have to use the model, this technique can not be applied to our use case. As a result we propose several simplified neural network configurations in Section 5.4. Despite the expectation of very good performance, the data that these networks base their predictions on is not available on the base station side of the network. This means that for the purpose of forming an accurate environmental state estimation at the base station we can not use these networks. Thus we end up with a neural network that is aware of the notion of what data is communicated, which is described in Section 5.5.

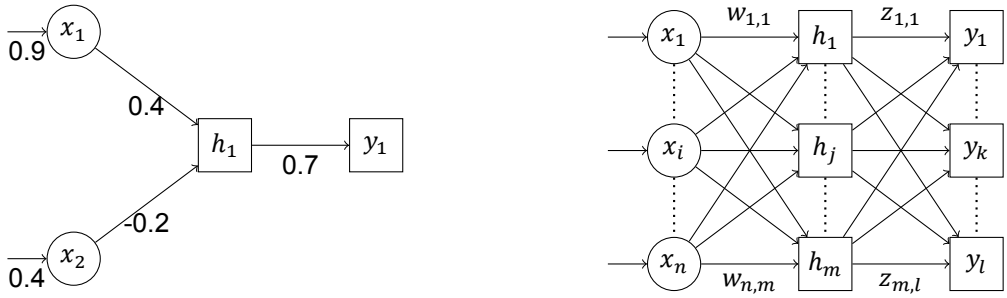
5.1. Baseline prediction mechanisms

Two of the most basic methods that can “predict” the phenomenon under study are what we refer to as the “constant predictor” and the “linear predictor”. These predictors use no knowledge of the phenomenon under study, but are based on simple assumptions. The constant predictor simply assumes the phenomenon under study to not change at all. It will take an initial value or state for the phenomenon under study and use that as its prediction for all time steps that follow. Only when the prediction differs sufficiently from the observation, the predictor is updated and the observation is communicated. When the predictor is updated, the last last communicated observation is taken as the new prediction. This predictor is described in Algorithm 5.

The second naive prediction mechanism is the linear predictor, which is a generalisation of the constant predictor. Instead of assuming that the phenomenon remains constant, we instead assume that it exhibits perfectly linear behaviour. We again assume an initial slope for the behaviour and are only updating it when a sufficiently large difference with the observation followed by a send action has occurred. For this we need the last two values to be remembered, so that we can compute a new slope from them. Optimisations that determine from what two points in recent history a new slope needs to

Algorithm 5 The behaviour of the constant predictor.

```
1: function getConstantPrediction()
2:     return pred
3: end function
4:
5: function updateConstantPrediction(lastObservation)
6:     pred ← lastObservation
7: end function
```



(a) A simple artificial neural network with two input nodes, one node in the middle layer, and one output node.

(b) An abstract formulation of a three-layered neural network that includes the notion used in text.

Figure 5.1: A concrete example of a neural network is shown on the left, with an abstract formulation of neural networks shown on the right.

be computed could be considered, but we do not do so in this work, as this predictor simply serves as a baseline. The resulting predictor is described in Algorithm 6.

Algorithm 6 The behaviour of the linear predictor.

```
1: function getLinearPrediction()
2:     last ← last + Δ
3:     return last
4: end function
5:
6: function updateLinearPrediction(lastObservation, secondLastObservation)
7:     Δ ← lastObservation - secondLastObservation
8:     last ← lastObservation
9: end function
```

We use these two predictors as a baseline when studying the performance of the neural network predictors discussed in the rest of this chapter.

5.2. Theoretical background: artificial neural networks

One of the models used in the field of supervised learning is the artificial neural network. Given the right conditions, a simple neural network is able to approximate any function [23]. In this section we describe how a neural network is constructed, how it performs its computations, and how it can be trained.

5.2.1. The construction of an artificial neural network

Similar to how the brain consists of neurons linked by synapses, artificial neural networks also feature two main components: neurons and the links between them. The architecture and functions described in this thesis are based on the book “Artificial Intelligence: A Guide to Intelligent Systems” by Negnevitsky [60].

In a typical artificial neural network, we can see three types of layers (composed of neurons) in the network: the input layer, one or more middle layers, and the output layer. A node in one layer of the

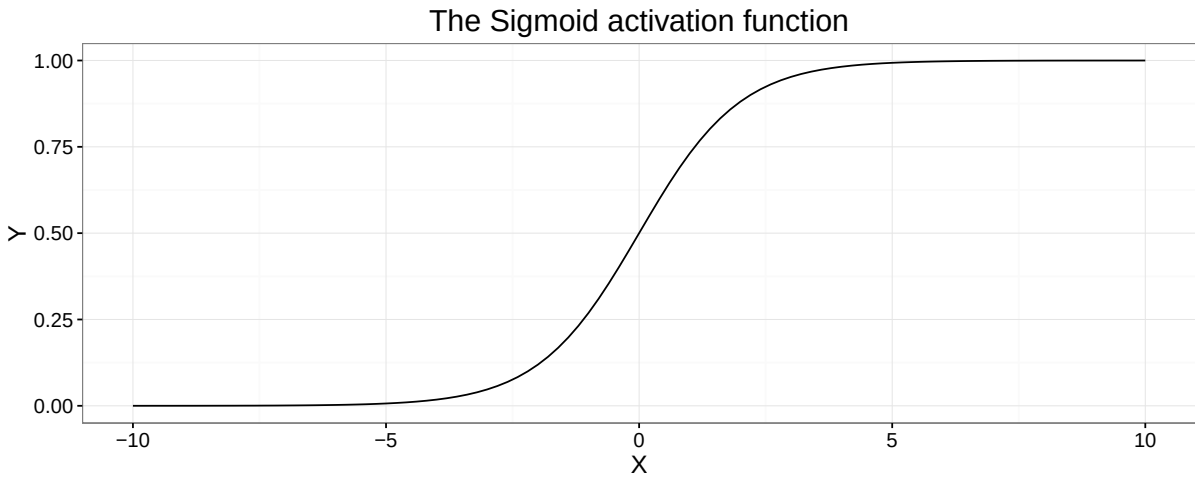


Figure 5.2: The sigmoid activation function that can be used in neurons, where Y is the output and X is the weighted sum of the inputs.

network is connected to all nodes in the previous and to all nodes in the next layer by links. Each of these links has a weight that determines the strength of the link, thus the impact of one node on the other. Figure 5.1a shows an example of a very simple neural network with two input neurons (x_1 and x_2), one hidden neuron (h_1) and one output neuron (y_1). The numbers next to the edges denote the weight of that edge, used in the computations of the network.

Such a neural network can be used as a simple computing element producing an output based on the inputs of the node. The function used to compute the output is often kept simple. As McCulloch et al. proposed in 1943 [58], simply taking the sum of the weighted inputs as the input of an "activation function" works properly for many (current) use-cases. Different version of these activation functions exist, of which Negnevitsky lists four. For our purposes we only require the sigmoid activation function which is shown in Figure 5.2. This function computes an output Y for a neuron with threshold θ, based on inputs x_i over links with weights w_i as follows:

$$Y^{\text{sigmoid}} = \frac{1}{1 + e^{-X}} \qquad X = \sum_{i=1}^{n} x_i w_i - \theta \tag{5.1}$$

For our example depicted in Figure 5.1a, the computation results in the following (assuming the threshold θ of both h_1 and y_1 to be zero):

$$\begin{aligned} x_1 &= 0.9 \\ x_2 &= 0.4 \\ h_1 &= \frac{1}{1 + e^{0.9 \cdot 0.4 - 0.2 \cdot 0.4}} \\ &= \frac{1}{1 + e^{0.28}} \\ &\approx 0.43 \\ y_1 &= \frac{1}{1 + e^{0.43 \cdot 0.7}} \\ &\approx 0.74 \end{aligned}$$

In order to use these networks for the purposes of prediction, we require a method that allows the networks to learn. To describe how this learning method works, we first introduce some mathematical notation for the different neurons and weights found in the network. These definitions are also included in an abstract visualisation of a neural network in Figure 5.1b.

- n input signals labelled x_1 through x_n.
- m neurons labelled h_1 through h_m in the middle layer.
- l output signals labelled y_1 through y_l.
- $n{\cdot}m$ links between the input signals and the middle layer, with weights $w_{i,j}$ for $1 \leq i \leq n, 1 \leq j \leq m$.
- $m \cdot l$ links between the middle layer and the output signals, with weights $z_{j,k}$ for $1 \leq j \leq m, 1 \leq k \leq l$.
- θ the threshold of the different neurons. This threshold is included as an extra node in the preceding layer that has no inputs, but always carries output 1. By learning the appropriate weight for the link connecting the θ-node of the previous layer to a neuron, we essentially simulate such a threshold of value θ. Note that figure 5.1b does not show this θ-node so as to not over complicate the figure.
- α the so-called learning rate of the network. It determines how quickly the network can adapt to new input/output pairs and is used in the training algorithm below. It is not used during computations made by the neural network, in contrast to the other values in this list.

5.2.2. Training the network using the back-propagation algorithm

The method we use for training a neural network is called the back-propagation algorithm which involves repeatedly feeding the inputs to the network and changing the link weights until the outputs match our expected outputs to acceptable levels. One such iteration involves feeding a set of input values to the network, computing the output, comparing these outputs to the expected output and then back-feeding the errors to the network. We will examine each of these steps in turn:

1. Initialise network

Before we can train the network, we need to initialise the weights and thresholds. We do so randomly by taking the required amount of random numbers from a uniform distributions $U(-\frac{2.4}{F_i}, \frac{2.4}{F_i})$ as prescribed by Kubat et al. [49], where F_i is the number of inputs of the neuron for which we are settings the weights.

2. Provide inputs and compute output

With a network in place we can apply a training input. Using the sigmoid activation function in both the middle and output layers, and following the procedure outlined in the previous section, we then compute the output of the network for the given training input.

3. Update the weights

To improve the accuracy of the network, we need to change the weights in the network so that the inputs are mapped to their desired outputs. To this end we compute the current error for every output using Equation 5.2. These errors are the basis of how we compute the weight differences (Equation 5.3, where m_j is the output of middle node j) that should be added to the weights between the middle neurons and the output neurons. In this step the learning rate parameter α plays a role.

$$e_k = y_{\text{desired},k} - y_k \tag{5.2}$$

$$\delta_k = y_k \times (1 - y_k) \times e_k$$

$$\Delta z_{j,k} = \alpha \times m_j \times \delta_k \tag{5.3}$$

The computation for the links between the input signals and the middle neurons follows a similar pattern, the only difference being the computation of the δ value. This takes into account all of the outputs the neuron is connected to as illustrated by Equations 5.4 and 5.5.

$$\delta_j = m_j \times (1 - m_j) \times \sum_{k=1}^{l} \delta_k \times z_{j,k} \tag{5.4}$$

$$\Delta w_{i,j} = \alpha \times x_i \times \delta_j \tag{5.5}$$

Having computed these Δ-values they are added to the current weights of the network to create a modified network that returns a more accurate result.

4. Repeat

During every iteration we compute the sum of squared errors over all outputs computed for all training inputs and use that as our stopping criterion. When we consider it to be sufficiently small we stop the repetition process and assume the network has learned the observations sufficiently well.

Several optimisations in this training process exist. In our implementation we use both an adaptive learning rate [42, 91] as well as a momentum constant [35] to accelerate the training of the neural networks.

5.3. Modelling periodic behaviour with neural networks

Our main interest in neural networks stems from the need to model the phenomena that the wireless sensor networks monitor. In our use case we are monitoring environmental properties (e.g. temperature) inside bird nests, offices and other environments, for an extended period of time. In this section we describe how neural networks have been used to model time-series data in the past, why this does not work for our application, and what environmental properties are relevant for a neural network-based predictor.

5.3.1. Neural networks for modelling time-series data

The main feature of the environmental trends that we want to exploit is the fact that we expect these properties to exhibit periodic behaviour influenced by the day-night cycle. This information can not easily be captured by our base-line predictors, but the periodicity can help to predict how the environment changes over time. Such structured, periodic behaviour is also referred to as time-series data. Gashler et al. [28] describe a method for constructing neural networks that deal specifically with predictions for periodic time-series data.

Rather than having only one hidden layer and using the Sigmoid activation function as described in Section 5.2, this approach features three hidden layers and uses three different activation functions. The first two hidden layers each consist of 24 nodes, 12 nodes using a simple linear activation function ($Y = X$) and 12 nodes using a "softplus" activation function ($Y = \ln(1 + e^X)$). The main function of these two layers is to "warp time" as Gashler et al. call it. Essentially this should allow the network to realise it when the data has a slight offset in time compared to the training set, a technique similar to Dynamic Time Warping [12]. In the third and final hidden layer, we again require 12 linear and 12 softplus nodes, but we also add k sinusoid nodes. These nodes use the sine function as their activation function ($Y = \sin(X)$). They are able to encapsulate the periodic behaviour of the time-based data. Gashler et al. recommend to take k as a power of two for optimal performance.

Despite the good results that Gashler et al. obtain for these networks, these techniques can not be used for our use case. The reasoning for this is twofold, the first being a very practical argument and the second a more conceptual one.

1. A preliminary calculation tells us that this neural network will simply not fit in the memory of most sensor nodes. Excluding any operational logic or other models that need to be stored in the memory, we will at least have to store the weights for all interconnections between neurons in the memory of the node. With only one input (the time) and one output (the prediction), we come to the following number of weights required:

$$\begin{aligned}\text{number of weights} &= 1 \cdot 24 + 24 \cdot 24 + 24 \cdot (24 + k) + (24 + k) \cdot 1 \\ &= 1200 + 25k\end{aligned}$$

   Depending on how we store these numbers, this is unfeasible for even a small k. Assume for instance that each weight is stored as a two-byte value, we require over 2.5KB even when $k = 1$. As many models feature little RAM where 2.5KB would take up half the memory, or at least be a significant portion, fitting in such a neural network as well as the finite state controller introduced in the previous chapters is unfeasible. Additionally the amount of processing power and complexity

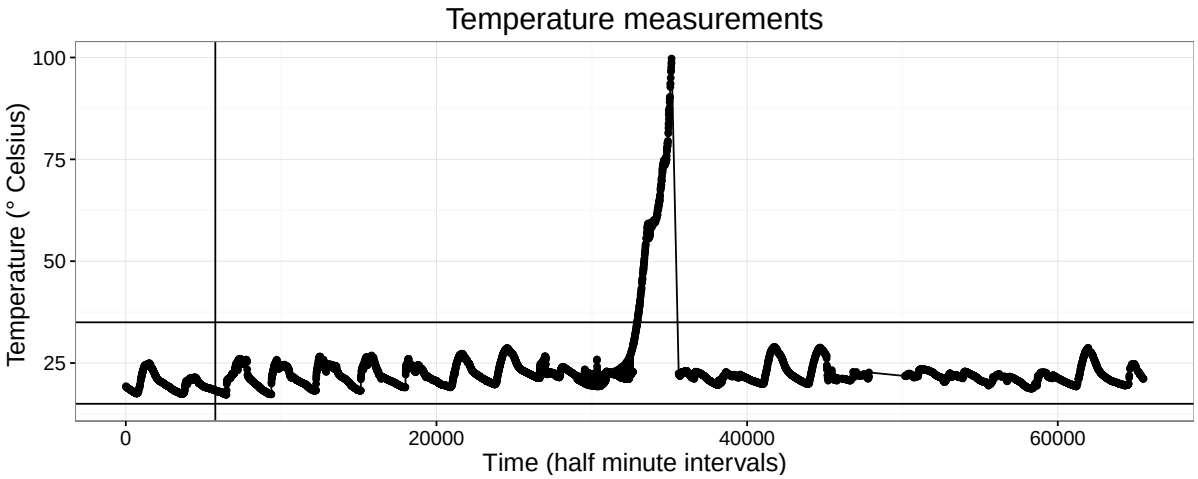


Figure 5.3: The temperature trace from sensor mote one of the Intel Lab data set [15].

associated with updating the network to recalibrate with the phenomenon is likely to be too tasking for the hardware found on these sensor nodes.

2. Whereas this more complex model can work well with time-series based data, Gashler et al. already observe that despite their attempts to prevent this, this network can still get out of phase with the actual data, something that happened for one of their presented experiments. Whereas this still gives a very good impression of the trend that the data is following, this information is meaningless when trying to construct an actual measurement log based solely on the model, as every individual measurement will be off by varying amounts. The dynamic time warping implemented in the first two layers has ensured the trend be continued, but it is still out of sync with the actual data.

Another proposed technique which focuses on applying neural network specifically in wireless sensor networks was introduced by Park et al. back in 2007 [68]. Park et al. describe how a certain neural network architecture could be applied, but conclude with: “The total amount of energy consumption ... will be compared through extensive simulations in further research.” Unfortunately to the best of our knowledge such further research has not been conducted. Perhaps because the hardware requirements of their proposed technique could not be fulfilled by the sensor motes. Similarly work by Shen et al. [74] also features only simulation. It seems that for a feasible real-world implementation we require a much simpler neural network.

5.3.2. Relevant input parameters for the neural network

As a result we explore simpler network configurations that take several inputs to help with the prediction, as opposed to doing very complex computations on just the single input representing time. We identify several such inputs that are relevant to make a good prediction. We have taken a temperature trace for office buildings based on the Intel labs [15] and plotted the temperature as measured by sensor mote one in Figure 5.3. As can be observed the temperature exhibits clear but imperfect periodic behaviour. Additionally a large peak to 100 degrees Celsius can be observed. Although no explanation is given by the authors of the temperature trace, we suspect the sensor was simply malfunctioning for an extended period of time. We do not consider this erroneous data for our training or evaluation of the neural networks.

Based on this temperature trace we believe that the following three input parameters are fundamental parts of a neural network configuration:

Time

To encapsulate the notion of periodic behaviour in the neural network, it is necessary to include the current time of day in the network. This way the network can exploit the periodic behaviour that is directly caused by the time of day.

The previous temperature

As the periodic behaviour does not exhibit a perfectly repeating form, the previous temperature should be able to help with detecting and reacting to oddities in the behaviour. It serves as an offset, as opposed to the time that serves as an indicator for the slope.

A time difference

One additional input parameter should be mentioned here, which is specific to our use case of wireless sensor networks. When working with unreliable hardware as well as with potential for communication failure, it is entirely possible that some measurements are not recorded or communicated properly. To keep the base station and wireless sensor node synchronised, it is important to know how much time has passed since this "previous temperature" input. So this third parameter that could be relevant is exactly that: the time since the previous temperature used as an input.

These three inputs can be combined and interpreted in a variety of ways, which we discuss in the next sections.

5.3.3. Training and evaluating neural network predictors

To evaluate and compare different neural network predictors, we require a training set and a metric for the performance of a prediction mechanism.

For this purpose we again refer to Figure 5.3, which shows the full data trace used for evaluation. The vertical bar separates the training data from the evaluation data. Left of the vertical bar is training data and right of the bar is the evaluation data from which we only use the data points between 15 and 35 degrees Celsius (as indicated by the horizontal bars). The training set comprises the first 3109 data points. Since data for several time steps is missing, these data points end at time stamp 5763 and thus correspond to two days worth of data (measurements are approximately half a minute apart). For training we use the following metric to determine when to stop the training procedure (a metric commonly used in neural network training). The metric is chosen as the sum of the squared error at every time step, as is often done when training neural networks:

$$\text{error}_{\text{training}} = \sum_{i=0}^{n} (\text{output}_i - \text{actual}_i)^2$$

It should be noted that the output and actual value have both been normalised to be in a range of 0 to 1 to be used in the neural network. The value of 0 is associated with 15 degrees Celsius, whereas 1 corresponds to 35 degrees. Any values that are outside of this range, are simply stripped from the data set and dismissed as the sensor malfunctioning. Thus an error of 1 corresponds to a 20 degree Celsius error. During the training phase $n = 3109$. We stop training the network when the training error is smaller than 0.01. The reason this threshold is chosen is that initial experiments show that training the network more either degrades performance or takes a very long time. Larger training errors also show worse performance, meaning there is room for improvement.

