Reducing wireless control communication for a water irrigation system

Bas Boot

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Reducing wireless control communication for a water irrigation system

Master of Science Thesis

For the degree of Master of Science in Embedded Systems at Delft University of Technology

Bas Boot

August 18, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis is supported by the European Research Council through the SENTIENT project (ERC-2017-STG #755953)

Abstract

For this thesis we have compared communication times of different control schemes on a wireless control network for a water irrigation system. Therefore a control application which can run different protocols was created. We have shown that a significant reduction of communication time can be achieved by using event-triggered control (ETC), and by using self-triggering techniques we could reduce this even further. Wireless control networks (WCN) are networks of one or multiple controllers, sensors and actuators which share a digital communication network. Sensors send their measurements to the controller, and the controller sends a control signal to the actuators. Because the nodes are digital devices, periodic control is the standard approach. Communication therefore is not needed all the time, and nodes can sleep in between updates to save energy. Event-triggered control is a technique to further reduce communication of resource constrained nodes in the network by only updating control when an event, typically a significant change since the last control update, occurs. Water irrigation systems (WIS) are networks of channels transporting water from main rivers, to smaller rivers which need to be controlled to avoid losses. At TU Delft a model of a water irrigation system with three pools has been connected to a WCN to simulate this. To create an application for this testbed, supporting different control schemes, a low power embedded OS capable of doing multitasking with a replaceable network stack is needed. The Contiki Operation System is capable to do this and has been used by the D3S Research Group of the University of Trento to build wireless control bus (WCB), and has also been used for the WIS control application. WCB uses network floods from a protocol called Glossy and a schedule to synchronise communication with low power, and supports both periodic and event-triggered control. Experiments with WCB show periodic control is possible with only 7% communication time for this setup, and event-triggered control with only 2% without much loss of performance. Because the testbed unit has some technical problems, a hardware-in-the-loop simulation was created to connect to the WCN to do the experiments. A further reduction in communication has been achieved by combining a predictive control method called PSTC with ETC, which we call ETC+. Advantages over normal ETC are not large in our experiments, and there are still practical problems, but they show this is a promising research area.

Table of Contents

List of Figures

List of Tables

Preface and acknowledgements

The past year I have been working on building a control application for the wireless control network of the water testbed unit at TU Delft. This was not easy, because to do the measurements to create the application and determine the requirements an application to do these measurements was also needed. I have been pragmatic and have first built the application to do the measurements, and built the control application on top of it. Because there were a lot of problems with the testbed unit I decided to switch to a hardware-in-the-loop simulation, which took me some extra time at first but also gave me the opportunity to speed up by fully working from home during the Covid-measures.

Of course I would like to thank dr. ir. Manuel Mazo Jr. for giving me the opportunity to do my master thesis at his research group, and supporting me during the weekly group meetings via Teams... which I was always allowed to start first thanks to alphabetical ordering of attendants in video calls.

I also want to thank my daily supervisor ir. Gabriel de Albuquerque Gleizer for his help and advice. Especially during the 'final push'!

Besides my supervisors I had help from Jacob Lont to start with the testbed unit and Wim Wien who helped me with all kinds of problems on the testbed. Also Iimofei Istomin and Matteo Trobinger from the University of Trento helped me by sharing the wireless control bus code with me. Thank you.

Another person I would like to mention is John, a friend and colleague, who seemed to enjoy my study the past few years just as much, or sometimes even more than me, and kept me motivated.

But most of all I would like to thank my kids, Gijs en Jet, for cheering me up, and my wife Mirja for supporting me along the way of my master studies.

Thanks!

Delft, University of Technology — Bas Boot
August 18, 2021

Chapter 1

Introduction

In control networks, idle listening is often the most energy consuming task of the network adapter. Networked control systems [18] are sensors, actuators and controllers which share a band limited digital communication network. Data from the sensor nodes is used by the controller(s) to create the control signals for the actuator nodes. Control can be done centralised on a dedicated controller to control the entire network, but also distributed to control local portions. Because the nodes are digital devices, control is inherently discrete and communication is periodic. In a wireless control system the nodes have radios to communicate. When using normal computers communicating over traditional wifi [6], radios are always on, making idle listening the most energy consuming task of a node as shown in Figure 1-1a. For resource constrained devices it would be better to optimise this to reduce energy consumption.

