Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Sensor-Based Control of Water Samplers in WSNs

Master's Thesis

Daniel Burgener
burgdani@gmx.ch

Computer Engineering Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Dr. Jan Beutel, Dr. Philipp Schneider
Prof. Dr. Lothar Thiele

June 22, 2012

Abstract

Hydrologists use water samplers in order to research the linking process between catchment hydrology and stream water chemistry during storm events. Deciding when to trigger a series of samples, which is subsequently analyzed in the laboratory, is difficult with a single locally connected sensor. Due to the water sampler's stand-alone system design it is not possible to use multiple distributed sensors as a trigger criteria, synchronize the sampling of multiple water samplers, or to remotely query the sampler status.

To tackle these problems we explore in this thesis the possibilities to (i) integrate the water sampler into the existing PermaSense wireless sensor network, and (ii) to control the water sampler based on online sensor data. To this end, the end-to-end integration of a water sampler into the PermaSense wireless sensor network is detailed, which leverages on a feature rich sensor platform for rapid development and efficient debugging. A control algorithm is devised, evaluated and embedded into the backend of the PermaSense system where sensor data is used to control the water sampler.

The system is evaluated in a real-world wireless sensor network deployment at river Thur. The results show the successful integration of the water sampler and its sensor-based control. Specifically, the control algorithm generates a static sampling scheme consisting of 24 samples at an interval of 30 minutes when the electrical conductivity sensor in the wireless sensor network exceeds a predefined threshold. Further steps include the use of multiple sensor channels as trigger input, synchronized water sampling of multiple water samplers, as well as the dynamic adaption of the sampling scheme based on sensor data.

Acknowledgments

First, I would like to thank Prof. Dr. Lothar Thiele for giving me the opportunity to write this master thesis in his research group.

I would also like to thank my advisor Dr. Jan Beutel for his support. His expert knowledge and his commitment were very helpful for the realization of this work. Many thanks also to my co-advisor Dr. Philipp Schneider who introduced me into the field of hydrology. Thanks to both, Jan and Philipp, it was possible to achieve an exciting and challenging interdisciplinary cooperation.

A special thanks goes to Tonio Gsell for his straightforward support in the lab as well in the field. He always assisted me with his advice and took his time for answering my questions. I would also like to thank the following people who helped me during this thesis: Ben Buchli, Matthias Keller and Christoph Walser (Computer Engineering Group) as well as Alberto Calatroni (Electronics Laboratory). Furthermore, I'm very grateful for the help of Felix Sutton who proofread my report.

Last but not least, I would like to thank my family and girlfriend for their moral support and their patience during my master thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Water sampling is an important process in different areas of application, such as water and wastewater monitoring. For example, governmental monitoring networks use automated samplers to survey water quality in ground and surface water systems as defined by the water framework directive (WFD) [1]. Its focus is on long-term continuous monitoring to detect water quality trends in river basins. In environmental research the focus is typically on short-term campaigns and experiments with an emphasis on hydro-chemical dynamics during storm events. While for governmental water quality monitoring the catchment area is in the order of river basins ($> 250km^2$) the research catchments are typically of meso-scale of up to $50km^2$. Figure 1.1 shows a possible headwater research catchment to monitor.

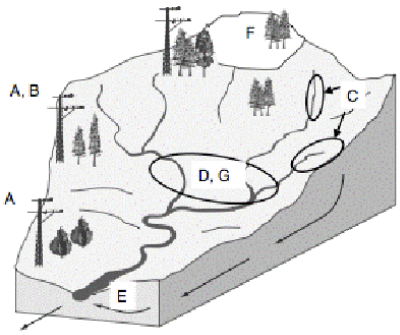


Figure 1.1: Hydrological catchment area.

Automated water samplers are an established technology for monitoring water quality. The water samplers are placed at the location of interest (e.g. a stream, river or lake) from where they draw water according to a programmed sampling scheme and store it into bottles located inside the water sampler. The number of samples drawn by a water sampler is limited by the number of available bottles, making continuous high frequency sampling very labor-intensive due

to the frequent collection of the samples and subsequent analysis in the laboratory. Therefore, water samplers are mainly used for low frequency sampling or in combination with a local sensor for high frequency sampling during a specific event. To enable sampling at their specific time of interest, e.g. storm events of short duration (1-2 hours), sensor-based event triggering is used. The drawback of existing event triggered water samplers is their dependence on a single local sensor, typically a water level sensor. It is not possible to use multiple, distributed sensors to determine the trigger. Because of the limited number of samples a water sampler can draw, the threshold of such water level sensor is set rather high to prevent false triggering. This in turn often has the drawback that the sampling onset is delayed and no samplings are drawn at the beginning of the rising limb (see Figure 1.2(a)). In addition, it may happen that events are not detected because the stream has eroded its river bed and thus changed the cross-section area and thus the local rating curve. Another drawback of existing water samplers is their lack of networking capabilities. That is, it is not possible to synchronize multiple water samplers allowing to draw samples at different locations of the catchment area at the same time nor is it possible to remotely observe the current state of a water sampler.

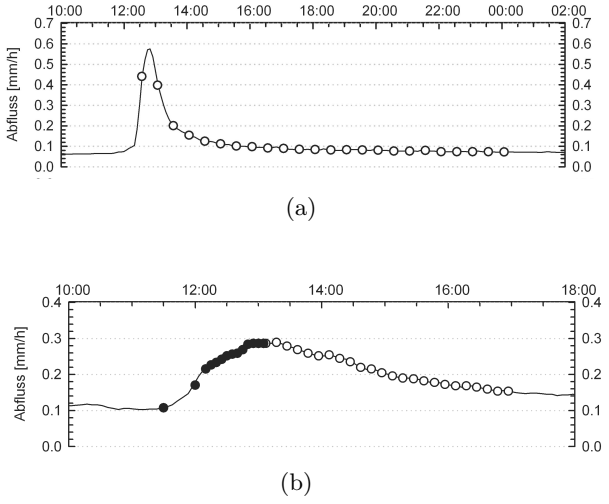


Figure 1.2: The figure shows two sampling series [2]: a) automated water sampling with delayed onset, and b) manual on-site water sampling without delayed onset. The X axis represents the time and the Y axis the runoff. The time of sampling is represented as circle or dot.

In this thesis, we propose a concept to integrate the ISCO 6712 water sampler from Teledyne [3] into the existing PermaSense wireless sensor network (WSN) [4]. Furthermore, the control of the water sampler is integrated into the PermaSense system. This enables the control algorithm to trigger one or more water samplers based on data from one or multiple sensors in the WSN.

1.2 Contribution

The contributions of this thesis are the following:

- Integration of the 6712 water sampler into the PermaSense WSN.
- Control of the 6712 water sampler based on sensor data.
- Automated and manual control of the water sampler.
- Remote information about the current state of the 6712 water sampler.
- Possibility to remotely modify the water sampler control algorithm.

1.3 Outline

Chapter 2 discusses related works and concepts of WSNs, wireless sensor actor networks (WSANs) and water sampling strategies. Chapter 3 describes possibilities how to integrate the 6712 water sampler into the existing PermaSense WSN and where to run the control algorithm controlling the water sampler. In Chapter 4 relevant information about the 6712 water sampler are summarized. Chapter 5 contains a detailed concept of the in-system control of water sampler as well as general information about the different software implementations. Chapter 6 illustrates the problem when using raw data for the control and examines some data cleaning approaches. First evaluation results from tests at the river Thur are shown in Chapter 7. Finally, Chapter 8 concludes the master thesis and gives an outlook on possible further steps.

Chapter 2

Related Work and Project Background

This chapter contains some general information about wireless networks consisting of sensors only or sensors and actuators. Furthermore, some sampling strategies used for water quality assessment are shown.

2.1 Wireless Sensor and Actuator Networks

2.1.1 Why Actors?

According to [5], actuators and actors are not exactly the same. A device used to convert a control signal into a physical action is called an actuator. An actor on the other hand consists of one or several actuators and additionally features networking-related functionalities (receive, transmit, process, and relay data). For example, a robot may consist of several actuators used to perform actions on the physical environment. In terms of networking, the robot is a single entity and therefore considered as an actor.

2.1.2 Implication on Network Architecture

WSNs aim to sense the physical environment and store sensor data. WSANs, in turn, can additionally perform certain actions on the environment in response to the sensed data. That is, the information transfer is not only in one direction like in a WSN but in both directions. A typical WSAN network architecture is shown in Figure 2.1. The sensors sense the physical environment and forward this information to the actors which, in turn, use the information to perform an action on the environment. Examples of WSANs are the light sensing and control in home automation [6], smart homes for supporting elderly and handicapped people [7], or the health monitoring of infrastructure such as an airplane [8].

The two main components of a WSAN are sensors and actors. Sensors are small devices with limited power, communication and computation capabilities while actors have more resources and are able to perform appropriate action based on sensor values [9]. Additionally, they may have a sink responsible for monitoring and managing the overall network [10].

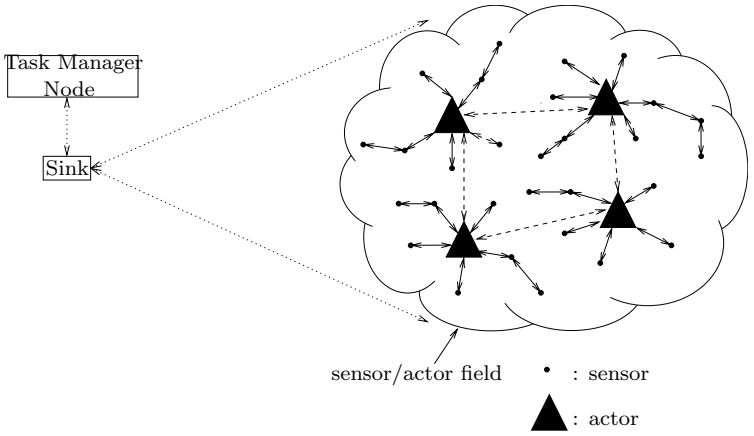


Figure 2.1: The architecture of WSANs including sensors, actors, and a sink [11].

WSAN architectures can be grouped in two categories [12]: (i) semi-automated architecture, and (ii) automated architecture. In semi-automated architectures the sensed data is collected at a sink from where the acting process is centrally controlled (see Figure 2.2(a)). That is, there is an explicit controller entity, e.g. located at the sink. With this architecture, the sensor data as well as the control commands to control the actuator are transmitted over the WSAN. Figure 2.3(a) shows the corresponding abstract high-level control application view. An example of a centralized control is the networked lighting system for optimizing energy savings where the optimal illumination of a shared-office space is controlled from a central point [13]. Conversely, in the automated architecture actions are not controlled from a centralized controller but from the actor itself, i.e. controllers are embedded into the actors (see Figure 2.2(b)). The actor collects sensor data from its environment and initiate appropriate action. Depending on the application, actor-actor communication may be required to coordinate the acting process. If no communication between actors is necessary, the WSAN network is only used for the transmission of sensor data (see Figure 2.3(b)).

Because the semi-automated architecture is similar to WSN architectures, existing WSNs may be adapted to accommodate actors. The disadvantage of the semi-automated architecture compared to the automated architecture is its longer latency because all data has to be routed to the sink where decision about the appropriate action is taken. But this drawback only holds when the sensor data origins from nearby sensors. Depending on the size of the event area, it may be necessary for the actor to receive data not only from nearby sensors

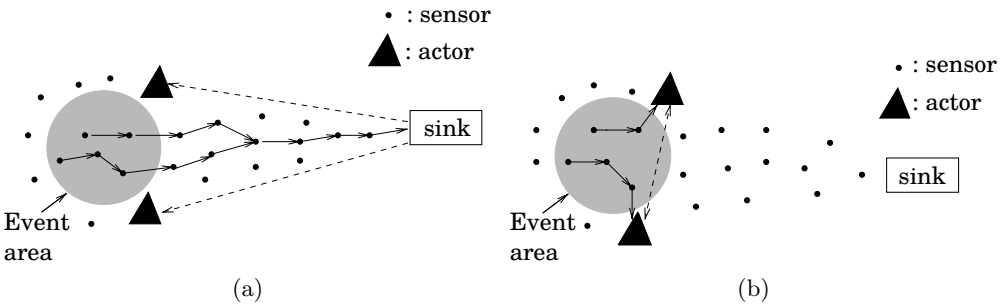
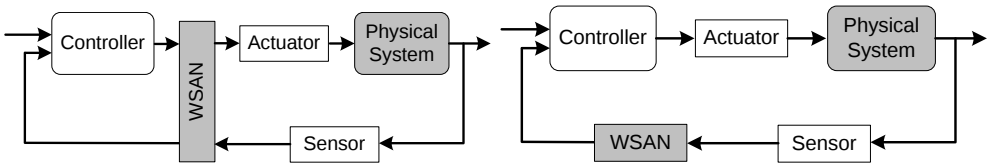


Figure 2.2: a) Semi-automated vs. b) automated architecture.



(a) Semi-automated architecture with explicit controllers.

(b) Automated architecture without explicit controllers.

Figure 2.3: Abstraction of control application [10].

but also from sensors which are further away or even in another WSN. Due to the fact that all data is processed by a central controller in the semi-automated architecture, the controller is a single point of failure.

2.2 PermaSense WSN

The PermaSense project [14] aims to collect geophysical data for permafrost monitoring in alpine regions. The system's heart is a WSN consisting of multiple simple sensor nodes able to measure temperature, electrical conductivity, water pressure etc. (see Figure 2.4). Recently the system was extended by more complex sensors with higher and variable data rates requiring user-interaction, or in-network data fusion. This successor project is called X-Sense [15] and strives to monitor the environment under extreme conditions. The PermaSense architecture, illustrated in Figure 2.4, is structured along the following tiers: (i) WSN including sensor nodes, (ii) *Base Station*, and (iii) backend. The WSN consists of multiple sensor nodes (TinyNodes [16]) which communicate over an 868 MHz radio. The wireless sensor nodes use the PermaDozer [17] data gathering protocol for sending their data to the sink. The PermaDozer protocol is based on the Dozer protocol [18]. The network topology of the WSN is a tree which may dynamically change. The tree is rooted at a single sink which is a

Base Station in the PermaSense WSN. Each message sent by a node is explicitly acknowledged by the receiving neighboring node in the tree topology and resent by the sender if no acknowledgment has been received. That is, there is no end-to-end message acknowledgment, but each single hop message is acknowledged. Although the Dozer protocol is mainly designed for data flow towards the sink, it features a lightweight backward channel enabling the sink to send data to sensor nodes. The backward channel data is included in the beacon message and limited to a few bytes. Furthermore, beacon messages sent by the sink are not acknowledged by the receiver, i.e. data send to sensor nodes over the WSN are not acknowledged at all.

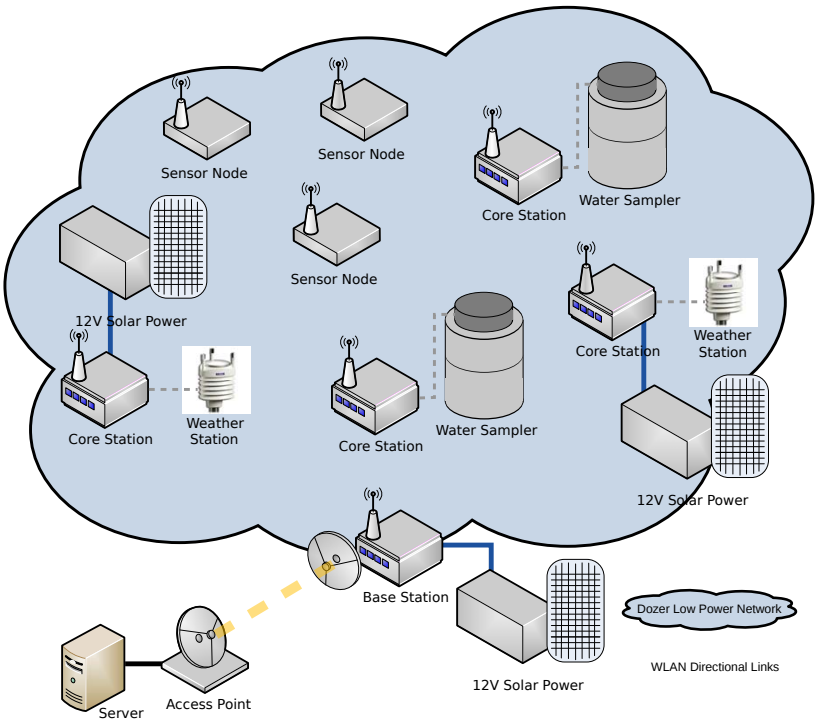


Figure 2.4: PermaSense system architecture.

The *Base Station* corresponds to a first level sink which forwards the received data over a wireless local area network (WLAN), general packet radio service (GPRS) or Ethernet link to the final sink (backend system). The PermaSense network may consist of several sub-WSNs, each connected to a single sink (*Base Station*). The *Base Station* is equipped with an 868 MHz radio interface allowing to communicate with the WSN. Additionally, it features a WLAN, Ethernet or GPRS interface for the communication with the backend. The backend system consist of one or multiple servers running the global sensor networks (GSN) software [19]. The GSN allows to integrate multiple WSNs, e.g. two different WSNs with individual sinks (*Base Station*s) connected to the backend system running the GSN. GSN is used for data collection, storage and data management.

Currently the following different sensor integrations are possible: (i) sensor node in WSN, (ii) *Core Station* in WSN (iii) *Core Station* with WLAN, GPRS or Ethernet link only, or (iv) *Core Station* in WSN with WLAN, GPRS or Ethernet link. The case (iv) is used in the concept of a wakeup-radio where the low power, lower bandwidth radio interface (here WSN) is used to temporarily enable the radio interface with the higher bandwidth (here WLAN, see [20]).

2.2.1 Time Information

Different time information exists in the PermaSense system. Figure 2.5 illustrates the relation between the different time information. The `generation_time` is the absolute time when the data was generated. It is calculated based on the absolute time when it was received by the *Base Station* (`timestamp`) and the relative time it required to propagate through the Dozer low power WSN to the *Base Station* (`atime`). The `timed` time corresponds to the absolute time when the data was finally received by the GSN and stored in a database. For data generated by the *Base Station*, the Dozer low power network is not involved in the transmission, and therefore the `atime` is zero and the `generation_time` is identical with the `timestamp`.

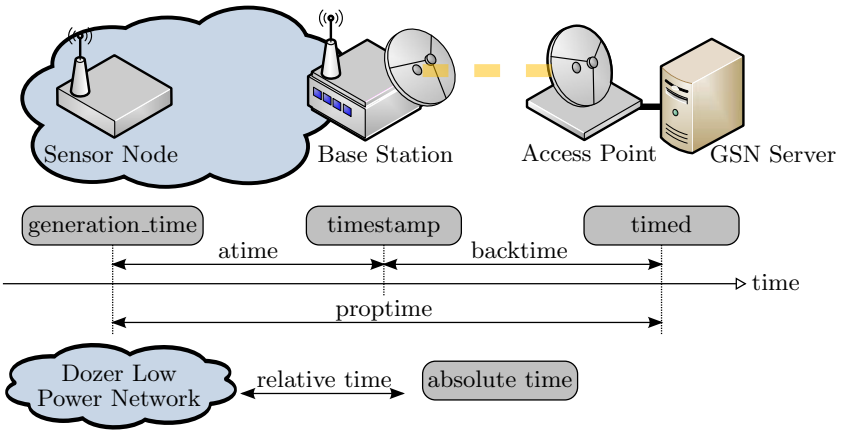


Figure 2.5: Time information in PermaSense.

2.2.2 Core Station

The *Core Station* is a generic platform which can be used as a sensor node or a *Base Station* [15]. The software running on the *Core Station* is called backlog and defines the platform's functionality. It is organized with plugins executing different tasks such as receiving sensor data from the WSN and forwarding them to the backend or reporting the *Core Station* and backlog status to the backend. The backlog software is configured by means of a configuration file stored on the

Core Station (`backlog.cfg`). When a plugin in the plugin section of the configuration file is set to 1, it is loaded during startup. After a plugin is instantiated, the `__init__` function of the plugin is called before the `run` function is executed. The `action` function is called when there is a schedule entry in the schedule file. The `action` function's parameters are those listed in the schedule file. If the plugin has not been loaded before, the plugin is instanciated before the `action` function is called. The `stop` function is called when the backlog software is stopped or the maximal runtime of the plugin is expired (`max_runtime_minutes` as schedule entry parameter or `max_runtime` value in the config file).

The *Core Station* can be operated in a duty cycle mode to save energy. In the duty cycle mode the *Core Station* is shutdown if the plugins do not have to execute any tasks. After the *Core Station* is shutdown, it is only turned on if a schedule job has to be executed, the daily service wake up is enabled, or a Dozer wake up command is received over the WSN. To activate the duty cycle mode, the duty cycle variable in the `options` section of the backlog config file has to be set to 1 (`duty_cycle_mode = 1`). In the duty cycle mode, the *Core Station* is shutdown if no plugin needs to run. The individual plugins can decide whether or not they allow to power off. If the plugin does not allow to power off, the function `isBusy` needs to return `TRUE` else `FALSE`. Before the system is shutdown, the TinyNode (running the PowerControl code) is configured to wake up the system at the time of the next schedule entry. The *Core Station* can be remotely powered on by the Dozer beacon command `GUMSTIX_CTRL_CMD` with the command value `1`. If the system has been enabled by this command, it has to be cleared with the same command and the command value `4` to return to the normal schedule mode. The command value `2` can also be used to power on the *Core Station*. With this command value the *Core Station* starts normally. After the boot process the *Core Station* is shutdown as soon as no more tasks have to be executed. That is, no subsequent Dozer beacon command is necessary to set the *Core Station* back to sleep as it was the case with the Dozer command value `1` mentioned before.

2.3 Sampling Strategies for Water Quality Assessment

There exist different sampling strategies to monitor the water quality. The applied strategy depends on the focus of the water quality monitoring. In research water analysis is used to understand the linking process between the catchment hydrology and the stream water chemistry [21]. Therefore, the sampling frequency has to be adapted to the time scale of the catchment's hydrological response. For research catchments of small scales, the hydrological response is in the order of minutes or hours requiring high-frequency water sampling. On the contrary, sampling frequencies prescribed by WFDs for water quality monitoring

are often in the order of weeks or months [22] (e.g. acidification status of rivers: one sample every 3 months [1]).

Manual sampling is a simple sampling strategy and used when low frequency monitoring is required. The drawback of low frequency sampling is that it only gives information about a single point in time which may be inappropriate if the water quality fluctuates over a short time period [23]. Automated water samplers are an alternative to manual sampling. They are able to automatically draw water from a stream or lake and store the samples into bottles. Due to the limited number of bottles, the automated water sampler has to be serviced once all bottles have been filled. Automated water samplers are used to draw time proportional samples [24, 25, 22] or, with a locally connected sensor, flow/volume proportional samples [26]. In addition, an on-site sensor can be used to trigger a sampling series in case of a specific event such as a storm event [27]. When continuous high-frequency sampling is required, sample bottles pile up fast leading to time consuming and expensive laboratory measurements of water chemistry. Therefore, high frequency sampling with automated water samplers is typically restricted to event triggered sampling [21]. In-situ analytics with field-deployable autoanalyzers enable long-term continuous sampling with higher frequency. The availability of environmental data with a high temporal resolution allows to observe previously unattainable insights into the hydrochemical evolution [21]. Although, some online measurements such as temperature, pH or electrical conductivity are already available at favorable prices, autoanalyzers for chemicals are still expensive. Another drawback of autoanalyzers is their limited number of chemicals they can analyze.

Chapter 3

Integration Concept

This chapter examines the different methods of integrating the actuator and its control into the existing PermaSense WSN.

3.1 Introduction

The approach of an event driven sampling system is based on the infrastructure of the WSN project PermaSense [4]. Although actuators have previously been integrated into the PermaSense system (e.g. controllable cameras [20]), the fact that an actuator has to be controlled based on sensor data is new. By the integration of actuators, the former WSN is extended to a WSAN. The presence of a control algorithm arises new questions such as where to implement the control algorithm, i.e. on which tier of the system (i.e. WSN, *Base Station*, or backend).

The integration of the in-system controlled actuators can be divided into two parts: (i) the actuator integration, and (ii) the control algorithm integration. The actuator integration deals with the physical integration of the actuator, i.e. how can the actuator interact with the existing wireless network. The control algorithm integration addresses the location where the control algorithm is implemented, i.e. where sensor data and other inputs are converted into actor control commands. Figure 3.1 shows the integration concept of the actuator and the control algorithm. The following sections will discuss the actuator and the control algorithm integration and how they relate to the WSAN network architecture presented in Section 2.1.2.

3.2 Network Architecture

The integration of water sampler actuators requires the extension of the PermaSense WSN to a WSAN. But our WSAN slightly differs from the WSAN

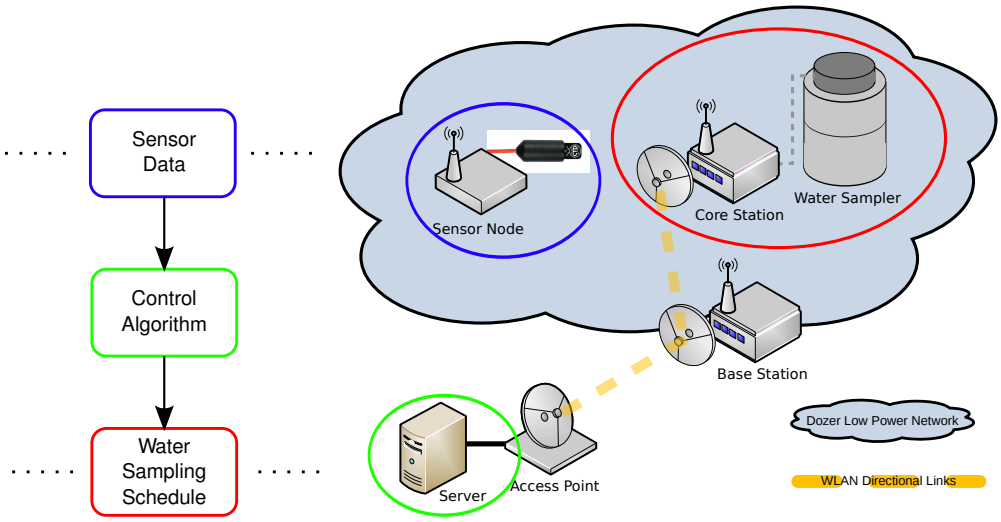


Figure 3.1: a) Abstract view of the water sampler control. b) PermaSense system architecture view of the integration concept.

described in Section 2.1. The actuator used in this project has little impact on the environment because it only draws a small amount of water from its environment. Additionally, the actuator does not only rely on sensor nodes located in the same WSN but potentially also on data from other networks (WSN, Internet etc.). Thus, the event area may be bigger than the WSN itself. Furthermore, actions to be performed cannot be split among multiple actors or delegated to another actor. Beside applying actions based on sensor data, the sensor data has to be gathered and stored, which requires forwarding all data to the sink.

In this project we focus on the semi-automated architecture which has the advantage that its architecture is similar to the existing WSN architecture. This allows the reuse of the existing architecture and its protocols. Furthermore, the actuator is not used to control an environmental variable, thus the hydrological catchment system is not a closed-loop system with potential instabilities due to high latencies. Also network lifetime is not considered to decrease significantly by the choice of a semi-automated architecture because the sensed sensor data still needs to be forwarded to the sink and control commands for the actors can be piggybacked on the Dozer beacon messages which are send anyway.

3.3 Actuator Integration

The actuator (6712 water sampler) does not feature a wireless interface supported by the PermaSense infrastructure (868 MHz radio, WLAN, GSM/GPRS). Therefore an interface unit is required to integrate the actuator into the PermaSense system. The combination of the actuator and the interface unit is referred to as actor (see Section 2.1.1). Currently two hardware platforms exist which allow to integrate sensors and actuators into the network: (i) sensor node, and (ii) *Core Station*. Some interface features of the two hardware platforms are listed in Table 3.1. Both platforms are equipped with a UART interface required for the communication with the actuator.

Table 3.1: Interfaces of sensor node and *Core Station*.

Interface	Sensor Node	Core Station
UART	x	x
Ethernet		x
868 MHz Radio	x	x
WLAN		x
GSM/GPRS		x

In a first step the actuator is connected to a powerful generic platform (see Section 2.2.2). The PermaSense *Core Station* serves as generic node, which in a later step may be replaced by a node with reduced hardware (e.g. sensor node hardware). This approach allows faster implementation and easier observation. The interface unit is integrated via the 868 MHz radio as well as the WLAN into the existing system. Care has to be taken that the data transmitted over the WLAN link is kept low so that it is possible to replace it at a later time by an interface with lower data rate, e.g. an 868 MHz radio.

3.4 Control Algorithm Integration

The aim of the control algorithm is to control actors based on sensor data and other inputs such as human commands or actor states (see Figure 3.1(a)). As mentioned in Section 3.2, we will focus on the semi-automated architecture where the control algorithm is located in a central controller, e.g. the sink. In the following the different control algorithm integration possibilities are reviewed. The review is based on the following assumption:

- The control algorithm requires data from different networks such as a different WSN or Internet.
- Actuators may reside in different WSAN.
- Wireless link failures may occur.

Based on the PermaSense system architecture with its three tiers, the control algorithm may be integrated into a single tier or a combination of different tiers. In the following, the different tiers are analyzed qualitatively with respect to the following criteria: link failure, processing power, availability of energy and sensor data, observability of the control algorithm, reaction time and data traffic.

3.4.1 Sensor Nodes

Nodes within the WSN are connected to the *Base Station* via ISM-band radio interface. The multi-hop protocol PermaDozer is used for the communication between nodes and *Base Station*. Due to the power restrictions of the sensor nodes in the WSN and the protocol, the bandwidth is limited. Because of the potentially large amount of data required by the control algorithm, the integration of the control algorithm into the WSN, i.e. into the interface unit, is not possible due to PermaDozer data limitation. Although the WSN tier is not appropriate for the control algorithm integration, it may need to carry out some tasks of the control algorithm in case of link failure (fallback scenario). Furthermore, it may also take over some control algorithm parts where short latencies are required.