In order to evaluate and compare the configurations, we require a score or metric to base the comparison on. For evaluation we simply use the average error in degrees Celsius:

$$\text{error}_{\text{eval}} = \frac{\sum_{i=0}^{n} |\text{output}_i - \text{actual}_i|}{n}$$

5.4. Neural networks that predict based on past measurements

Based on the input parameters identified in the previous section, we formulate several different neural network configurations that combine these inputs in different ways. All of these networks use the previous temperature as one of the inputs. In this section we give a description of their input/output configuration, as well as make predictions about their performance.

Name	Time	Temperature	Time difference
Absolute3	System up-time	Previous Measurement	Time since previous measurement
Relative2	Time of Day	Previous Measurement	-
Relative3	Time of Day	Previous Measurement	Time since previous measurement
LastX-1	-	Previous X Measurements	-
LastX-2	Time of Day	Previous X Measurements	-
LastX-3	Time of Day	Previous X Measurements	Time since previous measurement

Table 5.1: The configurations of input parameters for neural networks we investigate for modelling temperature. The final number of the name indicates the number of inputs that are used in the network.

5.4.1. Network design

Initially we investigated using neural networks that only use the previous temperature, but as was to be expected, this network is very difficult to train. After all, with only the previous temperature, the network does not know if the temperature should increase or decrease next, thus two training points can easily contradict each other. Similarly a network that only takes the time of day as the input, is also unable to learn the behaviour properly, as a slight shift in temperature exactly 24 hours later again results in contradictory training data. It seems that a just a time or temperature is not sufficient to train these types of network, as even with a moderate amount of middle layer nodes, it was impossible to reduce the error in the training phase sufficiently to have the network output useful predictions.

It should also be noted that there are two different ways in which we could include the time parameter, either as an absolute value since the start of our analysis or as a periodic value in the form of the time of day. The rationale behind these choices is that in the first case the network should learn about the periodicity by itself, we do not prescribe that the period of our data is 24 hours. If a slightly different period works better, the network can potentially discover that by itself. In the second case we do already provide that domain knowledge, as we know that for these office buildings specifically this periodicity holds. We expect that the second case allows for much easier training and better results, but we still investigate the system up-time as a parameter for use cases in which information about the periodicity might not be available.

Considering these initial results and alternatives, we set out to investigate several configurations that combine the time and previous temperature in different ways. Table 5.1 offers an overview of the different configurations we investigate.

5.4.2. Predictions on performance

We now formulate a hypothesis about the performance of the network configurations described above, using the evaluation error metric as a performance measure.

Hypothesis 7 (More information makes a better predictor)**.** *Network configurations that consider more information are able to make better predictions.*

To test this hypothesis, we break it down into two follow-ups:

Sub hypothesis 7.1 (Relative)**.** *Relative-3 performs at least as good as Relative-2.*

Relative-3 introduces the notion of the time between the last measurement and the current measurement. Whereas this seems irrelevant in simulations as we an assume the last measurement was one time step ago, in real wireless sensor networks, all kinds of failures can lead to gaps in measurements. Figure 5.3 clearly shows such a gap around the 50.000 mark where no data was recorded for a certain period. Relative-3 can compensate for these blackout periods by weighing the previous temperature less as the time since that measurement becomes larger. Depending on the amount of gaps in your data however, this parameter could be of very little influence.

As Relative-3 only contains little more information than Relative-2, in the form of a parameter that will very often be set to the normalised equivalent of a “1” to indicate that the previous measurement was one time step ago, we expect it to perform on an equal level or better than Relative-2. To ensure enough training data is available, we also offer the data as if there was a gap of lengths up to 10 time steps (using the data from $t - 10$ to train for time t).

We expect that, if nothing else, adding this extra information should not be able to significantly decrease performance. This has resulted in the weak formulation of the hypothesis.

Sub hypothesis 7.2 (LastX)**.** *LastX-3 performs best.*

The culmination of all the relevant information is found in the LastX-3 category. It contains the time in the cycle to give us an indication of whether temperature should be going up or down, it contains the previous X measurements that can inform us of the speed with which the temperature is changing and finally the time since previous measurements that can tell us to what extent the previous measurements are relevant. However it does not know about the time period that was between these X measurements, only how long ago the most recent of these X measurement was recorded. Thus for data that contains many gaps, for instance when the last measurement was a single time step ago, but the other (X-1) were over one hundred time steps ago, LastX-3 may unjustly give these measurements too much weight in the next prediction. Even so, we expect it to win out over the others on average.

Similarly to our note for the previous hypothesis however, we should note that the difference between LastX-3 and LastX-2 is likely to be relatively small if there are few gaps in the data collection process.

5.5. Neural networks that predict based on the last communication

As the temperature changes gradually, the neural networks shown in the previous section are likely to be very accurate in terms of prediction. This is very useful if they are only used on the sensor nodes themselves. However they are not usable when the base station has to have an accurate environmental state estimation at all times. After all the base station does not have access to the inputs that the neural network uses for its predictions, namely the last observation. In this section we discuss a modification of the inputs, that allows the base station to produce the same predictions and thus produce such an environmental state estimation.

5.5.1. The network configuration

Rather than keeping track of the last time we measured in the neural network inputs, we instead provide the last time we communicated data as that input. The “last temperature” input parameter is then also changed to the last temperature that was communicated. During training we use data for up to one hundred points in the past as the “last communicated measurement” and we also discount training errors for larger time gaps. In other words we focus on predicting very well for short time periods and accept losing some accuracy as time goes on. The main reason for this is that it allows us to train the network more easily and keep the size of the network relatively small. The new training metric thus becomes:

$$\text{error}_{\text{training}} = \sum_{i=0}^{n} \sum_{t=1}^{100} \left(\frac{\text{output}_{i,t} - \text{actual}_{i,t}}{t} \right)^2 \tag{5.6}$$

Now that we have created a neural network configuration that the base station can execute independently, we can go one step further and allow the neural network to update itself every time the sensor node performs a “report” action. This update action allows it to adapt to changes in the environment that were not present in the training data, thus potentially enabling the neural network to become a more accurate predictor as time goes on. For this update we use only the last data that is communicated to the base station, that is the current observation that differs from the prediction and the time since the last measurement. To update we use a single round of the training algorithm, outlined in Section 5.2. The α factor used in this update, the learning rate, should be chosen in such a way that the new update changes the network for the variations observed, but not so much that the old training data is completely ignored.

5.5.2. Predictions on performance

As these network configurations no longer have access to the previous measurement, we expect their performance in terms of average error to drop compared to the networks introduced in the previous section. To combat this problem we allow the neural network to update itself after reporting a measurement. This introduces the trade-off that is central to this thesis for the predictors. Minimising the number of reported measurements, whilst also obtaining the minimum average error.

As we expect our trained neural network predictors to outperform the baseline predictors, we formulate the following hypothesis:

Hypothesis 8 (Trained predictors dominate generic predictors)**.** *A prediction mechanism designed and trained for our use case outperforms generic prediction mechanisms that do not consider environmental behaviour.*

Since our variant of the trained predictor has several parameters that influence its performance, we first investigate the influence of these different parameters (the learning rate α, the maximum allowed error x, the effect of different data sets). To this end we formulate several sub hypotheses that illustrate how the parameters influence the performance of the network. These parameters when set correctly should then allow us to test Hypothesis 8. Although the influence of the parameters is predictable, we formulate claims about their influence as sub hypotheses to structure our analysis of them:

Sub hypothesis 8.1 (Maximum allowed error)**.** *As we increase the maximum allowed error x, fewer messages will be sent, at the cost of a higher average error.*

As x is the parameter that solely determines when data should be communicated, increasing x means we allow a higher error before reporting a measurement. This means that we expect fewer messages to be communicated as x increases, but as x also forms the upper bound on the average error we expect the average error to increase with it. Different values of x are likely to form a set of Pareto-optimal solutions, with none outperforming the others in *both* the number of reports and the average error.

Sub hypothesis 8.2 (Learning rate)**.** *An α of either 0 or 1 will not perform well.*

Although formulated as a weak hypothesis, it seems likely that both the training data and the live-updates can help to improve the quality of the predictions made by the neural network. By either ignoring the live-updates ($\alpha = 0$) or by quickly overwriting the training data ($\alpha = 1$), we lose part of the information that form these predictions and thus both of these α values are likely to decrease performance. We expect an α that balances the use of old and new information to perform best.

Sub hypothesis 8.3 (Data)**.** *The networks perform better on the data akin to the training data, than to dissimilar data.*

It seems likely that the networks will perform better when confronted with data that matches their training data well. However for α values that allow the network to adapt to changes in the data trace, these differences in performance may be overcome through the adaptation.

Sub hypothesis 8.4 (NN outperforms naive)**.** *The neural network predictors outperform the linear and constant predictors.*

As the embodiment of a trained prediction mechanism, we expect the neural network predictors to outperform the untrained naive baseline predictors. Through their training they should be able to foresee periodic changes in the data set, whereas a constant predictor can not. We expect this foresight to help the neural network predictors obtain both a smaller report rate, as well as a smaller average error when compared to the constant or linear predictors.

6

Evaluation of different prediction mechanisms

The design of the neural network predictors in Chapter 5 has put forth several hypotheses about the performance of these predictors. In this chapter we perform an experimental evaluation of the network designs to test these hypotheses.

In Section 6.1 we focus on the networks that use the last measurement to predict the next. These networks are designed to show that neural networks can indeed predict environmental states. Using these networks we test Hypothesis 7 (More information makes a better predictor). In Section 6.2 we turn to the networks that incorporate the notion of communication in their prediction mechanism. These networks can actually be used in our application of wireless sensor networks and we perform the multi-objective evaluation associated with our use case. Using these networks we test Hypothesis 8 (Trained predictors dominate generic predictors).

6.1. Past-based neural networks

To show that neural networks can accurately predict an environmental state change, we have trained networks according to the specifications outlined in Table 5.1. With these networks we aim to test Hypothesis 7. In this section we describe how the networks are trained and evaluated, what the results of the evaluation are and what this means for the hypothesis, and why these networks can not be used in the use case of wireless sensor networks.

6.1.1. Experimental setup

To show that the neural networks can predict environment state change, we have taken the temperature trace as report by sensor mote one from the Intel research labs data set [15] as it is one of the most complete traces with only one clear error-prone period. This data is plotted in Figure 5.3. As described in Section 5.3.3, we have selected the first 3109 data points (two days worth of data) for training the networks. For the evaluation we use the full data trace (including the training data, but excluding data points outside of the 15 to 35 degrees Celsius range).

As the final neural network depends on the random weights chosen before training, we have generated one hundred random instances of each configuration, which are then trained until the normalised error was smaller than than the threshold of 0.01. Each training was stopped as the target error was achieved, but there is no lower bound on the training error. As a result not all networks are trained to the exact same degree. Furthermore for some networks it was exceedingly hard to train them to the required training level, for which training was stopped after 10 minutes. These networks are left out of the analysis. All results given below are averages over at least 70 and at most 100 networks.

To ensure that such a neural network is of a size that can be used in an actual wireless sensor mote (which has very little memory), we have limited the number of neurons in the middle layer to the number of input nodes + 8. In this way the total number of cross-connections is limited to 128 between input and middle nodes. This should only take only half a KB of RAM to store in memory (again assuming 2-byte weights), ensuring it would fit in the memory of the sensor motes.

Configuration	Average Training Error		Average Evaluation Error	
absolute-3	0.046 ±	0.008	0.24 ±	0.05
relative-2	0.034 ±	0.007	0.10 ±	0.04
relative-2	0.097 ±	0.015	0.13 ±	0.02
relative-3	0.038 ±	0.007	0.11 ±	0.05
relative-3	0.097 ±	0.015	0.12 ±	0.02
last2-1	0.040 ±	0.008	0.09 ±	0.04
last2-2	0.036 ±	0.008	0.10 ±	0.04
last2-3	0.038 ±	0.008	0.10 ±	0.04
last3-1	0.042 ±	0.009	0.10 ±	0.04
last3-2	0.041 ±	0.008	0.11 ±	0.04
last3-3	0.041 ±	0.008	0.11 ±	0.04
last4-1	0.042 ±	0.009	0.10 ±	0.04
last4-2	0.042 ±	0.009	0.11 ±	0.04
last4-3	0.043 ±	0.009	0.11 ±	0.04
last5-1	0.043 ±	0.009	0.09 ±	0.04
last5-2	0.041 ±	0.009	0.11 ±	0.04
last5-3	0.043 ±	0.009	0.11 ±	0.04

Table 6.1: The average performance and standard deviation of the different configurations on the full data set for sensor 1 after training on the first two days of data points (3109 points). The errors shown here are the average deviation in degrees Celsius between the prediction (the output of the neural network) and the measurement. Note that the more untrained configuration results are not shown.

6.1.2. Evaluation of the configurations

The average error per measurement for the training set, as well as for the full evaluation set after validation over the one hundred networks for the different configurations are shown in Table 6.1. To give some indication as to what the values in this table represent, Figure 6.1 plots the error in predictions for two networks with the relative2 and last3-1 configurations. As can be observed the errors for measurements with index 19000 to 23000 are not shown. These errors are quite a lot larger and are left out of the analysis. This interval corresponds to a period of time where the measurements are inconsistent before the data this is removed due to it being larger than 35 degrees Celsius. To remove any faults these faulty measurements might introduce, these predictions and measurements are left out of the averages presented in this section.

With these results we can now revisit the hypotheses formulated in the previous chapter and see to what extend they hold.

Sub hypothesis 7.1 (Relative)**.** *Relative-3 performs at least as good as Relative-2.*

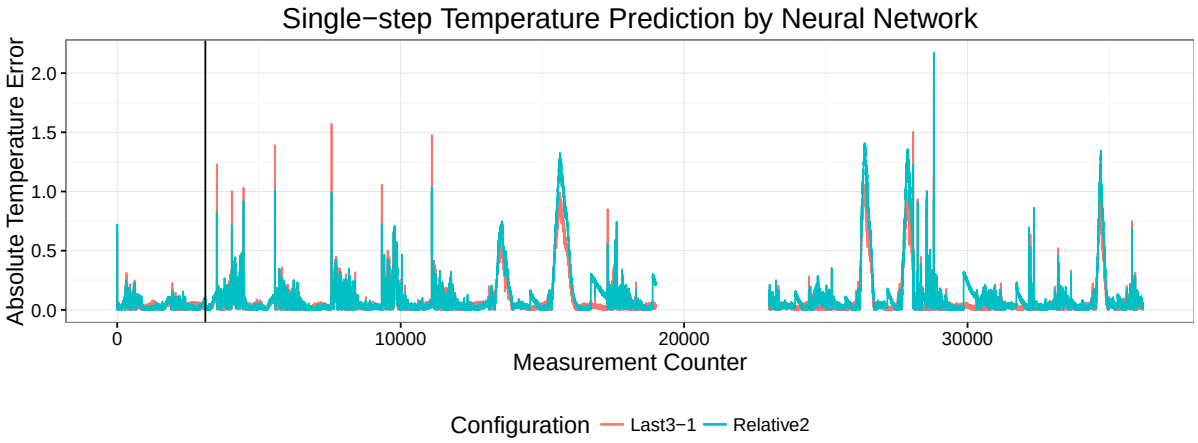


Figure 6.1: The errors of the predictions made by a well-trained Relative3 and Last3-1 network compared to the real data.

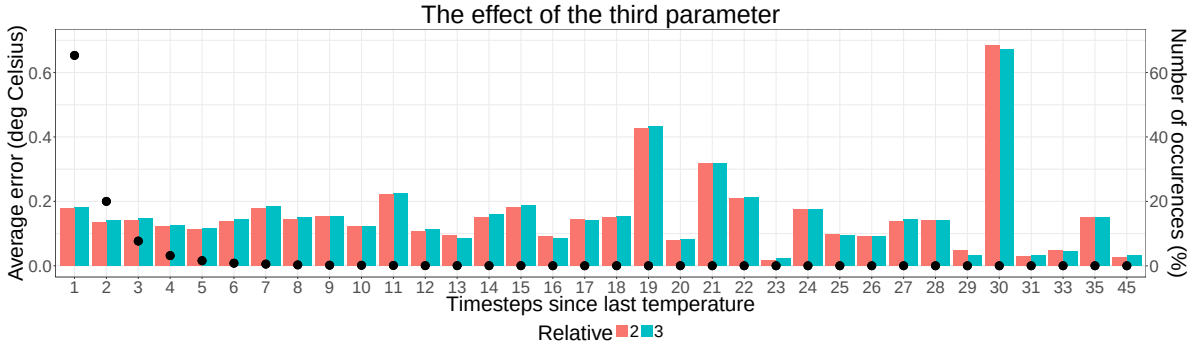


Figure 6.2: The average errors of the Relative2 and Relative3 configurations for different time deltas, as well as the number of times a time delta occurs (indicated in black).

Indeed, Relative3 seems to outperform Relative2 when we train for an average error of 0.1 degree Celsius in the training phase, but it does not outperform it when we train to an average error below 0.04 degrees Celsius. So it this possible improvement, or slight degradation really due to the extra input parameter (the time since the last measurement) or could it be that this network just happened to be configured slightly differently and thus perform better by chance? In other words are these "improvements" significant?

To determine this we focus on the better trained network and plot the average error vs the time delta. In other words we want to determine if the third parameter of the Relative-3 network has a (positive) impact on the average error. We would expect that the third parameter can help decrease the error when two measurements are separated by a longer period of time. We formulate this as a follow-up:

Follow-up hypothesis 6.1.1. *Relative3 performs better than Relative2 when a larger period of time separates two measurements.*

Figure 6.2 shows a plot of the average error for increasing time deltas for both the Relative-2 and Relative-3 configurations. Additionally the black points indicate the number of occurrences of the different time deltas. Based on this figure we draw two conclusions. First over 90% of the data features a time delta of less than 5 time steps. In other words impact of this parameter is very likely to be minimal, if there is to be any impact at all. Secondly, for a large time delta the average error is not necessarily smaller for the Relevant-3 configuration than it is for the Relevant-2 configuration. The difference in error is often insignificant (especially with the small sample sizes for large time deltas). Thus we have to reject the follow-up. As for Hypothesis 7.1, this can be considered valid. Although Relative-3 is not better than Relative-2, it certainly is not significantly worse than Relative-2 either.

Sub hypothesis 7.2 (LastX)**.** *LastX-3 performs best.*

With the knowledge of our observations about Relative2 and Relative3 we would expect the differences between LastX-2 and LastX-3 to be very small as well, which is indeed the case. What is perhaps more surprising is that LastX-1 seemingly outperforms both LastX-2 and LastX-3. The inclusion of previous measurements therefore seems to outweigh the impact of the current time of day.

In LastX-3, as noted before, the final parameter indicates the time since the last measurement, it says nothing about how far apart the other X measurements were. As a result when the last measurement was a single time step ago, but the other $X - 1$ were a hundred time steps ago, then this is not known to the network and they might be given undeserved weight. This will quickly repair itself after X time steps, but for these X time steps the predictions are potentially off by a large amount.

So it seems that there is little difference between LastX-3 and other network configurations. LastX-1 seems to be on par with Relative2 and so do others. When plotting the errors in the predictions for a Relative2 and a Last3-1 network, we see that these show very similar trends (see Figure 6.1). This means that we have to reject Hypothesis 7.2 as the LastX-3 configuration is not necessarily best. As a result we also reject Hypothesis 7. Adding more information does not necessarily result in a better predictor.

6.1.3. Applying these networks in wireless sensor networks

Whereas the LastX-2 and Relative2 configurations have shown promising results in terms of being able to predict the temperature based on the data available in the sensor mote, this configuration can not be used in the wireless sensor network applications we work on in this thesis. In these real-world settings the base station is tasked with producing an environmental state estimate at every time step. To this end, we need to simulate this neural network in both components, the wireless sensor mote and the base station. But if the past measurements are an input to the neural network, then the base station can only get a prediction if they also have the last few measurements. This in turn means that the sensor mote has to report every measurement, thus nullifying the effect of the prediction based sending.