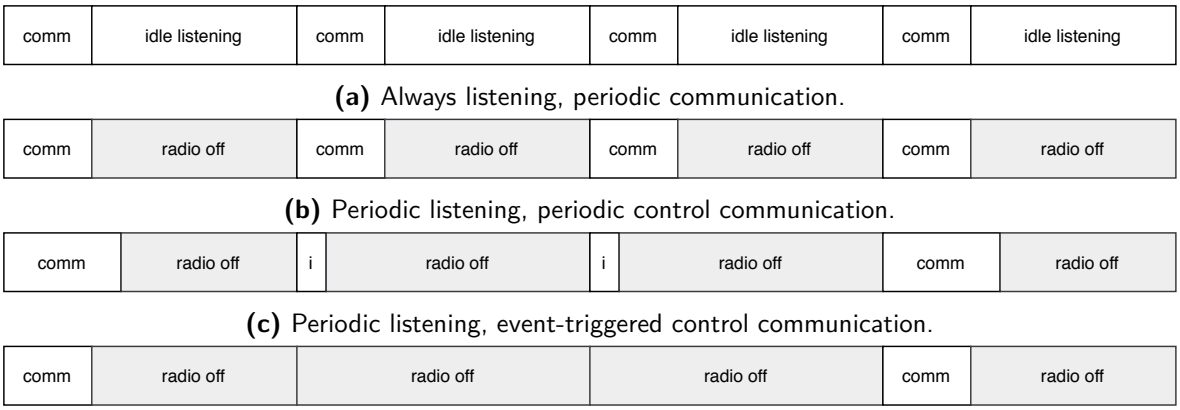
A more efficient approach therefore is to turn off the radio in between communications as shown in Figure 1-1b. Traditional periodic control is conservative in the sense that the period is chosen based on the worst case scenario. When using event-triggered control, where the system state is checked first to see if control is really necessary for this period, this radio scheme can be changed to further reduce radio on time. A periodic wake-up will still be necessary to check for events, but when there is no event, communication can be reduced for that period, as shown in Figure 1-1c. It will however have a little overhead because of the extra communication of the events.

Event-triggered control [17] (ETC) will be explained in Section 2-2, and the protocol used to reduce idle listening is Wireless Control Bus (WCB) developed by the D3S Research Group of the University of Trento [30], which will be explained in Section 2-3.

In self-triggered control, as described in Section 2-2-2, the controller dictates the sleeping times based on a prediction when an event will occur. The benefit is that this reduces radio on time, as shown in 1-1d. The downside is that it cannot react, like ETC, to unpredictable events during sleep which results in worse disturbance attenuation.

To overcome this a hybrid approach of event-triggered control and a predictive method can be considered for a practical implementation. In this approach the network sleeps a number of periods where no event should occur based on a prediction, but this prediction does take a

(bounded) disturbance into account. Because this prediction is conservative the event might not happen directly after waking up, so the control network can wait for it just like in ETC. We will call this hybrid approach ETC+, and the conservative estimations will be made using preventive self-triggered control [11], which will be discussed in Section 2-2-3.



(a) Always listening, periodic communication.

(b) Periodic listening, periodic control communication.

(c) Periodic listening, event-triggered control communication.

(d) Self-triggered listening and control communication only when an event occurs.

Figure 1-1: Idle listening comparison for different control schemes, during four periods of periodic control for a a case where only at the first and fourth period an event occurs which requires control.

1-1 Wireless water irrigation system

As a case to implement and compare different communication schemes a testbed of a water irrigation system (WIS) will be used. A WIS is a network of channels transporting water from main rivers, to smaller rivers. From these smaller rivers farmers take water to irrigate their land, causing disturbance in the WIS. Because water resources are becoming scarce in many areas [26] and losses can be large [32] in these networks, it is important that they are controlled. Because WISs are typically large, a network of communicating controllers can be effective to control such a system [23] [10]. At TU Delft a model of a water irrigation system with three pools [28] has been connected to wireless microcontrollers, called Fireflies [34], to simulate such a setup.

1-2 Project outline

In this thesis we use the results from Jacob Lont's master thesis [24] to implement a wireless control network on the water testbed unit at TU Delft using the protocols developed by the D3S Research Group of the University of Trento [30]. The goal is to compare energy usage and performance of periodic and event-triggered control over these protocols by comparing radio on times for different event-triggering configurations. Our last goal is to further improve energy savings by making the protocols aperiodic by adding extra sleeping time.

We start by presenting the preliminaries of this thesis in Chapter 2. In Section 2-1 we will look at the procedure followed [24] to control water irrigation systems. In Section 2-2 event-based

control is described and in Section 2-3 we look at the protocols to implement control on the wireless sensor network.

After that, in Chapter 3 the testbed at TU Delft will be described and identified so in Chapter 4 we can design for it using the preliminaries. Control implementations on the sensor network will be described in Chapter 5 and the results of experiments of the different implementations will be presented in Chapter 6. We end this thesis with our conclusions and recommendations for future work in Chapter 7.

The contributions of this thesis are:

1. Improvement of the hardware and software of the water testbed (Section 3-4, 3-7).
2. Identification of the pools of the testbed (Section 3-5).
3. Adaption of the model, to use undershot gates which are smart (Section 2-1-1, 3-6).
4. Controller design for the testbed (Chapter 4).
5. Adaptation of WCB-E and WCB-P to support aperiodic communication schemes (Section 5-1, 5-2).
6. Documentation of the WCB protocol implementation (Section 5-3, B-2).
7. Control implementation on the Fireflies (Section 5-4 to 5-9).
8. A hardware-in-the-loop simulation (Section 5-11)
9. Experimental results of the different control strategies (Chapter 6).