3.4.2 Base Station vs. Backend (GSN)

If the control algorithm would depend only on the sensor data of sensor nodes connected to a single *Base Station*, then the integration of the control algorithm into the *Base Station* would be an advantage because of the availability of data in case of a *Base Station* - backend link failure. But as the control algorithm possibly relies on data from other WSNs only accessible via the backend, none of the two tiers have an advantage over the other. Furthermore, the current *Base Station* software architecture does not allow plugins running on the *Base Station* to access data of other sensors. Also the reaction time will be similar for both tiers, because the *Base Station* and the backend need information from each other. In terms of processing power the backend has more resources which can be increased easily. Also the availability of energy is often better for the backend because the *Base Station* is usually powered by a battery combined with a solar panel. Concerning the observability of the control algorithm, the backend does not have a big advantage because the *Base Station* - backend link allows fast access to the *Base Station*. In terms of data traffic, the *Base Station* does not have an advantage because all sensor data needs to be stored in the backend system. Furthermore, an integration of the control algorithm into the *Base Station* would increase the data traffic between *Base Station* because all relevant information not available from the WSN needs to be send from the backend to the *Base Station*.

3.4.3 Summary

The WSN tier alone is not an option for the implementation of the in-system control algorithm, but it has to undertake some fallback tasks if no connection is available between the actor and the main control algorithm. The implementation of the main control algorithm in the *Base Station* tier is not favorable because its main advantage, the availability of sensor data of the connected WSN in case of a *Base Station* - backend link failure, is no longer given when data from other networks is required. For this reasons, the backup tier is the preferred location for the implementation of the control algorithm (see Figure 3.1(b)).

Chapter 4

ISCO 6712 Water Sampler

This chapter aims to give some relevant information about the 6712 water sampler from Teledyne ISCO (hereafter referred as 6712). The information mainly origins from the 6712's instruction manual [3] or was gained through tests with the 6712.

4.1 Introduction

The 6712 is a portable water sampler which can easily be moved from site to site (Figure 4.1). It is able to draw water from a nearby stream or lake by means of a tube. Water samples are stored in bottles located inside the sampler (see Figure 4.1(b)). The 6712 features a controller which can be programmed to run different programs. The controller has a 4×20 character display as well as a keypad for human interaction. Different modules (700 Series modules) may be connected to the controller to monitor physical phenomena such as temperature or flow rate. These monitoring devices may be used to trigger the sampler to draw a water sample. As an option, SDI-12 sensors may be connected to the controller. The water sampler is extensively described in the installation and operation guide [3]. In the following sections some relevant parts from the guide as well as information gained during the project are described.

4.2 Programming Level

When the water sampler is used as stand-alone device, it may be programmed based on two programming levels: (i) standard programming level, and (ii) extended programming level. The standard programming level allows to set up typical sampling programs while the extended programming level allows to set additional features. The programming level can be selected by entering a numerical command on the keypad when the 6712 is in the main menu: (i) `6712.1`

(a) (b)

Figure 4.1: 6712 water sampler.

for the standard programming level, and (ii) `6712.2` for the extended programming level. The main menus of the standard and the extended programming level are shown in Figure 4.2.

RUN PROGRAM VIEW REPORT OTHER FUNCTIONS	RUN "EXTENDED 1" **PROGRAM** VIEW REPORT OTHER FUNCTIONS

(a) Standard programming level (b) Extended programming level

Figure 4.2: Programming level dependent main menus [3].

4.2.1 Standard Programming Level

The language and the unit for the length can be selected by typing `6712.8` at the main menu. In the standard programming level the following settings may be configured:

- Site description (text)
- Bottle kit (number of bottles and bottle size)
- Suction line length
- Pacing, defining at which rate the 6712 takes samples
- Distribution, defining how the drawn samples are distributed over the bottles (one sample per bottle, several bottles per sample, several samples per

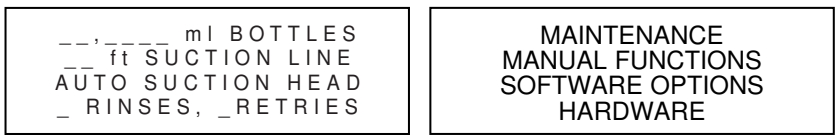
bottle etc.)

- Sample volume
- Sampling start, defining when to start with the sampling (immediate start, delayed start, start at a specific time, start after remote command)

The 6712 may be disabled to prevent samples being taken while a program is running. This can be done by an external module or a remote control command (see Section 4.6.1.1). A standard program may be started by selecting `RUN` in the main menu. Afterwards, the running program may be interrupted by pressing the STOP key or sending the remote command `PAUSE`. These actions are logged in the event log contained in the sampling report (see Section 4.6.1.4). In the pause state, the program operates as normal, except that no samples are taken. Skipped samples are recorded in the event log. By selecting `RESUME PROGRAM` the paused program can be continued.

4.2.2 Extended Programming Level

All the features available in the standard programming level are also available in the extended programming level. The 6712 can store four different extended programs. In the following some relevant configurations available in the *Program* and *Software Options* menu are described. The *Program* menu is available in the main menu (Figure 4.2(b)) and the *Software Options* menu is a sub-menu of the *Other Functions* menu (Figure 4.3(b)) available in the main menu. Figure 4.3(a) shows the *Bottle* menu where the number of bottles, the bottle volume and other settings can be configured.



(a) *Bottle* menu (b) *Other Functions* menu

Figure 4.3: 6712 extended programming menus.

4.2.2.1 Rinses and Retries

In the extended programming mode it is possible to program the 6712 to automatically rinse the suction line before taking a sample. The number of rinse cycles is in the range of 0 to 3. The 6712 is able to detect liquid. If the 6712 does not detect any liquid when pumping, it may retry to take a sample before it skips. The number of retries may be configured and is between 0 and 3.

4.2.2.2 Pump Counts for Purge Cycle

Purge cycles are used to clear the strainer at the end of the suction line before and after taking a sample. During the purge cycle the pump runs in reverse. Pre- and post-sample purges are defined in pump counts and can be configured in the *Software Options* menu.

4.2.2.3 Periodic Serial Output

The 6712 can be configured to periodically output its status to the interrogator port (see Figure 4.7). The format of the serial data (`DATA`) is the same as described in Section 4.6.1.2. While the 6712 is in communication mode (Section 4.6) the status is not sent. Only after the communication timeout the status is periodically sent again.

4.2.2.4 Interrogator Connector Power

The interrogator connector features a +12VDC power output (Section 4.6.2). This power output can be programmed to be either always on/off or on/off during certain time intervals.

4.3 Power

4.3.1 Power Consumption

Table 4.1 shows some typical power consumptions of the 6712. The measurements were accomplished with the multimeter Agilent U1253A. For the current measurements the handheld terminals 'A' and 'COM' with a maximal current of $10A$ were used.

Table 4.1: Power consumption of 6712.

Sampler Status (see Figure 4.6)	**Back Light**	**Current [mA]** avg[1]	peak[1]
Sampler Off	OFF	14.6	23.1
at Standby	OFF	16.2	29.8
at Standby	ON	225.9	239.6
Program Running (disabled)	OFF	18.3	28.9
Take Sample	OFF	3100	5322

[1] Peak and average current were measured during at least 100s

4.3.2 Power Failure

When a running 6712 program is interrupted due to a power failure, it is continued after the power is restored. When requesting the 6712 status in the *Menu Control* mode (see Section 4.6.1.1), the sampler status is extended with an error in brackets (see Listing 4.1). Furthermore, the time when the power fail occurred and the time when the power has been restored is listed in the sampling report (`POWER FAILED!` and `POWER RESTORED`).

Listing 4.1: *Menu Control* sampler status after power fail.

```
Sampler Status: Program Running (ERRORS)
Program Status: 0 samples in bottle 1 (Disabled: remote)
...
```

4.4 Fuel Gauge

The 6712 has an internal fuel gauge monitoring the power consumption. By pressing the STOP key, the current and the previous power consumption are displayed in Ampere-hours. The previous power consumption denotes the power consumed before the battery has been changed and the current power consumption denotes the power consumed since the battery has been changed. It has to be kept in mind that the power consumption does not give an absolute information about the battery's state because the initial state of the battery is not known by the 6712.

4.5 Report

The 6712 is able to store 1000 sampling events in its internal battery-backed RAM. It is able to store additional readings from SDI-12 sensors, 700 Series modules and other sensors. Those sensor readings and the program settings are available as reports which may be collected in different ways, e.g. via serial RS-232 interface. The format of the report can be configured. Figure 4.4 lists two different types of report. For the description of the source and error codes in the sampling report we refer to Table 4-3 in [3]. The data for the reports are stored in the memory until a new program is run (selecting `RUN`). Starting a new program clears the memory and stores the data for the current program. That is, the program settings in the report are only updated after a program has been started.

```
SAMPLER ID# 3687447734  06:32 19-DEC-02
  Hardware: A0      Software: 1.02
*********** SAMPLING RESULTS ***********
   SITE: FACTORY051
Program Started at 15:03 WE 18-DEC-02
Nominal Sample Volume =  200 ml
                                        COUNT
                                          TO
SAMPLE  BOTTLE TIME  SOURCE ERROR LIQUID
------- ------ ----      -- --    ------
              15:03  PGM ENABLED
  1,1      1   15:03       S           250
  1,1      2   15:18       T           247
  1,1      3   15:33       T           247
  1,1      4   15:48       T           249
  1,1      5   16:03       T           247
  1,1      6   16:18       T           247
  1,1      7   16:33       T           247
  1,1      8   16:48       T           248
  1,1      9   17:03       T           237
  1,1     10   17:18       T           236
  1,1     11   17:33       T           237
  1,1     12   17:48       T           241
  1,1     13   18:03       T           238
  1,1     14   18:18       T           236
  1,1     15   18:33       T           237
  1,1     16   18:48       T           236
  1,1     17   19:03       T           242
  1,1     18   19:18       T           237
  1,1     19   19:33       T           235
  1,1     20   19:48       T           238
  1,1     21   20:03       T           237
  1,1     22   20:18       T           237
  1,1     23   20:33       T           236
  1,1     24   20:48       T           230
              20:48  PGM DONE 19-DEC

  SOURCE S  ==> START
  SOURCE T  ==> TIME
```

(a) Sampling results

```
SAMPLER ID# 3687447734  06:32 19-DEC-02
  Hardware: A0      Software: 1.02
*********** PROGRAM SETTINGS ***********
     ----------

 SITE DESCRIPTION:
    "FACTORY051"

     ----------
   UNITS SELECTED:
   FLOW RATE: cfs
 FLOW VOLUME: Mgal

     ----------
   BUBBLER MODULE:
        WEIR
        90
      V-NOTCH
     ----------
  24,  1000 ml BTLS

 10 ft SUCTION LINE

     ----------
      PACING:
    TIME, EVERY
 0 HOURS, 15 MINUTES

     ----------
    DISTRIBUTION:
     SEQUENTIAL


     ----------

   200 ml SAMPLES


     ----------

  5 MINUTE DELAY TO
    FIRST SAMPLE
```

(b) Program settings

Figure 4.4: 6712 report examples [3].

4.6 Remote Control

4.6.1 Computer Control

The 6712 can be controlled remotely via the RS-232 serial interface. Table 4.2 shows the settings required for the RS-232 serial communication. The baud rate is automatically detected by the 6712. Figure 4.5 shows an overview of the samplers remote operation control structure. It is distinguished between two different levels of computer control: (i) *Menu Control*, and (ii) *External Program Control*. To build up a connection to the 6712, question marks (`?`) need to be sent until the 6712 returns its banner string containing hard- and software revision, model number and ID followed by a greater-than sign (`>`). This needs to be done for both levels of control. After approximately 6 minutes, the communication times out and has to be re-established by sending question marks before new commands are accepted. It is not possible to remotely turn off the sampler. In the following the different remote operation modes illustrated in Figure 4.5 are explained in more detail.

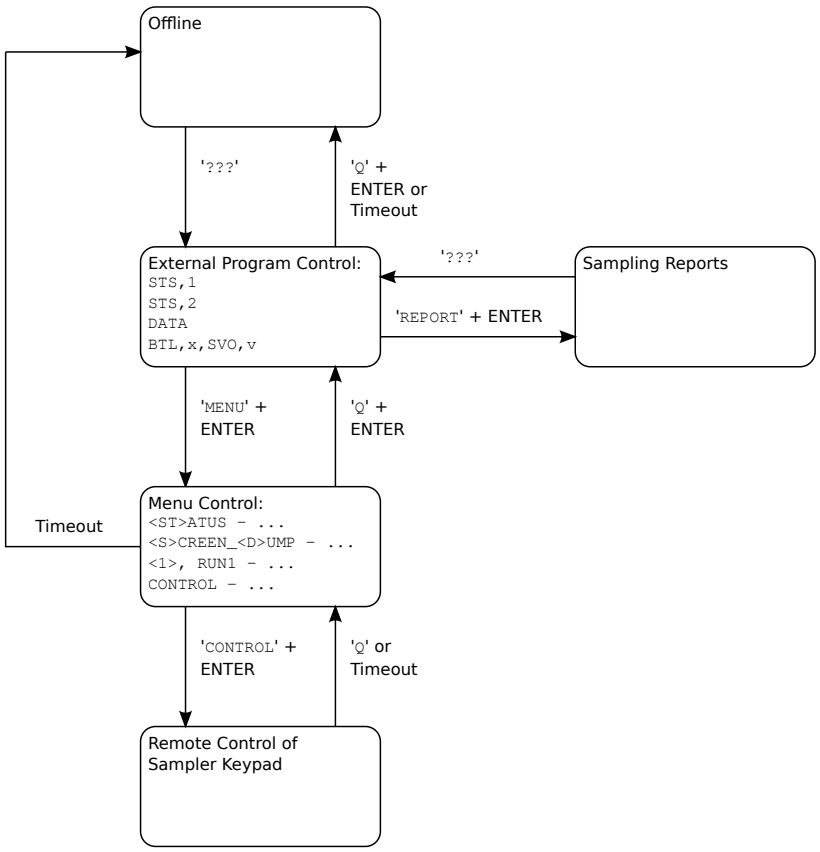


Figure 4.5: 6712 remote operation control structure.

Table 4.2: Serial settings for the communication with the 6712.

Baud Rate	2'400 - 19'200
Parity	none
Data Bits	8
Stop Bits	1
Flow Control	none

4.6.1.1 Menu Control

After establishing a connection to the 6712, menu commands can be sent to the 6712. `MENU` followed by an ENTER needs to be typed before menu commands are accepted by the 6712. Table 4.3 lists all known menu commands and indicates when those commands are available. For more detailed description we refer to Table 7-1 in [3].

Table 4.3: Remote menu commands of the 6712.

Menu Command/	**Standard Program**		**Extended Program**	
Short Form	**Stopped**	**Running**	**Stopped**	**Running**
`START / 0`	–	✓[1]	–	✓[1]
`RUN1 / 1`	–[3]	–	✓	✓[2]
`RUN2 / 2`	–	–	✓	✓[2]
`RUN3 / 3`	–	–	✓	✓[2]
`RUN4 / 4`	–	–	✓	✓[2]
`DISABLE / 5`	–	✓	–	✓[4]
`ENABLE / 6`	–	✓	–	✓[5]
`TAKE_SAMPLE / 7`	–	–	–	✓
`STATUS / ST`	✓	✓	✓	✓
`SCREEN_DUMP / SD`	✓	✓	✓	✓
`PAUSE / P`	–	✓	–	✓
`CONTROL`	✓	✓	✓	✓
`QUIT / Q`	✓	✓	✓	✓

[1] Available when program is set to WAIT FOR PHONE CALL
[2] Available when another program number is running
[3] Accoring to [3] Table 7-1, this should be possible
[4] Available when program is enabled
[5] Available when program is disabled

Depending on the 6712's current state (selected menu, programming mode, program state etc.) the available commands may change. The menu commands allow to load and start existing programs. Furthermore, running programs can be disabled to prevent the 6712 from drawing samples. While a program is running

on the 6712, it is possible to manually initiate a sampling. Full control over the 6712 can be gained by typing `CONTROL`. This command allows to remotely control the 6712's keypad (see Section 4.6.1.3). The *Menu Control* mode can be quit by typing `Q` and ENTER.

Note: When the main screen (see Figure 4.2) is not active, i.e. when another menu is selected or the `POWER USED` is displayed, the `RUNx` commands are not available in the *Menu Control* mode!

In the *Menu Control* mode the following states, requested by the `STATUS` command, are possible:

- Sampler Off
- Program Idle
- at Standby
- Program Waiting Start
- Program Running (enabled/disabled)
- Program Halted

The possible transitions between the states are shown in Figure 4.6. After the power on of the 6712, the *Menu Control* mode state changes from **Sampler Off** to the **Program Idle** before it reaches **at Standby** after the booting has completed. The 6712 is only in the **at Standby** state when the main menu (see Figure 4.2) is active. When a program is run, the next state depends on the start configuration of the 6712. When it is configured to start at a later point in time, the state **Program Waiting Start** is entered (see [1] Figure 4.6) else the state **Program Running (enabled)** is entered (see [2] Figure 4.6). If the sampler is in the **Program Waiting Start** state and the program is started because the configured 'later point in time' is reached, the 6712's state changes to **Program Running (enabled)** if it has not been disabled before (see [3] Figure 4.6). If it has been disabled before, the 6712's next state is **Program Running (disabled)** (see [4] Figure 4.6). While the 6712 is running a program, its program state can be enabled and disabled. Disabling and enabling is only possible in the *Menu Control* mode but neither in another remote operation mode nor by a manual keypad press. The state **Program Halted** is entered as soon as the 6712's program is stopped. The program can only be stopped by a keypad operation, which can take place manually on-site or remotely in the remote operation mode *Remote Control of Sampler Keypad*. It may happen that after the stopping of the 6712's program, the 6712 state stays in the **Program Halted** and does not automatically change to the **at Standby** state (see [5] Figure 4.6). The **Program Halted** state then can be left by pressing the STOP key again.

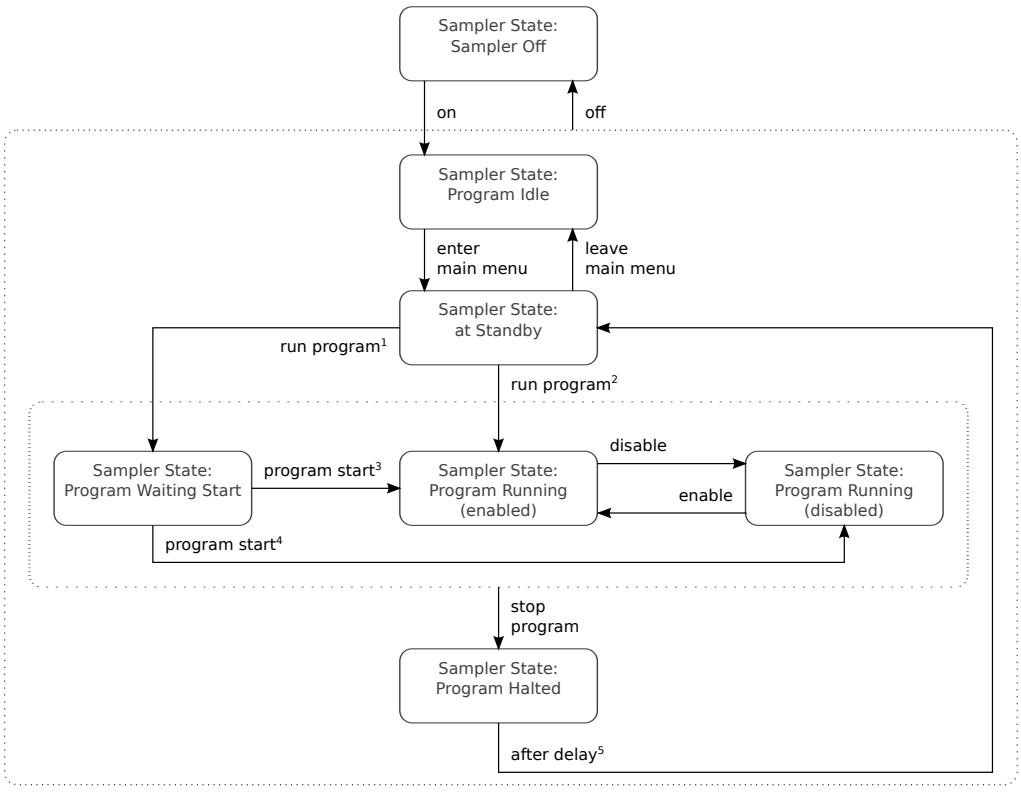


Figure 4.6: *Menu Control* mode state transitions.

4.6.1.2 External Program Control

The following four commands are available in the *External Program Control* mode:

- Turn on sampler
- Take a sample
- Send status
- Send data

The commands need to be followed by a carriage return (<CR>). The individual commands are described in the following paragraphs. It is possible to execute the commands while a program is running. Command driven samples are then marked in the sampling results report as M (command driven sample). When no program is running, command driven samples are not logged by the 6712.

Turn on the Sampler The 6712 can be remotely turned on by sending the command STS,2<CR>. This command is replied by the 6712 with the latest status. It is not possible to remotely turn off the 6712.

Take a Sample The 6712 can be remotely triggered to take a sample by the command BTL,2,SVO,100<CR>. The first number after BTL, specifies the bottle to place the sample in (here bottle number 2) and the second number specifies the sample volume in ml (here $100ml$). This command is only valid if the 6712 is waiting to sample (state WAITING TO SAMPLE = STS,1). The take a sample command is replied with a status string (see **Send Status** command below). When an invalid bottle number or an invalid sampling volume is specified, the reply status is INVALID BOTTLE=STS,22 or VOLUME OUT OF RANGE=STS,23 respectively. This status is only present for the immediate reply, that is, for a later status request with STS,1<CR> the status will not be INVALID BOTTLE or VOLUME OUT OF RANGE anymore (see Appendix C.1.3).

Most settings of the 6712 are ignored when the 6712 is in the *External Program Control* mode, but two settings need to be configured in advance (see p.7-4 in [3]): (i) number of bottles and (ii) suction line length (can be done in the standard or extended program mode). The rinsing and retry behavior depends on the selected program. Rinses and retries are only available in the extended programming level (see Section 4.2.2). If the selected extended program, which does not need to run, enables rinses and/or retries, a remotely triggered sample taking also does rinses and/or retries.

Send Status The status command `STS,1<CR>` returns a string containing the current status of the 6712. The string contains multiple comma-separated pairs of identifiers and values. The status string contains static information such as model number or ID number as well as dynamic information such as the current device state or information about the last drawn sample. All possible identifiers and the interpretation of the values are listed on page 7-5 in [3]. Line 1 in Listing 4.2 shows a possible reply on a status request command.

Listing 4.2: 6712 send status replies with identifier/value pairs.

```
1 MO,6712,ID,1281780884,TI,40889.61407,STS,9,STI,40889.58014,
    BTL,1,SVO,200,SOR,13,CS,4793
2 MO,6712,ID,1281780884,TI,40889.61407,STS,9,CS,6894
```

When the 6712 has been reset, no 'most recent sampling' data is available and therefore the status reply contains only the following identifiers: `MO, ID, TI, STS, CS` (see line 2 in Listing 4.2).

Send Data The send data command `DATA<CR>` is replied by the 6712 with a string containing standard and sensor information. The standard information part contains, among others, the model number, current time and information about the last three bottles used for storing the water samples. The sensor part contains data from the sensor modules connected to the 6712 (e.g. rain gauge, SDI-12 devices). Table 7-2 in [3] shows an extensive list describing different identifiers. Listing 4.3 shows a possible reply from a 6712 without any external sensors.

Listing 4.3: 6712 send data reply with identifier/value pairs.

```
DE,6712 SAMPLER,ID,1281780884,MO,6712,TI,40889.62551,SS,1,B1
    ,40889.58014,B1,40889.57059,B1,40889.49978,CS,5876
```

Checksum Each command can be extended by an optional checksum to prevent wrong commands. If the appended checksum is wrong, the command is ignore and the status is `CHECKSUM MISMATCH=STS,21`. The 6712's reply strings always contain the checksum. The calculation of the checksum is described on page 7-8 in [3].

4.6.1.3 Remote Control of Sampler Keypad

In this mode, the sampler keypad can be remotely controlled. The 6712 continously updates the terminal so that it corresponds to the 6712 display. The accepted computer keys and their corresponding functions are listed in Table 4.4. The remote control of the sampler keypad can be left by typing `Q` (see Figure 4.5).

Table 4.4: Remote control of sampler keypad (see Table 7-3 in [3]).

Computer Key	**Sampler Key**
<Esc>, S, s	STOP
L,l,U,u, <Backspace>	Left / Up
R, r, D, d	Right / Down
O, o	ON
<Enter>, arrows, decimal, numbers	Same as sampler

The 6712 uses VT100 terminal commands to move the cursor in the terminal and to format the displayed characters. All known commands used by the 6712 are listed in Tables 4.5 and 4.6. For a more detailed list about VT100 terminal commands we refer to [28].

Table 4.5: VT100 control characters.

Control Character		**Description**
Hex	**Char**	
0A	LF	Line feed
0D	CR	Carriage return
1B	ESC	Invokes a control sequence (see Table 4.6)

Table 4.6: VT100 control sequences.

Control Sequence		**Description**
Hex	**Char**	
1B 5B 30 6D	ESC [0m	Turn off character attributes
1B 5B 35 6D	ESC [5m	Turn blinking mode on
1B 5B 37 6D	ESC [7m	Turn reverse video on
1B 5B xx 41	ESC [xxA	Move cursor up xx lines
1B 5B xx 42	ESC [xxB	Move cursor down xx lines
1B 5B xx 43	ESC [xxC	Move cursor right xx lines
1B 5B xx 44	ESC [xxD	Move cursor left xx lines

Programming Style The 6712 features two different programming styles, that is, ways to navigate through the 6712's menu, for on-site configuration or in the *Remote Control of Sampler Keypad* mode: (i) *Normal*, or (ii) *Quick View*. The *Quick View* progamming style allows to view and modify the different configurations, while the *Normal* programming style guides through the menu step-by-step. The *Quick View* is faster and more flexible because not all possible configurations have to be entered.

Stopping a Program A program cannot be stopped by a *Menu Control* mode command or by an *External Program Control* mode command. To remotely stop a running program, the *Remote Control of Sampler Keypad* mode has to be activated (`CONTROL` in the *Menu Control* mode) before, by navigating through the menu, the program can be stopped by selecting `STOP PROGRAM`.

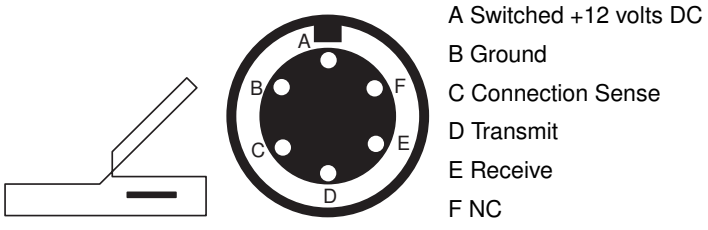
Configuring the 6712 It is not possible to configure the 6712 by means of a configuration file or a specific application programming interface (API). That is, the only way to remotely configure the 6712 is through the *Remote Control of Sampler Keypad* mode.

4.6.1.4 Sampling Report

The sampling report contains events occurred while a program was running (see Section 4.5). It can be read out via serial RS-232 interface. The format of the report is as configured (see Chapter 4.15.3 in [3]). The *Menu Control* mode has to be exit (`Q`) before requesting the sampling report by typing `REPORT` and pressing enter (see Figure 4.5). After receiving the report, question marks have to be sent until the banner string is sent by the 6712, otherwise no commands are accepted. The sampling report can be read out while a program is running.

4.6.2 Interface

Figure 4.7 shows the symbol of the interface plug and its pinning. The female connector on the 6712 side has 6 pins. Its counter piece is a 6 pin male cable connector (MIL-C-5015 type 3106A-14S-6P). The cable configuration of the interface cable is shown in Table 4.7. Pin C (Connection Sense) of the interrogator port needs to be set to ground to enable the communication. If pin C is left open, only model number, hardware and software revision as well as the ID can be read out before the connection times out. Therefore no commands other than `??` are accepted by the 6712 if pin C is left open.



(a) Symbol (b) Pinning (c) Pin function

Figure 4.7: 6712 interrogator port [3].

Table 4.7: Interface cable 6712 – computer.

6712 interrogator port 6-pin male			**Computer** 9-pin female	
A	+12V DC	open	-	-
B	GND	-	5	GND
C	Connection Sense	set to GND, pin B	-	-
D	Transmit	-	2	RXD
E	Receive	-	3	TXD
F	NC	open	-	-

Chapter 5

Realization

This chapter contains the detailed integration concept for the in-system control of the actuator. Furthermore, an overview about the individual software implementations is given. For detailed information about the software we refer to the source code.

5.1 Overview

The following chapters describe the tasks of the individual devices/tiers in connection with in-system controlled actuators. Figure 5.1 shows an abstract view of the control application. The control loop is open because the actuator does not have any influence on the environment, i.e. the physical system.

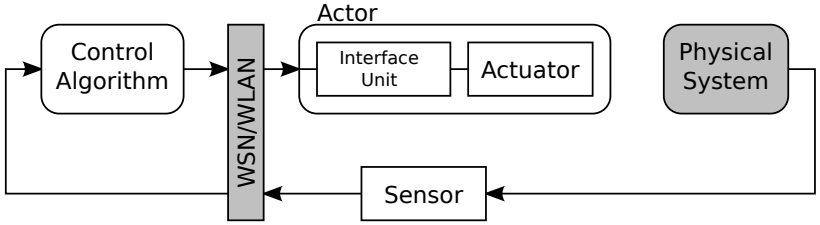


Figure 5.1: Abstraction of control application.

A PermaSense system architectural view with the different devices is illustrated in Figure 5.2. It shows an example with a single actor and a single sensor as well as the backend system consisting of a GSN server. Furthermore, the connections between the different devices are shown. The control algorithm residing in the GSN server controls the actor by means of a schedule file containing all relevant information about the samples to be taken. The schedule file is transmitted by the GSN server via WLAN to the *Core Station*. To save energy the *Core Station* is in the duty cycle mode. Therefore the *Core Station* cannot receive any schedule files over the WLAN link when it is shutdown. Sending a

Dozer wake-up beacon from the backend via *Base Station* to the the WSN allows to wake-up the *Core Station* so that it can receive a new schedule.

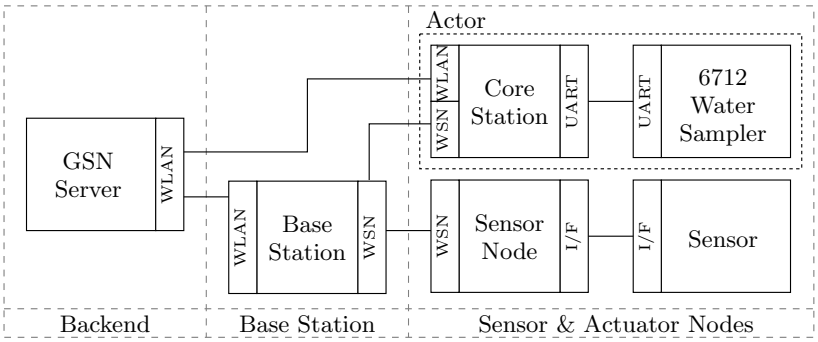


Figure 5.2: PermaSense system architectural view.