There are two potential methods to resolve this issue. One option is that rather than taking the actual last measurement, we use the last prediction as an input. After all both the sensor mote and the base station have access to the prediction data. As you can imagine however, an erroneous prediction will only ensure the next prediction is even worse if it is used as the input. Thus using the previous prediction to make new predictions is likely to lead to very bad prediction models.

The second option is to use the data that Relative3 already uses, but in a slightly different way. Rather than describing when the last measurement was taken and what this measurement was, we instead describe when the last measurement was communicated and what this measurement was. We make the neural networks “communication aware”, as described in Section 5.5. We evaluate these networks in the next section.

6.2. Communication-aware neural networks

Having established that, despite their good performance, past-based neural networks can not be applied in wireless sensor networks, we now turn to the communication-aware networks designed in Section 5.5. These networks are aware of the last communicated measurement and base their prediction on that. In this way both the base station and the sensor mote can reliably produce the same prediction. During the design we formulated a hypothesis about the performance of the communication-aware neural network compared to that of the naive predictors in Hypothesis 8. In this section we test this hypothesis.

6.2.1. Experimental setup

Since these neural networks are aware of the communication that is required in our application, we have them execute a generic policy that tells them whether or not they should report their data. If their prediction deviates more than x degrees from their measurement, they report the measurement and update the prediction model. This policy is described in Algorithm 7.

Algorithm 7 A simple policy that communicates every deviation from the prediction of more than x degrees.

Require: some prediction model that produces a prediction for every measurement.
Require: a threshold value x that determines whether or not the measurement should be reported.

```
1: function onMeasurement(measurement)
2:     p ← getPrediction(...)            ▷Arguments depend on predictor used
3:     if |p − measurement| > x then
4:         Report(measurement)
5:     else
6:         Sleep
7:     end if
8: end function
```

For our experimental validation we also require data sets to evaluate the networks against. As before, our networks are trained using data from sensor mote one of the Intel lab data. For evaluation we have selected four days worth of data, instead of the full data trace. Two of the evaluation data sets that we use are day one of the sensor mote one, which is part of the training data, as well as day three of sensor mote one, which is not part of the training data. Additionally an analysis of how much the different data sets for different sensors are correlated to one another, shows us that the data of sensor mote two is very similar to that of sensor mote one, but the data from sensor mote 51 (which is in a

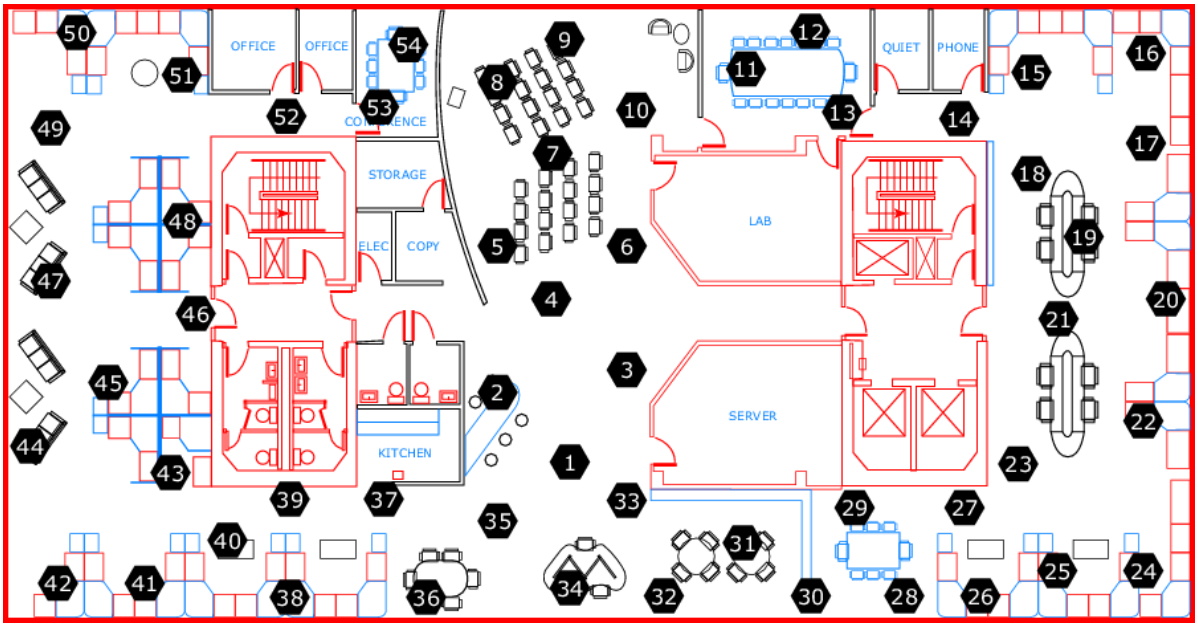


Figure 6.3: Lay-out of the Intel Labs [15], with the sensor mote placements indicated.

different part of the lab, see Figure 6.3) shows a number of deviations from sensors one and two. Thus we use day one from sensor mote two and day five from sensor mote 51 as well, as data that is very similar and very dissimilar respectively. In Figure 6.4 the temperature traces for the different sensors are shown.

For our experiments we vary a total of four parameters:

The predictor used

Depending on what neural network we use the results might differ. Thus the choice of neural network is the focus of a preliminary experiment described in the next section. Additionally we consider the two naive predictors, the linear and the constant predictor, to see if our neural networks can still outperform them with other configurations of the other parameters.

The data set used

As is already addressed above, we select four data sets for the evaluation. For these experiments, we again use only the data that is between 15 and 35 degrees Celsius and furthermore we use linear interpolation to substitute any missing data. This is done to ensure a continuous data trace, so that our policy can simply be executed at every time step in our data. The fact that we use a linear interpolation is an important detail, as this might give an advantage to the linear predictor. We come back to this in the next section when we discuss the results.

The learning rate used α

This parameter determines to what extent the neural network can adapt its behaviour when a wrong prediction is made. We test the values of 0, 0.05, 0.5, 0.95, and 1 in our our experiment. We include the values 0 and 1 to see how well the networks do without the ability to adapt and with an adaptation so strong that the training information is essentially lost.

The maximum allowed error x

With a larger allowed error, we expect fewer observations to be reported, accompanied with a larger error. The interplay between the learning rate and the maximum error is also important however, as with a small error, but without an opportunity to learn from the reported mistakes, we will repeatedly keep sending messages. We use the values 0.05, 0.1, 0.5, and 1 degrees Celsius in our evaluation.

The sub hypotheses 8.1 to 8.3 focus on the effects of these parameters and are tested below.

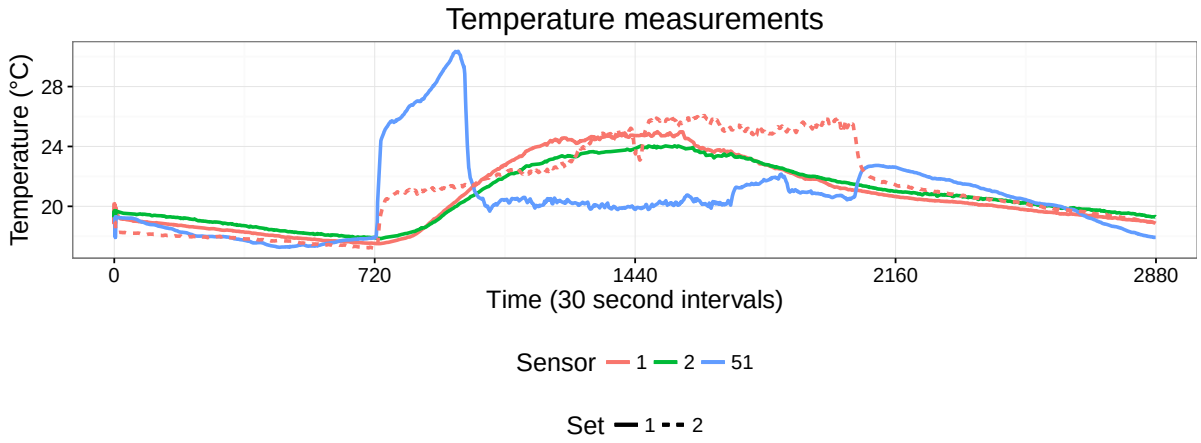


Figure 6.4: The chosen days of data for the communication-aware neural network evaluation. The time scale is in measurement numbers, with measurements taken at 30 second intervals. In other words 2880 measurements corresponds to one day worth of data. The data of sensor two matches the training data of the first day of sensor mote one very well, whereas both the second set of mote one and especially the set of data from sensor mote 51 show different behaviour.

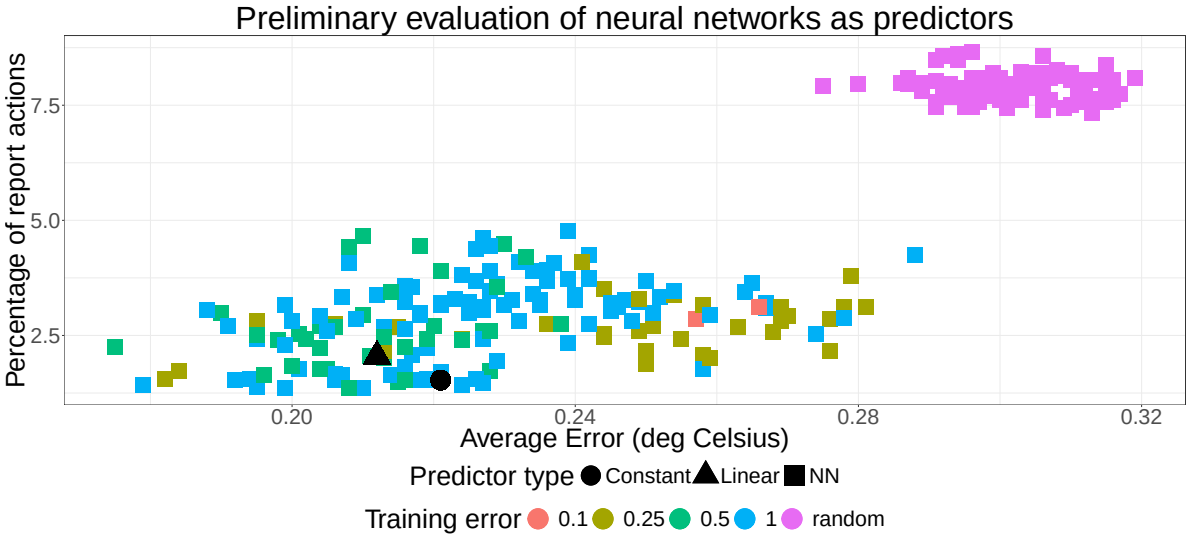


Figure 6.5: The average error and number of reported observations for one hundred randomly initialised neural networks trained to several different levels. The training error is still normalised, similar to the previous section, but the average error is given in degrees Celsius. These results are obtained from an evaluation on the third day of sensor mote one, after training the networks on the first two days of sensor mote one. The black symbols indicate the performance of the naive predictors on this data.

6.2.2. Preliminary experiment: neural network selection

Since we aim to vary four parameters, each taking at least four values, the number of experiments to run grows very quickly. If we were to again evaluate all combinations for one hundred trained neural networks this would result in 8000 evaluations. To make the number of evaluations more manageable we perform a small preliminary experiment to select the neural networks for the full evaluation.

We start with one hundred randomly initialised neural networks, that take the three inputs, have fifteen nodes in the hidden layer, and have one output node. Such a network can be saved in less than one KB of memory, so it can be saved and updated in a wireless sensor mote. These networks have been trained on the first 3109 data points (two days of data) to several different training errors, and were then evaluated against the third day of data, where all points originate from sensor one of the Intel lab data. For the maximum error x a value of 0.5 degrees Celsius is chosen, and a learning rate of 0.05 allows for small adaptations to the neural network if deviations were found. We have selected these two parameters in such a fashion that they allow for small changes, hopefully without obfuscating the effect of the initial configuration of the neural network.

The results of this initial analysis are depicted in Figure 6.5. In this figure the number of "report" actions performed (expressed as a percentage of the 2880 possible messages) is shown against the average error (in degrees Celsius). For the computation of the average error we have only included the errors during the "wait" actions, as no error is incurred from a wrong prediction when the measurement is reported. The colours represent different errors during the training phase (all using the error metric given by Equation 5.6). There are fewer networks with a training error of 0.1 and 0.25 as many networks did not manage to reach these thresholds within the time limit set for training the networks.

Based on this graph we draw two conclusions. First, perhaps unsurprisingly, the random networks perform very badly. They still manage to do relatively well, likely due to the fact that they can adapt themselves every time they deviate too much. However with an error that is 50% larger and approximately twice as many reported observations they perform much worse than the trained neural networks. Secondly, we have quite a number of networks that can accomplish an error that is smaller than the constant and linear predictors, without requiring extra report actions. For our more complete experimental validation, we use the seven neural networks that form the Pareto front found in this evaluation of the third day of sensor mote one.

6.2.3. Experimental validation: results

With the seven neural networks we obtain a total of 560 data points for different combinations of the parameters. Armed with these data points we set out to test Hypothesis 8:

Hypothesis 8 (Trained predictors dominate generic predictors)**.** *A prediction mechanism designed and trained for our use case outperforms generic prediction mechanisms that do not consider environmental behaviour.*

To do so we first explore the impact of the different parameters on the performance to find the optimal neural network predictor, which we then compare to the linear and constant predictors.

Sub hypothesis 8.1 (Maximum allowed error)**.** *As we increase the maximum allowed error x, fewer messages will be sent, at the cost of a higher average error.*

Figure 6.6 shows the average number of report actions and the average error over the different NNs we have used. The error bars indicate the minimal and maximal values observed. Without exception in all cases the number of messages sent goes down as x increases, whereas the average error increases. In the most extreme case tested here, a maximum error of one degree Celsius, the average number of reports required is only 29 (slightly more than 1%) and in two thirds of the configurations with $x = 1$ it requires even less than 29 reports. The x that is chosen will have to depend on the specific use case as it is clearly a large influence in the trade-off between the report rate and the accuracy. This hypothesis is thus confirmed.

Sub hypothesis 8.2 (Learning rate)**.** *An α of either 0 or 1 will not perform well.*

To investigate this hypothesis we have visualised the data in a different manner. Figure 6.7 denotes the results for different learning rates. Similar to Figure 6.6 for the maximum error comparison, we have averaged the results of the seven neural networks, with the black bars indicating the minimum and maximum scores of the seven networks.

Figure 6.6: Predictor results highlighting the impact of the maximum error x parameter. Every bar represents the average score of the seven neural networks used, with the error bars indicating the minimal and maximal scores of these seven networks.

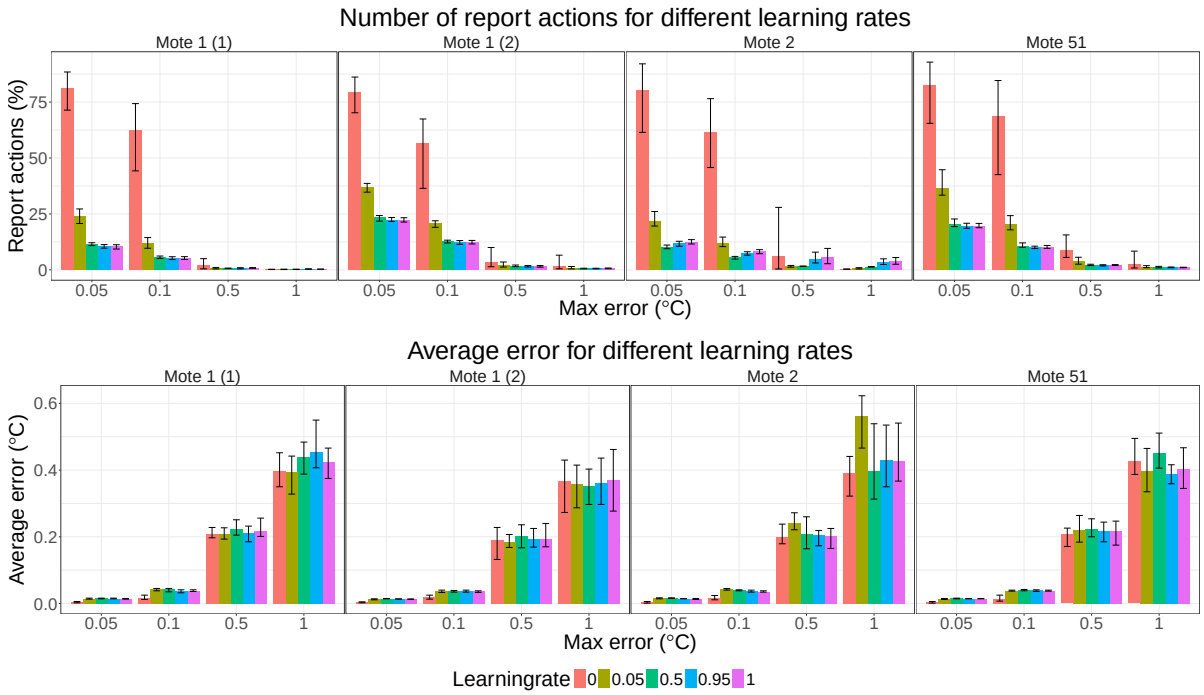


Figure 6.7: Predictor results highlighting the impact of the learning rate α parameter. Every bar represents the average score of the seven neural networks used, with the error bars indicating the minimal and maximal scores of these seven networks.

The first thing that is immediately clear is that not allowing the network to adapt, i.e., having a learning rate of zero, always leads to the worst performance in terms of the report rate. Since many of the observations are reported, the average error is often slightly lower, or at least on par with other learning rates. Allowing for small adaptions, with an α of 0.05, already drastically reduces the number of messages required. In all cases by more than 50%. Thus allowing the neural network to update itself leads to huge improvements in performance. There is only one exception to this, which is for the data of sensor mote two with a maximum error of one degree Celsius. The reason for this exception is likely found in the very small amount of report actions performed for this set with this maximum error. All networks feature ten or less report actions, furthermore as this data closely resembles that of sensor mote 1, any changes made here are more likely to correct for a small deviation from the pattern and thus make the network "overreact". The fact that a learning rate of zero outperforms the others here is thus best classified as an exception rather than rule.

However we also hypothesised that an α of 1 results in bad performance. This is not immediately clear and at first glance performance seems similar for an α of 0.5, 0.95, and 1. For three of our data sets indeed the learning rate does not change the performance significantly irregardless of the maximum error used. It is again sensor mote 2 that differs slightly in this regard. For all maximum error values used in these experiments the best performance is achieved when $\alpha = 0.5$. The reason is probably similar to that given before. The network is already trained well for the start and the end of the day (which are most similar to the training data), but will produce slightly wrong predictions for the middle of the day. If the network is trained too much to compensate for this, that also undoes the training for the later half of the day. Since performance is thus comparable for learning rates of 0.5 and up, but there are some cases for which 0.5 does outperform higher learning rates, it seems that 0.5 works well for this use case. This hypothesis is thus partially confirmed, as it is clear that an α of 0 will not perform well, but an α of 1 does not perform as bad as we had initially hypothesised.

Sub hypothesis 8.3 (Data)**.** *The networks perform better on the data akin to the training data, than to dissimilar data.*

Figure 6.8 shows the impact of using a different data set. Again scores have been averaged over the seven neural networks with the black bars indicating minimal and maximal scores. Since we have already established that a learning rate of 0.5 works best or at worst similar to other learning rates we have only plotted the results that use a learning rate of 0.5 for our analysis of this hypothesis.

Two main conclusions can be drawn from this figure. First it is clear that for small error tolerance (a low x value) the training data and the data most similar to the training data easily outperform the others. For larger error tolerance this distinction is less clear, for reasons already mentioned before. Thus having a well-trained predictor trained on data that matches your observations well, unsurprisingly results in less deviations that cross the threshold. The second main conclusion is slightly more surprising, namely the average error is relatively constant regardless of the data set used. This is the second time we have found a parameter that seems largely irrelevant for the average error incurred by our predictors (the first being the learning rate). Thus the hypothesis is confirmed, the networks do much better on data that matches the training set well.