1-3 Notation and definitions

We denote by $\mathbb{R}$ the set of real numbers, and by $\mathbb{R}_+ := \{x \in \mathbb{R} : x \geq 0\}$. For a vector $x \in \mathbb{R}^n$ we denote 2-norm by $\|x\| := \sqrt{x^\top x}$. For a matrix $A \in \mathbb{R}^{n \times m}$ we denote by A^T its transpose, by $rank(A)$ its rank and by $\lambda(A)$ its eigenvalues. A matrix A is said to be Hurwitz when all its eigenvalues have strictly negative real parts. We denote by I the identity matrix. A symmetric square matrix $P \in \mathbb{R}^{n \times n}$ is written as $P \succ 0 (P \succeq 0)$ if P is positive (semi) definite. $\|H\|_\infty := \max_\omega \bar{\sigma}(H(j\omega))$ is used to denote the infinity norm of H, where $\bar{\sigma}(\cdot)$ denotes the maximum singular value of the matrix argument. The magnitude of a transfer function $T(j\omega)$ at a certain frequency ω is denoted as $|T(j\omega)|$.

Symbols in formulas will be explained near their usage. They are used like they were used in the literature they came from, unless this leads to confusion because the symbols are already used with another meaning. If different literature about the same subject is referenced near each other but they do not use the same notation this is adapted for clarity.

Chapter 2

Preliminaries

2-1 Water irrigation control

The water testbed unit at the TU Delft has three pools for which we want to control the water levels, by actuating undershot gates via a wireless control network. To design a control system a model of the plant and control setup is necessary. In Section 2-1-1 therefore we model the pools and the gates of the plant, and the design of the controller will be explained in Section 2-1-2, 2-1-4 and 2-1-3. For simulation the plant and controller can combined (Section 2-1-5).

2-1-1 Downstream control model

To be able to use the downstream control model from [10] we need to convert the model from overshot gates to undershot gates for our testbed.

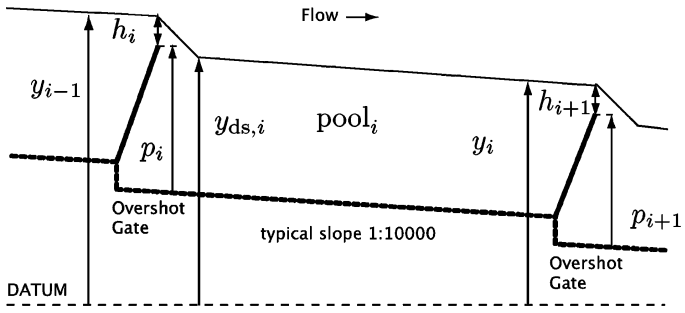


Figure 2-1: Stretch of open-water channel with over-shot gates from [10].

An irrigation network can be seen as a string of pools for which reference takes place at the end of each pools because that is where off-takes typically take place. In [10] a model for the water level y_i at time t of a single pool with overshot gates is considered in the form of

$$\pi_i \left(\frac{d}{dt}\right) y_i(t) = \gamma_i h_i^{3/2}(t-\tau_i) - \gamma_{i+1} h_{i+1}^{3/2}(t) - d_i(t).$$

In this model $\pi_i(.)$ is a polynomial describing the dynamics of the pool, τ_i is the delay caused by the water having to flow from the beginning to the end of $pool_i$ (Figure 2-1), and d_i is the disturbance at the end of the pool.

$\gamma_i h_i^{3/2}$ and $\gamma_{i+i} h_{i+i}^{3/2}$ are the inflow and outflow of the overshot gates of the pool. In which γ_i is a constant for $gate_i$ and h_i is the so called 'head over the gate' which is the difference between the water level (h_i) and the top of the gate (p_i). The dynamics of the pool can be modelled accurately using a third-order model [23] but if the controller is designed with a bandwidth below the dominant wave dynamics the model can be simplified to

$$\alpha_i \frac{d}{dt} y_i(t) = \gamma_i h_i^{3/2}(t-\tau_i) - \gamma_{i+1} h_{i+1}^{3/2}(t) - d_i(t),$$

where α_i is the surface area of the pool.

By defining the inflow $u_i = \gamma_i h_i^{3/2}$ and outflow $v_i = u_{i+1} = \gamma_{i+1} h_{i+1}^{3/2}$ and applying the Laplace transform a more general frequency domain model for $pool_i$ becomes

$$P_i : y_i(s) = \frac{1}{s\alpha_i} \left(\exp\left(-s\tau_i\right) u_i(s) - v_i(s) - d_i(s) \right).$$

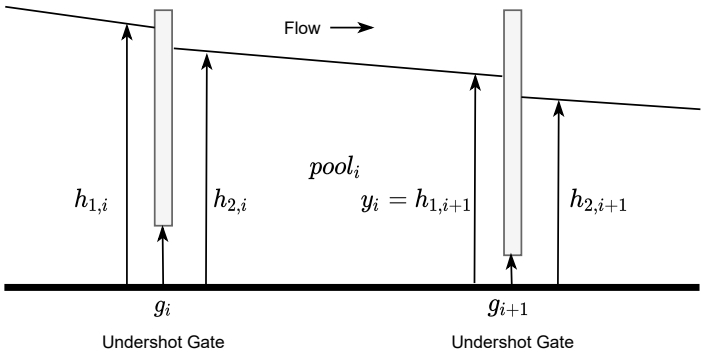


Figure 2-2: Stretch of the water testbed unit with undershot gates.