The *Core Station* and the 6712 water sampler are powered by the same 12VDC battery. The battery voltage is periodically measured by the *Core Station* and reported to the backend system. Therefore, the 6712 fuel gauge functionality (see Section 4.4) is not required to monitor the state of the battery.

5.2 Backend

This section describes the structure of the system from a GSN perspective. It describes how the different virtual sensors (VSs) are connected to each other.

5.2.1 Object Model

Figure 5.3 shows the GSN view of the sensor-based control of the 6712 water samplers. Physical devices such as the *Core Station*, sensors or 6712 are shown outside the GSN view as rectangular boxes. The control algorithm VS receives cleaned data (outlier and noise filtered data) from one or multiple sensors. Based on its control algorithm, it decides when and how much water the 6712 needs to draw. Before the 6712, respectively the *Core Station* the 6712 is connected to, can receive a new schedule file containing the water sampling timing information, the *Core Station* needs to be activated via the Dozer command VS.

5.2.2 Dynamic Model

This section describes the behavior of the individual VSs and the interaction between the VSs. The sequence diagrams show the interaction between the VSs for different scenarios.

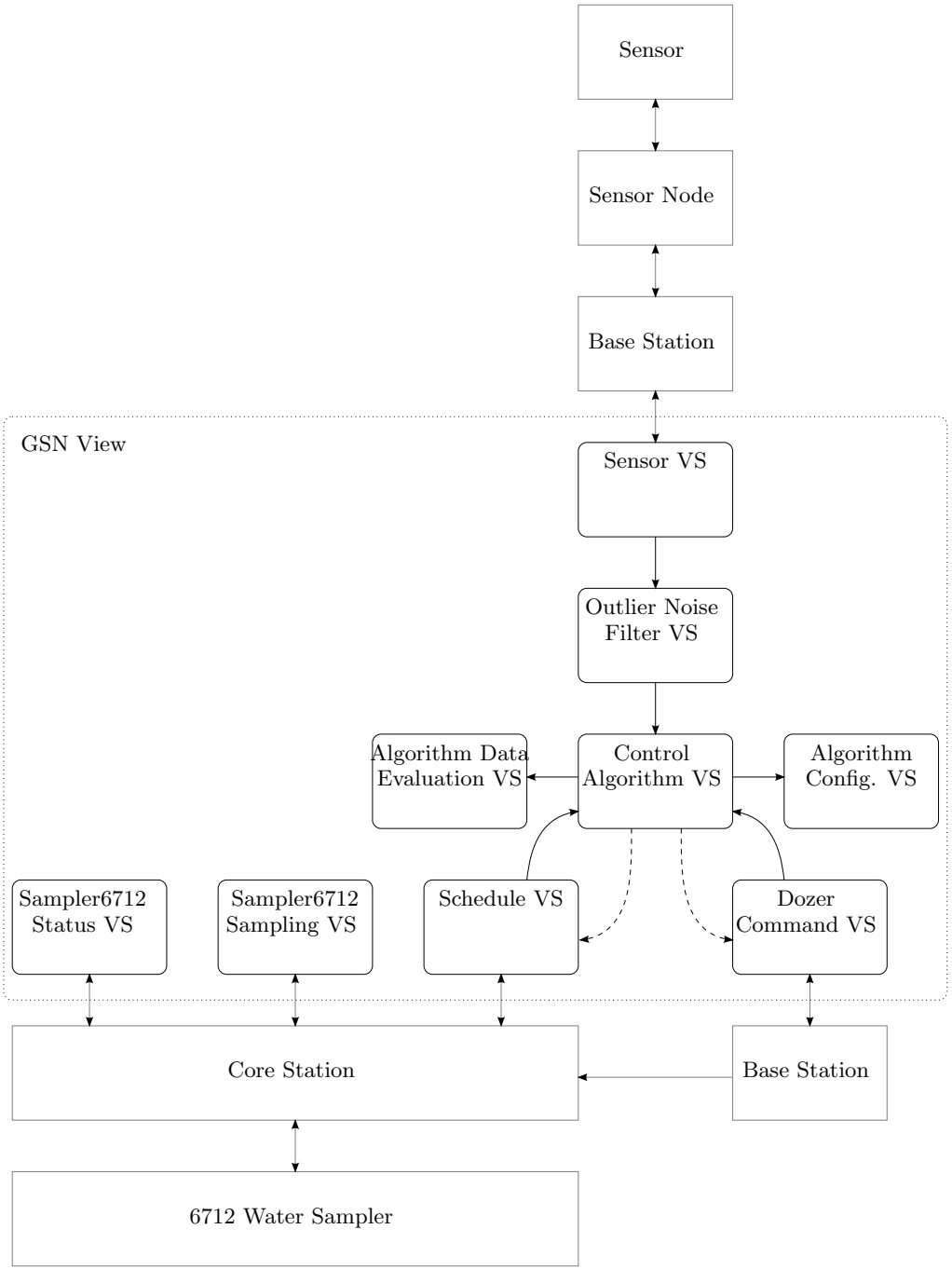


Figure 5.3: GSN view of VSs (rounded boxes) related to the control of the 6712 water sampler. An arrow with a solid line corresponds to the flow of streaming data within GSN. Dashed lines show the flow of data via HTTP post. The rectangular boxes outside the GSN view represent the devices the individual VSs are connected to (see also Figure 5.2).

5.2.2.1 Sequence Diagrams

The following sequence diagrams are shown only for a single sensor and a single Dozer command and schedule VS. But it is conceivable that the number of VS is increased in the future. The communication between the VSs in the following figures takes place over GSN data streams or HTTP posts. In the following, the expression 'acknowledge' is used in connection with the Dozer command VS and the schedule VS corresponds to the data streams generated by those two VSs when accessed with an HTTP post.

Figure 5.4 shows the sequence diagram of a successful schedule transmission. When the control algorithm VS decides to generate a new schedule after receiving a cleaned sensor value it sends the new schedule to the schedule VS and wakes up the *Core Station*. The wake-up of the *Core Station* is done via the Dozer command VS. A Dozer command is sent to the *Base Station* from where it is forwarded over the WSN to the destination (see Figures 5.2 and 5.3). In our case, the destination is the *Core Station* the actuator is connected to. The reception of a Dozer command is only acknowledged by the connected *Base Station*(s) but not by the destination (*Core Station*) of the command. That is, there is no direct feedback about the reception of the wake-up command and therefore you never know if the command reached its destination or if it is lost during transmission. For this reason the Dozer command acknowledge is ignored. The sending of the schedule disposes of a feedback mechanism. First of all, the schedule VS acknowledges the reception of the new schedule. As soon as the schedule has arrived at the *Core Station*, a second acknowledge is generated by the schedule VS (schedule transmitted). The time between the sending of the schedule including the wake-up command and the acknowledgment of the schedule by the *Core Station* may be in the order of minutes. The reason for this is: (i) the propagation time of the Dozer wake-up command (depends on the current WSN topology), (ii) the startup time of the *Core Station* after receiving a wake-up command (approx. 2 minutes), (iii) the time until the GSN connects to the *Core Station*, and (iv) the time until the *Core Station* requests for a new schedule.

The Dozer command used is a power control command (command id: `14`, command: `GUMSTIX_CTRL_CMD`) with the argument `2` which starts the *Core Station*. For more details about the Dozer command see chapter *TinyOS Dozer Beacons* in [29]. Because the *Core Station* is in duty cycle mode, after checking for a new schedule it is automatically shutdown as soon as it does not have anything to do. Therefore, after sending the power control command, no additional commands have to be sent to the *Core Station* in order to shut it down.

Figure 5.5 shows the sequence diagram of a failed transmission. If the schedule is not acknowledged by the schedule VS within the timeout, the schedule is retransmitted. Also the schedule transmission acknowledge, i.e. the confirmation that the schedule was not only received by the schedule VS but also by the

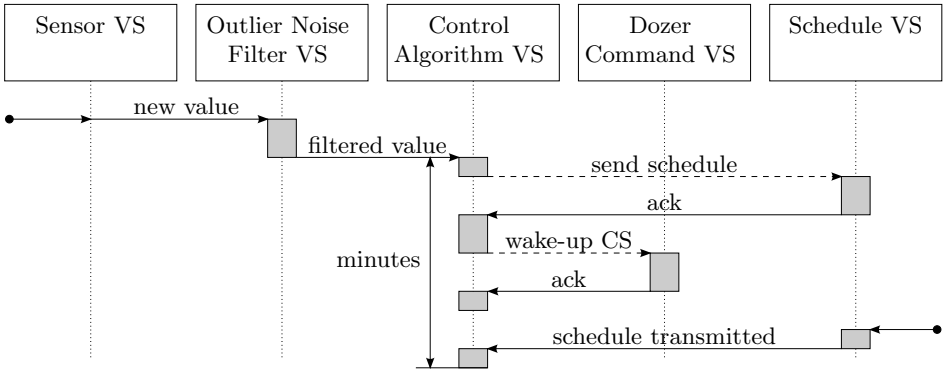


Figure 5.4: Sequence diagram of successful sending of a new schedule. Arrows with a solid line corresponds to the flow of streaming data within GSN, dashed lines show the data flow via HTTP posts.

Core Station, features a retransmission mechanism. If the schedule transmission is not acknowledged within a timeout, a new wake-up command is sent to the Dozer command VS. In this case the schedule does not have to be sent again because it is stored in the schedule VS, unless it needs to be modified.

5.2.3 Control Algorithm VS

5.2.3.1 Introduction

The aim of the control algorithm is to control the 6712 water sampler based on sensor values. Specifically, for a first system test the sensor is a single electrical conductivity (EC) sensor. As soon as the difference between the minimum and maximum electrical conductivity (EC), determined over a period of 12 hours, exceeds a certain threshold for more than 10 minutes (hold time), a static sampling schedule is generated and send to the actor. It must be taken into account that sensor values may be delayed or lost.

5.2.3.2 Data Processing

The GSN system provides aggregate functions which allow to calculate the minimum and maximum EC over a certain period [19]. For the calculation of the aggregate function, the GSN system uses the *timed* time information (see Figure 2.5) as the time reference. Hence, the result of the aggregate function may be wrong, especially in case of long delays between the generation of the data and its arrival at the GSN server (see Section 7.1.3.2). For this reason, the control algorithm needs to buffer all data of interest internally and apply its own aggregate function with the *generation time* as the time reference, i.e. the time

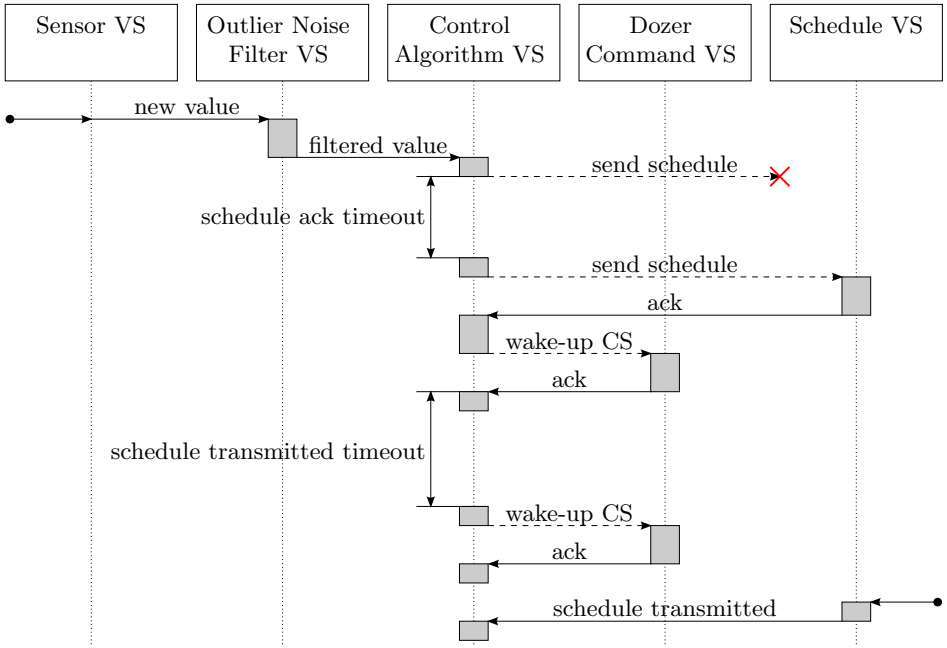


Figure 5.5: Sequence diagram of failed sending of a new schedule. Arrows with a solid line corresponds to the flow of streaming data within GSN, dashed lines show the data flow via HTTP posts.

when the data has been generated. If the VS delivering the sensor values is on the same GSN server, then the buffer is filled with historical data during the initialization of the control algorithm VS. Figure 5.6 illustrates how a new sensor value is processed. Only sensor values generated within the 12 hour window + hold time period are stored in the internal buffer, all older sensor values are skipped. Because of the limited buffer size, the oldest stored values are removed if the buffer is full.

After storing the new sensor value, the sensor values in the buffer need to be analyzed. There has to be a minimum number of sensor values within the hold time period. If this is not the case, the sensor values are not evaluated (see Figure 5.6). If enough values within the hold time period are available, then for each sensor value within the hold time period, the minimum and maximum over its preceding sensor values is calculated. Figure 5.7 illustrates with an example the sensor values stored in the internal buffer as well as the sensor values used for the minimum/maximum calculation of a single sensor value (red square) within the considered hold time period. If the start condition is fulfilled for all sensor values within the hold time period, a new sampling schedule is generated and send to the actor. The sampling schedule is generated according to the parameters in the virtual sensor description (VSD) file (see Section 5.2.3.3).

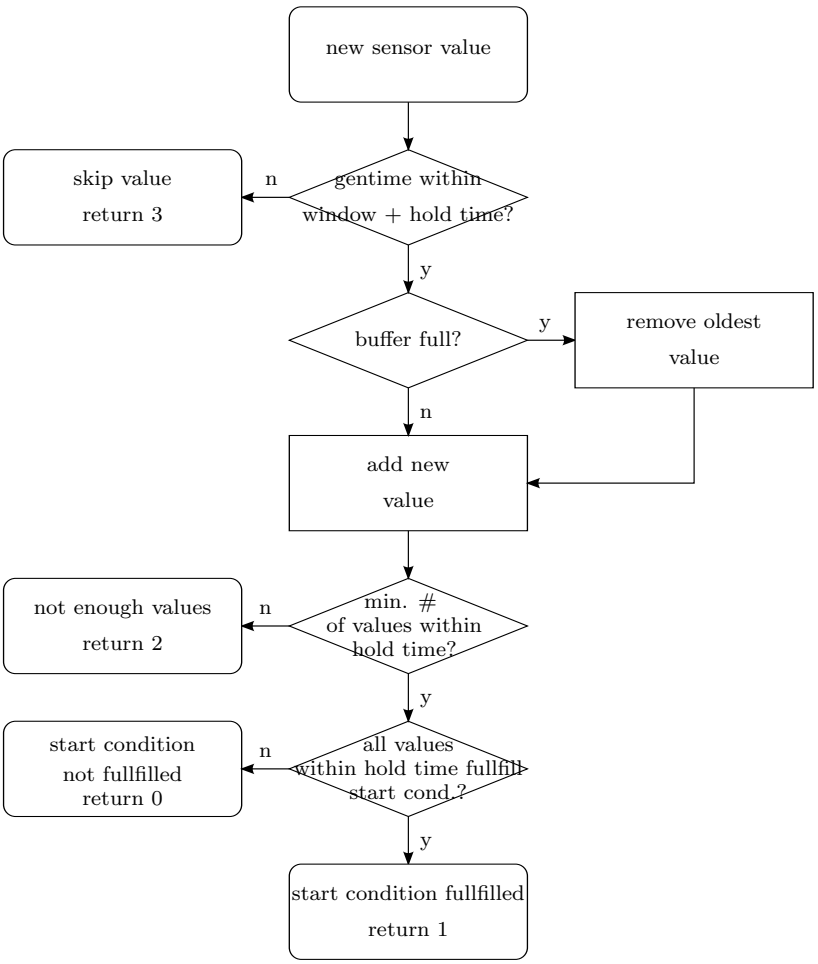


Figure 5.6: Flow chart of the processing of a new sensor value.

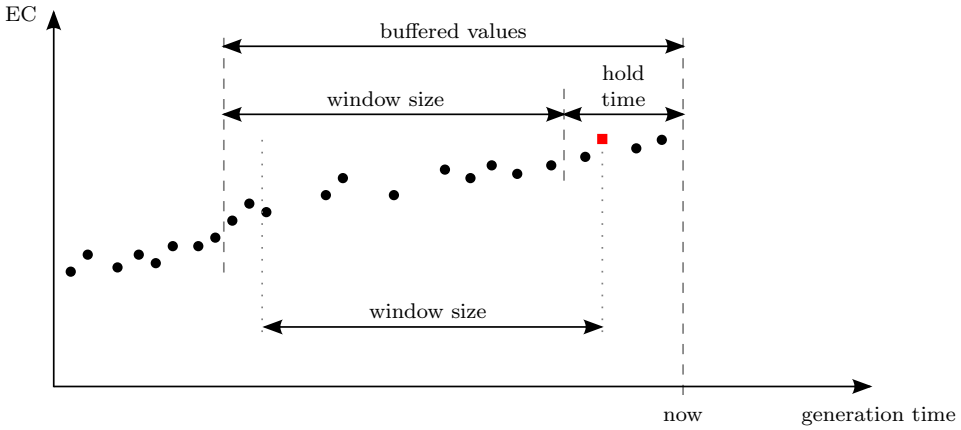


Figure 5.7: Data evaluation example to determine whether start condition is fulfilled or not.

The control algorithm can be enabled/disabled via the GSN web interface (see Appendix A.1.2). If the control algorithm is disabled, no schedule is sent to the actor. In case of an enabled control algorithm and a fulfilled start condition, the state machine in Figure 5.8 changes from the **Idle** state to the **Schedule Generation** state where a new sampling schedule is generated and sent to the schedule VS. This transition can also be triggered by a manual start initiated by an upload on the GSN web interface (see Appendix A.1.2). After the schedule VS has acknowledged the reception of the new schedule, the state **Wait for Transmission** is entered and a wake-up beacon is sent to the *Core Station*. As soon as the new schedule has arrived at the *Core Station*, the state **Schedule Transmitted** is entered and the control algorithm is disabled. This state is left when the GSN is restarted or a manual schedule reset is initiated on the GSN web interface.

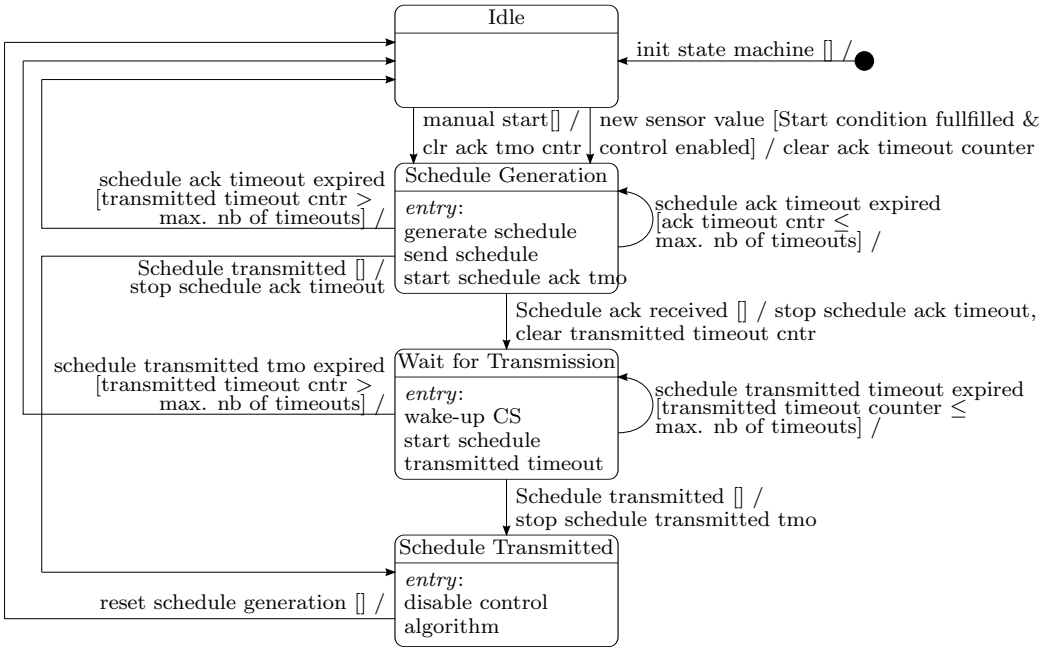


Figure 5.8: State diagram of the control algorithm VS. The transition arrows are encoded as follows: event [condition] / action. The filled circle marks the starting point.

5.2.3.3 Virtual Sensor Description File

The VSD file (*.xml) allows to set some parameters for the control algorithm VS. Listing D.3 shows such a VSD file for the control algorithm VS. The different values used for the sensor data evaluation are stored as parameters in the VSD file (threshold, hold time, window size, minimum number of sensor values within hold time etc.). Also the timeout for the wake-up of the *Core Station* as well as the number of retries can be parameterized. Additionally, some information required

for the generation of the sampling schedule, such as delay of first sampling and the sampling scheme, need to be parameterized. The sampling scheme is configured with a tuple of four elements: start bottle number, stop bottle number, interval in minutes, sampling volume in milliliter. When the start condition is fulfilled, the very first sample is not taken after the interval time but after the configured start delay. It is possible to add multiple sampling schemes by separating them with a semicolon. Listing 5.1 shows a configuration example of a possible sampling schedule and Figure 5.9 shows the resulting sampling series.

Listing 5.1: Configuration of sampling schedule in VSD file.

```
1 <param name="sampling_start_delay_in_min">20</param>
2 <param name="sampling_scheme">1,4,10,100;5,8,30,100</param>
```

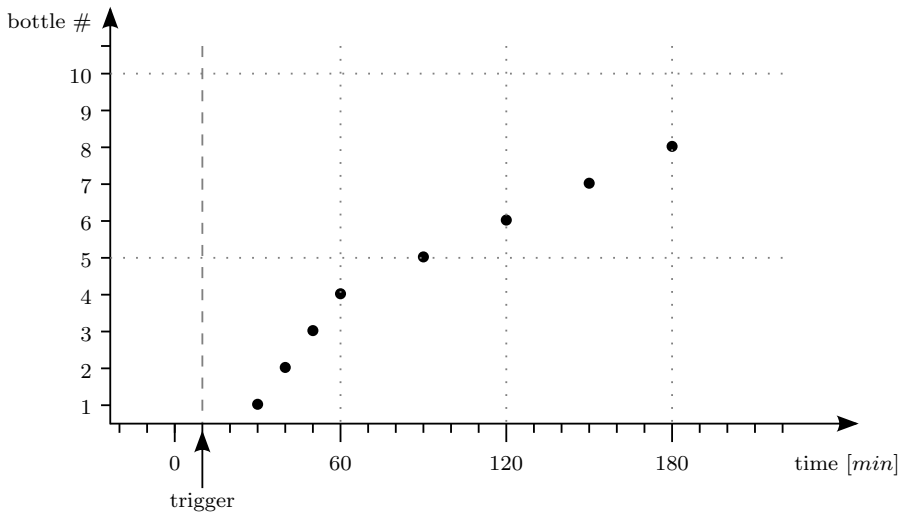


Figure 5.9: Sampling series as configured in Listing 5.1.

Because the schedule VS and the Dozer command VS are accessed via HTTP post, the host names as well as the VS names of those two VS need to be configured.

The control algorithm VS receives data streams from different VS which have to be processed differently. Therefore, the VS requires a method to distinct between the different streams. This differentiation is done by means of the stream name assigned in the VSD file (see Table 5.1 and line 62, 76, and 88 in Listing D.3).

Table 5.1: Stream names used in the VSD file of the control algorithm VS.

Stream Name	Virtual Sensor
sensor	VS delivering the sensor values
schedule	Schedule VS the schedule state is received from
dozer_command	Dozer command VS the wake-up beacon acknowledge is received from [1]

[1] This information is currently not used by the control algorithm VS.

The sensor value to be used by the control algorithm needs to be renamed with an SQL alias to `sensor_value` in order to be clearly detected by the VS (see line 69 in Listing D.3). The control algorithm VS generates data of different types: (i) sensor value evaluation data, and (ii) schedule file generation data. Because a VS can have only a single output structure, the differentiation is done by means of the `DATA_TYPE` field. This field can be used to demultiplex the output structure by other VSs (see Section 5.2.5). Table D.3 shows the output structure of the control algorithm VS.

5.2.4 Outlier and Noise Filter VS

5.2.4.1 Introduction

The aim of the outlier and noise filter is to reduce the number of outliers and the noise. The outlier cleaning is done with an approach presented in Section 6.2.3. A simple mean filter is used to filter the noise. Considering the implementation of both filters, it must be taken into account that sensor values may be delayed or lost.

5.2.4.2 Data Processing

Similar to the control algorithm VS (see Section 5.2.3), the outlier and noise filter VS also features an internal buffer for holding historical data. If the VS delivering the sensor values is on the same GSN server the buffer is filled with historical data during the initialization. The data cleaning is a two stage process. In a first step, the outliers are replaced by a more likely value and in a second step a noise filter is applied. For the filtering process, the relevant time information is the *generation time*. A new sensor value is evaluated only with respect to older sensor values.

For the outlier cleaning, only the buffered raw sensor values within the window width, with respect to the sensor value to be evaluated, are considered (see Figure 5.10). A minimum number of sensor values within the window is required.

If this is not the case, then the new sensor value is not classified. If more than the minimum number of sensor values are within the window, the sensor value is classified according to the outlier filter presented in Section 6.2.3. That is, if the absolute difference between the new sensor value and the median of the sensor values within the window, including the new sensor value, is smaller or equal than a certain threshold, then the new sensor value is classified as non-outlier. If the difference is more than the threshold, then the new sensor value is classified as outlier. An outlier is replaced by the last sensor value, with respect to the new sensor value, classified as non-outlier. If no such non-outlier-classified sensor value exists, the outlier is replaced by the calculated median.

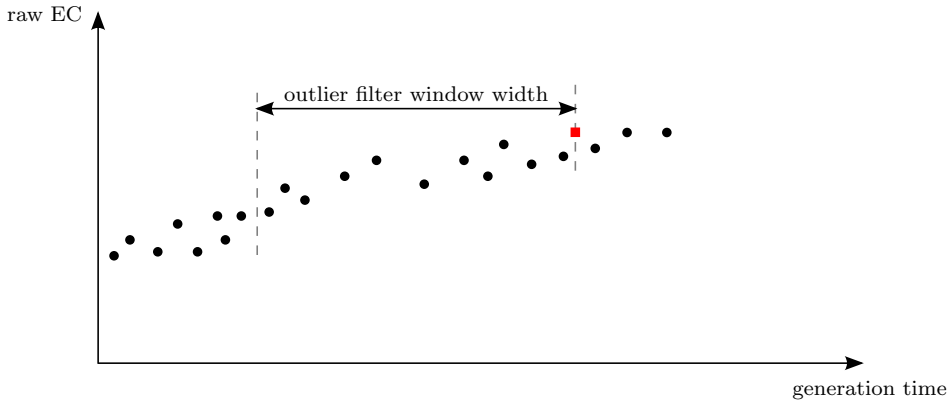


Figure 5.10: Outlier filtering example to determine whether the sensor value is an outlier or not. The red square represents the sensor value to be evaluated.

The noise filter applied after the outlier filter only takes into account values from a certain window width with respect to the sensor value. This window width may be different than the window width used for the outlier filter. The noise filter only considers outlier filtered sensor values, i.e. sensor values which were classified. In contrast to the outlier filter, the noise filter does not require a minimum number of sensor values within the considered window.

After the completion of both filtering steps, the new sensor value is stored in the internal buffer including its classification and filtered values. Furthermore, a new data stream is generated so that it can be further processed by other VSs, such as the control algorithm VS.

5.2.4.3 Virtual Sensor Description File

The VSD file contains some parameters for the outlier and noise filter. Listing D.6 shows such a VSD file for the outlier and noise filter VS. The size of the buffer holding historical sensor data can be configured in that file. Also the type of outlier and noise filter are parameterizable. The applied noise filter requires some configuration values such as threshold, window width and the minimum

number of values within the window. Also the window width of the noise filter can be configured in the VSD file.

In order to be flexible in terms of sensor data to be filtered, the outlier and noise filter VS expects the sensor value to have the stream element name `raw_value`. If the stream element name of the sensor data to be filtered has a different name, which it certainly will, then it can be renamed with an SQL alias to `raw_value` (see line 41 Listing D.6). The outlier and noise filter VS does not only generate a single output but also the intermediate results of the filtering process. Table D.4 shows the output structure of the outlier and noise filter VS.

5.2.5 Other VSs

The status VS and sampling VS receive data streams directly from the actor. The sampling VS shows the result of a sampling executed by the 6712 water sampler. Its output structure is illustrated in Table D.2. A sampling can be initiated either by the scheduler or by an upload on the GSN web interface (see Appendix A.1.4.2). The status VS shows the current status of the 6712 water sampler. It contains some configuration data of the 6712 water sampler, the *Menu Control* mode state (see Section 4.6.1.1) and the *External Program Control* mode state (see Section 4.6.1.2). Its output structure is illustrated in Table D.1. A daily status report can be configured in the schedule file or manually requested by an upload on the web interface (see Appendix A.1.4.4).

The control algorithm configuration and evaluation VSs are used to demultiplex data generated by the control algorithm VS. That is, they show data which are derived from the data stream of another VS. The demultiplexing is done based on the field `data_type`.

5.3 Actor

The actor consist of the interface unit (*Core Station*) and the actuator (6712 water sampler, hereafter referred as 6712). As already mentioned in Section 3.3, in a first step the water sampler is integrated by means of a powerful *Core Station* featuring a WSN as well as a WLAN interface. Although the 6712 is able to run its own sampling program, the 6712 is controlled by the *Core Station* in a way so that it only takes samples when it is told to do so. That is, the 6712 does not execute any samplings on its own initiative. The *Core Station*, in turn, is controlled by means of a schedule file containing the necessary sampling information (time, bottle, and volume). As soon as it is time to take a sample, the *Core Station* triggers the 6712 to draw a sample.

5.3.1 Bottle Change

Because the 6712 water sampler's sampling is controlled by the *Core Station*, a procedure is required to prevent automated sampling while the user collects the bottles. That is, the user being on-site needs to inform the *Core Station* that he intends to collect the 6712's bottles. As the *Core Station* does not feature an on-site user-interface, the 6712's internal state is used for this.