Sub hypothesis 8.4 (NN outperforms naive)**.** *The neural network predictors outperform the linear and constant predictors.*

Once again we plot the data in a different form to more easily distinguish between the different types of predictors. This time we again freeze the learning rate to 0.5 and average the neural network scores, whilst also showing the lowest and highest scores of the neural networks. Figure 6.9 shows the comparison of the different predictor types.

There are two main conclusions to draw from these results. First we notice that for small maximum errors the choice of predictor has surprisingly little influence on the average error. Any differences are likely compensated by the number of report actions performed. This brings us to our second conclusion, which is that the linear predictor often outperforms the neural network predictor in terms of report rate. Overall it seems that the neural network predictor is often the worst predictor for small maximum errors. There are several factors that likely influence this worse performance.

First notice that the linear predictor very often also outperforms the constant predictor. Remember that we have used linear interpolation of the data to fill in any missing data points, which is likely the reason for the improved performance of the linear predictor. As the linear predictor can find and use

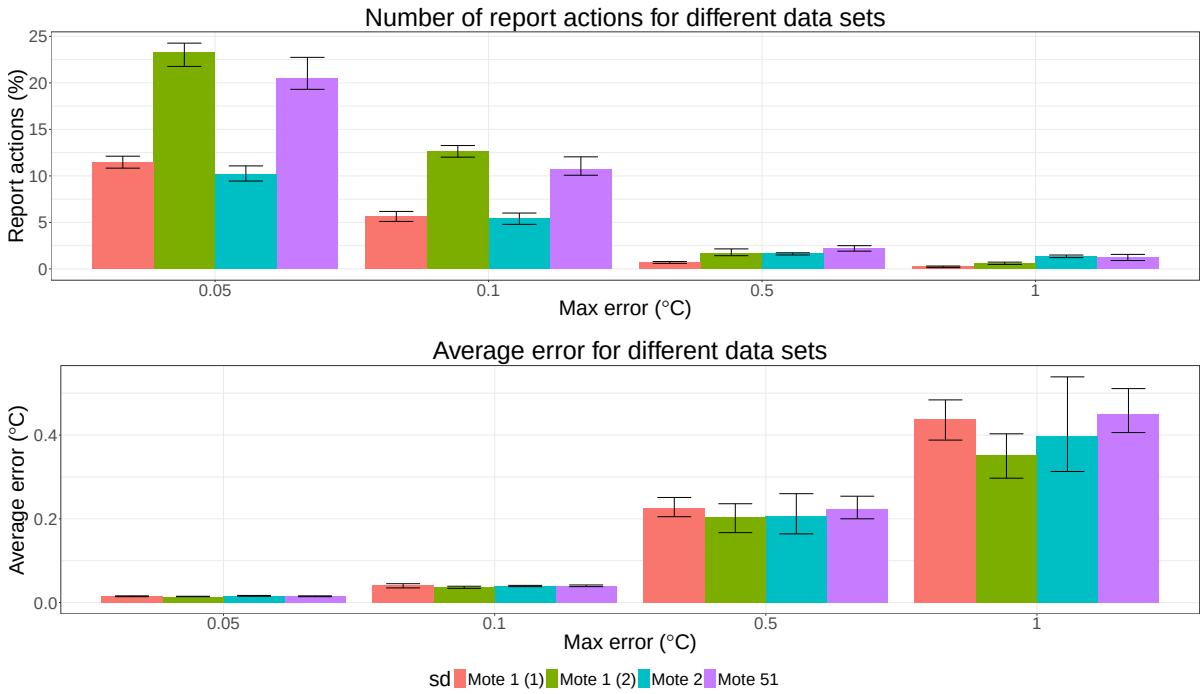


Figure 6.8: Predictor results highlighting the impact of the data set used. Every bar represents the average score of the seven neural networks used, with the error bars indicating the minimal and maximal scores of these seven networks.

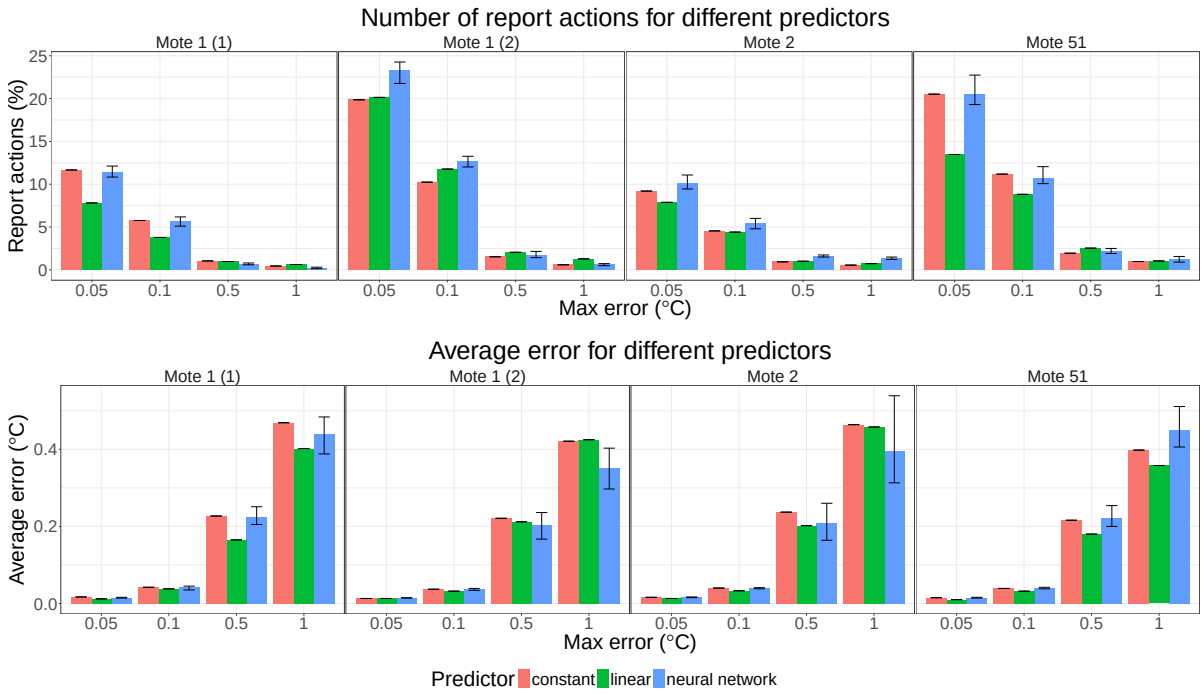


Figure 6.9: Predictor results highlighting the impact of the predictor type used. Every bar represents the average score of the seven neural networks used, with the error bars indicating the minimal and maximal scores of these seven networks.

the same slope that was used in the interpolation after one report operation, it will incur zero error for a few time steps and thus also have no reason to report. The neural networks on the other hand will incur small errors and especially for the small maximum error of 0.05 degrees this may be sufficient to warrant a report operation. Secondly the neural networks have been trained on the incomplete data set that does not feature this interpolation. This was done on purpose as for real-world deployments, data to train the networks on may also be incomplete. After all the data used here is from a real-world data set and data collection rates vary from 10% per day to over 90%. This also means however that it was not possible to train all the neural networks to a sufficiently small error in an acceptable amount of time. In our preliminary experiments we have selected seven neural networks with different training errors[1]. Suffice to say that better trained networks might outperform a linear/constant predictor, though the time-intensive nature of training as well as the limited data set used for training make it difficult to verify this.

As a result we can only reject Hypothesis 8.4 in part and thus not confirm Hypothesis 8. For large error margins, the neural networks clearly can outperform the linear and constant predictors. But for small error margins, the linear predictor works better for the evaluation circumstances used here. Depending on whether or not a perfect linear interpolation of the data is realistic, and how much data is available to sufficiently train the neural networks, this may also be true for real-world use cases. However as we expect the data to exhibit non-linear patterns and that more time and data can be used in training the neural networks, perhaps even with more advanced techniques than those deployed in this thesis, the neural network predictor certainly has the potential to outperform the linear predictor for small error margins as well.

One unexpected conclusion that we take away from the evaluation is that out of the four parameters that we have varied here (predictor type, data set, learning rate, and maximum error) only one seems to have a significant impact on the average error incurred by the predictor. Only the maximum allowed error x has a significant influence. The fact that this parameter influences the average error is not surprising, after all it serves as an upper bound on the average error, but the fact that none of the other parameters significantly influence the average error is remarkable. To work around this dependency on the maximum allowed error and allow for more flexibility in balancing the trade-off we consider the base station logic in the next chapter. This allows us to reason about the reported predictions, potentially gaining more insight into the environmental state.

[1] Due to the inclusion of the drop-off for larger time periods this is impossible to convert to an average error in degrees Celsius. Additionally the requirement of sending messages every so often to update the “last communication” inputs of the network make it hard to determine an average error during for the training data set.

7

The evaluation of the base station logic: different controllers

During the design of the base station logic and the controllers in Chapters 3 and 4 we formulated several hypotheses about the relative performance of the different policies. In this chapter we test these hypotheses through experimental evaluation. The main goal of the evaluation is not to experimentally determine the optimal controller for our use case, but rather show the potential of the base station logic and the potential effectiveness of the systematically created controllers.

To this end we evaluate the controllers both on samples from a transition model based on a real world data trace, as well as on instances from this data trace. The exact set-up of the experiments, including how the data sets are chosen and sampled, is described in Section 7.1. The rest of the chapter considers each of the hypotheses in turn. In Section 7.2 we evaluate Hypothesis 1, setting a baseline using only transition data and no controller. Next we evaluate Hypotheses 2 and 3 in Section 7.3, using the periodic controllers CC and PCC. Hypothesis 4 is tested in Section 7.4 using the SPCC and SDC controllers. The last two hypotheses, Hypothesis 5 and 6 are tested in Section 7.5 using the POMDP-based controllers. Finally we discuss some weaknesses of our methodology discovered during the evaluation in Section 7.6.

7.1. Experimental setup

There are several factors that influence the performance of the base station logic. We can control most of them throughout the experiments with the different FSCs. In this section we list these factors, how they can influence the performance, and how we vary them throughout the experiments.

Data sets

To evaluate the different FSCs presented in this chapter we again use the data from the Intel labs. In fact we use the same data sets as used in the Neural Network evaluation. The used data is shown in Figure 6.4 on page 52. Remember that the first data set of sensor mote one is part of the training data (for our predictor, as well as for the environmental model below) and that the first data set of sensor mote two is very similar to this. The second data set of sensor mote one, featuring the third day of data, and the first data set of sensor mote fifty-one are both chosen as they differ significantly from the training set.

Additionally we also evaluate the policies against data sets sampled from the environmental model described below. The reasoning for this is two-fold. First we want to ensure that the four days of data we have chosen do not (by chance) represent very unlikely scenarios in the environmental model we construct. If this is the case then results from these data sets are likely to misrepresent the effect of our policy. Secondly, it allows us to discuss the average performance of the policies over a wide variety of data traces adhering to the same transition model.

Environment model

We require a transition and observation model for the experiments, the T and O functions of our model. We create these based on the first two days of the measurement of sensor mote one. We discretise the twenty-four hour period into 48 periods of half an hour. For every half hour we examine all the consecutive measurements made by the sensor and based on these transitions we get $T_0(t'|t,x)$ that prescribes the probability of moving from temperature state t to temperature state t' in time state x. However we should be aware that these first days only give a course approximation, meaning other transitions within such a period may also exist even though they are not included in these first two days. To that end we create the final transition function T as such:

$$T(t'|t,x) = \begin{cases} 0.95 \cdot T_0(t'|t,x) & \text{if } T_0(t'|t,x) > 0 \\ 0.99 & \text{else if } t' = t \text{ and } T_0(t'|t,x) = 0 \forall t' \\ 0.01/(Z-1) & \text{else if } t' \neq t \text{ and } T_0(t'|t,x) = 0 \forall t' \\ 0.05/Z & \text{else} \end{cases}$$

Where Z is the number of instances where $T_0(t'|t,x) = 0$. This allows for deviations from the training data to occur without system failures due to being in impossible states. We do something similar for the observation function O, which we simply assume to report the correct observation at every time step with a probability of 0.95.

A discrete state space

The discretisation of the temperature state space determines the size of S and is thus important in the run-time and space-time complexity of the algorithm, but also affects the accuracy of the method. For instance a discretisation of states that represent a range of 10 degrees Celsius will be very quick and space-efficient to work with, but does not allow us to differentiate between an office that is a comfortable 21 degrees or a rather hot 28 degrees. For the experiments in this section we use a discretisation of 1 degrees Celsius, as preliminary experiments indicate this provides good results for comparison, whilst also providing sufficiently quick run times for the evaluation.

Predictors

For the prediction mechanism used in the experiments we have several options, as explored in Chapter 5. In order to use the linear predictor either the last two observations need to be communicated, or the last communicated observation and the current observation should be used to create a new delta. Both methods would mean a deviation from the intention of this predictor or the base station logic. Reporting the last two observations deviates from the current policy in the base station, which assumes only the last observation to be communicated. Using the previously communicated observation as the second point deviates from the original intention of the predictor, which was to use the most recent data to predict how the temperature might evolve. Additionally this predictor is likely to be most influenced by the fact that the data is linearly interpolated, as we have seen in our evaluation in Section 5.5. The neural network predictor is configured to use an α of 0.5 based on the previous evaluation and we use the neural network that Pareto dominated 4 of the other 6 neural networks in the evaluation. The other two neural networks are “equally good” as the chosen network, outscoring it either in the average report rate or the average error in the aggregation of all performed experiments.

Thus in our experiments we use the constant and neural network predictors unless indicated otherwise. The notable exception to this rule is the evaluation of the sampled data sets. Since we sample these from a discretised transition model, we can not produce the continuous temperature inputs required for the NN predictor.

7.2. The transition-based policy

The first policy we investigate is the transition-based policy as described in Section 3.3. Remember that this policy only reports every X’th measurement and uses the transition-model known by the base station to infer information about the unreported observations as well as the state of the environment. We formulated the following hypothesis:

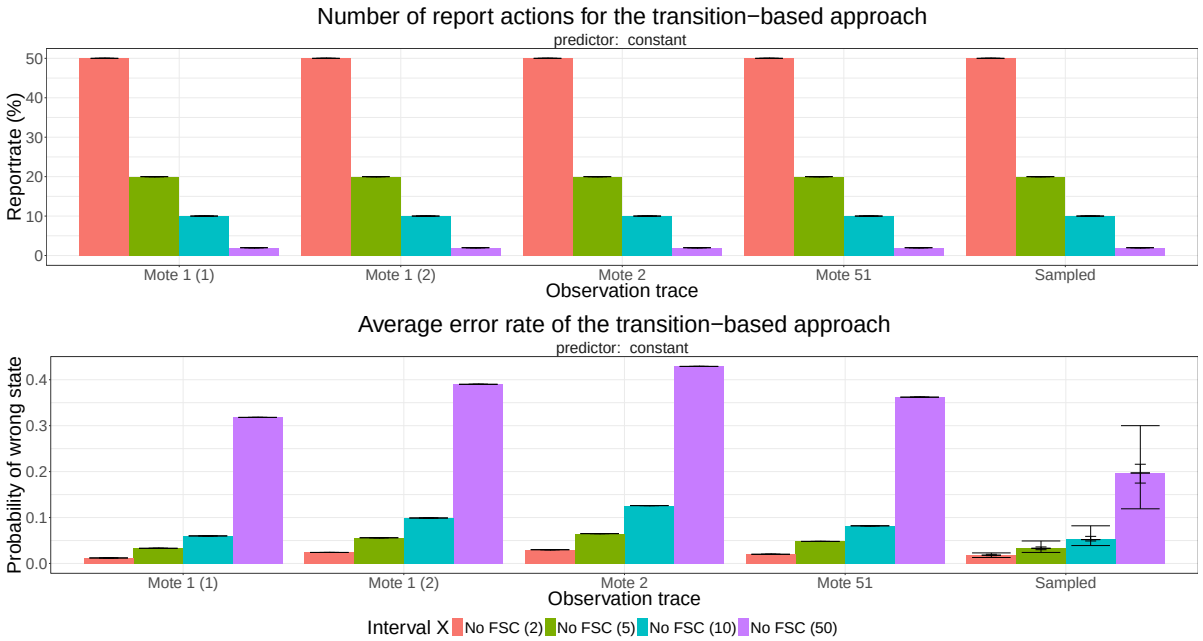


Figure 7.1: Performance of the transition-based policy for different data sets, different predictors, and different values of the X parameter. The report rate is expressed as the percentage of actions that are "report" actions. The average error is expressed as the average probability assigned to incorrect states.

Hypothesis 1 (Transition-based policies require frequent communication)**.** *An increasing X quickly decreases the accuracy of a policy that sends observations every X'th time step, irregardless of the contents of the observation.*

As the communication grows more infrequent we expect a lower accuracy of the system, as well as fewer messages to be sent. Figure 7.1 shows the results for a variety of X parameters. It is clear that the number of messages sent is directly correlated to the X parameter used. Since a choice for $X = x$ means one message is sent every x time steps, thus the fraction of actions that are report actions is exactly $1/x$. This is also clearly visible in the figure, with the report rate ranging from fifty to two percent for a X of two to fifty respectively. We also observe that the error indeed grows quickly as X increases. Since we can only base our log on the occasional observation combined with the a priori transition probabilities, longer periods of time increase uncertainty about the state the environment is currently in. Thus as X increases the periods of uncertainty grow larger and accuracy decreases. This hypothesis is thus confirmed.

Although it is now clear that the accuracy does indeed suffer, it is perhaps not immediately clear *why* this happens. After all the transition model should represent the environment accurately and should therefore serve as a good baseline. We hypothesise that the main reason for this drop in accuracy is that state-changes are found too late, or not noticed at all.

Follow-up hypothesis 7.2.1 (The transition-based policy does not notice changes between two reports)**.** *Deviations of length $t < X$, and state changes between two reports are unnoticed by the transition-based policy.*

Figure 7.2 show the probabilities assigned to different states over time. As can be seen the probabilities are very high surrounding the report operations, but at the times between two report operations the probabilities become much lower. In the lower half of the figure we zoom in on a time period that shows a clear deviation that is undetected around t=1935. Since the 1900 and 1950 reports both report the same observation, any deviation between those seem "unlikely" based on the transition function (especially since for this time and this temperature the probability of changing state is low). Notice also how the deviations between 2000 and 2050 are not picked up. As there is no evidence for a state change in that time period at all (in the transition function) the policy has no reason to assume a state change, other than the new evidence from 2050 which says that at some point the state must have changed. This too shows that oscillations and changes within the period are not properly recorded by

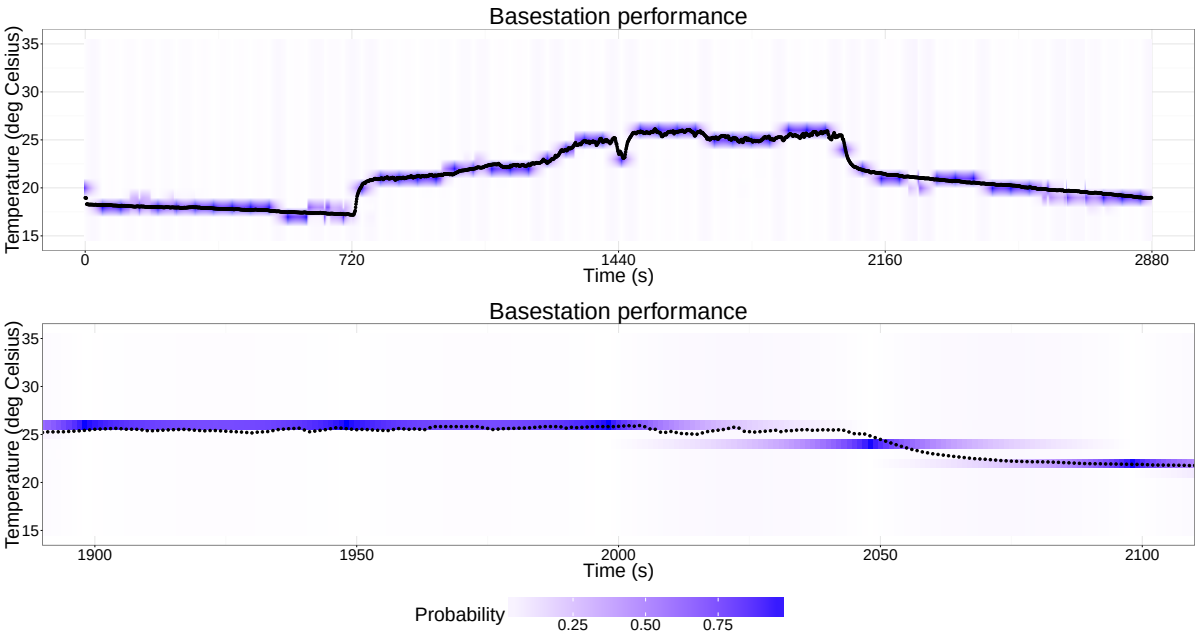


Figure 7.2: The “Sleep 50” configuration misses the deviations between 1900 and 1950, as well as the changes between 2000 and 2050.

the transition-based policy. The follow-up is therefore confirmed, we have found a plausible reason for the inaccuracy of the transition-based method.