This general model with flows instead of gate-properties can also be applied to the testbed unit with undershot gates (Figure 2-2) if we redefine u_i and v_i to model the flow over an undershot gate.

The model derived in [24] for the flow over an undershot gate does take flows in both directions into account, but under the assumption that the downstream level is always lower, and there is no sill under the gates so they can be lowered to the bottom of the pool, it can be simplified to

$$u_i = b_i \cdot C_{e,i} \cdot \sqrt{q_i} \cdot y_{jet,i} \cdot \sqrt{2 \cdot g},$$

in which $C_{e,i}$ and b_i are the flow efficiency and width of gate i, g is the gravitational acceleration,

$$q_i = h_{1,i} - \max\left[h_{2,i}, \min\left(h_{1,i} \cdot \frac{2}{3}, \mu_i \cdot g_i \right) \right],$$

and

$$y_{jet,i} = \min\left[\mu_i \cdot g_i, \max\left(h_{2,i}, h_{1,i} \cdot \frac{2}{3} \right) \right],$$

in which μ_i denotes the flow-depth constant. When we make the assumption that the gates are always fully submerged, and the water level differences are normally small around the reference point, and take from [24] that $\mu_i = 1$, we can make an extra simplification of the model:

$$u_i = g_i \gamma_i \sqrt{h_{1,i} - h_{2,i}}, \tag{2-1}$$

because

$$q_i = h_{1,i} - h_{2,i}, \qquad y_{jet,i} = g_i, \qquad \gamma_i := b_i \cdot C_{e,i} \cdot \sqrt{2 \cdot g}.$$

The outflow then becomes

$$v_i = g_{i+1} \gamma_{i+1} \sqrt{h_{1,i+1} - h_{2,i+1}}.$$

In these formulas γ_i now is a constant for the undershot gate, g_i is the gate opening and $h_{1,i}$ and $h_{2,i}$ are the water levels on both sides of the gate as illustrated in Figure 2-2.

The full irrigation system can be seen as a string of pools, Figure 2-3, with $v_i(s) = u_{i+1}(s)$ and boundary condition $v_N \equiv 0$ so there is no flow over the last gate, and the outflow of $pool_i$, is the inflow for $pool_{i+1}$.

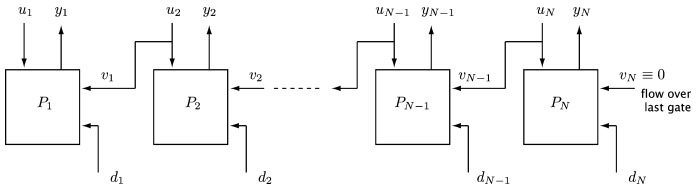


Figure 2-3: Irrigation channel modelled as a string of N pools from [10].

2-1-2 Controller design

Design of the controller can be split in a local part to control a single pool, and a global part that takes the full system into account.

When designing water irrigation control, [24] followed the distributed approach from [23] in which a localised portion (single pool) of a channel is considered.

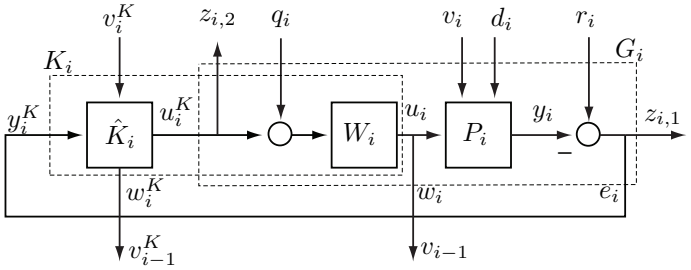


Figure 2-4: Localised portion of a controlled channel from [23].

In this approach the controller K_i is split in a part $\hat{K}_i$ and a part W_i. This way a local controller for the plant P_i can be created using classic loop-shaping techniques only aimed at local characteristics of the pool, and the resulting loop-shaping weight can be used as an input for H_∞ loop-shaping of the distributed controller which aims at global performance. This way the local controllers treat outflow (v_i) like normal disturbance (d_i), while $\hat{K}_i$ can exploit this knowledge. Other disturbances are the input disturbance q_i and output disturbance r_i.

The full system can then be seen as a series of distributed controllers ($\hat{K}_i$) which control the generalised pools (G_i) formed by combining the local controller and the plant, as shown in Figure 2-5.

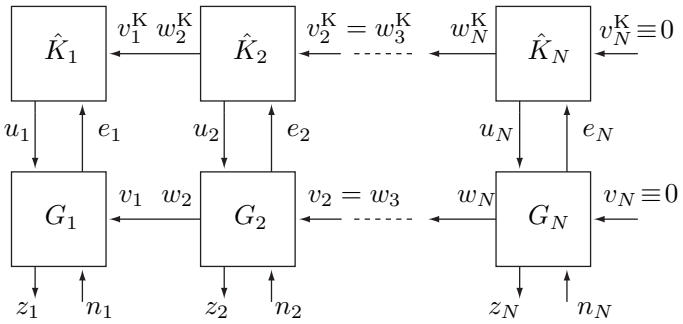


Figure 2-5: Generalised distributed structure for synthesis from [23]

Because we want to implement global control on a single node in the network, the distributed controllers will be combined later.