Before the bottles can be collected, the user needs to stop the running program by pressing the STOP key on the 6712's keypad. By the stopping of the running program, the 6712's *Menu Control* state will change into one of the states in the green box of Figure 5.11, labeled with *Collecting Bottles*. Now, the user can safely collect the bottles because no sampling commands are send by the *Core Station* as long as the 6712 is not in the **Sampler Off** or **Program Running (disabled)** state. As soon as the bottles have been collected and replaced by empty bottles, the user has to turn off the sampler to reactivate the remote sampling. When the *Core Station* detects at a later point in time, that the sampler has been turned off (state **Sampler Off**), the sampler is initialized and the *Core Station*'s internal bottle states are reset.

5.3.2 Initialization

Before the *Core Station* initiates a sampling, it first checks the 6712's *Menu Control* mode state. When the sampler is in the state **Sampler Off**, the 6712 needs to be initialized by the *Core Station* before a sample can be taken. The following 6712 settings are initialized:

- Number of bottles
- Bottle volume
- Suction line length
- Suction head
- Suction line rinses
- Sampling retries
- Start time
- Report configuration

After the initialization, the 6712 is set into the *Menu Control* mode state **Program Running (disabled)**. This state can only be reached by means of remote operation and therefore allows the *Core Station* to verify if the 6712

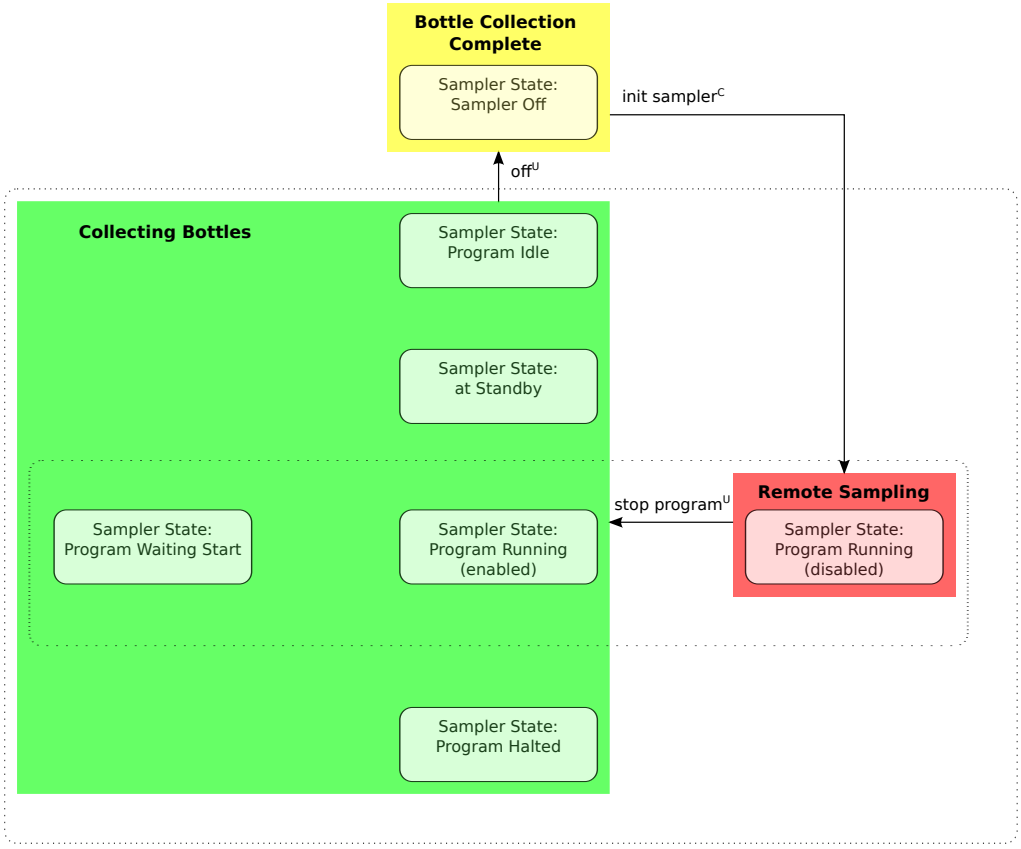


Figure 5.11: 6712 water sampler's internal menu control state transitions. ^{U} = user interaction, ^{C} = *Core Station* interaction.

has already been initialized before. The start time of the 6712's program is always configured to `WAITING FOR PHONE CALL`. This allows to disable the 6712's internal program before it is started. Although the 6712's internal program is disabled, the 6712 accepts remotely triggered samples. Furthermore, those remotely triggered samples are stored in the sampling result report (see Figure 4.4(a)). The **Program Running (disabled)** is reached by the following program sequence executed by the *Core Station*:

- run program (when 6712 is in the *Remote Control of Sampler Keypad* mode)
- disable program (*Menu Control* mode command `DISABLE`)
- start program waiting for call (*Menu Control* mode command `START`)

5.3.3 Sampling

Sampling is only allowed when the sampler is in the *Menu Control* mode state **Program Running (disabled)** or **Sampler Off**. If the sampler is not in one of those two states and a sample needs to be taken, the sampling is reported to the GSN to have failed. All samples are triggered by the *Core Station* by means of the *External Program Control* command `BTL,x,SVO,x<CR>` (see Section 4.6.1.2). This command does not only take a sample but also, when configured, rinses the suction line and retries if sampling fails. Furthermore, the sample events are logged in the 6712's sampling result report when a 6712 program is running. The sampler's *External Program Control* mode state is periodically read by means of the `STS<CR>` command while the sampler is taking a sample. As soon as the most recent sample result (`SOR` identifier) and the sampler's current status (`STS` identifier) is not anymore `SAMPLING IN PROGRESS`, the sampling is complete and the result is sent to the GSN.

5.3.4 Software-Design

5.3.4.1 Overview

The backlog software running in the *Core Station* is responsible to initialize the actuator, that is, the 6712 water sampler, and to initiate samplings. The functionality to access the 6712 is implemented in a plugin (Sampler6712Plugin.py). To save power, the backlog software runs in the duty cycle mode so that the *Core Station* is off if no sample has to be taken. In the following, we refer to the Sampler6712Plugin as software, unless otherwise mentioned. The backlog software is programmed in Python.

5.3.4.2 6712 Sampler Plugin

The behavior of the software is controlled by the tasks in the task queue (see Figure 5.12). Multiple sources can put new tasks into the task queue: (i) scheduler (schedule plugin Section 5.3.4.3), or (ii) GSN message generated through the GSN web interface.

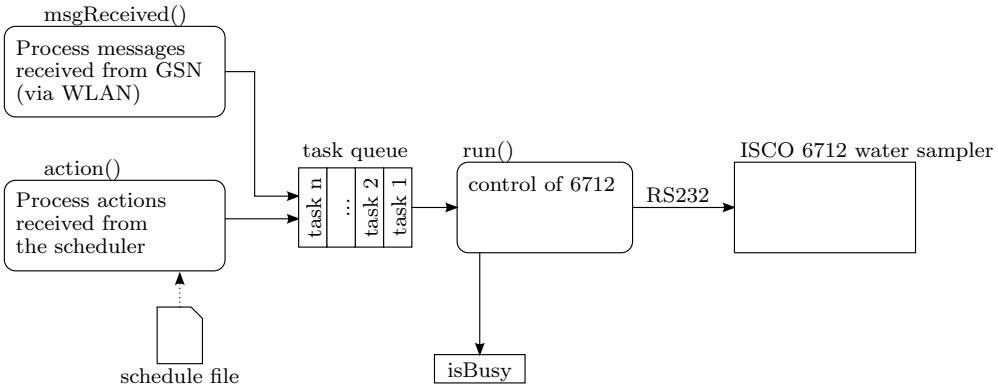


Figure 5.12: 6712 plugin overview.

The queue element is defined as a list where the first element is the task origin and the second element is the task type (Table 5.2). The following list-elements are optional parameters and depend on the task type.

Table 5.2: Definition of queue element.

Task Origin	Task Type	Parameters
[1]	Take sample	Sampling object [2]
[1]	Reinit 6712	-
[1]	Report status	-

[1] Scheduler or GSN
[2] The sampling object contains the bottle number and the sampling volume in *ml*

In the following some software parts of the plugin are described. For more detailed information we refer to the source code.

Task Take Sample If a `Take Sample` task has been received, the behavior of the software depends on the status of the 6712 obtained in the *Menu Control* mode (see Figure 4.6). It is noted that there is also another status which can be remotely obtained, the *External Program Control* mode status (command `STS,1<CR>`). A sampling task is only forwarded to the 6712 if it is in the status **Program Running** and the program's state is **disabled** (see Algorithm 1).

If the sampler is turned off (status **Sampler Off**), the 6712 is first initialized before a sampling is initiated. For all other states, the sampling is skipped.

Algorithm 1 6712 sampler plugin (`run()` function)

```
1: while plugin not stopped do
2:   if task queue is empty then
3:     isBusy = False
4:   end if
5:   read task queue (blocking)
6:   isBusy = True
7:   if task type == Take sample then
8:     read samplerState and programState (menu control mode)
9:     if samplerState == Program Running then
10:      if programState == disabled then
11:        take a sample
12:      else if programState == enabled then
13:        skip sampling
14:      else
15:        raise exception (invalid program status)
16:      end if
17:    else if samplerState == Sampler Off then
18:      initialize 6712
19:      take a sample
20:    else
21:      skip sampling
22:    end if
23:  else if task type = Reinit 6712 then
24:    initialize 6712
25:  else
26:    warning: unknown queue element
27:  end if
28: end while
```

When a sampling is possible according to the sampler and program state the bottle number is verified first (Algorithm 2). If the bottle number is valid, the bottle capacity is also checked. The water level of each bottle is tracked in the software. This allows to prevent overfilling bottles. If the bottle number is valid and the bottle is capable to collect the new sampling, a sampling command is send to the 6712. This sampling command (`BTL,x,SVO,y<CR>`) is send in the remote operation mode *External Program Control*. After sending the sampling command, the sampler's *External Program Control* mode status is periodically requested (`STS,1<CR>`) as long as either the sampler state or the last sample result is `SAMPLE IN PROGRESS`. As soon as the sampling is complete, the internal water level of the filled bottle is update, independent of the sampling result. As a

final action, the sampling result is sent to the backend system. The information contained in the GSN message is listed in Table D.1.

Algorithm 2 Take a sample

```
1: if bottle number is not valid then
2:    samplingResult = skipped because of invalid bottle number
3: else
4:    if bottle has capacity to carry new sample then
5:       send sample command to 6712 and read state (BTL,x,SVO,y)
6:       if samplerState == SAMPLE IN PROGRESS then
7:          while samplerState == SAMPLE IN PROGRESS or
            mostRecentSamplingResult == SAMPLE IN PROGRESS do
8:             read samplerState and mostRecentSamplingResult from 6712
9:          end while
10:         update bottle water level
11:         samplingResult = done
12:      else
13:         samplingResult = skipped (for reason see samplerState)
14:      end if
15:   else
16:      samplingResult = skipped because of exceeded bottle capacity
17:   end if
18: end if
19: send samplingResult and other data to GSN
```

Task Reinit Sampler The task `Reinit Sampler` can be used to remotely force a reinitialization of the 6712 (see Appendix A.1.1). This command is useful when the 6712 was not manually turned off after it has been stopped to collect the bottles. The initialization procedure is the same as for the regular initialization executed when the 6712 is in the **Sampler Off** state.

Task Report Status The task `Report Status` can be initiated either by an upload on the GSN web interface or by the scheduler (see Appendix A.1.4.4). An example of a schedule file entry is shown in Listing 5.2. The `Report Status` task reports the different states of the 6712 as well as some configuration of the 6712 to the GSN. The following list shows a general overview of the information contained in the status report. For a more detailed list we refer to Table D.2.

- Sampler state (*Menu Control* mode)
- Program state (*Menu Control* mode)
- Program state (*External Program Control* mode)
- Program configuration

Listing 5.2: Schedule file entry for periodic status report.

```
1 0  9  *  *  *  plugin Sampler6712Plugin report_status
```

Actuator Initialization The extended programming level allows to configure more settings than the standard programming level (see Section 4.2). Therefore the 6712 is set into the extended programming level before it is configured. Because only some configuration of the 6712 are relevant, the programming style *Quick View* is used for the configuration of the 6712 (see Section 4.6.1.3). The *Quick View* allows a faster and more flexible configuration of the 6712 than the *Normal* mode. The programming style is essential for the navigation in the *Remote Control of Sampler Keypad* mode and therefore it has to be set before the actual configuration starts. As soon as the 6712 is in the *Quick View* programming style, the configuration starts. At first, partial configuration of the *Software Option* menu, such as pump purge counts, are carried out. Afterwards some settings in the *Program* menu, such as number of bottles or bottle volume, are configured by navigating through the 6712's menu (see Figure 4.3(a)). After the configuration has been completed, the 6712's program is run, then disabled, and finally started. This procedure ensures that the 6712 is in the state **Program Running** with the program state **disabled** after the initialization. During the initialization, the remote operation mode is changed multiple times because for different actions different remote operation modes are necessary.

Determination of Remote Control Mode Different remote control modes are used for the communication with the 6712. Therefore it is important to reliably determine the 6712's remote operation mode. The remote control mode is determined by sending `?` to the 6712. Because the remote operation modes *Offline* and *Sampling Reports* (see Figure 4.5) are left when sending `?`, only three modes are possible: (i) *External Program Control*, (ii) *Menu Control*, and (iii) *Remote Control of Sampler Keypad*. The mode *Remote Control of Sampler Keypad* is determined by sending a `S` or a `Q`. Therefore (i) the current menu in the *Remote Control of Sampler Keypad* mode is modified (`S`), or (ii) the *Remote Control of Sampler Keypad* is quit to enter the *Menu Control* mode (`Q`) through the determination of the remote control mode. Figure 5.13 shows the program flow of the remote control mode determination.

Remote Control of Sampler Keypad The *Remote Control of Sampler Keypad* mode is designed for human navigation and not for machine controlled navigation. When the software communicates with the 6712 in the *Remote Control of Sampler Keypad* mode, it needs to act like a VT100 terminal emulator. As a VT100 terminal emulator, the software has to interpret the incoming data and store it somewhere if it is intended for further use. In our case, the incoming

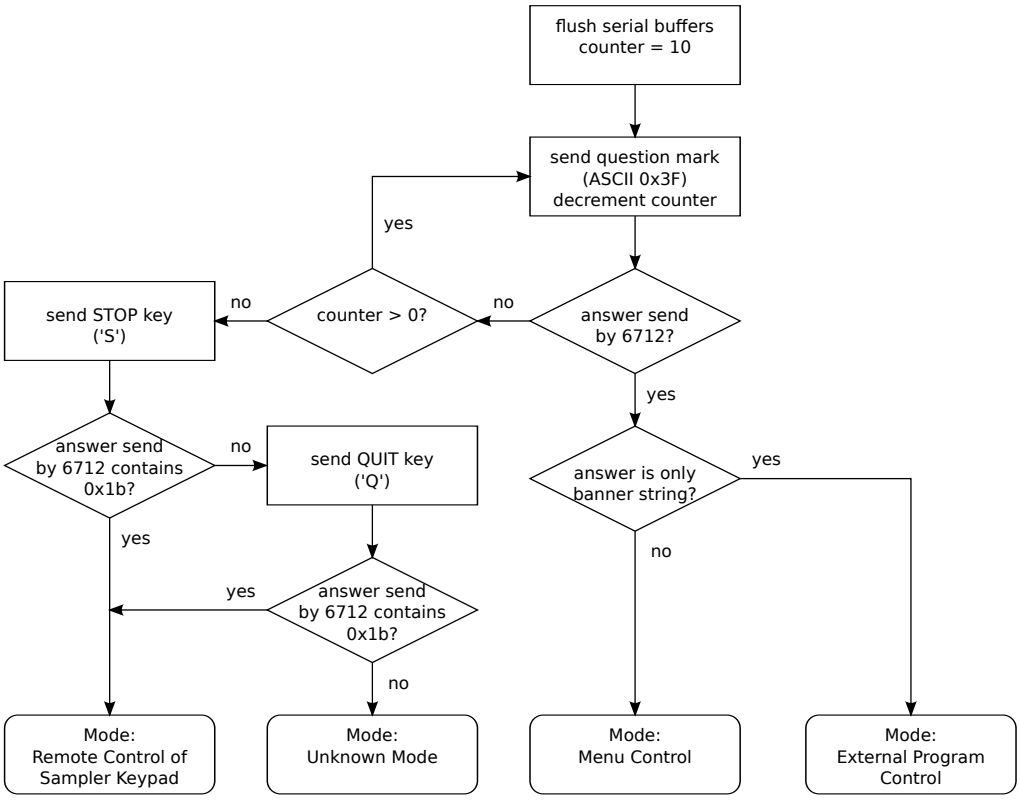


Figure 5.13: Determination of remote control mode.

data is stored in a virtual display (i.e. virtual because the display information is available in some memory but is not humane readable). Not all incoming data is available on the virtual display. Control commands cause the terminal emulator to change the character representation, move the cursor, etc. Only a few control characters and control sequences from the VT100 are used by the 6712. Other control commands than those listed in Tables 4.5 and 4.6 are ignored and cause an error. The virtual display is a 4×20 matrix similar to the 4×20 character display on the 6712. Each matrix element has an assigned character and a character attribute. Valid character attributes are: (i) no character attribute, (ii) blinking attribute and (iii) reverse attribute. Figure 5.14 illustrates an example of a display representation.

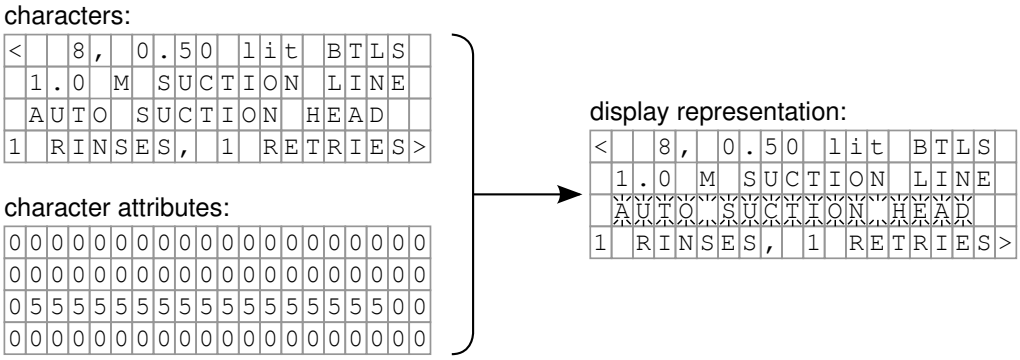


Figure 5.14: Each element on the 4×20 matrix has a character and a character attribute (left). Combining those information will result in the same display representation as the display on the 6712 (right). Here the `AUTO SUCTION HEAD` part is blinking because its character attribute is 'blinking' (`5`, see Table 4.6).

The virtual display is used to navigate through the menu. The navigation through the menu is configured by means of a navigation object which allows a flexible navigation procedure. Each navigation object has six attributes:

- search text (list)
- search type (screen or selection)
- found action
- maximum number of found actions
- found expiration action
- not found action
- maximum number of not found actions
- not found expiration action

The **search text** is a list with strings the navigator should navigate to. In case of **search type** 'screen', the navigator tries to navigate to the screen

containing all elements of the **search text** list. If the **search type** is set to 'selection', only characters with the character attribute 'blinking' are considered. The **found action** attribute defines the action to be executed when the **search text** was found and the **maximum number of found actions** has not been reached yet. If the **search text** has been found and the **maximum number of found actions** is reached, then the **found expiration action** is executed. The behavior is analog if the **search text** has not been found, but with **not found action** and **not found expiration action** as actions and **maximum number of not found actions** as maximum counter value. The different action attributes can be of varied type:

- **String** to send to the 6712
- Another **navigation object**
- **Delay object** defining a delay before the virtual display is again tested on the **search text**
- **None** if no action should be done
- **Exception** to be raised

The navigation procedure is listed in Algorithm 3.

5.3.4.3 Scheduler Plugin

The sampling of the 6712 is controlled by means of the schedule plugin. That is, the times when to take a sample have to be specified in the cron-like schedule file (see Listing 5.3). Each line in the schedule file corresponds to a job which consists of a time expression, the keyword `plugin` followed by the plugin name and optional parameters. The scheduler calls the plugin's `action()` function with the optional parameters, when the specified time is reached. For the Sampler6712Plugin the parameters are the bottle number (`bottle(x)`) to use for the sampling and the sampling volume in milliliter (`volume(y)`). If the loading of the schedule plugin during the backlog boot process is not completed before a new job in the schedule file needs to be executed, the job is discarded. To prevent this, the option `backward_tolerance_minutes` enables the scheduler to execute jobs which have expired by the defined number of minutes. The time when to generate a status report of the 6712 can also be configured in the schedule file. For an example see Listing 5.2.

Listing 5.3: Schedule file.

```
1 0  19  11  2   *  plugin Sampler6712Plugin bottle(9) volume
    (10) backward_tolerance_minutes=5
```

Algorithm 3 Navigation

```
1: foundCounter = maximum number of found actions
2: notFoundCounter = maximum number of not found actions
3: while foundCounter ≥ 0 and notFoundCounter ≥ 0 do
4:    update virtual display according to incoming data
5:    read virtual display text (according to search type)
6:    if all search text elements are available in the read display text then
7:       if foundCounter ≤ 0 then
8:          action = found expiration action
9:       else
10:         action = found action
11:      end if
12:      decrement foundCounter
13:   else
14:      if notFoundCounter ≤ 0 then
15:         action = not found expiration action
16:      else
17:         action = not found action
18:      end if
19:      decrement notFoundCounter
20:   end if
21:   do action
22: end while
```

5.3.4.4 Backlog Startup

The backlog software runs in duty cycle mode. That is, the *Core Station* is turned off when nothing has to be done and turned on when a new schedule job needs to be executed. When the *Core Station* is shut down, the backlog software running on the *Core Station* is booted a defined time before the next schedule job needs to be executed. This time can be configured in the configuration file with the entry `approximate_startup_seconds = 120`. To enable the starting of the Sampler6712Plugin during the backlog startup the configuration file entry `Sampler6712Plugin = 1` is required. If the plugin does not have to accomplish any tasks (that is, it is not busy), and all the other plugins are finished, it is stopped at the earliest after the minimum runtime (configuration file entry `min_runtime`). Figure 5.15 shows the backlog startup (solid line) followed by a take sample task (dashed line). If the 6712 has not been initialized before the take sample task, the take sample task is preceded by the initialization of the 6712.

If the plugin's minimum runtime expires before the scheduler executes the plugin's `action()` function, the plugin may already have been stopped so that the

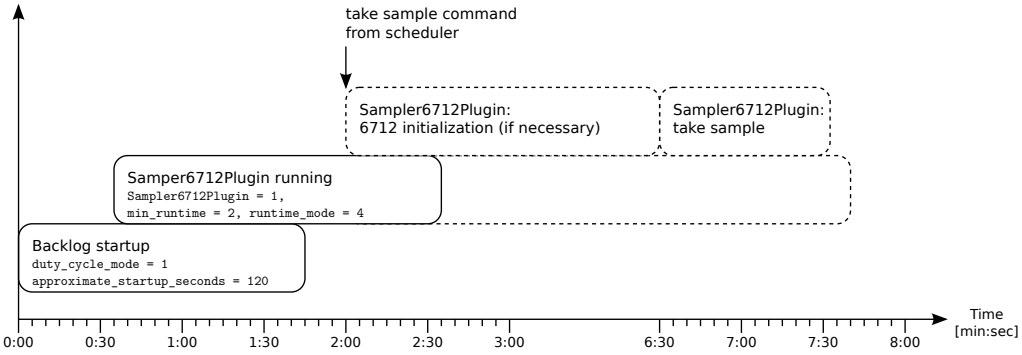


Figure 5.15: Backlog startup. The solid line corresponds to the actions/states always executed during the backlog startup. The dashed line are the actions/states if additional tasks have to be executed. The time information are not accurate but give a rough overview of how long the individual tasks last.

plugin needs to be loaded again. This behavior can be prevented by configuring the minimum runtime (`min_runtime`) greater than the approximated startup time (`approximate_startup_seconds`).

5.3.4.5 Config File

Some relevant entries for the Sampler6712Plugin in the backlog configuration file are listed in Listing 5.4. The backlog software runs in the duty cycle mode (`duty_cycle_mode = 1`). To allow for the reception of GSN messages, the Sampler6712Plugin is always started during the backlog startup (`Sampler6712Plugin = 1`) and runs at least a minimum runtime (`min_runtime`). The options `device_name` and `baudrate` specify the serial port and its baud rate to be used for the communication with the 6712. The option `bottle_status_file` defines where the bottle state (i.e. current level of each bottle etc.) is stored.

Listing 5.4: Backlog configuration file `backlog.cfg`.

```
1  [options]
2  duty_cycle_mode = 1
3  ...
4  [schedule]
5  ...
6  approximate_startup_seconds = 120
7  ...
8  [plugins]
9  ...
10 Sampler6712Plugin = 1
11 ...
12 [Sampler6712Plugin_options]
13 priority = 10
14 backlog = True
```

```
15 runtime_mode = 4
16 min_runtime = 2
17
18 device_name = /dev/ttyUSB0
19 baudrate = 19200
20
21 bottle_status_file = /media/card/backlog/bottle_status
22 ...
```

5.3.4.6 Debugging

Serial Interface To simplify the debugging of the serial communication, a transfer loop can be set up allowing to log all communication (see Appendix G.2). It is important to disable the canonical mode (`icanaon=0`) in order to ensure that all data is transmitted immediately. Listing 5.5 shows how the transfer loop can be installed. This command can be executed manually or automatically (e.g. in the backlog file). The command has to be executed from a writable location (e.g. `/media/card`) so that all traffic is logged into the file `screenlog.0`. For the listed example, the port `/dev/ttyUSB0` would be mapped to the serial port `/tmp/myttyUSB0`. That is, the serial port to be used by the Sampler6712Plugin would be `/tmp/myttyUSB0`.

Listing 5.5: Schedule file.

```
1 screen -L -S serial socat -v -x PTY,link=/tmp/myttyUSB0,raw,
    echo=0,isig=0,icanon=0 /dev/ttyUSB0,raw,echo=0,isig=0,
    icanon=0
```

Report It is possible to remotely retrieve the sampling report (see Section 4.5). For this, first a remote connection to the *Core Station* via SSH needs to be established. To get access to the *Core Station*'s serial interface connecting the *Core Station* with the 6712, the backlog software running on the *Core Station* needs to be stopped. Afterwards a terminal, such as minicom or socat, can be started on the *Core Station* to enable the direct communication with the 6712. Then the report can be download as illustrated in Figure 4.5. The connection with the 6712 not only allows to see the report file but also enables other levels of computer control (see Section 4.6.1).

Chapter 6

Data Cleaning

6.1 Introduction

The sensor value used as input signal for the control of the actuator originates from an EC sensor (Decagon 5TE, see Appendix B). The sensor data does not always represent the physical process properly because the time-series data contains sensor value steps of more than $\pm$ $40\mu S/cm$ within a few minutes (see Figure 6.1) which is not possible in ground water. It is unclear why this phenomena occurs. Additionally, the sensor data contains EC drops in absolute value to approximately $0\mu S/cm$. The second case can be explained by the exposure of the sensor to the air. Apart from outliers, the sensor data contains also noise. It is not always obvious to differentiate between outliers and noise [30]. Before the data can be analyzed, outliers as well as noise needs to be removed as much as possible. We will tackle those two problems sequentially. First we will filter the outliers and then the noise.

6.2 Outlier Filter

Outliers appear in many different applications such as aerial surface data measurement [31], intensive care patient monitoring [32], and closed-loop control [33]. In literature different names, such as anomalies or exceptions, are used for outliers [30]. The survey [30] shows a broad overview of different anomaly detection techniques. We will focus on a rather simple technique which is based on a median filter, modified so that it can be applied on streaming data [33, 34]. In the following, the terms outlier and anomaly are used interchangeably.

The focus of the filter design is the removal of the relative step changes in the EC sensor data (see Figure 6.1). The relative step changes of the EC sensor value is a contextual anomaly, i.e. its value actually is not incorrect because it is still within the valid range of possible water EC, but in context with its temporal neighborhood, the sensor value is anomalous. We assume the outliers

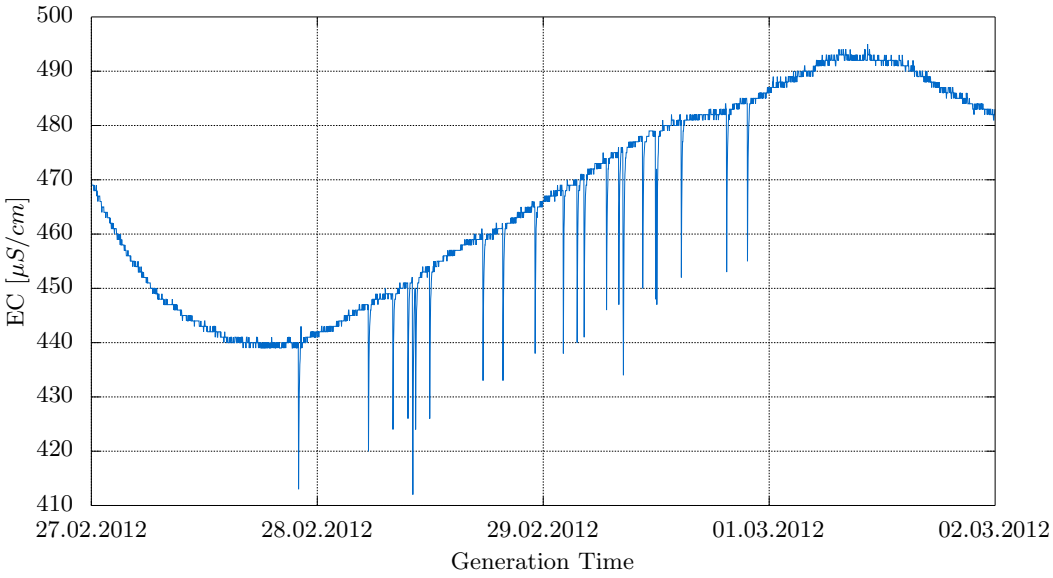


Figure 6.1: Raw EC values from the Decagon 5TE sensor at position 5 (R042), channel 0 of the Thur deployment.

to be additive. These additive outliers can be modeled with the additive outlier model [33]:

$$y_k = x_k + o_k \tag{6.1}$$

where y_k is the measured sensor value at time k, x_k the real but unknown physical value and o_k an additive which is mostly zero and a few times 'large' compared to the nominal variation of y_k.