7.3. Periodic controllers

We now move on to the policy introduced in Section 3.4 which introduces the notion of a controller in the sensor mote. This controller reacts to the observations and by also reporting the node of the controller periodically, the base station has some indirect information about the uncommunicated observations. We hypothesised that this additional information can lead to a better accuracy in the base station, provided of course that the FSC is constructed in such a way that its node provides insight into the observations that were made. To this end we evaluate the counting controllers introduced in Section 4.1.1.

7.3.1. Benefits of using the controller

To start with, we consider the simple counting controller (CC) which compares the measurements to the average temperature (25 degrees Celsius for our data). Using this controller, we test Hypothesis 2:

Hypothesis 2 (FSC-based dominates transition-based)**.** *The FSC-based approach outperforms the transition-based approach for an equal report interval* X*.*

Figure 7.3 compares the performance of the transition-based policies with that of the controller-based policy that uses the CC configuration. The results are very clear. The added benefit of the reported count ensures that the accuracy is higher for the same report rate. This effect becomes stronger as the report interval grows larger. For a report interval of two the differences are negligible. This is not unexpected as there is little improvement to be offered by the counting controller here. There is only one observation that is not known for every interval and this observation is likely to be similar to the reported observation preceding it. Only for cases where this does not hold, and where the observation falls at the different side of the average, can the counting controller actually offer additional insights.

For larger intervals, the power of the counting controller increases as it significantly reduces the observation space from which the unreported observations must have originated. It is also clear that this impact is much larger for the sampled data sets than for the real world data. The reasoning for this is twofold. First of all, some of these data sets hardly ever measure temperatures above 25 degrees

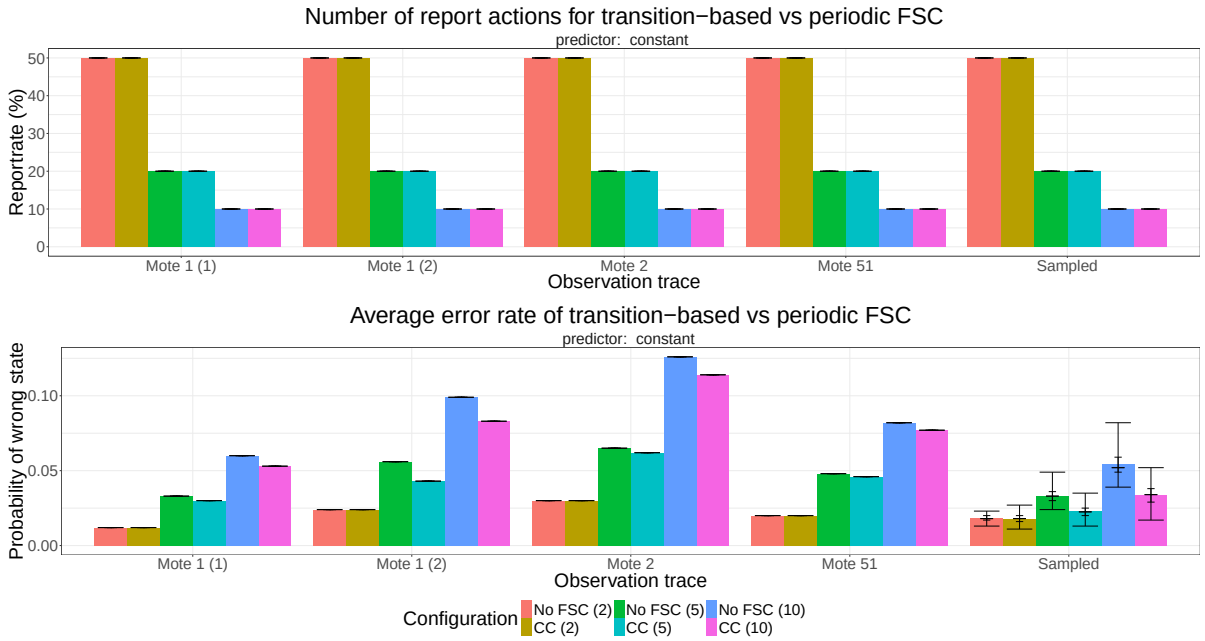


Figure 7.3: Performance of transition-based and controller-based policies for different data sets, different predictors, and different reporting intervals. The report rate is expressed as the percentage of actions that are “report” actions. The average error is expressed as the average probability assigned to incorrect states.

(the used average). In fact the first data set of sensor mote two never reaches a temperature above 25 degrees. Whereas the counter still reduces the observation space in those cases, it only strips off those observations that were unlikely to begin with. Secondly, for the sampled data set we use the corrected transition model which allows for unexpected changes. This means that the sampled data traces may feature many more “large” transitions than the real world data.

Nevertheless, the controller-based policy, even with a very simple controller, clearly outperforms the transition-based policy. Thus Hypothesis 2 is validated.

7.3.2. Potential improvements and shortcomings

Despite the improved performance compared to the transition-based approach, we already described how 25 degrees is no ideal threshold for the real world data sets. In fact we would rather like to have a dynamic threshold that changes as the temperature changes. This is where we can use the notion of the prediction mechanism. The predictors allow us to use a dynamic threshold that the base station and sensor mote can share without needing to explicitly communicate it. As the predictor is assumed to be a good indication of what the temperature is going to be, the counts reported by the controller are more likely to represent small offsets to the predictor. This should increase the accuracy of the log in the base station, without incurring extra reports (as the report intervals are unchanged). We formulate this as:

Follow-up hypothesis 7.3.1 (Adding the predictor)**.** *Counting deviations to the predictor dominates counting deviations to the average.*

For this follow-up we use the PCC configuration introduced in Section 4.1.1. The resulting report rate and accuracy are depicted in Figure 7.4. The impact of using the predictor is immediately clear. Again the difference for a reporting interval of two is minimal, but for larger reporting intervals, the difference is accuracy is very significant. For the data that is part of the training set this difference can even be as much as 50%. The follow-up hypothesis is thus confirmed, using the extra information in the form of the predictor can significantly improve results.

Even with this improved version, which features a dynamic threshold, the accuracy still degrades for larger reporting intervals. In Section 3.4 we also hypothesised that the controller-based policy still suffers as the reporting interval grows larger. We formulated this as:

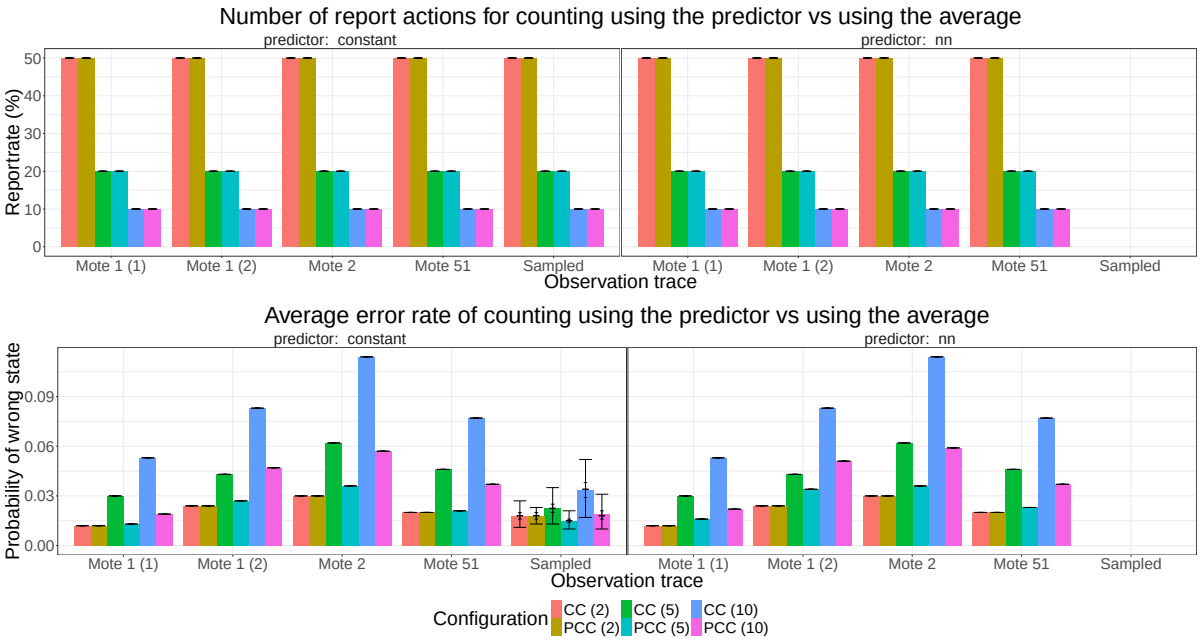


Figure 7.4: Performance of counting controllers that compare to the predictor or the average for different data sets, different predictors, and different reporting intervals. The report rate is expressed as the percentage of actions that are “report” actions. The average error is expressed as the average probability assigned to incorrect states.

Hypothesis 3 (FSC-based still requires frequent communication)**.** *An increasing X decreases the accuracy of a controller-based policy that sends observations and nodes every X'th time step.*

The evaluation for the previous hypothesis and follow-up as depicted in Figure 7.3 and 7.4 already reveals that this is indeed the case. Whereas the counting controller always wins out over having no controller for a given reporting interval, larger intervals still result in a lower accuracy. To see why this happens consider the three functions depicted in Figure 7.5. The figure shows three trends: a constant trend (depicted in green), a linear trend (depicted in blue), and a sinusoid trend (depicted in red). All three trends spend an equal number of measurements above and below the threshold (which is assumed to be zero for this figure), and thus result in the same count if they are reported at time 62. The linear trend offers a slight advantage over the others, as the inclusion of the last observation means we can at least differentiate it from the constant trend, but differentiating between trends that result in the same count can not be done by the base station. This means that for larger report intervals, low counts would actually become less informative. We formulate this belief as follows:

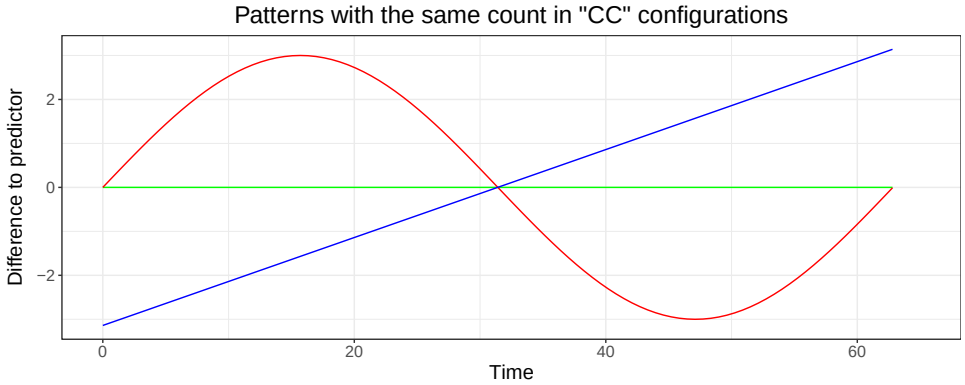


Figure 7.5: Two different functions (in red and green) that both result in a count of zero for CC configurations, meaning the configurations can not distinguish between them. The blue trend can not be confused with these however, as the last observation is also reported.

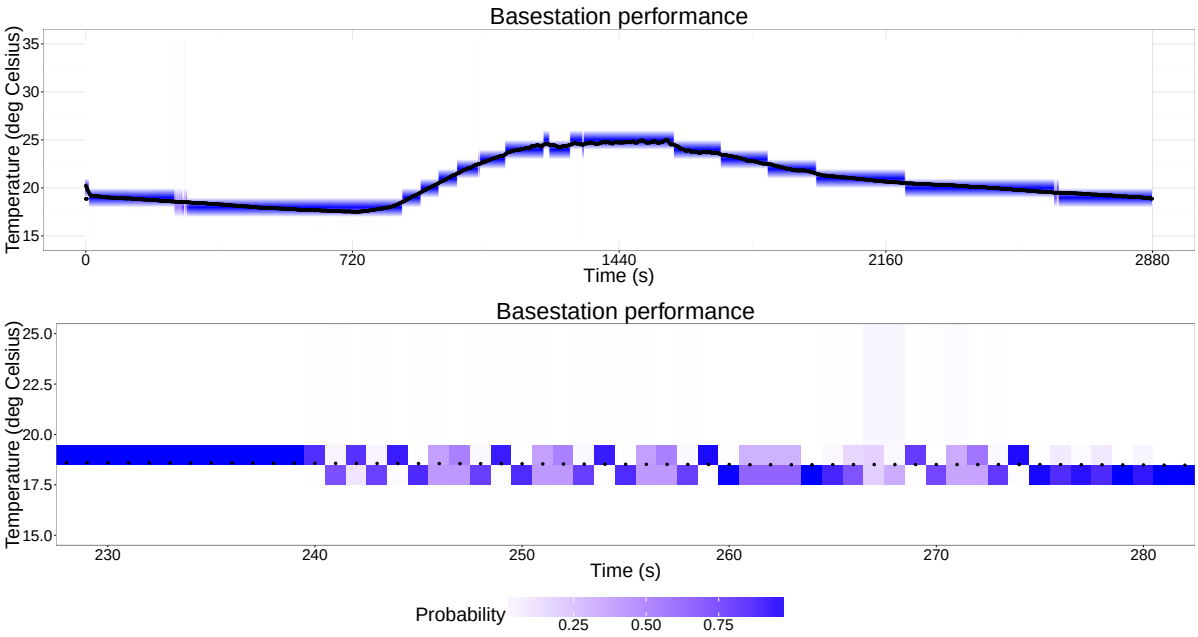


Figure 7.6: A counting configuration can not properly represent oscillations, with many probabilities ending up in a 50/50 or 60/40 split.

Follow-up hypothesis 7.3.2 (Low counts are not informative)**.** *Counting deviations performs badly when the observations oscillate around the predictor.*

Figure 7.6 shows the probability as logged by a counting controller with a report interval of five, for the first data set of sensor mote 1. As can be observed there are many measurements surrounding a state border here. We use a reporting interval of 5 here, meaning every fifth measurement is communicated and thus every fifth state can be estimated with better accuracy. However as previous measurements surround the predictor (which is the constant predictor in this figure), the FSC can not accurately tell how the temperature must have behaved. From 275 onward, observations are consistently equal to the predictor for some time, thus allowing the FSC to accurately report this period of time. This hypothesis is also confirmed.

7.4. Using the FSC to determine the action

As we have seen the counting controllers considered so far suffer from at least one clear weakness, namely their inability to distinguish between different trends that result in the same count. This means that some of their reports may be mistimed. Had they reported after a quarter of the period of a sinusoid for instance, this would be much more informative than after a full period. To this end we now turn to the SPCC configuration outlined in Section 4.1.1, which determines the action based on the current count. A larger count equals a larger probability of reporting.

Using this controller, we evaluate Hypothesis 4:

Hypothesis 4 (FSC with varying report rates)**.** *Controllers that base their actions on the node in the FSC outperform controllers with fixed reported intervals in terms of report rate for a similar accuracy.*

7.4.1. Improvements from the stochastic controller

To test the hypothesis we compare the deterministic and stochastic counting controllers. As these controllers do not directly share a parameter that allows for a simple comparison (M and C perform subtly different functions), we instead consider a variety of values for their parameters. In the previous section the analysis of the deterministic counting controllers features three values for its reporting interval. These three values result in a set of Pareto optimal solutions. To consider this hypothesis validated, we are looking for stochastic counting controllers that dominate the deterministic versions.

Figure 7.7 shows the performance of the stochastic and deterministic counting controllers. It is

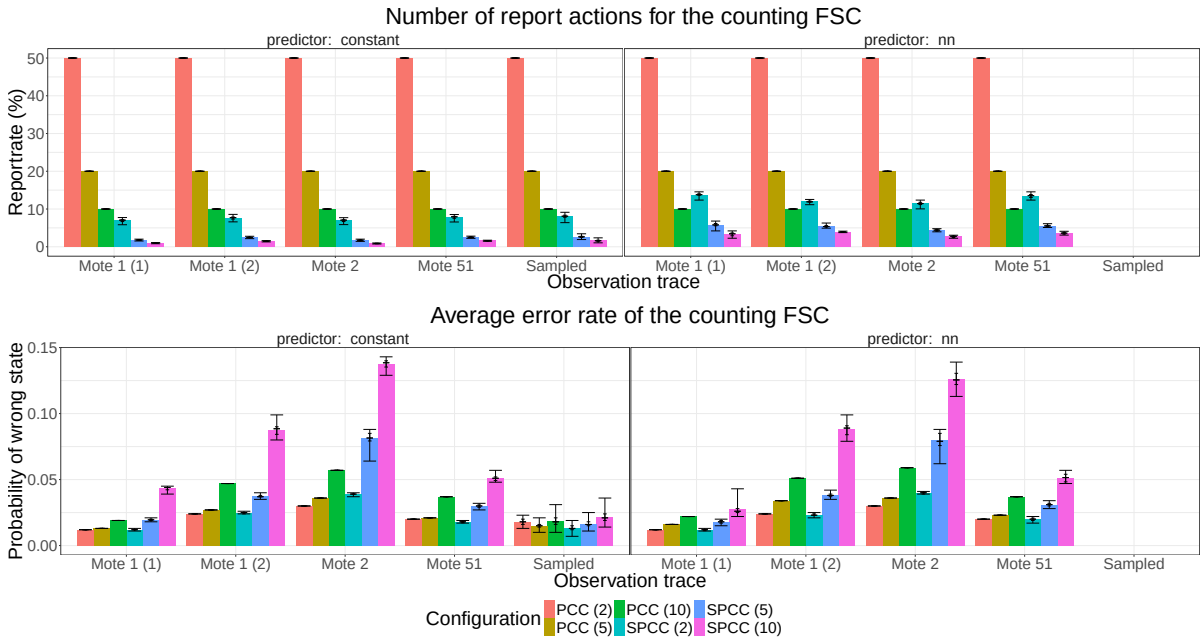


Figure 7.7: Performance of counting controllers that have a fixed report interval versus those that use stochastic actions for different data sets and different predictors. The report rate is expressed as the percentage of actions that are "report" actions. The average error is expressed as the average probability assigned to incorrect states.

clear that the stochastic controllers also represent a set of Pareto-optimal solutions. As we increase the M-parameter of the controllers, the report rate decrease and the average error increases. However for $M = 2$ we have found a dominating solution when compared to the deterministic equivalents.

There is one thing we should not overlook however: $M = 2$ only dominates the controllers tested here. Using the counting controllers, we can make their report interval as small as we want it to be (even zero for single-day evaluations). This means that the stochastic controller does not necessarily dominate all counting controllers. It does however dominate all feasible counting controllers. Recall that the counting controllers require $O(C^2)$ nodes where C is the report interval. For the evaluation of the controller with $C = 10$, performing the analysis only at every tenth time step, an evaluation using four cores on a high-end desktop computer already requires an hour. It seems fair therefore to state that we have found a stochastic controller that dominates its deterministic equivalent, as it requires less report actions for a similar (or even significantly better) accuracy. The hypothesis is thus validated.