2-1-3 Local compensator

The local controller only has to take one pool into account, and can be realised with just a PI-compensator. To design a compensator for local control of a pool the first order model of Section 2-1-1 can be used. The compensator will have to shape the loop-gain $L_i(s) = W_i(s)/s\alpha_i$. Because $L_i(s)$ does not reflect the wave dynamics which can disturb the flow over the gate, and also does not reflect the delay τ_i of the system it is necessary to make sure the bandwidth of the loopgain $|L_i(j\omega)|$ lies below the dominant local wave dynamics and $1/\tau_i$ rad/min. The off-takes of the irrigation channel are step disturbances to the system, so from the final value theorem it follows that the local compensator $W_i(s)$ should have at least one pole at $s = 0$ to achieve zero steady-state error. Both criteria and large loop-gain at low frequencies for set-point regulation and disturbance rejection can be achieved with a compensator of the form:

$$W_i(s) = \frac{\kappa_i \left(1 + s\phi_i\right)}{s\left(1 + s\rho_i\right)}.$$

In this compensator κ_i is used to set the loop-gain bandwidth, ϕ_i for phase lead in the cross-over section and ρ_i to add additional roll-off beyond the loop-gain bandwidth.

2-1-4 Global controller

The global controller takes global knowledge of the system into account. For this robust control techniques will be used. To design a global controller using H_∞ synthesis the loop-shaping weights (Section 2-1-3) need to be combined in a finite-dimensional state-space model that approximates G_i (Section 2-1-2) and delays of the pools. To represent the delay a Padé approximation is used.

$$\begin{pmatrix} \dot{x}_i \\ w_i \\ z_i \\ y_i^K \end{pmatrix} = \begin{pmatrix} A_i^{tt} & A_i^{ts} & B_i^{tn} & B_i^{tu} \\ A_i^{st} & A_i^{ss} & B_i^{sn} & B_i^{su} \\ C_i^{tz} & C_i^{sz} & D_i^{zn} & D_i^{zu} \\ C_i^{yy} & C_i^{sy} & D_i^{yn} & D_i^{yu} \end{pmatrix} \begin{pmatrix} x_i \\ v_i \\ n_i \\ u_i^K \end{pmatrix}$$

$$:= \left(\begin{array}{cccc|c|cc|cc} 0 & \frac{1}{\alpha_i} & \frac{-1}{\alpha_i} & 0 & \frac{-1}{\alpha_i} & 0 & \frac{-1}{\alpha_i} & 0 & 0 \\ 0 & \frac{-2}{\tau_i} & \frac{4}{\tau_i} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \frac{\kappa_i\phi_i}{\rho_i} & \frac{\kappa_i\phi_i}{\rho_i} \\ 0 & 0 & 0 & \frac{-1}{\rho_i} & 0 & 0 & 0 & \frac{\kappa_i(\rho_i-\phi_i)}{\rho_i^2} & \frac{\kappa_i(\rho_i-\phi_i)}{\rho_i^2} \\ \hline 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right) \begin{pmatrix} x_i \\ v_i \\ n_i \\ u_i^K \end{pmatrix} \tag{2-2}$$

The local state is $x_i = (y_i, \Delta_i, u_i, \Omega_i)^T$, in which the sub-state Ω_i corresponds to the loop-shaping pole at $s = \frac{-1}{\rho_i}$ and Δ_i to the Padé approximation of the delay, $w_i = u_i$, $z_i = (e_i, u_i^K)^T$. The disturbances our grouped in the signal $n_i := (r_i, d_i, q_i)^T$.

The full formulation of the H_∞ problem to stabilise the interconnection $G = (G_1, ..., G_N)$ of the weighted generalised pool models (Equation 2-2), where $v_i = w_{i+1}$ and $v_N = 0$ and achieves $\|H(G, K)\| < \gamma$, where $H(G, K)$ denotes the closed-loop transfer function from the vector of signals $(n_1^T, ..., n_N^T)^T$ to $(z_1^T, ..., z_N^T)^T$ shown in Figure 2-3, can be found in [23]. The scripts to perform this robust control optimisation [24] will be used in Chapter 4 to create the controller.

2-1-5 Centralised control and simulation

For simulation and implementation, the distributed controllers $(\hat{K}_i)$ can be combined into a single centralised controller. Also the generalised pool descriptions (G_i) can be combined to create a model of the water irrigation system which is fully explained in [24] for simulation.

2-2 Event-based control

To reduce control communication in wireless control networks, periodic control can be replaced by aperiodic control. Event-triggered control is a reactive approach to do aperiodic control and is described in Section 2-2-1. Self-triggered control is a proactive approach (Section 2-2-2) which can be extended to preventive self-triggered control to take bounded disturbances into account (Section 2-2-3).