6.2.1 Median Filter

A widely used filter is the nonlinear median filter [35]. The median of n values is defined as follows [35]:

$$med(y_k) = \begin{cases} y_{(v+1)} & n = 2v+1 \\ \frac{1}{2}(y_{(v)} + y_{(v+1)}) & n = 2v \end{cases} \tag{6.2}$$

where $y_{(i)}$ is the i-th value in the rank-ordered list. The corresponding median filter can be expressed as:

$$m_k = med(y_{k-v}, \ldots, y_k, \ldots, y_{k+v}) \tag{6.3}$$

where m_k is the filter output at time k and $y_{k-v} \ldots y_{k+v}$ is the considered data window. The breakdown point of the median filter is 50%, i.e. it can reject

up to 50% of outliers in the data window. If the variation of the data within the data window is high, the median filter leads to distortion [33] and the a linear trend is approximated by a step function [32]. Because in our application, the data is changing rather slowly, the main disadvantage of the median filter is the delay v introduced by the median filter when applied on streaming data.

6.2.2 Two- and One-Sided Median Filter

The outlier filter proposed in [34] consists of two steps: (i) the outlier detection, and (ii) the outlier handling, i.e. what to do with the outliers. They proposed two filters that are based on the median from neighboring data values: (i) two-sided median method, and (ii) one-sided median method. The two-sided median method considers the median of the neighboring data to determine whether a data value is an outlier or not. The neighborhood of a particular data value y_k at time k is defined as $\eta_k^{(v)} = y_{k-v}, \ldots, y_{k-1}, y_{k+1}, \ldots y_{k+v}$. If the difference between the consider data value y_k and the median of the neighborhood $m_k^{(v)} = med(\eta_k^{(v)})$ is larger than a specified threshold T, then y_k is declared to be an outlier. Figure 6.2 illustrates the two-sided median outlier detection method. This approach prevents distortion of the signal within the boundaries. But the drawback of the two-sided median method is the delay introduced by the consideration of the positive and negative neighborhood.

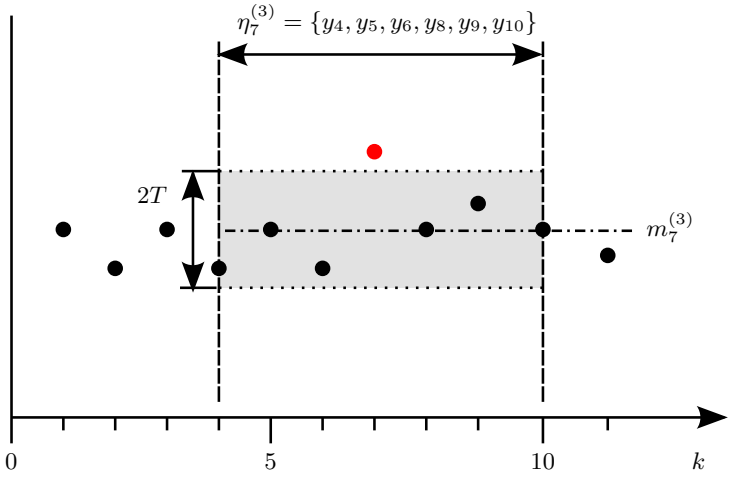


Figure 6.2: Example of two-sided median method [34]. The shaded area illustrates the threshold T. In this example y_7 is declared as outlier.

To overcome this drawback, a one-sided median method which considers only the current and old data values was also proposed [34]. Similar to the two-sided median method, the one-sided median is defined as $\tilde{m}_k^{(y)} = med\{y_{k-2v}, \ldots, y_{k-1}\}$. Additionally, the first difference of the considered data series y_k is calculated: $z_k = y_k - y_{k-1}$. Based on the median of the first difference $\tilde{m}_k^{(z)} = med\{z_{k-2v}, \ldots, z_{k-1}\}$

and the median of the observed time series $\tilde{m}_k^{(y)}$ a prediction for y_k is calculated. The prediction $\tilde{m}_k^{(2v)}$ is calculated as follows: $\tilde{m}_k^{(2v)} = \tilde{m}_k^{(y)} + v \cdot \tilde{m}_k^{(z)}$. If the difference between the prediction $\tilde{m}_k^{(2v)}$ and the data value y_k is larger than the threshold $\tilde{T}$, i.e. $|y_k - \tilde{m}_k^{(2v)}| \geq \tilde{T}$, then y_k is replaced by the prediction $\tilde{m}_k^{(2v)}$, else y_k is left unmodified.

6.2.2.1 Simulation

The following simulation was performed with Matlab based on GSN data from the Decagon sensor 5TE at position 3 (R073), channel 4 (conductivity4) from the Thur deployment (virtual sensor: thur_dozer_decagonmux__conv). The simulation was done offline, that is all data was available at the time of the simulation. The *generation time* was used as time reference and the applied outlier filter technique is based on the one-sided median method.

Figure 6.3 shows an example of instable behavior of the one-sided median outlier filter. The value of the last non-outlier data point is $463 \mu S/cm$. The median first difference for the next data point is $\tilde{m}_k^{(z)} = -0.5$ what results in a prediction of $\tilde{m}_k^{(2v)} = \tilde{m}_k^{(y)} + v \cdot \tilde{m}_k^{(z)} = 463 + 15 \cdot (-0.5) = 455.5$. Because of this incorrect prediction and the small threshold of $\tilde{T} = 5$, the data point with the value $464 \mu S/cm$ is declared as outlier and replaced by the (incorrect) prediction. As the replacement takes place in the original data, the outlier filter may become unstable as it is the case in Figure 6.3. Therefore, this outlier filter method is no longer considered.

6.2.3 Median Filter by Menold et al.

An approach very similar to the two-sided median method was proposed in [33]. In contrast to the two-sided median method they consider only the current y_k and past $N-1$ values, i.e. $y_k^\dagger = med(y_{k-N+1}, \ldots, y_k)$. If the distance d_k between the current value y_k and the median of the current and past $N-1$ values $y_k^\dagger$ is larger than the threshold T_k, then the current value is declared as an outlier. Two different strategies are proposed in [33] for the selection of T_k. One basic approach is to declare T_k as a constant value: $T_k = c \; \forall \; k$. In the other strategy T_k is varied in time and depends on the current and past data. Because in our case the typical variation of the data values is known, we will focus on the basic approach where the threshold is constant.

6.2.3.1 Dimensioning of Window-Width and Threshold

The window-width N has to be chosen large enough so that the number of outliers within the window stay below the breakdown point of 50%. For the

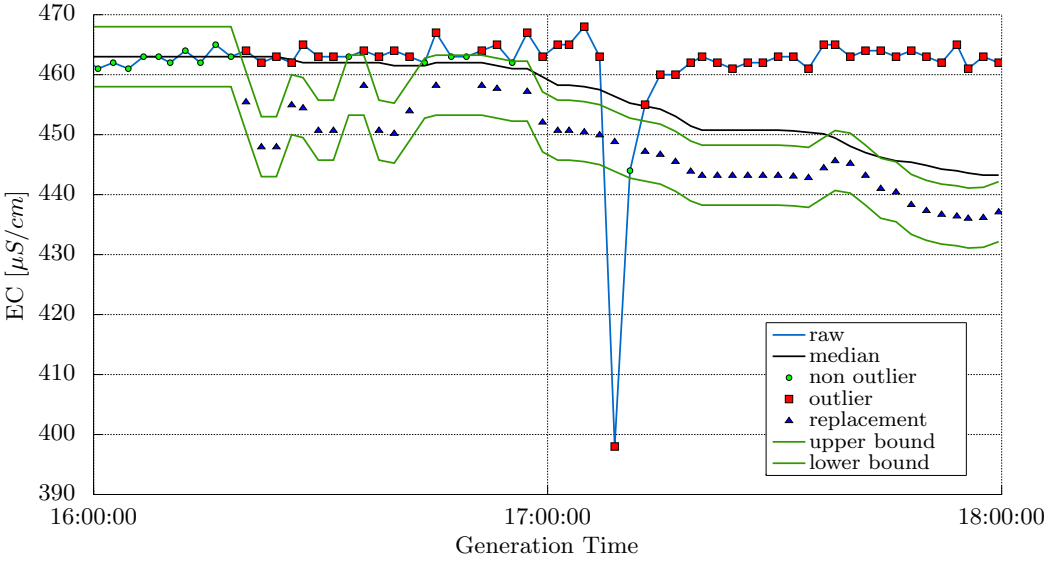


Figure 6.3: Simulation of one-sided median outlier filter with EC values from sensor at position 3, channel 5 from the Thur deployment with classification (date: 27.02.2012). Threshold $\tilde{T} = 5$, window width $v = 15$.

data of the Thur deployment, accumulation of outliers occurred (e.g. ≈ 10 out of 17). Therefore the window-width is set to 30 data values, what corresponds to approx. $60min.$ (based on a measurement interval of $2min$).

For a step change of the data value, it last half of the median window width $(N/2)$ until the median value $y_k^\dagger$ reacts. Therefore, the constant threshold T can be determined as follows:

$$T = EC_{dyn}\frac{W}{2} \tag{6.4}$$

where EC_{dyn} is the maximal temporal change of the EC and W is the window length. For $EC_{dyn} = 15\mu S/cm/h$ and $W = 1h$, the threshold is $T \approx 8\mu S/cm$.

6.2.3.2 Outlier Replacement Strategy

When a data value is declared to be an outlier, it has either to be removed or replaced by a prediction $y_k^\star$. A simple strategy is the replacement by the median, $y_k^\star = y_k^\dagger$ [33, 34]. Another strategy is the replacement with the last valid point [33]: $y_k^\star = y_{k-j}$ where j is the smallest value where the condition $|y_k^\dagger - y_{k-j}| \leq T_k$ is fulfilled. We will use the latter replacement strategy because this strategy performs better than the median replacement for monotonically increasing and decreasing sequences [33].

6.2.3.3 Simulation

The following simulations were performed with Matlab based on GSN data from the Decagon sensor 5TE at position 5 (R071), channel 0 (conductivity0) from the Thur deployment (virtual sensor: thur_dozer_decagonmux__conv). The simulation was done offline, that is all data was available at the time of the simulation. The *generation time* was used as time reference and the applied outlier filter technique is based on [33], see Section 6.2.3.

Figure 6.4 shows the EC over a period of four days (see also Figure 6.1). All the visible outliers were detected. It can be seen that the median curve, corresponding to $y_k^\dagger$, approximates the raw curve without outliers by a step function.

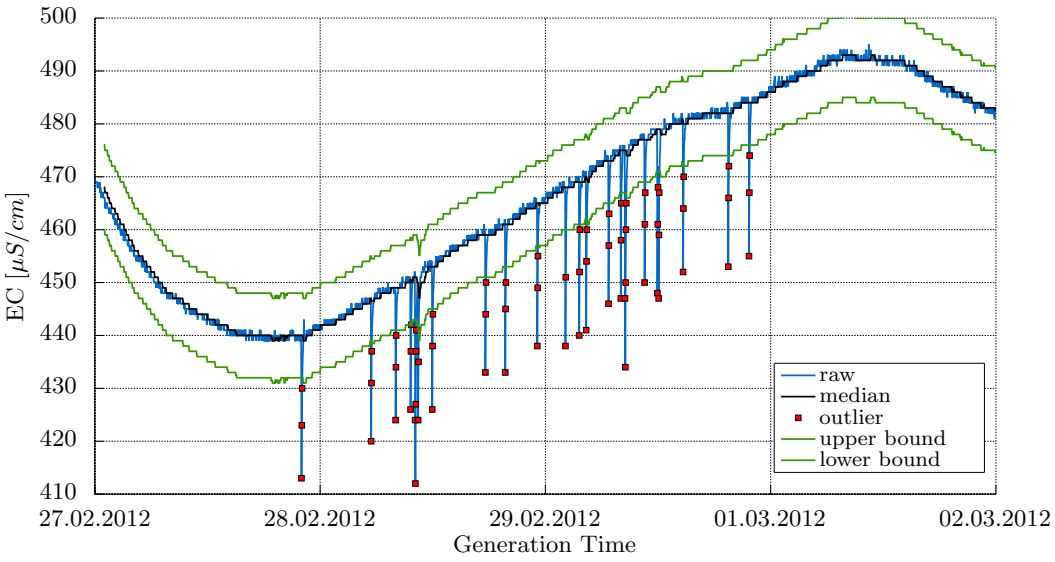


Figure 6.4: Simulation of median outlier filter from Menold et al. with EC values from sensor at position 5, channel 0 from the Thur deployment with classification. Threshold $T = 8$, window width $N = 30$.

Figure 6.5 shows a zoom of Figure 6.4. It can be seen that the data value declared as outliers (red squares) are replaced by the last valid value (blue triangles). The present case shows an accumulation of outliers where significant raw value drops are not recovered immediately but within the following $4-6$ samples. Due to the temporal accumulation of outliers the median shortly deviates from the overall tendency (see also Figure 6.4). Furthermore, as the raw data after an outlier does not immediately return to the general trend, some data points of a single outlier peak fall within the filter boundaries and are therefore not classified as outliers.

Figure 6.6 shows the data after the noise filtering with a mean filter (mean line). It is applied on the outlier filtered data. It smooths the outlier filtered

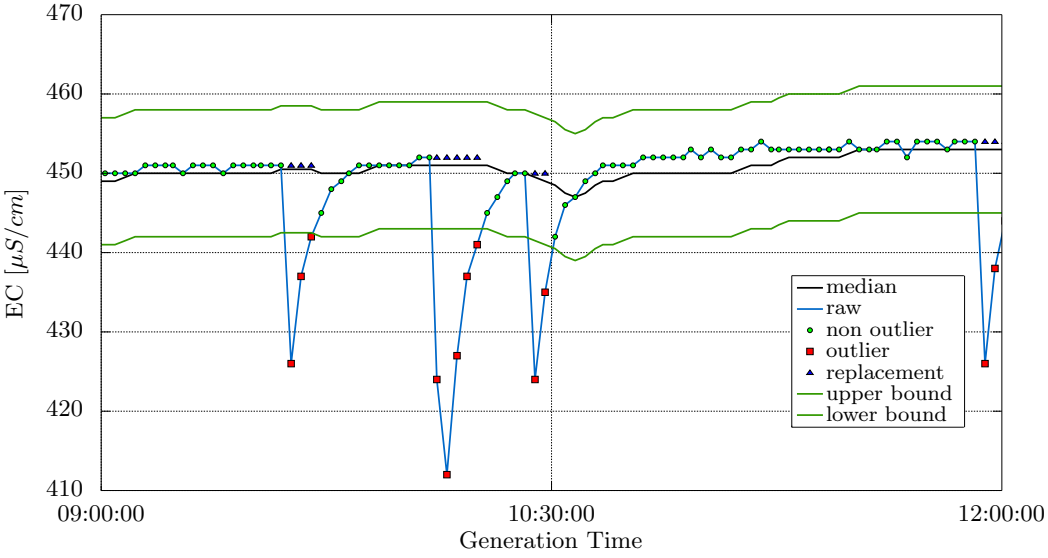


Figure 6.5: Simulation of outlier filter from Menold et al. with EC values from sensor at position 5, channel 0 from the Thur deployment with classification (date: 28.02.2012). Threshold $T = 8$, window width $N = 30$.

data and attenuates the effect of remaining outliers not removed by the outlier filter.

6.3 GSN-Test

The following tests were performed online with data from the Thur deployment. The applied outlier-filter is the median filter by Menold et al. (Section 6.2.3, [33]). Data points declared as outliers are replaced by the last valid data point. The filter settings were as follows:

- window-width = $60min$.
- threshold = $8\mu S/cm$

The noise filter applied on the outlier filtered data was a simple mean with a window width of $20min$.. Figure 6.7 shows the raw EC values and the output after the outlier- and noise-filtering. The green lines corresponds to the boundaries used by the outlier filter. All occurring outliers are properly filtered. Although the EC shows a steep rise on the 16.04.2012, the filtered output reflects the behavior of the raw EC.

Figure 6.8 shows the raw and the outlier- and noise filtered data from another sensor. Due to the steep rising of the EC, some raw data points are incorrectly declared as outliers. Therefore the filtered output does not properly reflect the

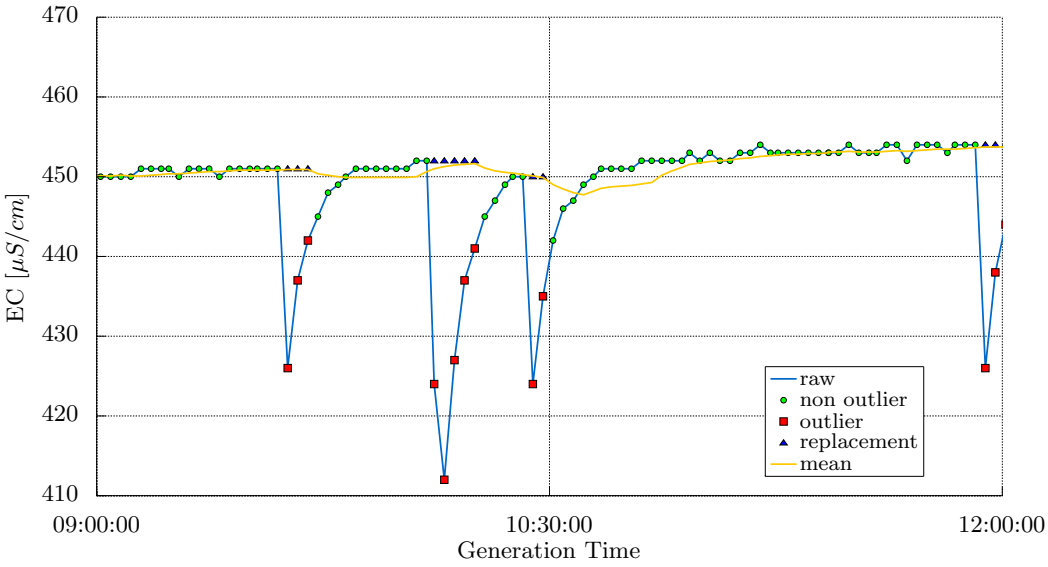


Figure 6.6: Simulation of the outlier and noise filter with EC values from sensor at position 5, channel 0 from the Thur deployment with classification (date: 28.02.2012). Mean window = 10.

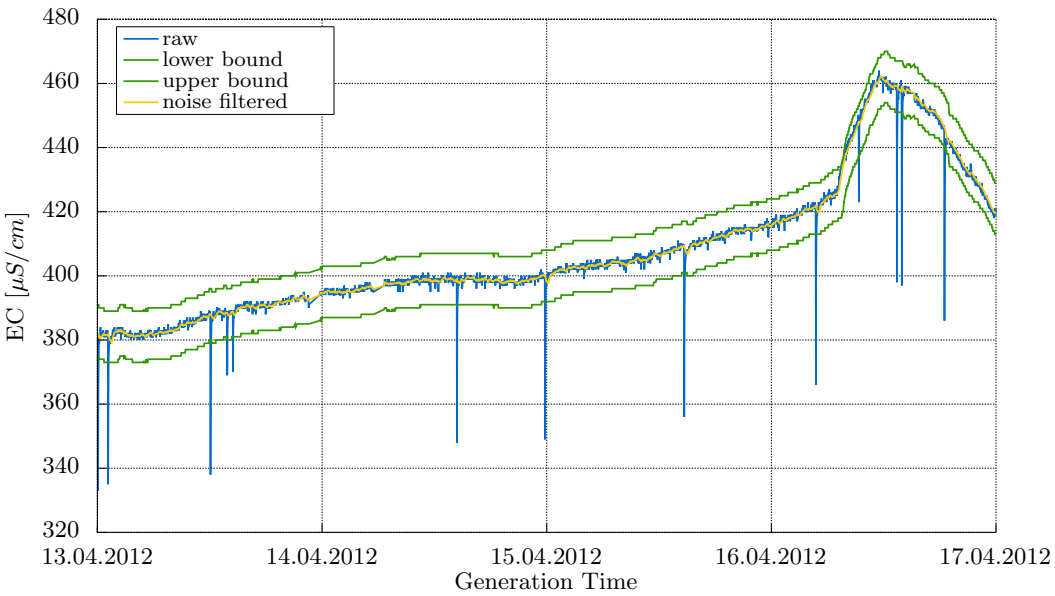


Figure 6.7: GSN test of the outlier and noise filter with EC values from sensor at position 3, channel 3 from the Thur deployment. The red line is the outlier- and noise filtered data.

behavior of the raw EC in the region of the steep rising. This last case shows the limitation of the filter in case of sensor signals with high dynamics. When the dynamics of the sensor signal is higher than the fixed boundaries of the outlier filter, the sensor data outside the boundaries is declared an outlier. Stretching the boundaries has the disadvantage that outliers with small amplitude are not detected anymore. That is, there is a trade off between the maximum dynamic of the sensor signal and the cleaning of outliers when using the outlier filter of Menold et al. [33] with a pre-defined threshold.

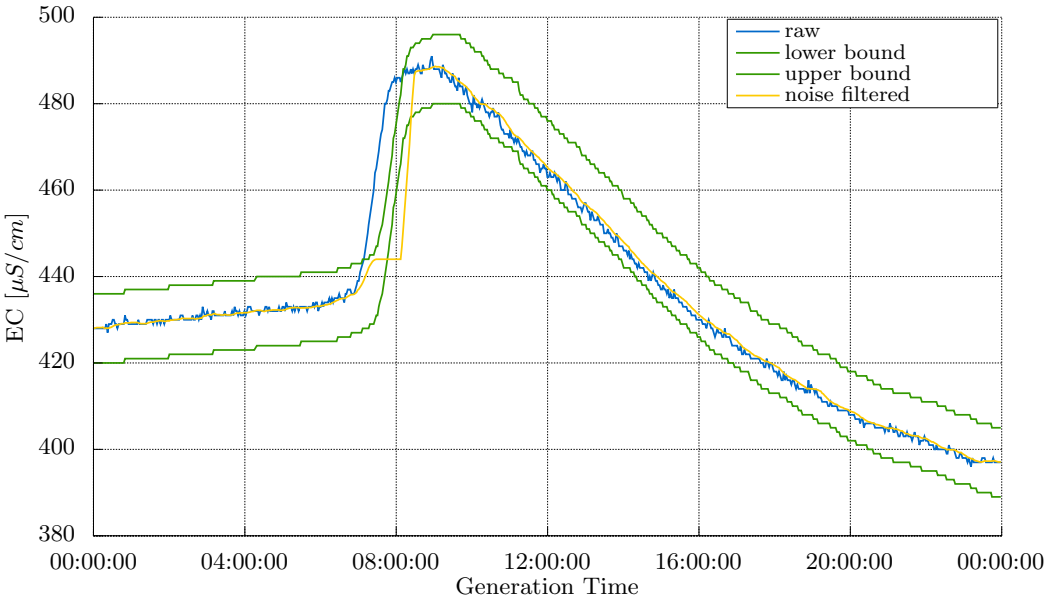


Figure 6.8: GSN test of the outlier and noise filter with EC values from sensor at position 5, channel 2 from the Thur deployment (date: 16.04.2012). The red line is the outlier- and noise filtered data.

Chapter 7

System Evaluation

7.1 Test at Thur Deployment

7.1.1 Overview

At the Thur deployment the electrical conductivity of groundwater is measured at different positions (R042, R071, R073 and R074 see Figure 7.1(b)) and depths. At each position, the conductivity is measured in five different depths with $1m$ distance. In this first test the EC of a single sensor is used to determine when to start with the sampling program.

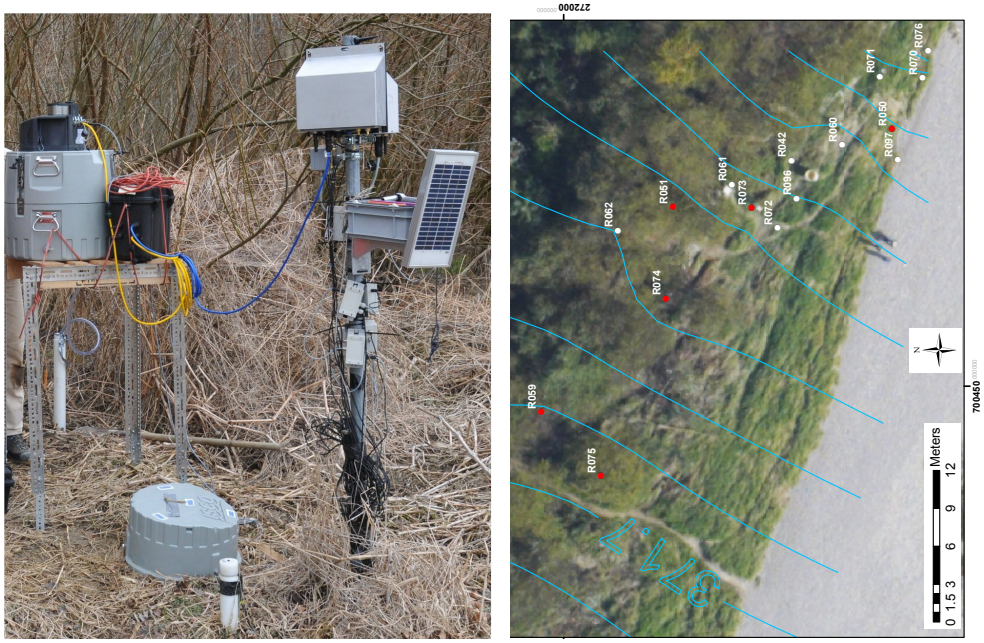
7.1.2 Deployment Configuration

Figure 7.1 shows the installation at the Thur deployment. Both, the 6712 water sampler and the *Core Station* are battery powered.

The *Core Station's* hostname is 'permasense-thur-hyd01.ethz.ch' and the channel of the WSN is channel number 8. The following plugins are enabled on the *Core Station* the 6712 is connected to:

- BackLogStatusPlugin
- CoreStationStatusPlugin
- SyslogNgPlugin
- Sampler6712Plugin

The duty cycle mode is also enabled and the service wakeup is at 08:00 and lasts $15min$.



(a) (b)

Figure 7.1: a) 6712 water sampler with battery (black box) next to position R061 and *Core Station* (gray box) mounted on the top of the mast at position R073. b) Description of positions.

7.1.3 Test

7.1.3.1 Configuration

The test round was performed between 01.05.2012 and 13.05.2012. During that period, the GSN server was down from 07.05.2012 11:21:23 UTC until 09.05.2012 08:17:36 UTC due to maintenance work. Furthermore, the GSN application was restarted multiple times.

For the test, the virtual sensors were configured as shown in Figure 7.2. The single sensor (dozer_decagonmux_conv) was the EC sensor channel 4 at position 4 (R042). The median filter presented in Section 6.2.3 was used as outlier filter. The filter's parameters were as follows: threshold = $8\mu S/cm$, minimum numbers of values in window = 10, window width = 60 minutes. The noise filter applied after the outlier filter was a mean filter with a window width of 20 minutes.

The control algorithm was configured to generate a static sampling schedule file as soon as the difference between the minimum and maximum EC within the last 12 hours is higher than $10\mu S/cm$. The water sampling schedule is a static schedule taking a sample of $10ml$ every 30 minutes into a new bottle, starting with bottle 1 and ending with bottle 24.

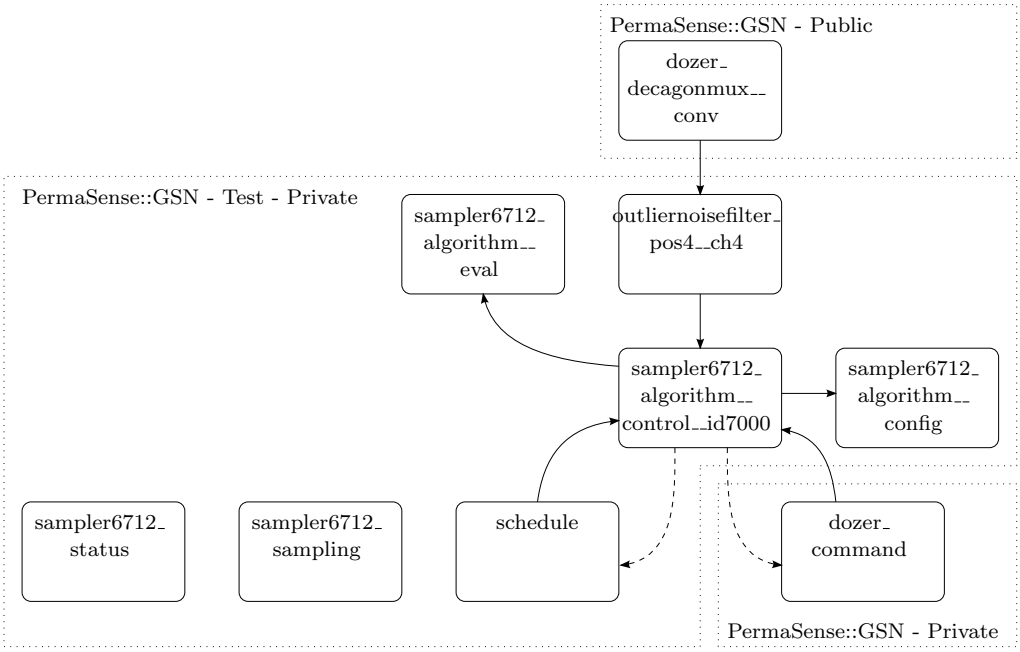


Figure 7.2: The virtual sensors for the test at the Thur deployment reside in different GSN servers. The 'thur_' prefix in the figure's virtual sensor names were omitted.

7.1.3.2 Results

Figure 7.3 shows an overview of the test period. As long as the EC min/max delta is below the threshold, no new schedule containing the water sampling information was generated and sent to the *Core Station*. The first time the EC min/max delta exceeded the threshold (just before 06.05.2012), a new sampling schedule was generated and sent to the *Core Station*. The schedule transmission was followed by 24 samplings in an interval of 30 minutes, starting with bottle number 1. The next exceeding of the threshold does not trigger a schedule generation because a manual reset is required to enable a new schedule generation (see Figure 5.8).