7.4.2. Stochastic actions in other controllers

As an alternative to the counting-based controllers, consider the deviation-based controllers DC and SDC introduced in Section 4.1.2. These controllers also feature a difference in when the actions are taken. In the DC-configuration we only report when a deviation of size D is encountered, whereas the SDC configuration has a probability bigger than zero for all deviations smaller than D, and is only guaranteed to send deviations of at least size D. Intuitively DC will report less for a given D, but also be less accurate, whereas SDC will report slightly more, but be more accurate. Although both DC and SDC are controllers that base their actions on the node in the FSC, and thus are not suitable for testing Hypothesis 4, they do provide a good contrast to evaluate the impact of stochasticity in the actions, as described by this follow-up:

Follow-up hypothesis 7.4.1 (Deviation-based controllers)**.** *The deviation-based controllers improve in terms of accuracy for a similar report rate, when we use a probability function based on the deviation that determines the action, rather than sending only deviations of size D.*

Figure 7.8 shows the performance of these deviation-based controllers. There are two main observations to be made here. First notice how the report rates are much higher for the configurations that use the neural network predictors. Especially the configuration that reports any deviation (DC-1, representing a DC configuration with $D = 1$) shows report rates that are more than five times as large

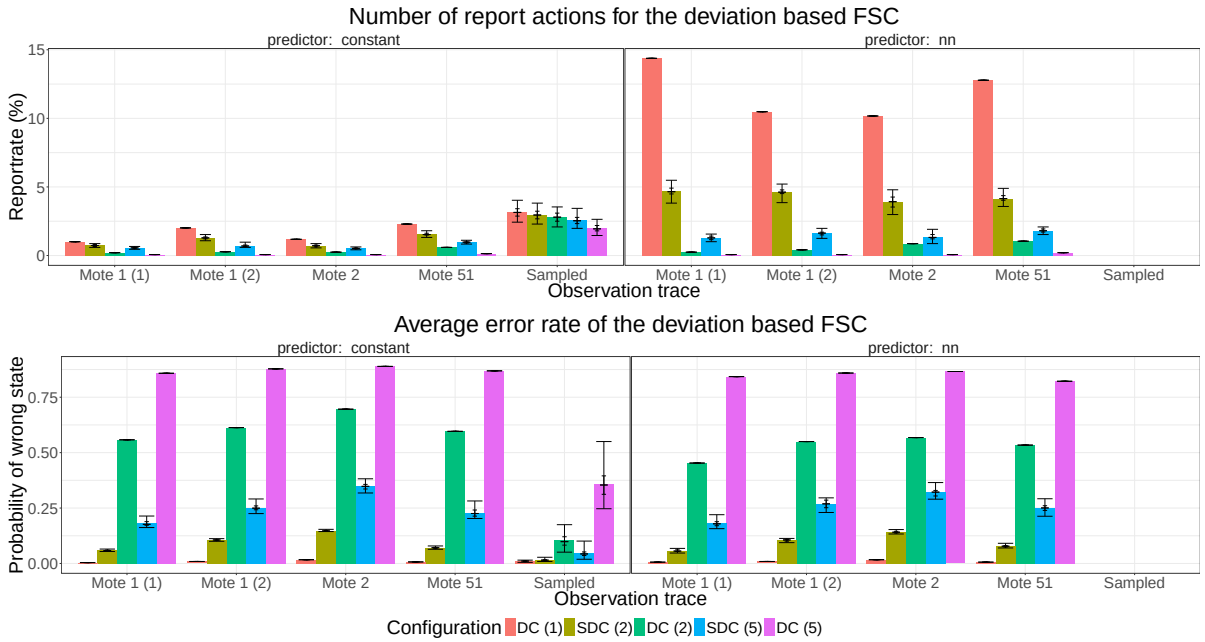


Figure 7.8: Performance of the deviation based controllers that for different data sets and different predictors. The report rate is expressed as the percentage of actions that are “report” actions. The average error is expressed as the average probability assigned to incorrect states.

for the NN-predictor. We expect this to be the result of the fact that the NN-predictors are trained for a continuous temperature trace, but those predictions are now discretised. Meaning that arbitrarily small errors that cross a state are already sufficient ground to report the measurement. In contrast the constant predictor does not suffer from such problems as it does use the discretised state space for its predictions.

Second, the configurations shown here contradict the follow-up. Whereas for sampled data the SDC configuration with $D = 5$ seems to improve over the DC configuration with $D = 2$, this result is not found in the real-world data sets. The reason is fairly simple. In the real world data set changes are much more likely to be single-state changes (which are not reported by DC configurations with $D = 2$), whereas the sample data includes the random jumps we allow in our transitions (see Section 7.1). This results in improvements in both report rate and accuracy in the sampled data, but also explains the far smaller report rate for real-world data. As a result it is also hard to find a DC and SDC configuration with similar report rates for the real-world data. Based on the results presented here, we reject the follow-up. However, the addition of stochastic actions does allow for much more control over the trade-off between accuracy and report rate (notice the difference between (S)DC-configuration $D = 1$ and the DC-configuration with $D = 2$ and compare that with the difference between (S)DC-configuration $D = 1$ and the SDC-configuration with $D = 2$).

7.5. Optimised controllers from the POMDP model

The final configuration of controllers that we test are the POMDP-based controllers. Using these controllers we evaluate the last two hypotheses, hypotheses 5 and 6:

Hypothesis 5 (Pareto optimal POMDP-based controllers)**.** *The POMDP-based controllers form a set of Pareto-optimal controllers for a set of different α values.*

Hypothesis 6 (POMDP-based controllers dominate)**.** *The POMDP-based controllers dominate the handmade controllers in terms of both report rate and accuracy.*

We first describe how the POMDP-based controllers are created and then evaluate the hypotheses using these controllers.

7.5.1. POMDP-based controllers

In the design as outlined in Section 4.3 we describe the three dimensions of the state and observation space. Since both $|S|$ and $|\Omega|$ determine the run-time of the POMDP-solver that we use, we want to keep them relatively small. Moreover the size of N and Ω are of importance for the time and space-complexity of the algorithm used in the sensor mote. Assuming a one degree Celsius temperature discretisation, we already require 441 states for just the temperature and prediction states. Assuming we have a temperature transition model for every half an hour, that makes another 48 time states, resulting in $|S| = |O| = 21168$ states in total.

Since this number of states is far too large to be tractable for the solver, we instead choose to factor out the time state. As a result we instead create 48 different POMDPs, one for each half hour of the day. Each POMDP uses a temperature discretisation of one degree Celsius and thus $|S| = 441$ (21 temperatures states, 21 prediction states and one time state). Each FSC has a limit to the number the number of nodes that it is allowed to have, namely five, ten, or twenty. To allow the BPI algorithm (see Algorithm 4) to add nodes for the purpose of escaping, the FSC starts with only half of the maximum allowed nodes. Thus when we report the results on a controller of five nodes, you should read that to mean a controller of two to five nodes (depending on how many escapes the BPI algorithm performed). As solving a POMDP to an FSC starting with twenty nodes (and allowing it to grow to 40) takes over 30 minutes in a parallelised implementation running on a high-end desktop computer, solving 48 of them takes over one day. As a result we have limited the number of α-values tested in this thesis. We focus on providing an impression of what the POMDP-based controllers can achieve, rather than finding the optimal α-value for our use case.

For the evaluation presented in this section, we have again presented the same input to the controllers multiple times. For the sampled data we have generated over 100 input traces and evaluated each of them ten times. For the real-world traces, we have evaluated each trace 40 times.

A final important note is that in contrast to results presented above, we consider only the constant predictor in the evaluation of the POMDP-based controllers. The reasoning for this is simple, we need to encode the way the predictor changes in the POMDP model. Since the neural network predictors are essentially a black box that can change in a myriad of ways when updated, it is impossible to accurately encode the prediction mechanism in a transition function for the prediction state space. The constant predictor however can easily be encoded in such a a way and is therefore used in this evaluation.

7.5.2. Evaluation: Pareto-optimal solutions

First we compare the POMDP-based controllers with each other to evaluate Hypothesis 5. Since α determines the penalty for reporting, we hypothesised that an α of zero would lead to a very high report rate and an α of one would lead to a very low accuracy. As Figure 7.9 shows both claims are indeed accurate. For $\alpha = 0$ almost all observations are reported. In fact most of the FSC with only two nodes feature fully deterministic action selections which always select the report action. For the FSC with a maximum of ten nodes, there are several which differ from this choice. Inspection of the controllers shows that the controllers for time period during which we expect very little fluctuation in temperature are to blame for this. There are some nodes in those FSC that have a non-zero probability of waiting, because those nodes are most likely reached when a matching temperature and prediction are observed.

For $\alpha = 1$ the accuracy is indeed much worse than for $\alpha = 0$. However, varying α does not form the Pareto front we had expected it to form. It seems for instance that $\alpha = 0.8$ is dominated by $\alpha = 0.2$ as it features both a better (average) report rate and a better accuracy. This is unexpected, since we expect an increasing α parameter to lead to a lower report rate rate, and thus also a lower accuracy. After all, the potential penalty for sending the wrong data becomes increasingly larger. This curious finding leads us to formulate this follow-up hypothesis:

Follow-up hypothesis 7.5.1 (Optimality of the controllers)**.** *The controllers found by the solving method are sub optimal, even more so for larger α-values.*

To evaluate this follow-up we compare the controllers with the solution found for the POMDP with the APPL Sarsop solver. This solver finds approximate solutions to POMDPs, returning a lower and upper bound of the expected reward. Our value function of a node in the FSC also represents this expected reward when executing the FSC starting from this node. Thus we have compared the expected reward from the FSC with the expected reward from Sarsop (computed as the average of the lower and upper

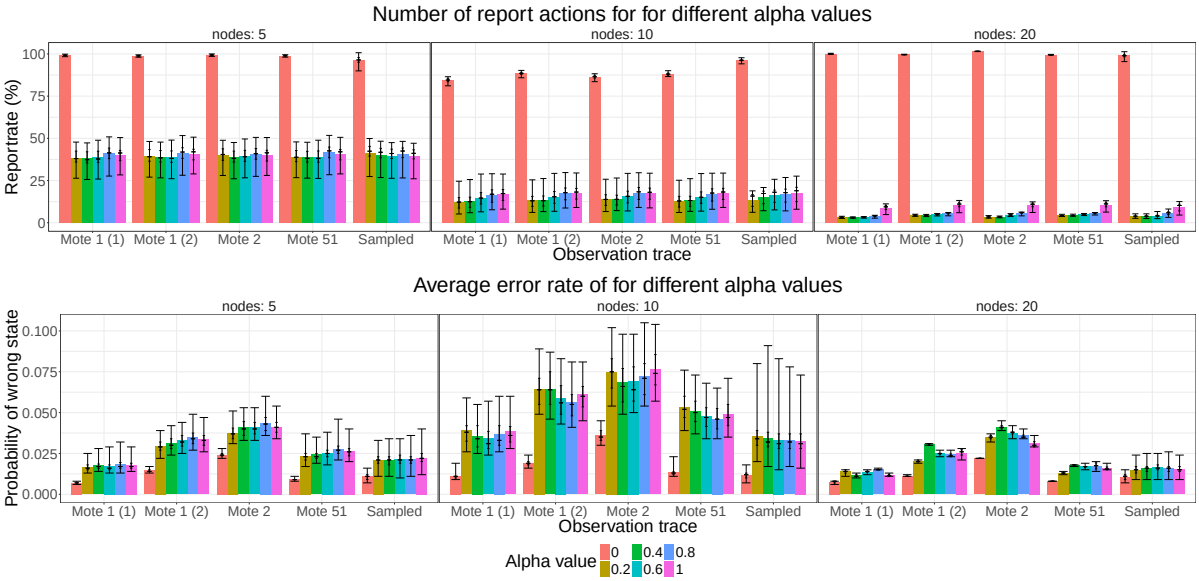


Figure 7.9: The effect of the α parameter on POMDP-based FSC performance. The scores represent average scores over 50 evaluations of the FSCs, with the error bars indicating minimum and maximum scores.

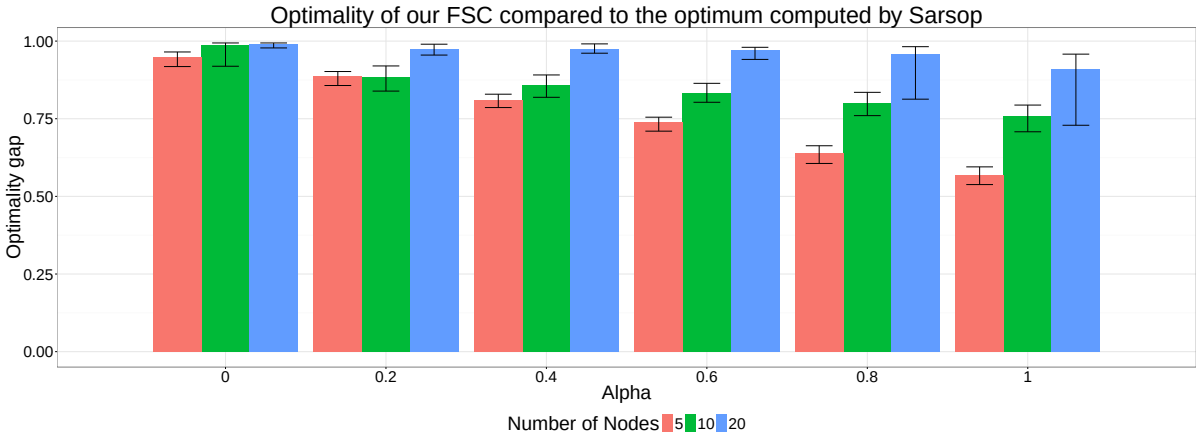


Figure 7.10: The optimality of our FSC solutions vs the solutions computed by Sarsop.

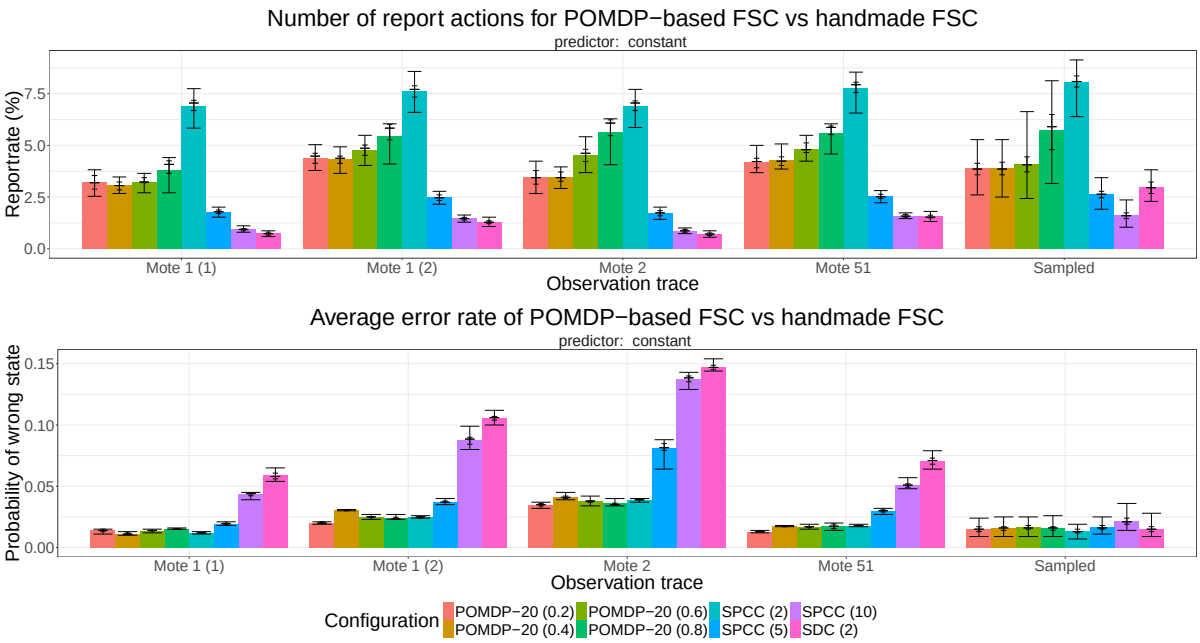


Figure 7.11: The performance of the POMDP-based FSCs versus the handmade FSCs. The scores represent average scores over 50 evaluations of the FSCs, with the error bars indicating minimum and maximum scores.

bound when running Sarsop for a precision of 0.001). The results are shown in Figure 7.10. The optimality gap depicted here, is the relative gap computed by dividing the score of the FSC by the optimal score found by Sarsop. The error bars indicate the minimum and maximum score of the 48 FSC in each category.

The conclusion is obvious. The FSC can not accurately represent the optimal solution to our POMDP in at most 10 nodes. As α increases, the optimality gap increases significantly. A larger number of nodes can help to combat this problem to some extent, but even starting with ten nodes and allowing ten nodes to be added does insufficient to alleviate this shortcoming, as some FSC still score below 75% of the optimum.

Regarding the hypothesis we can only conclude that the BPI method we have used to solve the POMDP to a FSC is not able to sufficiently encode an optimal solution in a small FSC. Other methods, such as the improved method introduced by Grzes et al. [33] or the alternative by Kumar et al. [51], might be able to improve on these results, but with this method we have to reject the hypothesis that varying the α parameter leads to Pareto front of scores.

7.5.3. Evaluation: POMDP-based controllers dominate handmade controllers

The final hypothesis remaining for evaluation is Hypothesis 6, in which we speculate that a controller based on an a commonly used model for this specific optimisation problem can outperform the FSC built on simple assumptions. Our investigation for the previous hypothesis has already cast a little doubt on this hypothesis, as results from the POMDP-based FSCs behaved differently than we initially expected. In Figure 7.11 we compare the most promising of the POMDP-based FSCs with the handmade FSCs.

The results are fairly positive. Even though some of the FSC score fairly badly in terms of optimality, the worst-case report rates improve significantly over the SPCC configuration with $C = 2$ and the worst-case accuracy improves significantly over the SCC configuration with $C = 10$. For some data sets the POMDP-based controllers dominate the handmade variants. Unfortunately the results are not sufficiently decisive to confirm the hypothesis.

The POMDP-based controllers certainly show potential, however with the current solving method and the limited controller sizes it is not possible to consistently outperform the handmade controllers. As a result from a systematic approach to finding controllers for a specific use case, the POMDP-based controllers do offer good performance that is as least as good as the handmade controllers that feature some requirements on the environment (such as observations being measurable on an interval scale).

7.6. Discussion: shortcomings and extensions

The evaluation performed in this chapter clearly shows that the choice of FSC used has a significant impact on the final output (the probability log) of the base station. Although we have shown that a POMDP-based FSC can compete with the handmade FSC based on simple principles, there are also some other (surprising) artefacts of this evaluation worth discussing. In this section we discuss several of the overarching (unmentioned) results, as well as some shortcomings and possible extensions to this methodology.

A comparison of different temperature discretisation is non-trivial.

Depending on the use case a different temperature discretisation may be required. Unfortunately comparing different temperature discretisations is non-trivial. In terms of computational effort and memory requirements, different discretisations can easily be compared. It changes the size of S and O and therefore has impact on both the run time and memory required by the methodology. Due to the nature of the metrics we defined however, it is hard to compare different discretisations in terms of accuracy. A finer discretisation (with smaller temperature states), is likely to lead to more reports, but since the accuracy metric compares states, the accuracy may not necessarily improve. Thus a more coarse discretisation is likely to outscore a finer discretisation when we apply our metrics.

To determine what discretisation works best, one should use a variation of the accuracy metric we propose. Say that for our use case we need a precision of β degrees Celsius. Then for a discretisation of β degrees, we can just use the metric proposed here. For a discretisation of β/x (assuming the same start point of the discretised range), x states would qualify as "correct" in the comparison used in the metric. This would correct the accuracy metric, but is still unlikely for a finer discretisation to dominate a coarser one. After all although the accuracy is now likely to improve, the report rate will still suffer. Ultimately the discretisation is just another parameter to tweak in balancing this trade-off, not only impacting the accuracy and report rate, but also the computational effort and memory required to create and use the controller.