2-2-1 ETC

Nowadays almost all control is implemented using digital platforms and is therefore inherently discrete. Periodic control theory is well-developed and the standard approach when implementing discrete controllers [27], but for low power sensor/actuator networks with communication constraints it is increasingly popular to look at aperiodic controllers to reduce power consumption by reducing communication [17]. In cases where a system operates as needed, without disturbances acting on it recalculating control and sending actuation commands over the network is a waste of resources [16] because both bandwidth and computation power could be reduced without effect on the system. In such a case it is possible to use a sample-and-hold implementation where the input (u) is kept the same based on a previous version of the state ($\hat{x}$) and only update control when an event-triggering condition is met, typically a bound on the difference between the real state x and $\hat{x}$, to reduce the number of control task executions, as illustrated in Figure 2-6.

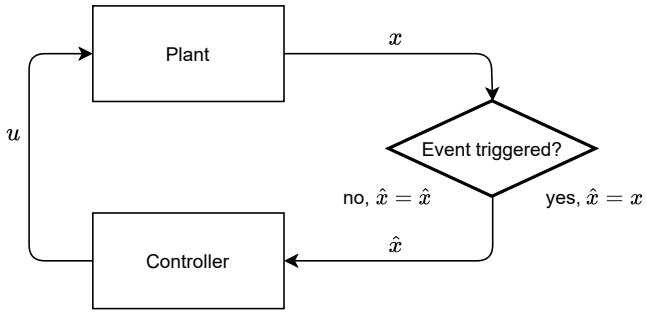


Figure 2-6: Event-triggered control schematic.

In a continuous control ETC approach (CETC) the triggering condition needs to be constantly monitored. With this approach it is necessary to avoid infinitely many triggers (Zeno behaviour [17]) by adding a threshold on the output triggering [12] to ensure a minimum inter-sampling time. When a periodic approach (PETC) is used however a minimum inter-sampling time will be implicitly set by the sampling period of the discrete controller. For a linear time-invariant (LTI) system, using the approach from [16] the plant can be described by:

$$\dot{x} = A^p x + B^p \hat{u} + B^w w, \text{ for } x \in \mathbb{R}^{n_x}, \hat{u} \in \mathbb{R}^{n_u}, w \in \mathbb{R}^{n_w},$$

where x and $\hat{u}$ are the state and input of the plant, and w is an unknown disturbance. For traditional periodic, or time-triggered, control a feedback controller would have the form of

$$\hat{u}(t) = Kx\left(t_k\right), \text{ for } t \in \left(t_k, t_{k+1}\right], t_k, k \in \mathbb{N},$$

in which the inter-sampling times for periodic control are always equal, so for a specific $h > 0$, chosen such that the system is stable and has desired performance, the control is updated based on the state x at time $t_k = kh$. For PETC this needs to be modified to only update control based on the actual state when a triggering-event occurs, which results in

$$\hat{u}(t) = K\hat{x}(t), \text{ for } t \in \mathbb{R}_+,$$

in which $\hat{x}$ is updated to the actual state after a trigger, and kept the same otherwise.

$$\hat{x}(t) = \begin{cases} x(t_k), & \text{when } \mathcal{C}(x(t_k), \hat{x}(t_k)) > 0 \\ \hat{x}(t_k), & \text{when } \mathcal{C}(x(t_k), \hat{x}(t_k)) \leqslant 0 \end{cases}$$

Note that an initial state $\hat{x}(0)$ needs to be set, which will will typically be $\hat{x}(0) = x(0)$.

A triggering condition is typically defined to update control when the state, plant or controller, has deviated so much that new control action is needed. An event-triggering conditions based on the state error [16] for example

$$\|\hat{x}(t_k) - x(t_k)\| > \sigma \|x(t_k)\|$$

triggers when the 2-norm of the difference between the kept and actual state is larger than a fraction (σ) of the 2-norm of the actual state. When σ is large there will be less triggers, than when it is small. If we assume the set point is at the origin, deviations will also trigger faster when we are near the origin than when we are not. Note that with $\sigma = 0$ every change will trigger, and $\sigma = 1$ no change will trigger anymore.

2-2-2 STC

Given the reactive nature of ETC, every sampling period the triggering conditions must be checked to see if control must be updated. Instead of checking the triggering conditions it is also possible to schedule the next control based on a prediction when an event will occur [31] called self-triggered control (STC). Because STC is proactive it is not necessary to monitor the plant every sampling period, and can therefore be used to reduce the transmission instances in a sensor/actuator network [7]. The prediction of the inter-sampling times must not only be calculated based on the plant model and the last measurement but also on a performance specification, because these inter-sampling times define both stability and performance of the controller. Longer inter-sampling times will result in a larger reduction, but also in smaller performance if external disturbances occur, or the model used for prediction is not perfect.

2-2-3 PSTC

STC does a one step prediction when an event will occur, but is mostly used in noiseless systems, or noise is disregarded like in [25]. Disturbance attenuation therefore is worse than with ETC which takes disturbances into account naturally. Because on the water testbed unit (Chapter 3) there are multiple disturbances present, we cannot ignore them and use this type of STC to predict events, to reduce radio on time by putting the network to sleep in between events. To still reduce radio on time, another method called preventive self-triggered control (PSTC) [11] is considered to do the predictions. PSTC does take unknown but bounded

disturbances into account, and makes a conservative prediction when an event might occur based on it. Note that because this is a conservative prediction of the earliest time an event can occur the actual control can be postponed until the event really occurs. The algorithm from [11] uses and observer to keep track of all possible states the plant and controller are in, for which it uses ellipsoids to describe the sets. To calculate the first possible instant an event might occur, the reachable sets for each observer state are calculated for a number of timesteps ahead, until a point in the reachable sets violates the triggering condition. This number of timesteps is then used as the prediction.