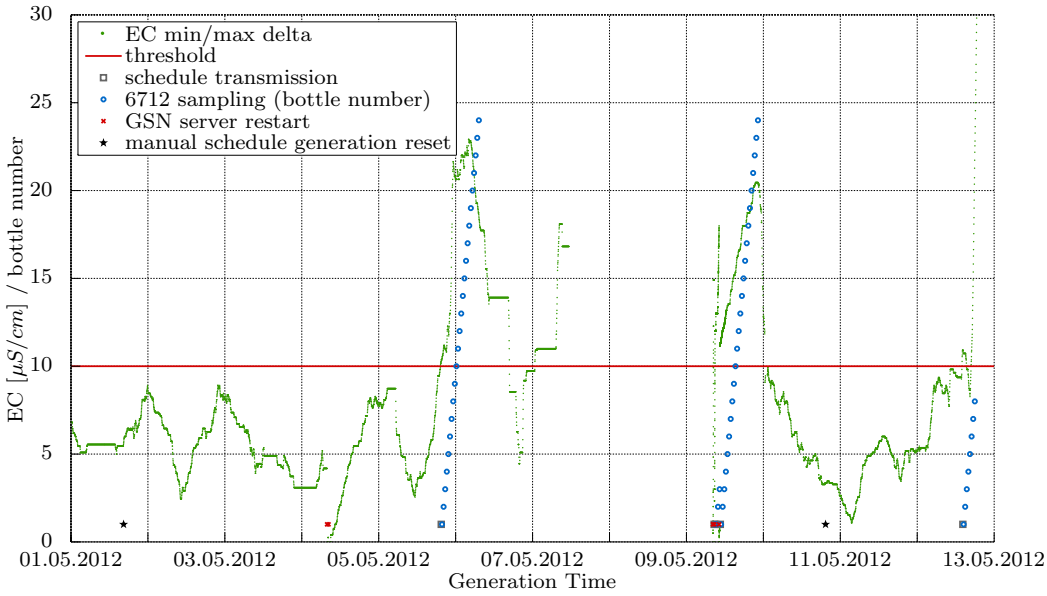


Figure 7.3: Overview of the test results. The data for the graphs originate from different virtual sensors. The EC min/max delta is the process observed by the control algorithm.

After the first sampling round, the GSN server was down for almost two days due to maintenance. The completion of the maintenance was followed by multiple restarts of the GSN server (see Figure 7.4). Because the EC min/max delta increased above the threshold shortly after the different GSN restarts, new sampling schedules were generated and overwrote the existing schedules. The reason for this overwriting is the fact that the internal state of the control algorithm VS is not stored persistently, i.e. after a GSN restart the control algorithm VS is ready to generate a new schedule (see Figure 5.8).

Table 7.1 shows an excerpt of the samples taken during the test run. It contains the sampling results reported by the *Core Station* to the GSN as well as the sampling report logged by the 6712. The sampling report of the 6712 was

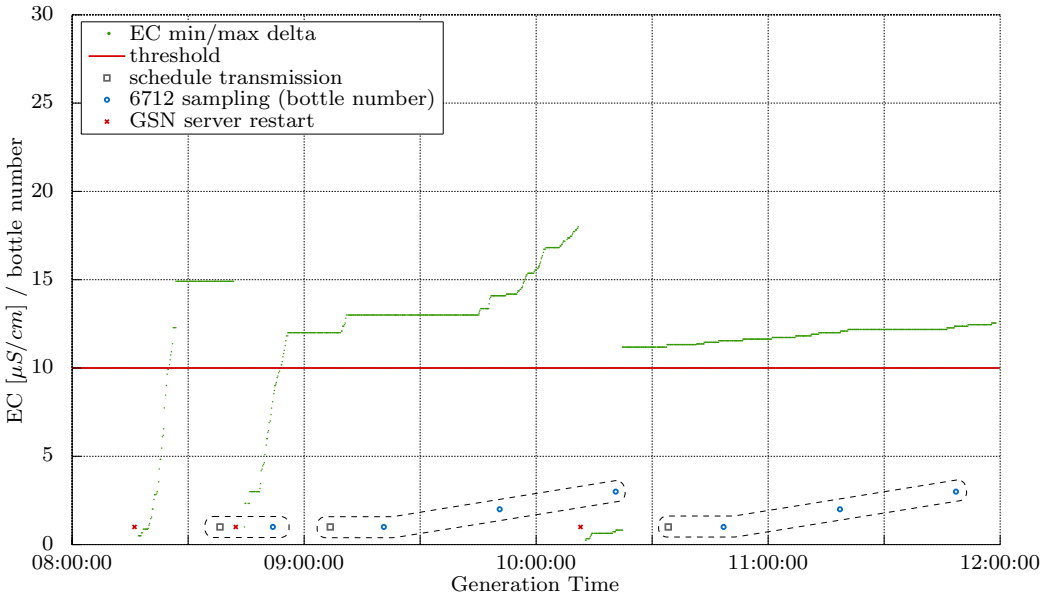


Figure 7.4: Zoom of the test (09.05.2012). The encircled 6712 samplings and schedule transmissions show to which schedule transmission the individual 6712 samplings belong to.

retrieved remotely (see Section 5.3.4.6). All samplings were successful and the bottle numbers and the sampling volume are identical. Only the time difference is between $1h$ $1min$ and $1h$ $2min$. The reason for the $1h$ difference is the winter time of the 6712's clock. One cause for the remaining $1-2min$ difference is that the time in the sampling report is the time of the event causing the sampling, i.e. the time the sampling was initiated by the *Core Station*, and the time in the GSN database entry corresponds to the sampling result after the completion of the sampling. Another cause for the $1-2min$ difference is the time difference between the *Core Station*'s clock and the 6712's clock of about $10s$. The static sampling started on 12.05.2012 was stopped after the 8th bottle because the battery was low.

By observing the behavior of the EC min/max delta after a GSN restart it can be seen that the EC min/max delta rises fast (after 1st and 2nd GSN server restart in Figure 7.4) or even does a jump (after 3rd GSN server restart). This behavior was not to be expected because the electrical conductivity of ground water changes rather slowly, i.e. not by $10\mu S/cm$ within a few minutes. The reason for this behavior is the way the min/max delta EC is calculated. For the calculation of the minimum and maximum value within the last 12 hours, the source query provided by GSN was used. Listing 7.1 shows the relevant extract from the VSD file (Thur_Sampler6712_Algorithm__control__Id7000.xml) which defines the calculation of the minimum and maximum. The window size of 12 hours is defined in line 2 (`storage-size`) and the aggregation functions

are visible in line 8 (`MIN()` and `MAX()` respectively).

Listing 7.1: Minimum and maximum aggregate function over a window of 12h.

```
1  <stream name="sensor">
2      <source alias="source" storage-size="12h" sampling-rate="1">
3          <address wrapper="local">
4              <predicate key="query">
5                  select * from Thur_OutlierNoiseFilter_Pos4__Ch4
6              </predicate>
7          </address>
8          <query> select MIN(noise_filter_value) as minValue, MAX(
               noise_filter_value) as maxValue from wrapper
9          </query>
10     </source>
11     <query> select * from source </query>
12 </stream>
```

If all sensor data would arrive in real-time, i.e. with a short propagation time so that the time difference between the *generation time* and the writing into the database is small (see Figure 2.5), then this approach would work. But due to the long downtime of the GSN server, the data partially arrived at the GSN server with very long delays. The lower graph in Figure 7.5 shows the filtered EC data once with the *generation time* as timeline (blue), i.e. the time the data was generated, and once with *timed* as timeline (red), i.e. the time the data was stored in the GSN database. Both curves are almost overlaying so long as the propagation time is short. But as soon as the propagation time gets large, the curve using the *timed* information as timeline does not represent the reality anymore. Such a case happened when the GSN server was under maintenance. While the GSN server was under maintenance, no data was stored in the database. But as soon as the server went back online, the data recorded in the WSN while the GSN server was down, is transmitted to the GSN and stored in the database.

As mentioned before, the aggregation function of the source query uses the *timed* information as time. Therefore it cannot distinguish between data which has just been generated by a sensor and data which was stored in the WSN/*Core Station* and only transmitted when the GSN server goes back online. This results in the fast increasing of the EC min/max delta (see Figure 7.6). The filtered EC values with the *generation time* as timeline in Figure 7.6 has gaps directly after GSN server restarts. The reason for those gaps is the outlier and noise filter which needs some sensor values before starting the filtering process.

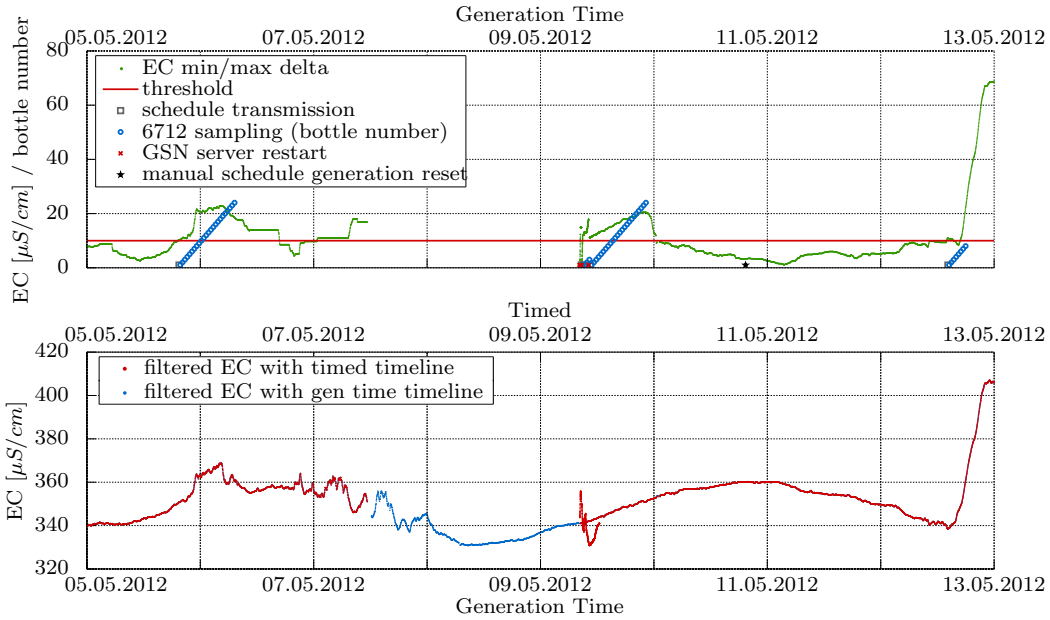


Figure 7.5: The *timed* and *generation time* time information may be very different when the GSN server is down for a longer time.

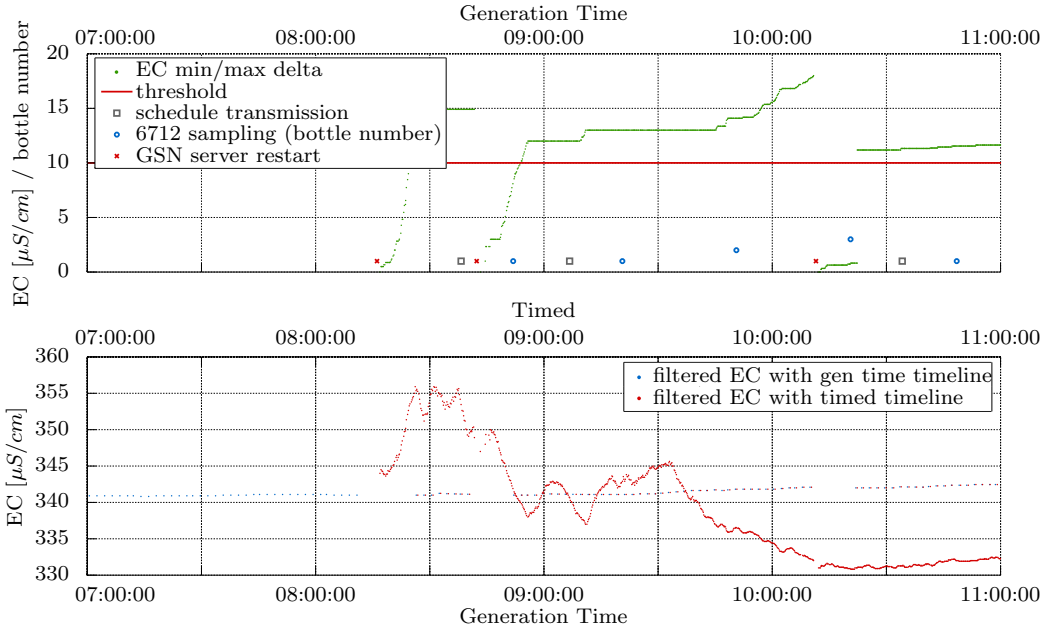


Figure 7.6: Zoom of Figure 7.5 (09.05.2012). When using GSN source queries to determine the minimum and maximum electrical conductivity, not the *generation time* but the *timed* timeline is relevant. Therefore the EC min/max delta rises fast although the real EC (*generation time* timeline) only changes slowly.

Table 7.1: Comparison between sampling report generated by 6712 and the GSN data generated by the *Core Station*.

6712 Sampling Report						GSN				
Date	**Vol**	**Btl**	**Time**[1]	**Source**[2]	**Error**	**Date and Time**	**Vol [ml]**	**Btl**	**R1**[3]	**R2**[4]
09-MAY-12	10 ml	1	09:50	M		2012-05-09T10:51:54.847+0200	10	1	0	0
	10 ml	1	10:19	M		2012-05-09T11:20:36.362+0200	10	1	0	0
	10 ml	2	10:49	M		2012-05-09T11:50:34.198+0200	10	2	0	0
	10 ml	3	11:19	M		2012-05-09T12:20:34.610+0200	10	3	0	0
	10 ml	1	11:47	M		2012-05-09T12:48:28.454+0200	10	1	0	0
	10 ml	2	12:17	M		2012-05-09T13:18:34.162+0200	10	2	0	0
	10 ml	3	12:47	M		2012-05-09T13:48:34.113+0200	10	3	0	0
	10 ml	4	13:17	M		2012-05-09T14:18:33.445+0200	10	4	0	0
	10 ml	5	13:47	M		2012-05-09T14:48:34.167+0200	10	5	0	0
...	...	...	...	...	...	...	...	...	...	...
	10 ml	21	21:47	M		2012-05-09T22:48:34.370+0200	10	21	0	0
	10 ml	22	22:17	M		2012-05-09T23:18:34.635+0200	10	22	0	0
	10 ml	23	22:47	M		2012-05-09T23:48:34.606+0200	10	23	0	0
	10 ml	24	23:17	M		2012-05-10T00:18:34.438+0200	10	24	0	0
12-MAY-12	10 ml	1	15:26	M		2012-05-12T16:27:56.386+0200	10	1	0	0
	10 ml	2	15:56	M		2012-05-12T16:57:36.713+0200	10	2	0	0
	10 ml	3	16:26	M		2012-05-12T17:27:36.740+0200	10	3	0	0
	10 ml	4	16:56	M		2012-05-12T17:57:36.258+0200	10	4	0	0
	10 ml	5	17:26	M		2012-05-12T18:27:36.206+0200	10	5	0	0
	10 ml	6	17:56	M		2012-05-12T18:57:36.897+0200	10	6	0	0
	10 ml	7	18:26	M		2012-05-12T19:27:37.106+0200	10	7	0	0
	10 ml	8	18:56	M		2012-05-12T19:57:36.826+0200	10	8	0	0
			19:01	POWER	FAILED!					

[1] Winter time (for summer time +1 hour)
[2] Source M = COMMAND SAMPLE
[3] R1 = sampling_result in Table D.1, 0 = sample done
[4] R2 = most_recent_sample_result in Table D.1, 0 = SAMPLE OK

Chapter 8

Conclusion and Outlook

In this thesis we successfully integrated the 6712 water sampler into the existing PermaSense WSN so that it is possible to control the water sampler based on sensor data. In a first step a concept was made to examine the integration of the 6712 water sampler as well as the control algorithm controlling the 6712 into the existing PermaSense system (see Chapter 3). The integration of the 6712 requires an interface unit to connect the 6712 with the existing system. Following the design approach where the integration of a new device is done with a feature-rich node followed by a second design step in which superfluous features are removed, we started with a *Core Station* to connect the 6712 with the PermaSense system. The 6712 and its serial interface had to be studied extensively in order to control the device by the *Core Station*, in particular the serial interface and the API required significant analysis, implementation and verification (see Chapters 4 and 5). The control algorithm was defined to be located at the backend, more precisely in the GSN. Because of the noise and the outliers in the sensor data used by the control algorithm, online data cleaning was required (see Chapter 6). The applied data cleaning was sufficient for the used control algorithm but new data cleaning approaches may be necessary when more complex control algorithms are used. First test at the Thur deployment showed that the basic mechanics work properly, i.e. after the control algorithm decided to start a static sampling series all samples were successfully drawn by the 6712 (see Chapter 7). Due to the fact that the 6712 is controlled with a schedule file containing the information for multiple samples, the 6712 together with the *Core Station* are able to run autonomous once received the schedule.

The control algorithm does not feature dynamic sampling yet. This requires some further information how such a dynamic schedule has to be in terms of the sensor data. Another next step may be the synchronized water sampling of multiple 6712 installed in the same catchment area. When aiming to have multiple 6712 water samplers connected to the PermaSense system, then the migration from the feature-rich *Core Station* to a smaller interface unit accessible through the WSN may be an option. This migration would not only contain a software migration but also the communication currently taking place over the WLAN would need to be moved to the WSN.

Appendix A

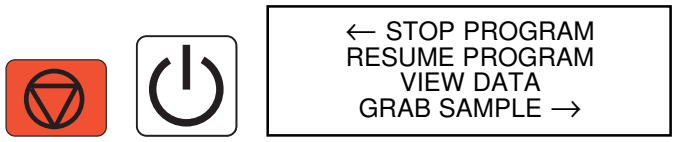
User's Guide

A.1 Operation

A.1.1 Bottle Change

To prevent sampling while the bottles are being changed, the bottles have to be replaced according to the following on-situ procedure:

1. Stop the running program of the water sampler by pressing the STOP key (see Figure A.1(a)) and selecting `STOP PROGRAM` on the menu pause screen (see Figure A.1(c)).
2. Change the bottles.
3. Turn off the water sampler by pressing the STANDBY key (see Figure A.1(b)).



(a) (b) (c)

Figure A.1: 6712 water sampler: a) stop key, b) standby key, c) menu pause screen.

This procedure is not only required to prevent sampling during the change of bottles but also to reinitialize the *Core Station's* internal bottle state. If the procedure is not performed, the *Core Station* skips samples when the bottles are full according to its internal bottle state. In case it was forgotten to stop the water sampler before changing the bottles or if the water sampler was not powered off after changing the bottles, the water sampler can be remotely reinitialized by a GSN web interface upload (see Figure A.2). It has to be ensured that the *Core Station* is powered on before executing remote reinitialization (see Appendix A.1.4.1).

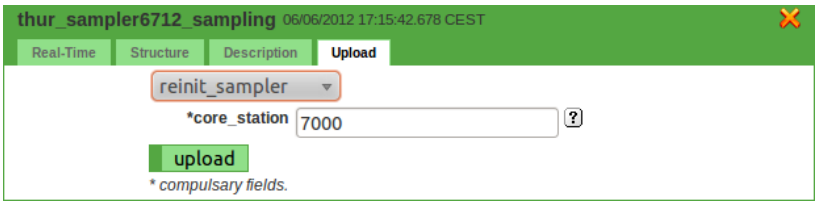


Figure A.2: GSN web interface upload tab of sampler6712_sampling VS.

Background Information When the water sampler's running program is stopped, the *Menu Control* mode state **Program Running (disabled)** is left (see Figure 5.11) so that no samplings are initiated by the *Core Station*. When the control algorithm generates a new sampling schedule while the bottles are being changed, all samplings falling into that period will be skipped. When the water sampler is turned off after the bottles have been changed, its state changes to **Sampler Off**. The next time the *Core Station* initiates a new sampling, the water sampler is initialized first and thereby set into the **Program Running (disabled)** state before a sampling is initiated by the *Core Station*.

A.1.2 Control Algorithm

The control algorithm generates two different types of data (see also Section 5.2.3.3). Two extra VS allow to demultiplex the control algorithm's output based on the `data_type` field. The eval VS (see Figure A.3) shows the evaluation criteria as well as the evaluation result of the sensor signal. The config VS (see Figure A.4) shows the state of the control algorithm (enable/disable) and the state of the state machine (see Figure 5.8). For the explanation of the different fields in Figures A.3 and A.4 we refer to Table D.3.

Figure A.3: GSN web interface real-time tab of sampler6712_algorithm__eval VS.

The control algorithm generates a new schedule file containing the sampling information (time, volume, bottle number) when the start condition is fulfilled and the control algorithm is enabled. Subsequently, the schedule file is sent to the *Core Station*. The control algorithm can be enabled and disabled by a GSN

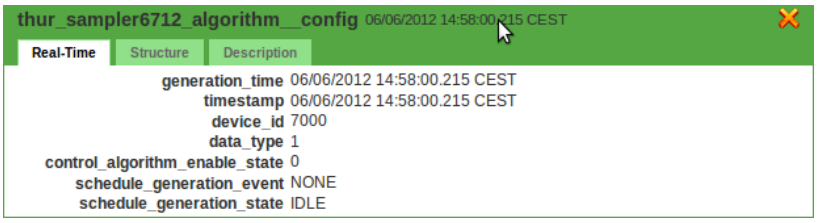


Figure A.4: GSN web interface real-time tab of sampler6712_algorithm__config VS.

web interface upload (see `control_state` drop down menu in Figure A.5).

As soon as the generated schedule is received by the *Core Station*, the control algorithm is automatically disabled to prevent a new schedule generation in case of a GSN restart (see Section 7.1.3.2). After the successful transmission of a new sampling schedule, the control algorithm can be re-activated by the following procedure:

1. Enable control algorithm by selecting `on` in the `control_state` drop down menu in Figure A.5
2. Reset schedule generation by selecting `reset_schedule_generation` in the `special_action` drop down menu in Figure A.5
3. Press `upload` button

Figure A.5: GSN web interface upload tab of sampler6712_algorithm_control VS.

It is also possible to manually start a sampling series. This can be achieved by selecting `manual_start_schedule_generation` in the `special_action` drop down menu (see Figure A.5).

A.1.3 View Generated Schedule

The last schedule sent to the *Core Station* can be viewed in the schedule VS by clicking on `download` next to the `schedule` field (see Figure A.6).

Listing A.1 shows an example of such a schedule file. For more details about the formatting of the schedule file we refer to Section 5.3.4.3.

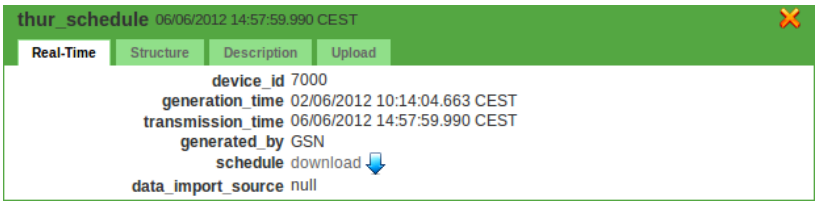


Figure A.6: GSN web interface real-time tab of schedule VS.

Listing A.1: Example of a schedule file.

```
1 0  9 * * *  plugin Sampler6712Plugin report_status
2 29 8 2 6 *  plugin Sampler6712Plugin bottle(1) volume(100)
    backward_tolerance_minutes=5
3 59 8 2 6 *  plugin Sampler6712Plugin bottle(2) volume(100)
    backward_tolerance_minutes=5
```

A.1.4 Advance

The following actions are only intended for expert users with GSN background knowledge!

A.1.4.1 Manual Power On/Off of Core Station

The *Core Station* is running in the duty cycle mode, i.e. it is turned off if it does not have to execute any tasks. Some manual actions (e.g. see Appendix A.1.4.4 or Appendix A.1.4.2) require the *Core Station* to be on. This can be achieved by sending a Dozer command over the WSN to the *Core Station* (see Figure A.7). To wake-up the *Core Station* the `*arg` has to be `1`. When the *Core Station* receives the wake-up command, it last about 2 minutes until it is ready to receive a manual action. After the *Core Station* has received the wake-up command it stays awake until the wake-up is cleared. That is, it has to be kept in mind that the *Core Station* needs to be set back to the duty cycle mode after the manual action is complete in order to reduce the power consumption! To return the *Core Station* into the duty cycle mode the `*arg` has to be `4`. When using `*arg=2` the *Core Station* wakes up and automatically goes back to sleep if it has nothing to do. This can be used for example when the *Core Station* only has to collect a new schedule file from the GSN (see Appendix A.1.4.3).

A.1.4.2 Manual Sampling

It is possible to manually initiate a single sampling. This can be done by a GSN web interface upload where the ID of the *Core Station* the water sampler is connected to, the bottle number, and the water volume to be drawn have to

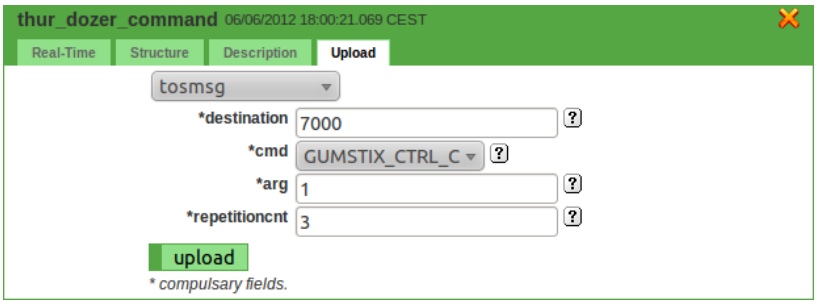


Figure A.7: GSN web interface upload tab of dozer_command VS.

be specified (see Figure A.8). Before executing the upload, it has to be ensured that the *Core Station* is powered on (see Appendix A.1.4.1). This upload is only intended for debugging and testing! The control algorithm is not aware of a manually initiated sampling, and therefore will overfill the manually filled bottle when generating a new sampling schedule.

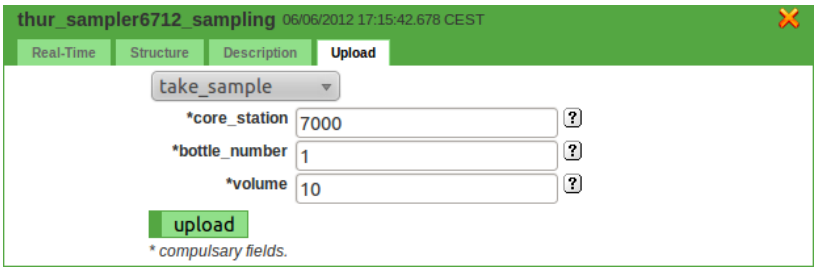


Figure A.8: GSN web interface upload tab of sampler6712_sampling VS.

A.1.4.3 Manual Schedule File Generation

It is possible to manually create a schedule file and subsequently send it to the *Core Station*. For the manual creation of the schedule file we refer to the example in Listing 5.3 as well as Section 5.3.4.3. After the creation of the schedule file the following procedure has to be followed in order to send the file to the *Core Station*:

1. Upload the schedule file with the device ID of the *Core Station* (see Figure A.9)
2. Wake-up *Core Station* (set `*arg` to `2`, see Appendix A.1.4.1)

The `transmission_time` field (see Figure A.6) has a valid time as soon as the schedule file was received by the *Core Station*, previously it was `null`. The control algorithm is not aware of a manually generated schedule file, and will

overfill the bottles when generating a new sampling schedule! Therefore, it is best to disable the control algorithm when controlling the water sampler with manually generated schedule files.

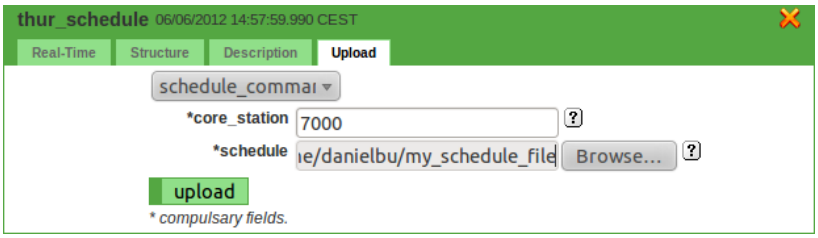


Figure A.9: GSN web interface upload tab of schedule VS.

A.1.4.4 Manual Status Request

The *Core Station* periodically transmits the status of the water sampler [1]. This status query can be initiated manually by a GSN web interface upload (see Figure A.10). It has to be ensured that the *Core Station* is powered on before executing the upload (see Appendix A.1.4.1).

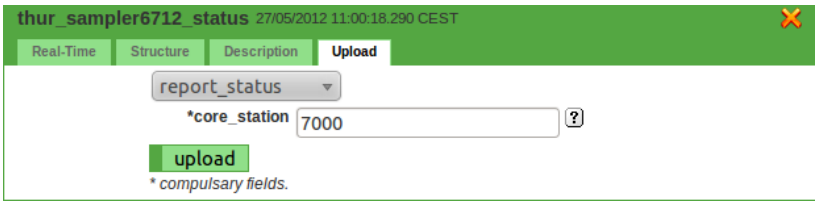


Figure A.10: GSN web interface upload tab of sampler6712_status VS.

A.2 Configuration

A.2.1 ISCO 6712

When installing a new 6712 water sampler, manual configurations need to be done prior to connecting the 6712 to the *Core Station*:

- Set language to English and unit to meter (type `6712.8` in main menu, see section 4.2.1)

[1] By the time of writing the status is sent every day at 09:00 UTC

Appendix B

Decagon 5TE

The 5TE sensor from Decagon can measure the volumetric water content, the electrical conductivity and the temperature [36]. The valid range of the raw electrical conductivity σ_{Raw} is 0 to 1022. A raw EC of 1023 is used to indicate a malfunction of the EC sensor. The bulk EC can be calculated as follows:

$$\text{If } \sigma_{Raw} \leq 700 \text{ then } EC = \frac{\sigma_{Raw}}{100}\left[\frac{dS}{m}\right] = \sigma_{Raw} \cdot 10\left[\frac{\mu S}{cm}\right] \quad \text{(B.1)}$$

$$\text{If } \sigma_{Raw} > 700 \text{ then } EC = \frac{700 + 5(\sigma_{Raw} - 700)}{100}\left[\frac{dS}{m}\right] \quad \text{(B.2)}$$

$$= \sigma_{Raw} \cdot 10(700 + 5(\sigma_{Raw} - 700))\left[\frac{\mu S}{cm}\right] \quad \text{(B.3)}$$

The resolution is

$$\text{If } EC \leq 7000\left[\frac{\mu S}{cm}\right] \text{ then the resolution is } 10\left[\frac{\mu S}{cm}\right] \quad \text{(B.4)}$$

$$\text{If } EC > 7000\left[\frac{\mu S}{cm}\right] \text{ then the resolution is } 50\left[\frac{\mu S}{cm}\right] \quad \text{(B.5)}$$

According to [37], sensors with firmware $\geq R3.18$ have a resolution of $0.001[dS/m] = 1[\mu S/cm]$ in SDI-12 mode.

$$\left[\frac{dS}{m}\right] = \left[10^3 \frac{\mu S}{cm}\right] \quad \text{(B.6)}$$

Appendix C

ISCO 6712 Water Sampler

C.1 Communication

The following listings show the communication between computer and 6712. The listings are produced by means of socat operating in the 'Data Transfer Loop' (see Appendix G.2). '>' corresponds to data send from the computer to the 6712, '<' corresponds to data send from the 6712 to the computer.