The worse performance for NN compared to the constant predictor

Whereas the constant predictor also outperforms the neural network predictor in Chapter 6 for some variations of the relevant parameters, the difference seems to have increased when compared to the experiments shown in Chapter 6. We suspect this is due to the fact that the NN have not been trained to output a temperature state, but instead to output a real temperature. Whereas we still feed the actual temperature to the NN and convert its output to a temperature state, many outputs may be around a state border. Especially for temperature discretisations of 0.25 degrees Celsius, the error may be small in degrees Celsius, but still cross a temperature state border. Training the NN to instead output temperature states could help alleviate this problem as this produces better trained NNs that focus on the finite output range.

Using a more realistic observation function

To perform our evaluations we require an "absolute truth" to compare our produced logs against. As a result we have assumed the observations made by the sensor node to be (relatively) perfect. For the observation function O however we can also base the probability distribution on the specifications of the temperature sensor of the Mica weatherboard which were used to record the Intel lab data. According to the specifications, these weatherboards use the SHT11 sensor module [66]. These sensor modules can accurately measure temperature within a range of 0.5°C (at 25°C). Thus an observation function should reflect this. However as the temperatures we use to evaluate our methods against are recorded by this same hardware, it is illogical to have an observation function reflect this (thus claiming that the observations are not perfect) and then penalise the system if the observations are not believed (thus claiming the observations are perfect). As a result we have not used a modified observation function in this evaluation (instead we simply allow for a uniform error with a probability of one percent), but if an "absolute truth" is available for comparison, the limitations of the sensor should not be overlooked.

8

Conclusion & Future Work

As one of the main challenges in wireless sensor networks, the efficient use of the limited battery has already sparked many different types of research. The need for an accurate environmental state estimation at the base station requires the sensor motes to communicate their measurements, but communicating such a measurement is a very expensive operation in terms of battery. This trade-off between battery life and accuracy is also the focus of this thesis, in which we describe an application-level methodology that uses data prediction techniques to reduce the number of measurements that has to be communicated. It allows the base station and sensor motes to work together in producing an environmental state estimation that field specialists can use to study the environment.

The sensor mote uses a controller that selects what actions to perform based on the taken measurements of the environmental state. This controller is created through a systematic approach that takes the behaviour of the environment as well as the requirements of the field specialists into account. Given the reports of the sensor mote the base station can then apply its logic to find an accurate environmental state estimation.

In the rest of this chapter we highlight the main contributions of this thesis work and present an outlook on future directions this research can take.

8.1. Contributions

The main contributions of this work are threefold:

An application of Bayesian probability resulting in base station logic that obtains a much more accurate log than solely transition-based methods.
Inspired by techniques often associated with Hidden Markov Models, we design base station logic that uses Bayesian probability operations to obtain an accurate environmental state estimation from the limited information reported by the sensor motes. This logic requires the sensor motes to execute simple finite state controllers that react to the observations made to determine what measurements are worth reporting. Due to the abstract nature of the logic, any controller can be used to determine the behaviour of the sensor mote. The controllers can even take other things than just the measurements as input. In this thesis for instance, we explore several controllers that (do not) use prediction mechanisms during their execution. Even simple controllers that are not specifically made for the use case have been shown to improve the accuracy of the system by over 30% given the same limited subset of measurements provided to a transition-only approach. This contribution answers research question 1, by outlining base station logic that derives an accurate environmental state estimation from a subset of measurements.

An analysis of existing prediction mechanisms, in the form of neural networks, that provides shared knowledge between the base station and a sensor mote without requiring communication.
Based on previous research [26, 28, 68, 74] we investigate neural networks as a potential predictor of the time-dependent period data traces we are interested in. Their methodology is infeasible in the

domain of sensor motes however, due to their limited memory and processing capabilities. Our analysis investigates four other configurations of neural networks, of which one can be directly applied in the context of sensor networks. Using the notion of a communicated measurement, it can predict how the environment changes over time. By allowing the neural network to update after deployment, we drastically reduce the amount of required communication. This answers research question 2 as with at most 10% of the measurements reported, we can bound the average error to 0.05 degrees Celsius. This error is sufficiently small to be acceptable, as hardware errors often result in larger deviations.

An evaluation of a systematic approach for obtaining controllers that balance the trade-off, showing good accuracy even when less than 10% of the measurements are reported.
In addition to an extensive evaluation of handmade POMDPs that aim to balance the trade-off using intuitive rules of thumb in their design, we also present a systematic approach to find performant controllers. To this end we first model the environmental monitoring problem as a Partially Observable Markov Decision Process (POMDP), which we then solve to a finite state controller using the BPI-algorithm [5]. We can indirectly control the trade-off between report rate and accuracy through the manipulation of the reward function. Our evaluation shows that the controllers from this POMDP-based approach show great potential in improving over the handmade controllers, allowing good control over the trade-off and dominating some of the configurations already. Other methods to solve the POMDP to a controller and more nodes in the controller might help to further boost the performance. This answers RQ3, as we show that POMDP-based controllers can perform on-par with the handmade controllers, but in their current state not necessarily outperform them.

Through these contributions we answer the three sub research questions we posed. As for our main research question, we have shown how intelligent sensor motes can provide a small subset of the measurements (even smaller than 10%) chosen in such a way that the base station can estimate the environmental state with an accuracy of over 95%.

8.2. Future Work

The results outlined in this thesis also raises new questions that should be addressed in future research. Section 7.6 already mentions several shortcomings and possible extensions that arise from the evaluation presented in that chapter. In this section we describe several other extensions to the methodology outlined in this thesis that could be considered to further improve the performance of this methodology.

Multiple observations and multiple sensors.
Our current methodology resolves around using only the reports of a single sensor mote to reason about the environmental state. In some environmental contexts however, such as the study of redwood trees in California, a strong correlation between measurements of different sensor nodes could be established [86]. As these sensors were placed at different elevation levels within the tree, it could be observed that high temperature fronts move downwards in the tree over time. Correlations like this would allow for sensors to be notified that a front is incoming. Thus allowing us to take a more proactive approach to changes in the environment, for instance by increasing measurement frequency, rather than only reacting after the changes have happened. Using this information we can perhaps focus our battery usage to the time periods that we deem to be “interesting”.

The base station also has access to other information that can be related. For instance the combination of communication at time t with previous communication at time $t - X$ can lead to better insights about periods further before time $t - X$. Furthermore as many sensor motes report to the base station, all observing the same or related environmental states, we can also combine their information to get a more accurate probability distribution as to what the environmental state is like.

Consider for instance a use case wherein we can monitor both the amount of sunlight and the temperature in a room. Both can tell us something about how the temperature is likely to change over the next period of time. If it is sunny outside, then the temperature is likely to rise more than when it is cloudy. Both of the observations relate to the same environmental state in a slightly different way and can thus be combined in computing an estimate of the environmental state.

Reacting to observations through actions in the base station.
Depending on the application, the base station may be more than a passive processor of information. In this thesis we have not considered any actions that can be taken by the base station, but this is another expansion that can be considered for many use cases. Consider for instance a simple expansion of our Intel labs use case, where the base station can control the heating. It may then want to turn on the heater when the temperature is lower than expected and vice versa.

For these kinds of applications it is important that the environmental state estimation is efficiently computable. Currently the implementation only performs calculations for every report action or when the end of the trace is reached. The multi-threaded version of this code that performs many of the computations in parallel, using a high-end desktop computer with four cores, evaluates 2880 actions (report or sleep) in approximately one hour of run time (for the smallest temperature discretisation used in this thesis). If computations are to be done in every time step, the required run time increases significantly. So if the base station needs to decide on taking an action every thirty seconds, the base station either needs sufficiently processing power to be able to do this, or offload its work. Further optimisations in the implementation of the base station logic are required to ensure this can be done in the thirty second time window.

Adaptive finite state controller.
We have already seen in Section 5.5 that allowing the predictor to update itself during deployment can result in much better performance. Similar improvements could be obtained by updating the finite state controller if our model of the environment is updated after receiving several measurements. By changing the transition/observation functions based on received observations and solving the updated POMDP model to a controller, we might be able to achieve better performance for data not properly matching the training data. As such complex computations can not be done on the sensor mote, this would involve sending a new FSC to the sensor mote. Such an operation is expensive in terms of communication, which means it is another operation that should be considered in balancing the trade-off.

Using a continuous temperature space.
The methodology proposed in this thesis requires the state space under observation to be discretised. There are use cases that naturally consist of such a discretised state space. Take for example the Great Duck Island use case. In this case it makes sense to describe the nest as “empty” or containing a “duck”, there is no need for a continuous state space. However for our Intel office labs, it makes much more sense to model the temperature using a continuous variable. Whereas techniques exist to solve continuous state space POMDPs [17, 70], the methodology introduced in this thesis will have to be adapted to handle the use of continuous state spaces and functions.

A

Mathematical derivations of base station logic

In this Appendix we give mathematical derivations for the posterior probabilities on the environmental state, given first in terms of the transition function T, the observation function O and the initial belief b_0. After which we also include T_{fsc}, and an initial node m in the derivations. Both derivations also feature a parallelised and efficient implementation of the mathematics needed to compute the posterior probability distribution.

During this analysis we apply several rules from Bayesian probability. First of all, Bayes' rule which allows us to invert a dependency shown in equation A.1. Secondly, a derivative of Bayes rule that allows us to transform a variable into a dependency shown in equation A.2. Thirdly, another derivative of Bayes rule that allows us to express a conditional probability as a joint probability instead, shown in equation A.3. Finally we use the law of total probability, which allows us to introduce a dependency as shown in equation A.4.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{A.1}$$

$$P(A,B|C) = P(A|B,C)P(B|C) \tag{A.2}$$

$$P(A|B) = \frac{P(A,B)}{P(B)} \tag{A.3}$$

$$P(A) = \sum_{B\in\Omega} P(A|B)P(B) \tag{A.4}$$

A.1. Transition model only

For our analysis of the desired probability we start by applying Bayes' rule (equation A.1).

$$P(s_j|o_t) = \frac{P(o_t|s_j)P(s_j)}{P(o_t)} \tag{A.5}$$

This rewritten function contains three new probabilities to examine. First we consider the two unconditional probabilities and apply the law of total probability (equation A.4) to both. For the unconditioned probability of s_j get the following recursive formulation:

$$P(s_j) = \begin{cases} b_0(s_j) & j = 0 \\ \sum_{s_{j-1}\in S} P(s_j|s_{j-1})P(s_{j-1}) & \text{else} \end{cases} \tag{A.6}$$

As for $P(o_j)$, this unconditioned probability can also be expressed in terms of the initial belief, by applying the law of total probability when conditioning on s_j.

$$P(o_j) = \sum_{s_j \in S} P(o_j|s_j)P(s_j) \tag{A.7}$$

Notice that $P(o_j|s_j)$ and $P(s_j|s_{j-1})$ are the observation function O and the transition function T respectively. This leaves just the conditional probability from equation A.5 which is just $O(o_j|s_j)$ when $j = t$, so we split it in two cases, applying equation A.4 to the recursive case.

$$P(o_t|s_j) = \begin{cases} O(o_t|s_j) & \text{if } t = j \\ \sum\limits_{s_{j+1}} P(o_t, s_{j+1}|s_j) & \text{else} \end{cases} \tag{A.8}$$

We consider this else case in a bit more detail. As o_t depends only on s_t which is dependent only on s_{t-1}, which is dependent only on s_{t-2} etc. we can state that $P(o_t|s_{j+1}, s_j) = P(o_t|s_{j+1})$, i.e. when s_{j+1} is given, o_t is independent of s_j. Thus our first step is to change s_{j+1} into a dependency using equation A.2.

$$\begin{aligned} P(o_t, s_{j+1}|s_j) &= P(o_t|s_{j+1}, s_j)P(s_{j+1}|s_j) \\ &= P(o_t|s_{j+1})P(s_{j+1}|s_j) \end{aligned}$$

Now we have once again reached a situation where we can easily compute all remaining terms. The first conditional probability is a recursive call to equation A.8. The second term is the transition probability $T(s_{j+1}|s_j)$. Putting all of this together, we get to Algorithm 8 to be executed by the base station. Once we get a reported observation we apply the equations described above to find the probability distributions over the state at the times during which we did not receive an observation.

A.2. Including the FSC

Rewriting this into a form using only the prior knowledge is slightly more cumbersome. To illustrate exactly how the equations can be applied, we again apply them to the toy example in the next section. We first apply equation A.1 to rewrite the dependency:

$$P(s_j|o_t, n_t) = \frac{P(s_j, o_t, n_t)}{P(o_t, n_t)} \tag{A.9}$$

The denominator of this equation can quite easily be rewritten to a sum over terms similar to the numerator with the law of total probability:

$$P(o_t, n_t) = \sum_{s_t \in S} P(o_t, n_t, s_t) \tag{A.10}$$

Now we turn to the numerator for which we consider two cases: $j = t$ and $j < t$ as they each involve different manners in which we condition the probability. First let us consider the case $j = t$. We use the law of total probability, conditioning on all the direct dependencies of s_t and n_t, namely s_{t-1} and n_{t-1}:

$$P(s_t, o_t, n_t) = \sum_{s_{t-1} \in S} \sum_{n_{t-1} \in N} P(s_t, o_t, n_t|n_{t-1}, s_{t-1}) \cdot P(n_{t-1}, s_{t-1}) \tag{A.11}$$

We further split this result and consider each term in turn. For the first term we turn s_t and o_t into conditions, starting with s_t:

$$P(s_t, o_t, n_t|n_{t-1}, s_{t-1}) = P(o_t, n_t|n_{t-1}, s_t, s_{t-1}) \cdot P(s_t|n_{t-1}, s_{t-1})$$

Observe now that s_t does not depend on n_{t-1}, when s_{t-1} is given. Similarly the given s_{t-1} is redundant in the first term, as s_t encapsulates this dependency:

$$= P(o_t, n_t|n_{t-1}, s_t)P(s_t|s_{t-1})$$

Algorithm 8 A policy for the base station that computes the probabilities for all uncommunicated time steps based on the received observation.

function onReceivedMessage(observation)
    computeStateProbabilities() ▷*Fills memSj*
    normaliser ← $\sum_{s \in S} O(\text{observation}|s) \cdot \text{memSj}[t][s]$ ▷*Equation A.7*

    mem ← 1D-array of size $|S|$
    memNew ← 1D-array of size $|S|$
    mem[s] ← $O(\text{observation}|s) \quad \forall s \in |S|$
    for $j = t \rightarrow t - X$ **do**
        parfor $s \in S$ **do**
            prob← $\frac{\text{mem}[s] \cdot \text{memSj}[j][s]}{\text{normaliser}}$ ▷*Equation A.5*
            log(j,s,prob)
            memNew$[s] \leftarrow \sum_{s' \in S} T(s'|s) \cdot \text{mem}[s']$ ▷*Equation A.8*
        end parfor
        mem ← memNew
    end for
end function

function computeStateProbabilities()
    memSj ← 2D-array of size $X \times |S|$
    memSj$[0][s] \leftarrow b_0(s) \quad \forall s \in |S|$
    for $t = 1 \rightarrow X$ **do**
        parfor $s \in S$ **do**
            memSj$[t][s] \leftarrow \sum_{s' \in S} \text{memSj}[t-1][s'] \cdot T(s|s')$ ▷*Equation A.6*
        end parfor
    end for
end function

Now we turn o_t in a dependency as well:

$$= P(n_t|n_{t-1}, o_t, s_t)P(o_t|s_t, n_{t-1})P(s_t|s_{t-1})$$

Again we can remove some redundant conditions, which are already redundant due to the other conditions to get:

$$\begin{aligned} &= P(n_t|n_{t-1}, o_t)P(o_t|s_t)P(s_t|s_{t-1}) \\ &= T_{\text{fsc}}(n_t|n_{t-1}, o_t)O(o_t|s_t)T(s_t|s_{t-1}) \end{aligned} \tag{A.12}$$

Now that we have completely rewritten the first term of equation A.11 to a function expressed in the known observation and transition functions, we turn our attention to the second term of equation A.11. This second term can be expressed as a recursive call with a smaller index:

$$P(n_{t-1}, s_{t-1}) = \begin{cases} 0 & \text{if } t-1=0 \text{ and } n_0 \neq m \\ b_0(s_0) & \text{if } t-1=0 \text{ and } n_0 = m \\ \sum\limits_{o_{t-1}\in\Omega} P(s_{t-1}, n_{t-1}, o_{t-1}) & \text{else} \end{cases} \tag{A.13}$$

With the case of $j = t$ done, we now turn our attention to the cases where $j < t$, in other words were we want to apply this new knowledge we have to learn more about the past, rather than about the present moment. We work towards a function that recursively uses s_{j+1} and n_j as this becomes s_t and n_{t-1} in the base case which allows for direct computation.

$$P(s_j, o_t, n_t) = \sum_{s_{j+1}\in S} \sum_{n_j \in N} P(s_j, o_t, n_t, n_j, s_{j+1}) \tag{A.14}$$

Consider now this probability of the five joined variables, where we turn three variables s_j, s_{j+1}, and n_j in to conditions:

$$\begin{aligned} P(s_j, o_t, n_t, n_j, s_{j+1}) &= P(o_t, n_t, n_j, s_{j+1}|s_j)P(s_j) \\ &= P(o_t, n_t|n_j, s_j, s_{j+1})P(n_j, s_{j+1}|s_j)P(s_j) \end{aligned}$$

We trim the condition on s_j in the first term as s_{j+1} and n_j capture all dependencies of o_t and n_t imposed by s_j:

$$= P(o_t, n_t|n_j, s_{j+1})P(n_j, s_{j+1}|s_j)P(s_j) \tag{A.15}$$

We now have three terms left. The last term $P(s_j)$ can already be found using equation A.6. The first and second are described here, starting with the first:

$$P(o_t, n_t|n_j, s_{j+1}) = \begin{cases} T_{\text{fsc}}(n_t|n_j, o_t)O(o_t|s_t) & \text{if } j = t-1 \\ \sum\limits_{s_{j+2}\in S} \sum\limits_{n_{j+1}\in N} P(o_t, n_t, s_{j+2}, n_{j+1}|s_{j+1}, n_j) & \text{else} \end{cases} \tag{A.16}$$

We consider the else-case of this equation in more detail, turning n_{j+1} and s_{j+2} into dependencies to get to a recursive call of this equation.

$$\begin{aligned} P(o_t, n_t, s_{j+2}, n_{j+1}|s_{j+1}, n_j) &= P(o_t, n_t, s_{j+2}|n_{j+1}, s_{j+1}, n_j)P(n_{j+1}|s_{j+1}, n_j) \\ &= P(o_t, n_t|s_{j+2}, n_{j+1}, s_{j+1}, n_j)P(s_{j+2}|n_{j+1}, s_{j+1}, n_j)P(n_{j+1}|s_{j+1}, n_j) \end{aligned}$$

We can again drop a condition on s_{j+1} and on n_j that are now obsolete in the first term, as s_{j+2} and n_{j+1} encapsulate the imposed dependencies. Similarly we can drop n_{j+1} and n_j as conditions on s_{j+2}:

$$= P(o_t, n_t|s_{j+2}, n_{j+1})P(s_{j+2}|s_{j+1})P(n_{j+1}|s_{j+1}, n_j)$$

Now we rewrite to get:

$$= P(o_t, n_t|n_{j+1}, s_{j+2})T(s_{j+2}|s_{j+1})P(n_{j+1}|s_{j+1}, n_j) \tag{A.17}$$

The first term is the recursive call to equation A.16, the second is the transition function, so that leaves the third. With the law of total probability we get:

$$P(n_{j+1}|s_{j+1}, n_j) = \sum_{o_{j+1}\in\Omega} P(n_{j+1}, o_{j+1}|s_{j+1}, n_j)$$
$$= \sum_{o_{j+1}\in\Omega} P(n_{j+1}|o_{j+1}, s_{j+1}, n_j)P(o_{j+1}|s_{j+1}, n_j)$$

Again we drop redundant dependencies to get:

$$= \sum_{o_{j+1}\in\Omega} P(n_{j+1}|o_{j+1}, n_j)P(o_{j+1}|s_{j+1})$$
$$= \sum_{o_{j+1}\in\Omega} T_{\text{fsc}}(n_{j+1}|o_{j+1}, n_j)O(o_{j+1}|s_{j+1}) \quad \text{(A.18)}$$

And with that we have rewritten the first term of equation A.15, which leaves the second term of that equation:

$$P(n_j, s_{j+1}|s_j) = P(n_j|s_{j+1}, s_j)P(s_{j+1}|s_j)$$

Removing the redundant dependencies we get:

$$= P(n_j|s_j)T(s_{j+1}|s_j) \quad \text{(A.19)}$$

We now look at the first term of that in more detail, applying equation A.3 to get:

$$P(n_j|s_j) = \frac{P(n_j, s_j)}{P(s_j)}$$

Both of these probabilities are already defined, by equations A.6 and A.13. With that we have rewritten the second term of equation A.15, which means that we have expressions that can compute $P(s_j|o_t, n_t)$ for all $t - X \leq j \leq t$. Algorithm 9 gives a parallelised and efficient implementation that computes the posterior probability distribution over the environmental states using these equations.