2-3 Wireless control protocols

To control the water testbed using the Firefly-nodes a wireless communication protocol supporting the intended control methods is needed. Wireless control bus (WCB) as implemented by the D3S Research Group of the University of Trento supports different control schemes. WCB was built on top of Glossy and Crystal which will be described first in Section 2-3-1 and 2-3-2. How periodic and event-triggered control are supported by WCB is described in Section 2-3-3, 2-3-4 and 2-3-5.

2-3-1 Glossy

The Glossy protocol provides a flooding architecture for wireless sensor networks. Where other architectures suffer from package loss due to interference [33], Glossy uses it to create constructive interference [14]. Reliability is created by synchronisation of the nodes to send a single flood, so all nodes send the same signal. Even when two nodes interfere, the capture effect [22] ensures that the strongest will be received. When a flood starts all nodes turn on their radios and relay received packages, which will trigger neighbouring nodes to do the same. This results in a single source being able to reach all other nodes in the network allowing a one-to-many communication scheme. Time synchronisation is implicit because there is only one source (initiator) in the network triggering a flood. To increase reliability event further the number of retransmission (NTX) of a node during a single flood can be increased.

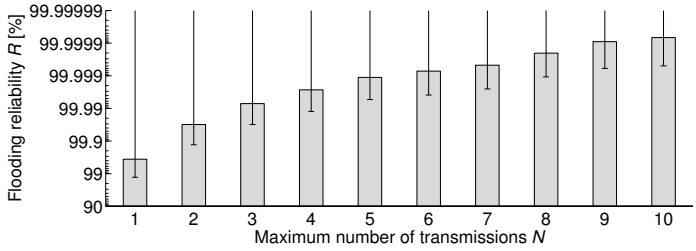


Figure 2-7: Flooding reliability for various N (NTX) from [14].

Experiments [14] show with NTX=1 Glossy's reliability is stable and nodes receive a package with a probability always above 98% and even 99.6% on average, and by increasing NTX flooding reliability increases almost logarithmically as shown in Figure 2-7.

Another benefit of Glossy is that its layout does not have to be configured. The range of the network can easily be increased by adding extra nodes for relay in between other nodes.

2-3-2 Crystal

Glossy allows only one-to-many communication. To allow other communication patterns (many-to-one and many-to-many) Low power wireless bus [13] (LWB) was created on top of Glossy. It uses a schedule so all nodes can be the initiator of one or multiple floods in turn during a round, and due to the way Glossy works all other nodes will receive the floods

and can pick the messages intended for them just like a bus in computer architecture. As illustrated in Figure 2-8 the network host controls the network and sends the schedule during the first flood (T_s) of a round. After that, all nodes send their data when it is their turn (T_d data) and at the end of the round they can send a package if they have different needs for the next round (T_d contention). The nodes can then turn off their radios until the next round which starts with a new schedule. Note that only one of the node's packages can be received per round by the host due to the capture effect [22].

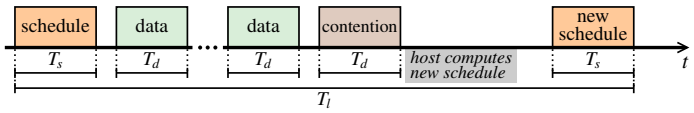


Figure 2-8: Communication slots within a round from [13].

The same idea as in LWB to create a wireless bus by using periodic schedules was used to create Crystal [19] which has also been built on top of Glossy. Crystal is a protocol to reliably send sensor data from multiple sensors to a single sink over a low power wireless network. There is no fixed schedule for each node, instead it has a flexible schedule which allows only nodes that need to send updated sensor data to send this data during a period (epoch). The sink (which is also the network controller) sends a Glossy package to synchronise (S) all nodes. Then each node that needs to sends data sends its data (T). Of all transmissions only one will reach the sink, but will be received with high probability due to the capture effects [22] exploited in Glossy as illustrated in Figure 2-9. The sink then sends an acknowledgement of the received data (A) after which the acknowledged node can go to sleep. The TA process repeats until no nodes send any data and the network controller can also go to sleep until the next epoch.

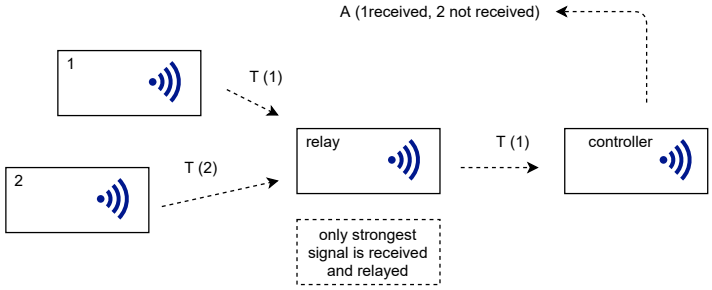


Figure 2-9: The capture effect ensures (only) one of the packages will be received during concurrent transmission.