C.1.1 Offline Mode after Reset

In this case five ? are necessary to connect to the 6712 Section 4.6.1. The time between the last ? and the banner string is approximately 0.17s. It last about 0.08s until the complete banner string, containing HW and SW revisions, is displayed.

Listing C.1: Socat serial log of offline mode after reset.

```
1
2  > 2012/01/05 15:43:38.703290  length=1 from=88 to=88
3   3f                                                ?
4  --
5  > 2012/01/05 15:43:39.105133  length=1 from=89 to=89
6   3f                                                ?
7  --
8  > 2012/01/05 15:43:39.505085  length=1 from=90 to=90
9   3f                                                ?
10 --
11 > 2012/01/05 15:43:39.905129  length=1 from=91 to=91
12  3f                                                ?
13 --
14 > 2012/01/05 15:43:40.305084  length=1 from=92 to=92
15  3f                                                ?
16 --
17 < 2012/01/05 15:43:40.317130  length=1 from=2635 to=2635
18  3f                                                ?
19 --
20 < 2012/01/05 15:43:40.489125  length=8 from=2636 to=2643
21  0d 0a                                             ..
22  2a 2a 2a 20 4d 6f                                 *** Mo
```

```
--
< 2012/01/05 15:43:40.498562  length=8 from=2644 to=2651
 64 65 6c 20 36 37 31 32                          del 6712
--
< 2012/01/05 15:43:40.505051  length=8 from=2652 to=2659
 20 20 48 57 20 52 65 76                            HW Rev
--
...
...
--
< 2012/01/05 15:43:40.541038  length=8 from=2684 to=2691
 30 30 30 20 20 49 44 20                          000  ID
--
< 2012/01/05 15:43:40.549017  length=8 from=2692 to=2699
 31 32 38 31 37 38 30 38                          12817808
--
< 2012/01/05 15:43:40.554024  length=8 from=2700 to=2707
 38 34 20 0d 0a                                   84 ..
 0d 0a                                            ..
 3e                                               >
--
< 2012/01/05 15:43:40.562641  length=1 from=2708 to=2708
 20
--
```

C.1.2 Menu Control Mode

In this case the computer is already connected to the 6712 Section 4.6.1. The time between the last `?` and the banner string is approximately $0.18s$. It last about $0.08s$ until the complete banner string, containing hardware and software revisions, is displayed. Approximately $1s$ after the banner string, the menu is displayd. It lasts about $0.25s$ until the complete menu is displayed. It has to kept in mind that this value depends on the menu to display.

Listing C.2: Socat serial log of menu control mode.

```
> 2012/01/05 16:05:05.945692  length=1 from=103 to=103
 3f                                               ?
--
< 2012/01/05 16:05:05.958006  length=1 from=3021 to=3021
 3f                                               ?
--
> 2012/01/05 16:05:06.766090  length=1 from=104 to=104
 3f                                               ?
--
< 2012/01/05 16:05:06.777039  length=1 from=3022 to=3022
 3f                                               ?
--
> 2012/01/05 16:05:07.586145  length=1 from=105 to=105
 3f                                               ?
--
< 2012/01/05 16:05:07.769130  length=8 from=3023 to=3030
 0d 0a                                            ..
 2a 2a 2a 20 4d 6f                                *** Mo
--
< 2012/01/05 16:05:07.778681  length=8 from=3031 to=3038
 64 65 6c 20 36 37 31 32                          del 6712
```

```
--
...
...
--
< 2012/01/05 16:05:07.821037  length=8 from=3071 to=3078
 30 30 30 20 20 49 44 20                          000  ID
--
< 2012/01/05 16:05:07.829995  length=8 from=3079 to=3086
 31 32 38 31 37 38 30 38                          12817808
--
< 2012/01/05 16:05:07.841085  length=5 from=3087 to=3091
 38 34 20 0d 0a                                   84 ..
--
< 2012/01/05 16:05:08.777112  length=2 from=3092 to=3093
 0d 0a                                            ..
--
< 2012/01/05 16:05:08.797004  length=8 from=3094 to=3101
 20 20 20 20 20 20 20 3c                                <
--
< 2012/01/05 16:05:08.807903  length=8 from=3102 to=3109
 53 54 3e 41 54 55 53 20                          ST>ATUS
--
< 2012/01/05 16:05:08.814533  length=8 from=3110 to=3117
 2d 20 47 65 74 20 63 75                          - Get cu
--
< 2012/01/05 16:05:08.821067  length=8 from=3118 to=3125
 72 72 65 6e 74 20 73 74                          rrent st
--
< 2012/01/05 16:05:08.830380  length=8 from=3126 to=3133
 61 74 75 73 20 69 6e 66                          atus inf
--
< 2012/01/05 16:05:08.837067  length=8 from=3134 to=3141
 6f 72 6d 61 74 69 6f 6e                          ormation
--
< 2012/01/05 16:05:08.846311  length=2 from=3142 to=3143
 0d 0a                                            ..
--
< 2012/01/05 16:05:08.858054  length=8 from=3144 to=3151
 20 20 20 20 20 20 3c 30                                <0
--
< 2012/01/05 16:05:08.865991  length=8 from=3152 to=3159
 3e 2c 53 54 41 52 54 20                          >,START
--
< 2012/01/05 16:05:08.873108  length=8 from=3160 to=3167
 2d 20 53 74 61 72 74 20                          - Start
--
< 2012/01/05 16:05:08.882094  length=8 from=3168 to=3175
 61 20 70 72 6f 67 72 61                          a progra
--
< 2012/01/05 16:05:08.890067  length=8 from=3176 to=3183
 6d 20 57 41 49 54 49 4e                          m WAITIN
--
...
...
--
< 2012/01/05 16:05:08.986323  length=8 from=3261 to=3268
 20 3c 51 3e 75 69 74 20                           <Q>uit
--
< 2012/01/05 16:05:08.994266  length=8 from=3269 to=3276
 2d 20 45 78 69 74 20 4d                          - Exit M
--
< 2012/01/05 16:05:09.006178  length=8 from=3277 to=3284
```

```
84   45 4e 55 0d 0a                                          ENU..
85   0d 0a                                                   ..
86   3e                                                      >
87  --
88  < 2012/01/05 16:05:09.010710  length=1 from=3285 to=3285
89   20
90  --
```

C.1.3 Status During Sampling

The following figures shows the behaviour of the 6712 state (`STS` in status reply, see section 4.6.1.2) and the result of the most recent sampling (`SOR` in status reply). The time of the 'sampling command' is not exact. It only shows between which two points the sampling was triggered. The status information shown in the figures result from the following commands:

- 1st data point: reply on `STS,2` (turn on 6712)
- 2nd data point: reply on `BTL,x,SVO,y` (take a sample)
- 3rd and following data points: reply on `STS,1` (request status)

C.1.3.1 Valid Sampling

Figure C.1 shows the states during sampling. The settings are as follows:

- Rinses: 0
- Retries: 0
- Bottle number: 24
- Sampling volume: 10ml

The time between initiating the sampling (approximately at time 4) and its completion (at time 32) last about $28s$ in this case. This time may be much larger in case of larger sampling volume, higher rinses and retries numbers. Furthermore, the sampling time also depends on the current position of the distributor and the bottle to use for storing the sample. The status of the 6712 is not continously on 'SAMPLE IN PROGRESS' during the sampling. It happen that the replies to two consecutive status request contained the same sampler's current time (`TI`), although the state was different! This has to be kept in mind when using the sampler's current time for some reasons.

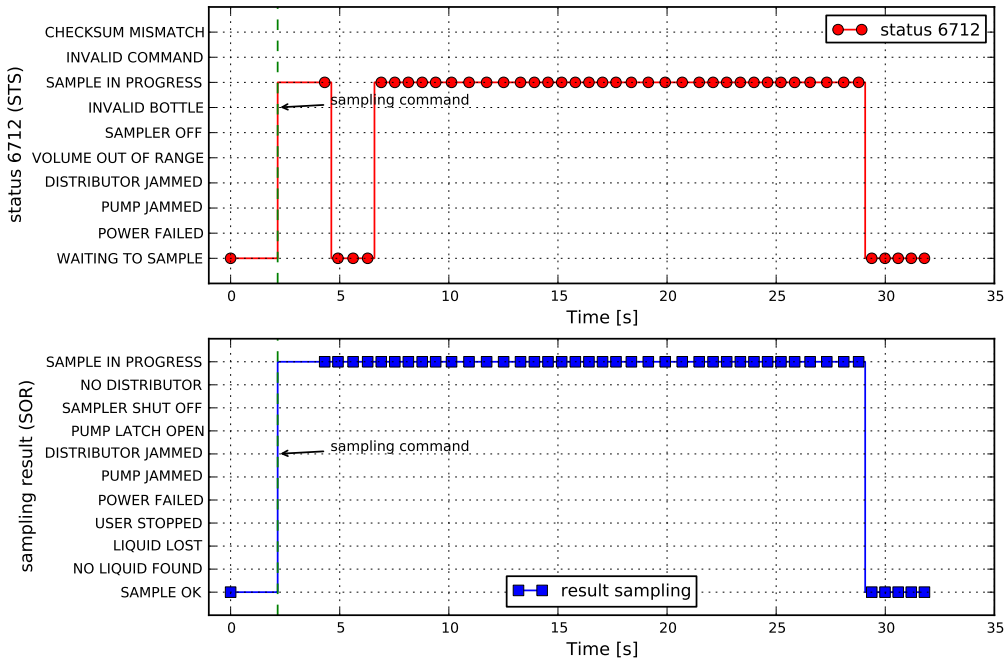


Figure C.1: 6712 water sampler state and sampling result during sampling.

C.1.3.2 Invalid Volume

Figure C.2 shows the states when trying to take a sample with an invalid sampling volume. The settings are as follows:

- Rinses: 0
- Retries: 0
- Bottle number: 24
- Sampling volume: 5ml

In case of an invalid sampling volume, the sampling result of the most recent sample is not changed. Only the 6712 status is returns 'VOLUME OUT OF RANGE' when trying to initiate a sampling (`BTL,24,SVO,5`).

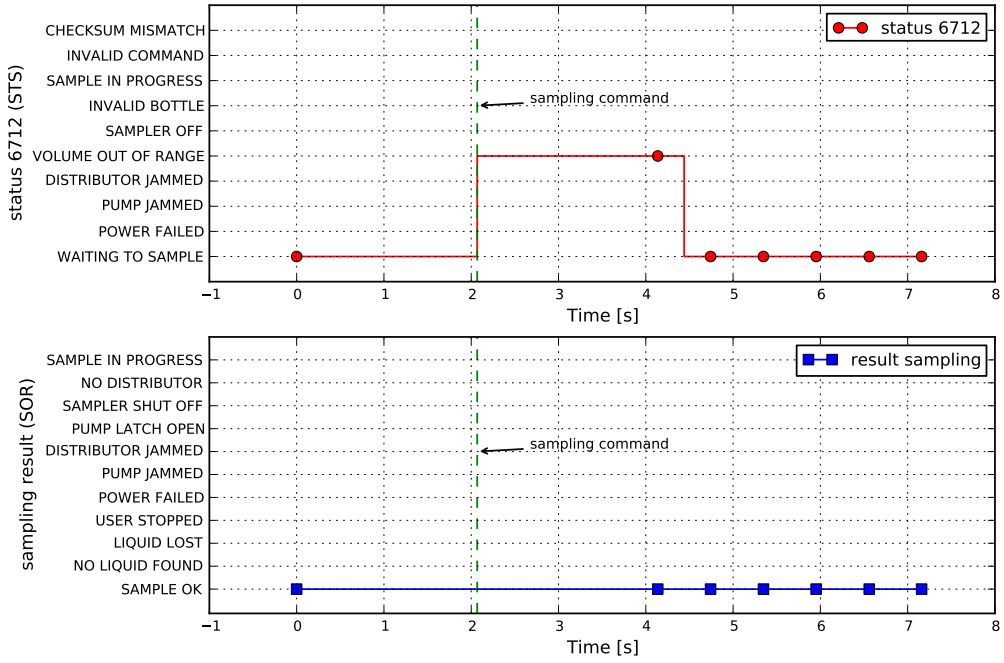


Figure C.2: 6712 water sampler state and sampling result when initiating a sampling with an invalid sampling volume.

C.1.3.3 Invalid Bottle

Figure C.3 shows the states when trying to take a sample with an invalid bottle number. The settings are as follows:

- Rinses: 0
- Retries: 0
- Bottle number: 30
- Sampling volume: 10ml

In case of an invalid bottle number, the sampling result of the most recent sample is not changed. Only the 6712 status is returns 'INVALID BOTTLE' when trying to initiate a sampling (`BTL,30,SVO,10`).

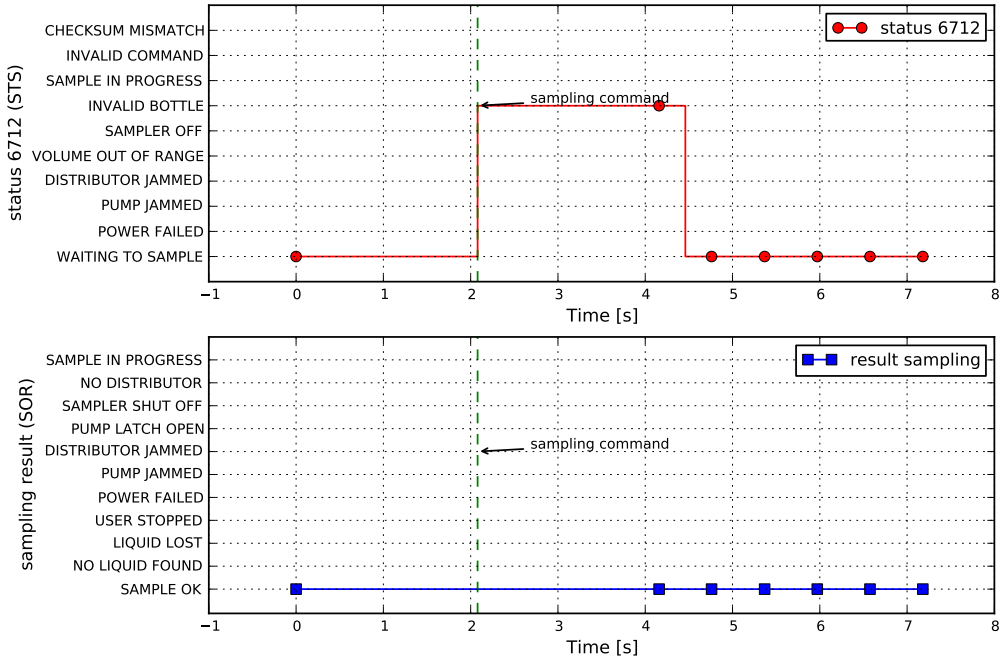


Figure C.3: 6712 water sampler state and sampling result when initiating a sampling with an invalid bottle number.

C.1.3.4 No Liquid

Figure C.4 shows the states when trying to take a sample when the suction head is placed outside water. The settings are as follows:

- Rinses: 0
- Retries: 0
- Bottle number: 24
- Sampling volume: 10ml

In case of placing the suction head on the air, the sampling result of the most recent sample is set to 'NO LIQUID FOUND'. If the number of retries is higher than 0, the process takes longer as the 6712 retries to draw water.

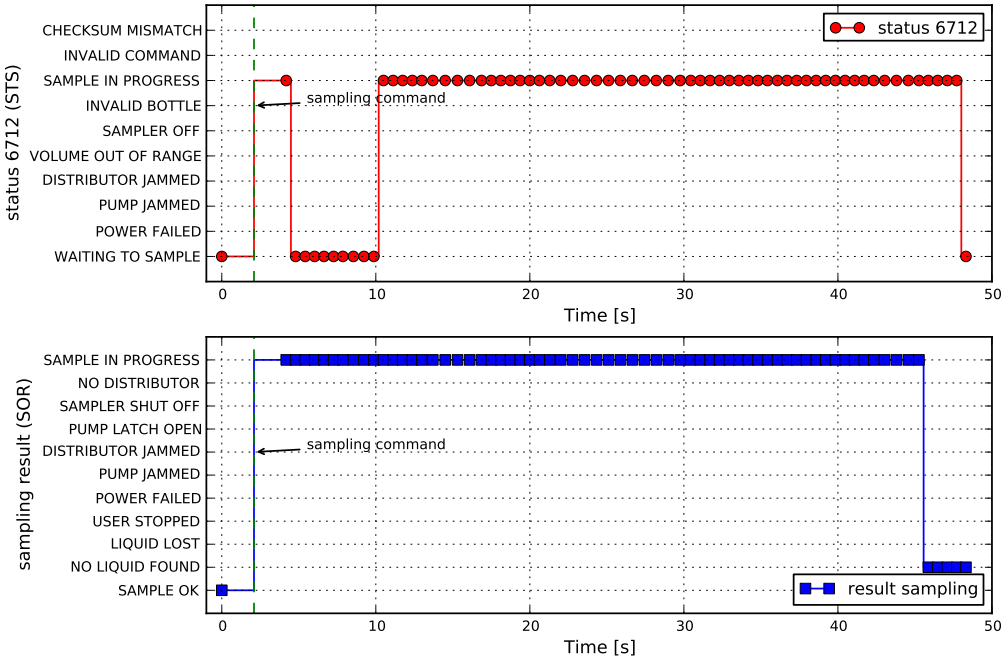


Figure C.4: 6712 water sampler state and sampling result when suction head exposed to the air.

C.1.4 6712 Program State Determination

Sometimes the 6712's reply on the `ST` command in the *Menu Control* mode is not complete. For example in the serial log Listing C.4 line 20 (see also Listing C.3 line 5), the program status was not properly send by the 6712. This caused a warning (Listing C.4) and a later retransmission of the `ST` command (not shown).

Listing C.3: Backlog info output

```
1  WARNING [UTC 08/02/2012 23:00:05,586] [Sampler6712-Thread] (
       Sampler6712Plugin.py:1035) Sampler6712Driver     -
2  Program status not found in: "
3  ST\r\n
4  Sampler Status: Program Running\r\n
5  PrograPogram Status      m\r\n
6     RAIN:  0.00 in       \r\n
7  \r\n
8  \n
9  \r\n
10        <ST>ATUS - Get current status information\r\n
11 <S>CREEN_<D>UMP - View 6712's display\r\n
12         <P>AUSE - Enter the MANUAL PAUSE menu if running a program\r\n
13       <2>, RUN2 - Runs extended program 2 if currently at standby\r\n
14       <3>, RUN3 - Runs extended program 3 if currently at standby\r\n
15       <4>, RUN4 - Runs extended program 4 if currently at standby\r\n
16     <6>, ENABLE - Enables a program previously DISABLEd\r\n
17         CONTROL - Get control of 6712's keypad\r\n
18          <Q>uit - Exit MENU\r\n
19 \r\n
20 > "
```

Listing C.4: Socat serial log

```
1      ...
2      >  53 54 0d                                             ST.
3      --
4      <  53 54 0d 0a                                          ST..
5      --
6      <  53 61 6d 70 6c 65 72 20 53                           Sampler S
7      --
8      <  74 61 74 75                                          tatu
9      --
10     <  73                                                   s
11     --
12     <  3a                                                   :
13     --
14     <  20 50 72 6f 67 72 61 6d 20 52 75 6e 6e               Program Runn
15     --
16     <  69 6e                                                in
17     --
18     <  67 0d 0a                                             g..
19     --
20     <  50 72 6f 67 72 61 50 6f                              PrograPo
21     --
22     <  67 72 61 6d 20 53 74 61 74 75                        gram Statu
23     --
24     <  73 20                                                s
25     --
26     <  20 20 20 20 20 6d                                         m
27     --
```

```
28      <  0d 0a                                                ..
29      --
30      <  20 20 20 52 41 49 4e 3a 20 20 30                        RAIN:  0
31      ...
```

Appendix D

Virtual Sensor Description Files

D.1 Virtual Sensor Description Files

The following Listings show a selection of VSD files.

Listing D.1: Thur_Sampler6712_Sampling.xml

```
<virtual-sensor name="Thur_Sampler6712_Sampling" priority="10" >
    <processing-class>
        <class-name>gsn.vsensor.BridgeVirtualSensorPermasense</class-name
           >
        <unique-timestamps>false</unique-timestamps>
        <web-input password="test">
            <command name="reinit_sampler">
                <field name="core_station" type="*text" defaultvalue="
                   7000">the device_id of the receiving core station</
                   field>
            </command>
            <command name="take_sample">
                <field name="core_station" type="*text" defaultvalue="
                   7000">the device_id of the receiving core station</
                   field>
                <field name="bottle_number" type="*text" defaultvalue="1"
                   >bottle to place sample in</field>
                <field name="volume" type="*text" defaultvalue="10">
                   volume to sample [ml]</field>
            </command>
        </web-input>
        <output-structure>
            <field name="DEVICE_ID" type="INTEGER" />
            <field name="GENERATION_TIME" type="BIGINT" unit="unixtime"
               index="true"/>
            <field name="TIMESTAMP" type="BIGINT" unit="unixtime" />

            <field name="SAMPLING_TRIGGER_SOURCE" type="INTEGER"/>
            <field name="BOTTLE_NUMBER" type="INTEGER"/>
            <field name="VOLUME" type="INTEGER" unit="ml" />
            <field name="SAMPLING_RESULT" type="INTEGER" />
            <field name="SAMPLER_MODEL" type="INTEGER" />
            <field name="SAMPLER_ID" type="INTEGER" />
            <field name="SAMPLER_TIME" type="DOUBLE" />
```

```
27              <field name="SAMPLER_STATUS" type="INTEGER" />
28              <field name="MOST_RECENT_SAMPLE_TIME" type="DOUBLE" />
29              <field name="MOST_RECENT_SAMPLE_BOTTLE" type="INTEGER" />
30              <field name="MOST_RECENT_SAMPLE_VOLUME" type="INTEGER" />
31              <field name="MOST_RECENT_SAMPLE_RESULT" type="INTEGER" />
32              <field name="CHECKSUM" type="INTEGER" />
33          </output-structure>
34      </processing-class>
35      <description>This plugin allows to send commands to the 6712 water
           sampler at the Thur deployment</description>
36      <life-cycle pool-size="10" />
37      <addressing/>
38      <storage />
39      <streams>
40          <stream name="data1">
41              <source alias="source" storage-size="1" sampling-rate="1">
42                  <address wrapper="backlog">
43                      <predicate key="remote-connection">permasense-thur-
                           hyd01.ethz.ch:9003</predicate>
44                      <predicate key="plugin-classname">gsn.wrappers.
                           backlog.plugins.Sampler6712Plugin</predicate>
45                      <predicate key="status-data-type">sampling_result</
                           predicate>
46                  </address>
47                  <query> select * from wrapper </query>
48              </source>
49              <query> select * from source </query>
50          </stream>
51      </streams>
52  </virtual-sensor>
```

Listing D.2: Thur_Sampler6712_Status.xml

```
1   <virtual-sensor name="Thur_Sampler6712_Status" priority="10" >
2       <processing-class>
3           <class-name>gsn.vsensor.BridgeVirtualSensorPermasense</class-name
               >
4           <unique-timestamps>false</unique-timestamps>
5           <web-input password="test">
6               <command name="report_status">
7                   <field name="core_station" type="*text" defaultvalue="
                        7000">the device_id of the receiving core station</
                        field>
8               </command>
9           </web-input>
10          <output-structure>
11              <field name="DEVICE_ID" type="INTEGER" />
12              <field name="GENERATION_TIME" type="BIGINT" unit="unixtime"
                    index="true"/>
13              <field name="TIMESTAMP" type="BIGINT" unit="unixtime" />
14
15              <field name="NB_OF_BOTTLES" type="INTEGER"/>
16              <field name="BOTTLE_VOLUME_IN_LIT" type="DOUBLE"/>
17              <field name="SUCTION_LINE_LENGTH_IN_M" type="DOUBLE"/>
18              <field name="SUCTION_LINE_HEAD" type="DOUBLE"/>
19              <field name="NB_OF_RINSE_CYCLES" type="INTEGER"/>
20              <field name="NB_OF_RETRIES" type="INTEGER"/>
21
22              <field name="MC_SAMPLER_STATUS" type="INTEGER"/>
23              <field name="MC_SAMPLER_STATUS_EXTENSION" type="INTEGER"/>
24              <field name="MC_PROGRAM_STATUS" type="INTEGER"/>
25
```

```
26              <field name="EPC_SAMPLER_MODEL" type="INTEGER" />
27              <field name="EPC_SAMPLER_ID" type="INTEGER" />
28              <field name="EPC_SAMPLER_TIME" type="DOUBLE" />
29              <field name="EPC_SAMPLER_STATUS" type="INTEGER" />
30              <field name="EPC_MOST_RECENT_SAMPLE_TIME" type="DOUBLE" />
31              <field name="EPC_MOST_RECENT_SAMPLE_BOTTLE" type="INTEGER" />
32              <field name="EPC_MOST_RECENT_SAMPLE_VOLUME" type="INTEGER" />
33              <field name="EPC_MOST_RECENT_SAMPLE_RESULT" type="INTEGER" />
34              <field name="EPC_CHECKSUM" type="INTEGER" />
35          </output-structure>
36      </processing-class>
37      <description>This plugin allows to request the 6712 water sampler
            status at the Thur deployment</description>
38      <life-cycle pool-size="10" />
39      <addressing/>
40      <storage />
41      <streams>
42          <stream name="data1">
43              <source alias="source" storage-size="1" sampling-rate="1">
44                  <address wrapper="backlog">
45                      <predicate key="remote-connection">permasense-thur-
                            hyd01.ethz.ch:9003</predicate>
46                      <predicate key="plugin-classname">gsn.wrappers.
                            backlog.plugins.Sampler6712Plugin</predicate>
47                      <predicate key="status-data-type">sampler_status</
                            predicate>
48                  </address>
49                  <query> select * from wrapper </query>
50              </source>
51              <query> select * from source </query>
52          </stream>
53      </streams>
54  </virtual-sensor>
```

Listing D.3: Thur_Sampler6712_Algorithm__control__Id7000.xml

```
1   <virtual-sensor name="Thur_Sampler6712_Algorithm__control__Id7000"
        priority="10" >
2       <processing-class>
3           <class-name>gsn.vsensor.Sampler6712ControlAlgorithmVS</class-name
                >
4           <unique-timestamps>false</unique-timestamps>
5           <init-params>
6               <param name="threshold">10</param> <!-- threshold in uS/cm-->
7               <param name="hold_time_in_min">10</param>  <!-- once signal
                    the input has rised above the threshold, the hold time
                    determines how long the output stays low -->
8               <param name="window_size_in_min">720</param>  <!-- window
                    size to use for the min/max value determination -->
9               <param name="min_nb_of_data_points_within_hold_time">3</param
                    >  <!-- min nb of data points within hold time, otherwise
                    start condition check is skipped -->
10              <param name="destination_device_id">7000</param> <!-- id of
                    device to control (id of core station 6712 water sampler
                    is connected to)-->
11
12              <param name="core_station_wake_up_timeout_in_min">15</param>
                    <!-- time before wake-up is sent again to CS-->
13              <param name="core_station_wake_up_repetition_cnt">3</param> <
                    !-- nb of attempts to wake up CS -->
14
15              <!-- settings for static sampling -->
```

```
16              <param name="sampling_start_delay_in_min">15</param>  <!--
                    delay between threshold condition fulfilled and first
                    sampling, minimum value is 5min -->
17              <param name="sampling_scheme">
                    1,6,10,1000;7,12,30,1000;13,24,60,1000</param>
18
19              <!-- settings for sending new schedule to core station (http
                    post) -->
20              <param name="schedule_host_name">http://whymper.ee.ethz.
                    ch:23001/</param> <!-- host name of schedule VS location:
                    PermaSense :: GSN - Test - Private -->
21              <param name="schedule_vs_name">thur_schedule</param> <!-- VS
                    name of schedule VS -->
22
23              <!-- settings for wake up of core station (http post) -->
24              <param name="dozer_command_host_name">http://whymper.ee.ethz.
                    ch:22001/</param> <!-- host name of dozer command VS
                    location: PermaSense :: GSN - Private -->
25              <param name="dozer_command_vs_name">thur_dozer_command</param
                    > <!-- VS name of dozer command VS -->
26          </init-params>
27
28          <web-input password="test">
29                  <command name="configuration">
30                  <field name="control_state" type="select:leave_unchanged|
                        on|off">on/off of control algorithm</field>
31                  <field name="special_action" type="select:none|
                        reset_schedule_generation|
                        manual_start_schedule_generation"></field>
32                  </command>
33                  </web-input>
34          <output-structure>
35              <field name="GENERATION_TIME" type="BIGINT" unit="unixtime"
                    index="true"/>
36              <field name="TIMESTAMP" type="BIGINT" unit="unixtime"/>
37              <field name="DEVICE_ID" type="INTEGER" /> <!-- device id of
                    core station (6712 water sampler) under control -->
38              <field name="DATA_TYPE" type="TINYINT" /> <!-- defines
                    whether sensor (0) or control (1) data -->
39
40              <field name="START_CONDITION_THRESHOLD" type="DOUBLE" />
41              <field name="START_CONDITION_HOLD_TIME_IN_MIN" type="INTEGER"
                     />
42              <field name="WINDOW_SIZE_IN_MIN" type="INTEGER" />
43              <field name="MINIMUM_NB_OF_DATA_POINTS_WITHIN_HOLD_TIME" type
                    ="INTEGER" />
44              <field name="START_CONDITION_FULFILLED" type="TINYINT" />
45              <field name="MIN" type="DOUBLE" />
46              <field name="MAX" type="DOUBLE" />
47              <field name="MIN_MAX_DELTA" type="DOUBLE" />
48
49              <field name="CONTROL_ALGORITHM_ENABLE_STATE" type="TINYINT" /
                    >
50              <field name="SCHEDULE_GENERATION_EVENT" type="VARCHAR(128)" /
                    >
51              <field name="SCHEDULE_GENERATION_STATE" type="VARCHAR(128)" /
                    >
52
53              <field name="AUTO_SCHEDULE_GENERATION_RESET_TIME_IN_MIN" type
                    ="INTEGER" />
54          </output-structure>
55      </processing-class>
```

```
56      <description>control algorithm for controlling 6712 water sampler at
           Thur deployment</description>
57      <life-cycle pool-size="10" />
58      <addressing />
59      <storage />
60      <streams>
61          <!-- sensor data stream for control algorithm -->
62          <stream name="sensor">
63              <source alias="source" storage-size="1" sampling-rate="1">
64                  <address wrapper="local">
65                      <predicate key="query">
66                          select * from Thur_OutlierNoiseFilter_Pos4__Ch4
67                      </predicate>
68                  </address>
69                  <query> select noise_filter_value as sensor_value,
                       generation_time from wrapper
70                  </query>
71              </source>
72              <query> select * from source </query>
73          </stream>
74
75          <!-- schedule stream -->
76          <stream name="schedule">
77              <source alias="source" storage-size="1" sampling-rate="1">
78                  <address wrapper="local">
79                      <predicate key="query"> select * from Thur_Schedule
                           where device_id = 7000 </predicate>
80                  </address>
81                  <query> select * from wrapper
82                  </query>
83              </source>
84              <query> select * from source </query>
85          </stream>
86
87          <!-- dozer command stream -->
88          <stream name="dozer_command">
89              <source alias="source" storage-size="1" sampling-rate="1">
90                  <address wrapper="remote-rest">
91                      <predicate key="query"> select * from
                           Thur_Dozer_Command where device_id = 7000 </
                           predicate>
92                      <predicate key="remote-contact-point">http://whymper.
                           ee.ethz.ch:22001/streaming/</predicate> <!--
                           PermaSense :: GSN - Private -->
93                  </address>
94                  <query> select * from wrapper
95                  </query>
96              </source>
97              <query> select * from source </query>
98          </stream>
99
100     </streams>
101 </virtual-sensor>
```