Algorithm 9 A policy for the base station that computes the probabilities for all uncommunicated time steps based on the received observation and node in the controller.

function onReceivedMessage(observation, node)
  computeStateNodeProbabilities()
  computeObsNodeGivenNodeStateProbabilities(observation, node)

  normaliser $\leftarrow \sum_{s\in S}\sum_{n'\in N}\sum_{s'\in S}$ memNS$[t-1][s'][n'] \cdot T(s|s') \cdot O($observation$|s) \cdot T_{\text{fsc}}($node$|n', o)$
  ▷*Equation A.11*
  for $j = t \rightarrow t - X$ **do**
    parfor $s \in S$ **do**
      prob$\leftarrow \frac{\sum_{n'\in N}\sum_{s'\in S} \text{memON}[t][s'][n']\cdot\text{memNS}[t][s'][n']\cdot T(s'|s)}{\text{normaliser}}$
      ▷*Equation A.9*
      log(j,s,prob)
    end parfor
  end for
end function

function computeStateNodeProbabilities()
  memNS ← 3D-array of size $X \times |S| \times |N|$
  memNS$[0][s][n] \leftarrow 0 \quad \forall n \in N, n \neq m$ ▷m *denotes the initial node*
  memNS$[0][s][m] \leftarrow b_0(s) \quad \forall s \in S$
  for $t = 1 \rightarrow X$ **do**
    parfor $s \in S$ **do**
      parfor $n \in N$ **do**
        memNS$[t][s][n] \leftarrow \sum_{o\in O}\sum_{n'\in N}\sum_{s'\in S}$ memNS$[t-1][s'][n'] \cdot T(s|s') \cdot O(o|s) \cdot T_{\text{fsc}}(n|n', o)$
        ▷*Equations A.11 and A.12*
      end parfor
    end parfor
  end for
end function

function computeObsNodeGivenNodeStateProbabilities(obs, node)
  memON ← 3D-array of size $X \times |S| \times |N|$
  memON$[X][s][n] \leftarrow T_{\text{fsc}}($node$|n,$ obs$) \cdot O($obs$|s) \quad \forall n \in N, s \in S$ ▷*Equation A.16*
  for $t = X - 1 \rightarrow 0$ **do**
    parfor $s \in S$ **do**
      parfor $n \in N$ **do**
        memON$[t][s][n] \leftarrow \sum_{o\in O}\sum_{n'\in N}\sum_{s'\in S}$ memON$[t+1][s'][n'] \cdot T(s'|s) \cdot T_{\text{fsc}}(n'|n, o) \cdot O(o|s)$
        ▷*Equations A.16, A.17 and A.18*
      end parfor
    end parfor
  end for
end function

Bibliography

[1] K. Akkaya and M. Younis. An energy-aware QoS routing protocol for wireless sensor networks. In *Proceedings. 23rd International Conference on Distributed Computing Systems Workshops*, pages 445–62, 2003. ISBN 0-7695-1921-0.

[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002. ISSN 13891286.

[3] J. N. Al-Karaki and A. E. Kamal. Routing Techniques in Wireless Sensor Networks: a Survey. *Ieee Wireless Communications*, 11(December):6–28, 2004. ISSN 15361284.

[4] H. Alemdar and C. Ersoy. Wireless sensor networks for healthcare: A survey. *Computer Networks*, 54(15):2688–2710, 2010. ISSN 13891286.

[5] C. Amato, D. S. Bernstein, and S. Zilberstein. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *Autonomous Agents and Multi-Agent Systems*, 21(3):293–320, 2010. ISSN 13872532.

[6] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537–568, 2009. ISSN 15708705.

[7] J. G. T. Anderson. Pilot survey of mid-coast Maine seabird colonies: an evaluation of techniques. Technical report, State of Maine Dept. of Inland Fisheries and Wildlife., Bangor, ME, 1995.

[8] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita. A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks*, 46(5):605–634, 2004. ISSN 13891286.

[9] T. Bäck. *Evolutionary Algorithms in Theory and Practice*, volume 2. Oxford University Press, jan 1996. ISBN 0195099710.

[10] R. Bellman. A Markovian decision process. *Journal Of Mathematics And Mechanics*, 6:679–684, 1957. ISSN 01650114.

[11] Y. Bengio and P. Frasconi. An Input Output HMM Architecture. *Advances in Neural Information Processing Systems*, pages 427–434, 1995. ISSN 15322092.

[12] D. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. *Workshop on Knowledge Knowledge Discovery in Databases*, 398:359–370, 1994.

[13] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The Complexity of Decentralized Control of Markov Decision Processes. *Mathematics of Operations Research*, 27(4):819–840, 2002. ISSN 0364-765X.

[14] D. S. Bernstein, N. Immerman, S. Zilberstein, and R. Givan. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002. ISSN 0364-765X.

[15] P. Bodik, W. Hong, C. Guestrin, S. Madden, M. Paskin, and R. Thibaux. Intel Lab Data, 2004. URL `http://db.csail.mit.edu/labdata/labdata.html`.

[16] M. Brand. Coupled hidden Markov models for modeling interacting processes. *Submitted to Neural Computation*, 405(405):1–28, 1996.

[17] A. Brooks, A. Makarenko, S. Williams, and H. Durrant-Whyte. Parametric POMDPs for planning in continuous state spaces. *Robotics and Autonomous Systems*, 54(11):887–897, 2006. ISSN 09218890.

[18] D. Bruckner, B. Sallans, and G. Russ. Hidden Markov models for traffic observation. *IEEE International Conference on Industrial Informatics (INDIN)*, 2:1015–1020, 2007. ISSN 19354576.

[19] A. E. Bryson and Y.-C. Ho. Applied Optimal Control: Optimization, Estimation, and Control. *IEEE Transactions on Systems, Man, and Cybernetics.*, 9(6):366–367, 1979.

[20] C. Bunks and D. Mccarthy. Condition-Based Maintenance of Machines Using Hidden Markov Models. *Mechanical Systems and Signal Processing*, 14(4):597–612, 2000. ISSN 08883270.

[21] M. Cattani, A. Loukas, M. Zimmerling, M. Zuniga, and K. Langendoen. Staffetta: Smart Duty-Cycling for Opportunistic Data Collection. *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM - SenSys '16*, pages 56–69, 2016.

[22] D. Chu, A. Deshpande, J. Hellerstein, and W. Hong. Approximate data collection in sensor networks using probabilistic models. *Proceedings of the 22nd International Conference on Data Engineering, 2006. ICDE'06.*, pages 48–48, 2006.

[23] G. Cybenko. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals, and Systems*, (2):303–314, 1989. ISSN 10009221.

[24] F. Ding, G. Song, K. Yin, J. Li, and A. Song. A GPS-enabled wireless sensor network for monitoring radioactive materials. *Sensors and Actuators, A: Physical*, 155(1):210–215, 2009. ISSN 09244247.

[25] A. Dunkels, J. Alonso, and T. Voigt. Making TCP / IP Viable for Wireless Sensor Networks. Technical report, 2004.

[26] N. Enami, R. A. Moghadam, K. Dadashtabar, and M. Hoseini. Neural Network Based Energy Efficiency in Wireless Sensor Networks: A Survey. *International Journal of Computer Science & Engineering Survey*, 1(1):39–55, 2010. ISSN 09763252.

[27] C. F. García-hernández, P. H. Ibargüengoytia-gonzález, J. García-hernández, and J. a. Pérez-díaz. Wireless Sensor Networks and Applications : a Survey. *Journal of Computer Science*, 7(3): 264–273, 2007. ISSN 00189162.

[28] M. S. Gashler and S. C. Ashmore. Modeling Time Series Data With Deep Fourier Neural Networks. *Neurocomputing*, pages 1–9, 2015. ISSN 09252312.

[29] F. Glover. Tabu Search Part I. *INFORMS Journal on Computing*, 1(5):190–206, 1989.

[30] F. Glover. Tabu Search Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990. ISSN 0899-1499.

[31] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, 1989. ISBN 0201157675.

[32] S. M. Goldfeld and R. E. Quandt. A Markov model for switching regressions. *Journal of Econometrics*, 1(1):3–15, 1973. ISSN 03044076.

[33] M. Grześ and P. Poupart. Incremental Policy Iteration with Guaranteed Escape from Local Optima in POMDP Planning. *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1249–1257, 2015. ISSN 15582914.

[34] M. Grześ, P. Poupart, and J. Hoey. Isomorph-free branch and bound search for finite state controllers. *IJCAI International Joint Conference on Artificial Intelligence*, pages 2282–2290, 2013. ISSN 10450823.

[35] I. Guyon. Applications of Neural Networks To Character Recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 5:353–382, jun 1991. ISSN 0218-0014.

[36] J. D. Hamilton. Analysis of time series subject to changes in regime. *Journal of Econometrics*, 45 (1):39–70, 1990.

[37] J. He, H. Li, and J. Tan. Real-time daily activity classification with wireless sensor networks using Hidden Markov Model. *Proceedings of the 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society.*, 2007:3192–5, 2007. ISSN 1557-170X.

[38] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1(4):660–670, 2002. ISSN 1536-1276.

[39] A. Hertz and D. Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4): 345–351, 1987. ISSN 0010-485X.

[40] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion. *Proceedings of the 6th annual international conference on Mobile computing and networking - MobiCom '00*, pages 56–67, 2000. ISSN 10636692.

[41] N. Itsuki. Hidden Markov Modeling for Multi-agent Systems. *PRICAI 2002: Trends in Artificial Intelligence*, pages 128–137, 2002.

[42] R. A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks, Elsevier*, 1(4):295–307, 1988.

[43] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking. *ACM SIGOPS Operating Systems Review*, 36(5):96, 2002. ISSN 01635980.

[44] S. Julier and J. Uhlmann. Unscented Filtering and Non Linear Estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004. ISSN 0018-9219.

[45] K. Kar, A. Krishnamurthy, and N. Jaggi. Dynamic node activation in networks of rechargeable sensors. *IEEE/ACM Transactions on Networking*, 14(1):15–26, 2006. ISSN 10636692.

[46] T. V. Kasteren and A. Noulas. Accurate activity recognition in a home setting. *UbiComp*, pages 1–9, 2008.

[47] K. K. Khedo, R. Perseedoss, A. Mungur, U. O. Mauritius, and Mauritius. A Wireless Sensor Network Air Pollution Monitoring System. *Science*, 2(2):15, 2010. ISSN 09754679.

[48] M. J. Kochenderfer, C. Amato, G. Chowdhary, J. P. How, H. J. D. Reynolds, J. R. Thornton, P. A. Torres-Carrasquillo, N. K. Üre, and J. Vian. *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015. ISBN 0262029251.

[49] M. Kubat. Neural networks: a comprehensive foundation. *The Knowledge Engineering Review*, 13(4):409–412, feb 1999. ISSN 02698889.

[50] A. Kumar and S. Zilberstein. History-Based Controller Design and Optimization for Partially Observable MDPs. *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 156–164, 2015. ISSN 23340843.

[51] A. Kumar, H. Mostafa, and S. Zilberstein. Dual Formulations for Optimizing Dec-POMDP Controllers. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, pages 202–210, 2016.

[52] H. Kurniawati, D. Hsu, and W. S. Lee. SARSOP : Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. *Proceedings of Robotics: Science and Systems*, 2008. ISSN 2330765X.

[53] J. Kwon and K. Murphy. Modeling Freeway Traffic with HMMs. 2000.

[54] S. H. Lee, S. Lee, H. Song, and H. S. Lee. Wireless sensor network design for tactical military applications : Remote large-scale environments. *MILCOM 2009 - 2009 IEEE Military Communications Conference*, 19:1–7, 2009. ISSN 0164-1212.

[55] J. Little. The Synchronization of Traffic Signals by Mixed-Integer Linear Programming. *Operations Research*, 14(4):568–594, 1966.

[56] M. Littman, T. Dean, and L. Kaelbling. On the complexity of solving Markov decision problems. *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 394–402, 1995. ISSN 1558603859.

[57] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 88–97, 2002. ISSN 00189162.

[58] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943. ISSN 00074985.

[59] K. P. Murphy. Dynamic Bayesian Networks: Representation, Inference and Learning. *Annals of Physics*, Ph. D.(November):225, 2002. ISSN 1098-6596.

[60] M. Negnevitsky. *Artificial Intelligence: A Guide to Intelligent Systems*. Addison-Wesley, 2005. ISBN 0321204662.

[61] R. Olfati-Saber. Distributed Kalman filtering for sensor networks. *Proceedings of the IEEE Conference on Decision and Control*, pages 5492–5498, 2007. ISSN 01912216.

[62] R. Olfati-Saber. Distributed Kalman filter with embedded consensus filters. *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference, CDC-ECC '05*, 2005:8179–8184, 2005.

[63] R. Olfati-Saber. Kalman-Consensus filter: Optimality, stability, and performance. *Proceedings of the IEEE Conference on Decision and Control*, pages 7036–7042, 2009. ISSN 01912216.

[64] F. A. Oliehoek, M. T. J. Spaan, and N. Vlassis. Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32:289–353, 2008. ISSN 10769757.

[65] S. C. W. Ong, S. W. Png, D. Hsu, and W. S. Lee. POMDPs for robotic tasks with mixed observability. *Proceedings of the Robotics: Science and Systems Conference (RSS)*, 2009. ISSN 2330765X.

[66] Parallax Inc. Sensirion SHT11 Sensor Module (# 28018) Precision Temperature and Humidity Measurement, 2003. URL `https://www.parallax.com/sites/default/files/downloads/28018-Sensirion-Temerature-Humidity-Sensor-Documentation-v1.0.pdf`.

[67] C. Park, J. Liu, and P. Chou. Eco: an ultra-compact low-power wireless sensor node for real-time motion monitoring. *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pages 398–403, 2005.

[68] I. Park and D. Mirikitani. Energy reduction in wireless sensor networks through measurement estimation with second order recurrent neural networks. *Third International Conference on Networking and Services, 2007. ICNS.*, pages 7–10, 2007.

[69] C. E. Perkins. *Ad Hoc Networking*. Addison-Wesley Professional, aug 2001. ISBN 0321579070, 9780321579072.

[70] J. M. Porta, N. Vlassis, M. T. J. Spaan, and P. Poupart. Point-Based Value Iteration for Continuous POMDPs. *Journal of Machine Learning Research*, 7:2329–2367, nov 2006.

[71] P. Poupart and C. Boutilier. Bounded finite state controllers. *Advances in Neural Information Processing Systems 16*, 2003. ISSN 10495258.

[72] M. Quwaider and S. Biswas. Body posture identification using hidden Markov model with a wearable sensor network. *BodyNets '08: Proceedings of the ICST 3rd international conference on Body area networks on Body area networks*, pages 1—-8, 2008.

[73] M. Roth, R. Simmons, and M. Veloso. What to communicate? Execution-time decision in multiagent POMDPs. *Distributed Autonomous Robotic Systems 7*, pages 177–186, 2006.

[74] Y. Shen and X. Li. Wavelet Neural Network Approach for Dynamic Power Management in Wireless Sensor Networks. In *2008 International Conference on Embedded Software and Systems*, pages 376–381. IEEE, 2008. ISBN 978-0-7695-3287-5.

[75] M. A. Shipp, K. N. Ross, P. Tamayo, A. P. Weng, J. L. Kutok, R. C. T. Aguiar, M. Gaasenbeek, M. Angelo, M. Reich, G. S. Pinkus, T. S. Ray, M. A. Koval, K. W. Last, A. Norton, T. A. Lister, J. Mesirov, D. S. Neuberg, E. S. Lander, J. C. Aster, and T. R. Golub. Diffuse large B-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning. *Nature medicine*, 8(1):68–74, 2002. ISSN 10788956.

[76] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993. ISSN 00043702.

[77] E. J. Sondik. The Optimal Control of Partially Observable Markov Processes. 1971.

[78] M. T. J. Spaan and F. A. Oliehoek. The MultiAgent Decision Process toolbox: Software for decision-theoretic planning in multiagent-systems. In *Proceedings of the Third AAMAS Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains (MSDM)*, pages 107–121, 2008. ISBN 9789064643262.

[79] M. T. J. Spaan and N. Vlassis. A point-based POMDP algorithm for robot planning. *Proceedings of the 2004 International Conference on Robotics & Automation*, pages 2399–2404, 2004.

[80] M. T. J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005. ISSN 10769757.

[81] M. T. J. Spaan, T. S. Veiga, and P. U. Lima. Decision-theoretic Planning under Uncertainty with Information Rewards for Active Cooperative Perception. *Autonomous Agents and Multi-Agent Systems*, 29(6):1157–1185, 2015.

[82] R. Steele, A. Lo, C. Secombe, and Y. K. Wong. Elderly persons' perception and acceptance of using wireless sensor networks to assist healthcare. *International Journal of Medical Informatics*, 78(12):788–801, 2009. ISSN 13865056.

[83] M. P. Steves. Utility Assessments of Soldier-Worn Sensor Systems for ASSIST. *Performance Metrics for Intelligent Systems Workshop*, pages 165–171, 2006.

[84] N. K. Suryadevara and S. C. Mukhopadhyay. Wireless sensor network based home monitoring system for wellness determination of elderly. *IEEE Sensors Journal*, 12(6):1965–1972, 2012. ISSN 1530437X.

[85] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An Analysis of a Large Scale Habitat Monitoring Application. *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys 2004)*, pages 214–226, 2004. ISSN 1-58113-879-2.

[86] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 51–63, 2005.

[87] J. E. Trowbridge and J. J. Mintzes. Alternative conceptions in animal classification: A cross-age study. *Journal of Research in Science Teaching*, 25(7):547–571, 1988. ISSN 1098-2736.

[88] Y. Wang, M. Martonosi, and L. S. Peh. Predicting link quality using supervised learning in wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 11(3):71–83, 2007. ISSN 15591662.

[89] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992. ISSN 0885-6125, 1573-0565.

[90] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, 1989.

[91] R. L. Watrous. Learning Algorithms for Connectionist Networks : Applied Gradient Methods of Non-Linear Optimization. *Technical Reports (CIS)*, (MS-CIS-87-51), 1988.

[92] M. Winkler, K.-d. Tuchs, K. Hughes, and G. Barclay. Theoretical and practical aspects of military wireless sensor networks. *Journal of Telecommunications and Information Technology*, 2:37–45, 2008. ISSN 1509-4553.

[93] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed Energy Conservation for Ad Hoc Routing. In *Proceedings of the Seventh ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM 2001)*, 2001. ISBN 1581137079.

[94] P. Zhang, C. M. Sadler, S. a. Lyon, and M. Martonosi. Hardware design experiences in ZebraNet. *Proceedings of the 2nd international conference on Embedded networked sensor systems - SenSys '04*, 7:227, 2004.