Listing D.4: Thur_Sampler6712_Status.xml

```
1   <virtual-sensor name="Thur_Sampler6712_Algorithm__eval" priority="10" >
2       <processing-class>
3           <class-name>gsn.vsensor.BridgeVirtualSensorPermasense</class-name
               >
4           <unique-timestamps>false</unique-timestamps>
5           <output-structure>
```

```
6             <field name="GENERATION_TIME" type="BIGINT" unit="unixtime"
                  index="true"/>
7             <field name="TIMESTAMP" type="BIGINT" unit="unixtime"/>
8             <field name="DEVICE_ID" type="INTEGER" />
9             <field name="DATA_TYPE" type="TINYINT" /> <!-- defines
                  whether sensor (0) or control (1) data -->
10
11            <field name="START_CONDITION_THRESHOLD" type="DOUBLE" />
12            <field name="START_CONDITION_HOLD_TIME_IN_MIN" type="INTEGER"
                  />
13            <field name="WINDOW_SIZE_IN_MIN" type="INTEGER" />
14            <field name="MINIMUM_NB_OF_DATA_POINTS_WITHIN_HOLD_TIME" type
                  ="INTEGER" />
15            <field name="START_CONDITION_FULFILLED" type="TINYINT" />
16            <field name="MIN" type="DOUBLE" />
17            <field name="MAX" type="DOUBLE" />
18            <field name="MIN_MAX_DELTA" type="DOUBLE" />
19
20        </output-structure>
21    </processing-class>
22    <description>sensor data evaluation of control algorithm for
          controlling water sampler at Thur deployment</description>
23    <life-cycle pool-size="10" />
24    <addressing />
25    <storage />
26    <streams>
27        <stream name="data">
28            <source alias="source" storage-size="1" sampling-rate="1">
29                <address wrapper="local">
30                    <predicate key="query"> select * from
                          Thur_Sampler6712_Algorithm__control__Id7000 where
                          data_type = 0 </predicate>
31                </address>
32                <query> select * from wrapper </query>
33            </source>
34            <query> select * from source </query>
35        </stream>
36    </streams>
37 </virtual-sensor>
```

Listing D.5: Thur_Sampler6712_Status.xml

```
1  <virtual-sensor name="Thur_Sampler6712_Algorithm__config" priority="10" >
2      <processing-class>
3          <class-name>gsn.vsensor.BridgeVirtualSensorPermasense</class-name
               >
4          <unique-timestamps>false</unique-timestamps>
5          <init-params>
6              <param name="scriptlet">
7                  <![CDATA[
8                          if (SCHEDULE_GENERATION_STATE == "
                               SCHEDULE_GENERATION") {
9                          // the mail recipients for error and exception
                               counter increase
10                         def mailrecipients = ["burgani@gmx.ch", "
                               tgsell@tik.ee.ethz.ch", "philipp.schneider@geo
                               .uzh.ch"];
11
12                                 // Notify by email
13                                 def emailTitle = "[PermaSense-GSN-Test] -
                                       Schedule generated";
```

```
14                                  def emailContent = "New schedule
                                       generated at " + (new Date(
                                       GENERATION_TIME)).getDateTimeString()
                                       + " for CoreStation with device id " +
                                        DEVICE_ID;
15
16                                                  org.apache.log4j.Logger.
                                                       getLogger(gsn.
                                                       processor.
                                                       ScriptletProcessor.
                                                       class).warn(
                                                       emailContent);
17                                  sendEmail(mailrecipients, emailTitle,
                                       emailContent);
18                      }
19                  ]]>
20              </param>
21          </init-params>
22          <output-structure>
23              <field name="GENERATION_TIME" type="BIGINT" unit="unixtime"
                    index="true"/>
24              <field name="TIMESTAMP" type="BIGINT" unit="unixtime"/>
25              <field name="DEVICE_ID" type="INTEGER" />
26              <field name="DATA_TYPE" type="TINYINT" /> <!-- defines
                    whether sensor (0) or control (1) data -->
27
28              <field name="CONTROL_ALGORITHM_ENABLE_STATE" type="TINYINT" /
                    >
29              <field name="SCHEDULE_GENERATION_EVENT" type="VARCHAR(128)" /
                    >
30              <field name="SCHEDULE_GENERATION_STATE" type="VARCHAR(128)" /
                    >
31
32          </output-structure>
33      </processing-class>
34      <description>algorithm configuration of control algorithm for
            controlling water sampler at Thur deployment</description>
35      <life-cycle pool-size="10" />
36      <addressing />
37      <storage />
38      <streams>
39          <stream name="data">
40              <source alias="source" storage-size="1" sampling-rate="1">
41                  <address wrapper="local">
42                      <predicate key="query"> select * from
                            Thur_Sampler6712_Algorithm__control__Id7000 where
                            data_type = 1 </predicate>
43                  </address>
44                  <query> select * from wrapper </query>
45              </source>
46              <query> select * from source </query>
47          </stream>
48      </streams>
49  </virtual-sensor>
```

Listing D.6: Thur_OutlierNoiseFilter_Pos4_Ch4.xml

```
1   <virtual-sensor name="Thur_OutlierNoiseFilter_Pos4__Ch4" priority="10" >
2       <processing-class>
3           <class-name>gsn.vsensor.OutlierNoiseFilterVirtualSensor</class-
                name>
4           <init-params>
```

```
5               <param name="outlier_filter">one_sided_median_menold</param>
6               <param name="outlier_filter_threshold">8</param>
7               <param name="outlier_filter_min_nb_of_values_in_window">10</
                   param>
8               <param name="outlier_filter_window_width_in_minutes">60</
                   param>
9
10              <param name="noise_filter">mean</param>
11              <param name="noise_filter_window_width_in_minutes">20</param>
12
13              <param name="buffer_size">100</param>
14          </init-params>
15          <output-structure>
16              <field name="DEVICE_ID" type="INTEGER" />
17              <field name="POSITION" type="INTEGER" />
18                          <field name="GENERATION_TIME" type="BIGINT" unit=
                               "unixtime" index="true"/>
19                          <field name="TIMESTAMP" type="BIGINT" unit="
                               unixtime"/>
20
21                          <field name="RAW_VALUE" type="DOUBLE" />
22              <field name="OUTLIER_FILTER_MEDIAN" type="DOUBLE" />
23              <field name="OUTLIER_FILTER_LOWER_BOUND" type="DOUBLE" />
24              <field name="OUTLIER_FILTER_UPPER_BOUND" type="DOUBLE" />
25                          <field name="OUTLIER_FILTER_CLASSIFICATION" type=
                               "INTEGER" />
26              <field name="OUTLIER_FILTER_VALUE" type="DOUBLE" />
27              <field name="NOISE_FILTER_VALUE" type="DOUBLE" />
28          </output-structure>
29      </processing-class>
30      <description>outlier/noise filter for conductivity channel 4 at
           position 4 at Thur deployment</description>
31      <life-cycle pool-size="10" />
32      <addressing />
33      <storage />
34      <streams>
35          <stream name="pos4_ch4">
36              <source alias="source" storage-size="1" sampling-rate="1">
37                  <address wrapper="remote-rest">
38                      <predicate key="query"> select * from
                           thur_dozer_decagonmux__conv where position = 4</
                           predicate>
39                      <predicate key="remote-contact-point">http://data.
                           permasense.ch:22001/streaming/</predicate>  <!--
                           PermaSense :: GSN - Public -->
40                  </address>
41                  <query> select device_id, position, timestamp,
                       generation_time, conductivity4 as raw_value from
                       wrapper </query>
42              </source>
43              <query>select * from source</query>
44          </stream>
45      </streams>
46  </virtual-sensor>
```

D.2 Output Data

Table D.1: Sampler6712 sampling VS output data.

Name	**Description**	**Val**	**Value Description**
sampling_trigger_source	source initiated the sampling	0 1	Scheduler (schedule file) GSN web interface upload
bottle_number	bottle number for sampling	[1]	bottle number
volume	sampling volume	-	in [ml]
sampling_result	sampling result	0 1 2..6 2 3 4 5 6 7 8	sample done (for result see most_recent_sample_result) sampling skipped (for reason see sampler_status) skipped because sampler status is: 'Program Idle' 'At Standby' 'Program Waiting Start' 'Program Running (enabled)' 'Program Halted' skipped because bottle capacity exceeded skipped because bottle number is invalid
sampler_model[1]	model number of the sampler	-	model number
sampler_id[1]	sampler's unique ID number	-	ID number
sampler_time[1]	sampler's current time	-	number of days since 00:00:00 1-Jan-1900, and the time shown as a fraction
sampler_status[1]	sampler's current status		see chapter 7.1.2 in [3]
most_recent_sample_time[1]	most recent sample time	-	
most_recent_sample_bottle[1]	most recent sample bottle	-	bottle number
most_recent_sample_volume[1]	most recent sample volume	-	in [ml]
most_recent_sample_result[1]	most recent sample result	-	see chapter 7.1.2 in [3]
checksum[1]	checksum of `STS,1` reply	-	for checksum calculation see section 4.6.1

[1] Data from `STS,1<CR>` reply (see section 4.6.1.2)

Table D.2: Sampler6712 status VS output data.

Name	Description	Val	Value Description
nb_of_bottles	number of bottles in 6712	1..24	number of bottles
bottle_volume_in_lit	bottle volume	-	in $[l]$
suction_line_length_in_m	length of suction line	0.9-30.2	in $[m]$
suction_line_head	length of suction line head	0.3-8.5	in $[m]$
nb_of_rinse_cycles		0..3	
nb_of_retries		0..3	
mc_sampler_status[1]	*Menu Control* mode state see also Section 4.6.1.1	0 1 2 3 4 5 6	'Program Waiting Start' 'Program Running' 'At Standby' 'Program Idle' 'Sampler Off' 'Program Halted' unknown status
mc_sampler_status_extension[1]		0 1 2	none errors [3] unknown
mc_program_status[1]	Only available when the *Menu Control* mode state is 'Program Running'	0 1 2	enabled, see also Section 4.6.1.1 disabled unknown
epc_sampler_model[2]	model number of the sampler	-	model number
epc_sampler_id[2]	sampler's unique ID number	-	ID number
epc_sampler_time[2]	sampler's current time	-	number of days since 00:00:00 1-Jan-1900, and the time shown as a fraction
epc_sampler_status[2]	sampler's current status		see chapter 7.1.2 in [3]
epc_most_recent_sample_time[2]	most recent sample time	-	
epc_most_recent_sample_bottle[2]	most recent sample bottle	-	bottle number
epc_most_recent_sample_volume[2]	most recent sample volume	-	in [ml]
epc_most_recent_sample_result[2]	most recent sample result	-	see chapter 7.1.2 in [3]
epc_checksum[2]	checksum of `STS,1` reply	-	for checksum calculation see section 4.6.1

[1] The prefix mc stands for *Menu Control* mode

[2] Data from `STS,1<CR>` reply (see section 4.6.1.2). The prefix epc stands for *External Program Control* mode.

[3] For more details about the error, the display of the 6712 has to be checked. A possible error is for example a power failure (see Section 4.3.2).

Table D.3: Control algorithm VS output data.

Name	**Description**	**Val**	**Value Description**
device_id	device id of *Core Station* connected with 6712 water sampler	-	device ID
data_type	type of generated data	0 1	sensor value evaluation data schedule generation data
start_condition_threshold	start condition threshold	-	in $[\mu S/cm]$
start_condition_hold_time_in_min	hold time periode	-	in $[min]$
window_size_in_min	window size of min/max calculation	-	in $[min]$
minimum_nb_of_data_points_within_hold_time	min nb of sensor values within hold time	-	[]
start_condition_fulfilled	start condition state	0 1 2 3	not fulfilled fulfilled not enough values within hold time sensor value outside window
min	minimum sensor value in window	-	in $[\mu S/cm]$
max	maximum sensor value in window	-	in $[\mu S/cm]$
min_max_delta	max-min	-	in $[\mu S/cm]$
control_algorithm_enable_state	state of control algorithm	0 1	disabled enabled
schedule_generation_event	event for state machine	-	see Figure 5.8
schedule_generation_state	new state of state machine	-	see Figure 5.8
auto_schedule_generation_reset_time_in_min	only for testing[1]	-	in $[min]$

[1] Obsolete

Table D.4: Outlier and noise filter VS output data.

Name	**Description**	**Val**	**Value Description**
raw_value	raw sensor value	-	in $[\mu S/cm]$
outlier_filter_median	median of sensor data within window	-	in $[\mu S/cm]$
outlier_filter_lower_bound	lower bound for classification	-	in $[\mu S/cm]$
outlier_filter_upper_bound	upper bound for classification	-	in $[\mu S/cm]$
outlier_filter_classification	classification of sensor value	0	outlier classified
		1	non-outlier classified
		2	not classified
outlier_filter_value	outlier filtered sensor value[1]	-	in $[\mu S/cm]$
noise_filter_value	noise filtered sensor value[2]	-	in $[\mu S/cm]$

[1] Equals to raw value if classified as non-outlier
[2] Noise filtering takes place after outlier filtering

Appendix E

CD Content

The following directory structure is contained on the enclosed CD:

- `Isco6712Tests`: Test software to evaluate the behavior of the 6712 water sampler.
- `Literature`: Literature used in this thesis (papers, datasheets etc).
- `PermaSenseGSN`: Source files for core station and GSN software written/added during this thesis. For the current version we refer to the PermaSenseGSN SVN repository.
- `Presentation`: Slides of initial, mid-term and final presentation.
- `Report`: The LaTeX sources of this thesis.
- `SimulationAndTest`: Matlab scripts for different simulations and tests.

Appendix F

Development Environment

The following chapters list information concerning the development environment collected during this thesis.

F.1 Core Station

Execution Environment Gumstix

Configuration The configuration of the *Core Station* is defined in the backlog.cfg file located at:

- CS: ./media/card/backlog/backlog.cfg
- CS: ./etc/backlog.cfg (link)
- PC: ./PermaSenseGSN/tools/backlog/python/

Plugin Plugins are programmed in Python. The plugins to be run need to be configured in the configuration file (backlog.cfg). The plugins are stored in the following folders:

- CS: ./media/card/backlog/python2.6/
- PC: ./tools/backlog/python/

A file can be uploaded from a PC to the *Core Station* with the following command: `scp file1 file2 root@host:/media/card/backlog/python2.6/` (e.g. `scp fileName.py ...`
`...  root@permasense-etz-bs02.ethz.ch:/media/card/backlog/python2.6/`)

Start/Stop Execution Before the backlog software on the *Core Station* can be started a connection with the *Core Station* needs to be established. A connection can be established with the SSH protocol (`ssh root@host`). After a SSH connection has been established, the backlog software can be started with the command `/etc/init.d/backlog start`. Starting the backlog software creates a new screen called `xxxx.backlog` in which the software runs. For screen commands we refer to appendix G.3. The backlog software is stopped by the short key `ctrl c` (for this the backlog screen has to be attached). The backlog start and stop procedure can be modified by manipulating the file `/etc/init.d/backlog`. Because this file is write protected, the system has to be remounted to gain temporary write access (listing F.1).

Listing F.1: Temporary write access on the *Core Station's* filesystem.

```
1 mount -o remount,rw /
2 ... do write stuff here ...
3 mount -o remount,ro /
```

Logging The logging can be configured in the configuration file. Logs are stored in `/var/volatile/log/backlog.*`.

Database The data from the sensors are temporarily stored in the following file `./media/card/backlog/backlog.db`.

Schedule The location and name of the schedule file of the schedule plugin is configured in the configuration file (e.g. `schedule_file = /media/card/schedule_file`).

F.1.1 Remote Control of Serial Interface

A remote control of the serial interface (`/dev/ttyUSB0`) can be established as follows:

1. Stop backlog software if a plugin uses the serial interface
2. Established ssh connection (`ssh root@permasense-etz-bs02.ethz.ch`)
3. Run terminal program (`socat -,raw,echo=0 /dev/ttyUSB0, raw,echo=0,b19200`)

F.2 GSN

Execution Environment PC or Server

Configuration There is not a specific configuration file to configure the plugins to be run. The plugins are loaded depending on the VSD file in the folder `./virtual-sensors/*.xml`. A general configuration file is `./conf/gsn.xml`.

Plugin Plugins are programmed in Java. The plugins are located in the following folder `./src/gsn/wrappers/backlog/plugins/`.

Start/Stop Execution The GSN server can be run on the computer. The application is started with the command `./PermaSenseGSN ant gsn`. To stop the GSN server `./PermasenseGSN/PermaSenseGSN ant stop` needs to be entered.

Logging The logging can be configured in the `./conf/log4j.properties` file. Logs are stored in `./logs/*`.

Position Mapping `./conf/permasense/location-positionmapping.cvs` location = virtual-sensor name prefix (i.e. name before underscore in xml-file)

Database The location of the database is configured in `./conf/gsn.xml`

F.2.1 GSN Test Private

Server Login ssh perma@whymper.ee.ethz.ch

Location of Test Private `/home/perma/permasense_gsn/gsn-test-private`

Start/Stop Execution The GSN server is running in the screen `gsn-test-private`. To stop the GSN server type `ant stop` in the location of the GSN test private (`/home/perma/permasense_gsn/gsn-test-private ant stop`), not in the screen! To start the GSN Test Private attach to the screen (`screen -r gsn-test-private`) and type `ant gsn` in the screen.

Plugin The plugins are located in the following folder

`/home/perma/permasense_gsn/gsn-test-private/ ... ... src/gsn/wrappers/backlog/plugins`.

Appendix G

Linux

The following chapters list Linux commands collected during this thesis.

G.1 Commands

cat filename displays the content of a file

dmesg — grep tty list serial ports

nano filename opens the file in a text editor

pgrep look up or signal processes based on name and other attributes

G.2 socat

socat is a relay for bidirectional data transfer between two independent data channels. socat is installed on the Gumstix and used to access the serial port.

```
socat -,raw,echo=0 /tmp/myttyUSB0,raw,echo=0,b19200
```

Data Transfer Loop Transfer loop between virtual port and real port and displaying of communication:

```
socat -v -x PTY,link=/tmp/myttyUSB0,raw,echo=0,isig=0,
  icanon=0 /dev/ttyUSB0,raw,echo=0,isig=0,icanon=0
```

`icanon=1`: Sets canonical mode to enable line buffering. If buffering is enabled

`-v`: Writes the transferred data not only to their target streams, but also to stderr. The output format is **text** with some conversions for readability, and prefixed with `>` or `<` indicating flow directions.

`-x`: Writes the transferred data not only to their target streams, but also to stderr. The output format is **hexadecimal**, prefixed with `>` or `<` indicating flow directions. Can be combined with -v .

`-d ...` `-d` log message

`-lf <logfile>` Writes messages to `<logfile>` instead of stderr

G.3 screen

Screen is a full-screen window manager that multiplexes a physical terminal between several processes. In the following some commands are listed.

G.3.1 Getting In

screen -ls list running sessions/screens

screen -x Attach to a not detached screen

screen -r <name> reattach to a detached screen process

screen -S <name>-d -m start a nwe screen session in detached mode

screen -S <name> start a new screen session with session name

screen -S <name>-X kill stop a detached session. There is a known problem that "A screen session started in detached mode cannot accept any commands unless the the session is first attached, and then re-detached." [38] Solution: **screen -S <name>-p 0 -X kill**

G.3.2 Command Line Options

-L tells screen to turn on automatic output logging for the windows. All data is logged into the file `screenlog.n` In case of logging on the core station it has to be ensured that the screen is started at a location where write access is available (e.g. `/media/card`)

G.3.3 Getting Out

Ctrl-a d detach screen

Ctrl-a k kill current screen

G.3.4 Logging

Ctrl-a H Begins/ends logging of the current window to the file 'screenlog.n'

Glossary

API application programming interface. 32, 79

EC electrical conductivity. 39, 61, 65–67, 69, 71, 73–76, 87

GPRS general packet radio service. 8, 9, 15

GSN global sensor networks. 8, 9, 35–40, 42, 44, 46, 49, 50, 52, 58, 73–76, 79, 81–84, 86

SDI-12 (Serial Data Interface at 1200 Baud) is a asynchronous, ASCII, serial communication protocol developed for intelligent sensory instrumentation. The communication is achieved by digital communications along a single line. 19, 23, 30

SQL (Structured Query Language) is a special-purpose programming language designed for managing data in relational database management systems [39]. 44, 46

SSH (Secure Shell) is a network protocol for secure data communication between two networked computers [40]. 59

VS virtual sensor. 36–40, 42–46, 74, 82, 83

VSD virtual sensor description. 40, 42, 43, 45, 46, 75, 99, 115

WFD water framework directive. 1, 10

WLAN wireless local area network. 8, 9, 15, 35, 46, 79

WSAN wireless sensor actor network. 3, 5, 6, 13, 15

WSN wireless sensor network. 2, 3, 5–10, 13–17, 36, 38, 46, 71, 76, 79, 84

Bibliography

[1] European Union. Directive of the european parliament and of the council 2000/60/ec establishing a framework for community action in the field of water policy. In *Official Journal of the European Community*, pages 1–72, 2000.

[2] P. Schneider. *Hydrologische Vernetzung und ihre Bedeutung für diffuse Nährstoffeinträge im Hotzenwald/Südschwarzwald.* PhD thesis, Geographisches Institut der Universität Basel, 2007.

[3] Teledyne ISCO. *6712 Portable Samplers, Installation and Operation Guide*, April 11 2011.

[4] J. Beutel, S. Gruber, A. Hasler, R. Lim, A. Meier, C. Plessl, I. Talzi, L. Thiele, C. Tschudin, M. Woehrle, and M. Yuecel. Permadaq: A scientific instrument for precision sensing and data recovery under extreme conditions. pages 265–276. ACM, 2009.

[5] Wireless sensor and actor networks (WSAN), "Why Actors?". `http://www.ece.gatech.edu/research/labs/bwn/actors/`. Accessed: 29.05.2012.

[6] S.F. Li. Wireless sensor actuator network for light monitoring and control application. In *Consumer Communications and Networking Conference, 2006. CCNC 2006. 3rd IEEE*, volume 2, pages 974 – 978, jan. 2006.

[7] S. Dengler, A. Awad, and F. Dressler. Sensor/actuator networks in smart homes for supporting elderly and handicapped people. In *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on*, volume 2, pages 863 –868, may 2007.

[8] X. Zhao, H. Gao, G. Zhang, B. Ayhan, F. Yan, C. Kwan, and J.L. Rose. Active health monitoring of an aircraft wing with embedded piezoelectric sensor/actuator network: I. defect detection, localization and growth monitoring. *Smart Materials and Structures*, 16(4):1208, 2007.

[9] E.C.H. Ngai, M.R. Lyu, and J. Liu. A real-time communication framework for wireless sensor-actuator networks. In *Aerospace Conference, 2006 IEEE*, page 9 pp., 0-0 2006.

[10] F. Xia, Y.C. Tian, Y. Li, and Y. Sung. Wireless sensor/actuator network design for mobile control applications. *Sensors*, 7(10):2157–2173, 2007.

[11] I.F. Akyildiz and M.C. Vuran. *Wireless Sensor and Actor Networks*, pages 319–348. John Wiley & Sons, Ltd, 2010.

[12] I.F. Akyildiz and I.H. Kasimoglu. Wireless sensor and actor networks: research challenges. *Ad Hoc Networks*, 2(4):351–367, 2004.

[13] Y.J. Wen and A.M. Agogino. Wireless networked lighting systems for optimizing energy savings and user satisfaction. In *Wireless Hive Networks Conference, 2008. WHNC 2008. IEEE*, pages 1–7, aug. 2008.

[14] A. Hasler, I. Talzi, J. Beutel, C. Tschudin, and S. Gruber. Wireless sensor networks in permafrost research – concept, requirements, implementation and challenges. In *Proc. 9th International Conference on Permafrost (NICOP)*, 2008.

[15] J. Beutel, B. Buchli, F. Ferrari, M. Keller, L. Thiele, and M. Zimmerling. X-sense: Sensing in extreme environments. In *Proceedings of Design, Automation and Test in Europe, 2011 (DATE 2011)*, Grenoble, France, 2011.

[16] H. Dubois-Ferrière, L. Fabre, R. Meier, and P. Metrailler. Tinynode: a comprehensive platform for wireless sensor network applications. In *Proceedings of the 5th international conference on Information processing in sensor networks*, IPSN '06, pages 358–365, New York, NY, USA, 2006. ACM.

[17] M. Keller, M. Woehrle, R. Lim, J. Beutel, and L. Thiele. Comparative performance analysis of the permadozer protocol in diverse deployments. In *Proceedings of the Sixth IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2011)*, pages 969–977, Bonn, Germany, 2011. IEEE.

[18] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: Ultra-low power data gathering in sensor networks. In *International Conference on Information Processing in Sensor Networks (IPSN)*, Cambridge, Massachusetts, USA, 2007.

[19] K. Aberer, M. Hauswirth, and A. Salehi. The Global Sensor Networks middleware for efficient and flexible deployment and interconnection of sensor networks. Technical report, 2006. Sumitted to ACM/IFIP/USENIX 7th International Middleware Conference.

[20] M. Keller, M. Yuecel, and J. Beutel. High resolution imaging for environmental monitoring applications. In *International Snow Science Workshop 2009: Programme and Abstracts*, pages 197–201, Davos, Switzerland, 2009.

[21] J.W. Kirchner, X. Feng, C. Neal, and A.J. Robson. The fine structure of water-quality dynamics: the (high-frequency) wave of the future. *Hydrological Processes*, 18(7):1353–1359, 2004.

[22] C. Neal, B. Reynolds, P. Rowland, D. Norris, J.W. Kirchner, M. Neal, D. Sleep, A. Lawlor, C. Woods, S. Thacker, H. Guyatt, C. Vincent, K. Hockenhull, H. Wickham, S. Harman, and L. Armstrong. High-frequency water quality time series in precipitation and streamflow: From fragmentary signals to scientific challenge. *Science of The Total Environment*, (0):–, 2012.

[23] A. Facchi, C. Gandolfi, and M.J. Whelan. A comparison of river water quality sampling methodologies under highly variable load conditions. *Chemosphere*, 66(4):746–756, 2007.

[24] C.E. Müller, N. Spiess, A.C. Gerecke, M. Scheringer, and K. Hungerbühler. Quantifying diffuse and point inputs of perfluoroalkyl acids in a nonindustrial river catchment. *Environmental Science & Technology*, 45(23):9901–9909, 2011.

[25] M. Hayashi, T. Vogt, L. Mächler, and M. Schirmer. Diurnal fluctuations of electrical conductivity in a pre-alpine river: Effects of photosynthesis and groundwater exchange. *Journal of Hydrology*, (0):–, 2012.

[26] L.E. Gentry, M.B. David, F.E. Below, T V. Royer, and G.F. McIsaac. Nitrogen mass balance of a tile-drained agricultural watershed in east-central illinois. *Journal of Environmental Quality*, 38(5):1841–1847, September 2007.

[27] T.V. Royer, L.E. Gentry, M.B. David, C.A. Mitchell, and K.M. Starks. Phosphorus transport pathways to streams in tile-drained agricultural watersheds. *Journal of Environmental Quality*, 36(2):408–415, March 2007.

[28] P. Williams. VT100 User Guide. `http://vt100.net/docs/vt100-ug/`. Accessed: 29.05.2012.

[29] PermaSense Wiki. `http://people.ee.ethz.ch/~nccr/permasense/wiki`.

[30] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[31] M.F. Ismail, K. Yanagi, and A. Fujii. An outlier correction procedure and its application to areal surface data measured by optical instruments. *Measurement Science and Technology*, 21(10):105105, 2010.

[32] P.L. Davies, R. Fried, and U. Gather. Robust signal extraction for on-line monitoring data. *Journal of Statistical Planning and Inference*, 122(1–2):65–78, 2004. Contemporary Data Analysis: Theory and Methods in.

[33] P.H. Menold, R. Pearson, and F. Allgöwer. Online outlier detection and removal. In *Mediterranean on Control and Automation*, pages 1110–1133, Haifa, Israel, 1999.

[34] S. Basu and M. Meckesheimer. Automatic outlier detection for time series: an application to sensor data. *Knowledge and Information Systems*, 11:137–154, 2007. 10.1007/s10115-006-0026-6.

[35] I. Pitas and A.N. Venetsanopoulos. *Nonlinear digital filters: principles and applications*. Kluwer international series in engineering and computer science: VLSI, computer architecture, and digital signal processing. Kluwer Academic Publishers, 1990.

[36] Decagon Devices. *5TE Water Content, EC and Temperatur Sensors Operator's Manual*, 7 edition, 2010.

[37] Decagon Devices. *5TE 5TM Integrators Guide*, 9 edition.

[38] Stop detached screen session. `http://fvue.nl/wiki/Screen:_Start/stop_detached_session_via_one_command`. Accessed: 22.06.2012.

[39] Wikipedia the free encyclopedia, SQL. `http://en.wikipedia.org/wiki/SQL`. Accessed: 22.06.2012.

[40] Wikipedia the free encyclopedia, Secure Shell. `http://en.wikipedia.org/wiki/Secure_Shell`. Accessed: 22.06.2012.