2-3-3 Wireless control bus

Crystal was further developed to design a protocol to not only receive sensor data at a central node, but also sending actuator data from this node back into the network. This way a control protocol for low power wireless networks has been created using flooding technology which is therefore called Wireless Control Bus [30] (WCB). There are two versions. WCB-P for periodic control, and WCB-E for periodic event-triggered control (PETC). It uses network

flooding through Glossy, the ideas from LWB to use synchronised protocol stages for multi directional communication and resending of data from Crystal to increase network reliability.

2-3-4 WCB-P

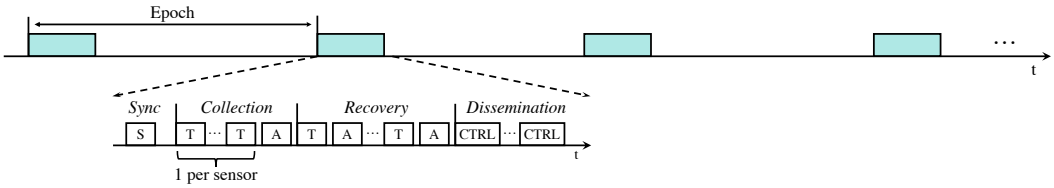


Figure 2-10: The WCB-P protocol from [30].

The WCB-P protocol has multiple stages which repeat each epoch as shown in Figure 2-10.

Sync

During the synchronisation phase the network controller sends a glossy package to mark the start of an epoch. Nodes that are not in sync yet listen for this to connect to the network, nodes that are in sync also listen and can fine tune their synchronisation.

Collection

In the collection phase there are multiple T-slots. One dedicated for each sensor node. The sensor nodes transmit their data in turn to the controller. When all T-slots are transmitted, the network controller sends an acknowledgement (A) with a bitvector indicating which nodes were received.

Recovery

During recovery the protocol works like Crystal with alternating TA-slots until all sensors have been received by the network controller. If no recovery is needed because all T-slots have been correctly received by the controller, this phase will be skipped.

Dissemination

After collection all sensor data, the network controller sends all plant control signals in a single slot (CTRL). From this data each actuator can pick the data intended for them. Because there is no acknowledgement the protocol allows to schedule multiple CTRL-slots to increase reliability of reception of the control signal by all nodes.

2-3-5 WCB-E

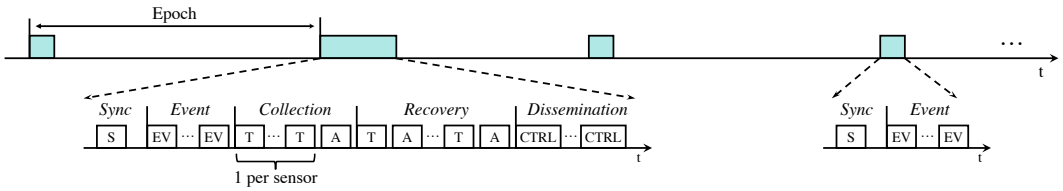


Figure 2-11: The WCB-E protocol from [30].

The WCB-E protocol is very similar to WCB-P and differs only by one extra stage, the event-stage, and the possibility to terminate an epoch early after it, as shown in Figure 2-11. The way WCB-P has been designed all control signals will be recalculated when an event triggers, not only the control signal for the part of the system where the event occured.

Event

In PETC a new control signal is only sent when it is triggered by an event. If no event occurs the previous calculated control signal is held, Section 2-2-1, (at least) until the next period. Therefore after the synchronisation phase the protocol does not go straight to Collection, but enters the Event phase first. The event phase only has an EV-slot in which every node that detects a triggering event can let this know to the network. It does not matter if one or multiple nodes trigger, because of the capture effect all nodes will receive one of the EV-packages with high probability [22]. When a node receives an event it knows the protocol will continue with the collection phase. If it does not receive an event the protocol terminates early and all nodes can sleep and turn off their radios until the next epoch. Note that WCB-E does not use a centralised triggering strategy like [16] which needs all states, but decentralises triggering which only requires local information.

To increase reliability of all nodes receiving an EV-package when there is in event, the EV slot can be repeated, just like the CTRL-slot.

Chapter 3

The Water Testbed Unit

To simulate a water irrigation system a water testbed unit at TU Delft has been revised last year, and connected to wireless nodes. Its layout and specifications are explained in Section 3-1, and its capabilities in Section 3-2 and 3-3. To be able to do wireless control, the software for the sensors and actuators needed to be improved (Section 3-4) and the system needed to be identified (Section 3-5). The model from Section 2-1-1 uses flows instead of gate openings. Therefore the gates have been made smart enough to control the opening themselves, based on a desired flow, which is explained in Section 3-6. Although being revised last year there is still room for improvement, therefore this chapter ends with an overview of the encountered problems and possible solutions in Section 3-7.

To not overcrowd this chapter with images, pictures of the actual testbed unit are shown in Appendix C.

3-1 Testbed overview

To simulate a wireless water irrigation system TU Delft has a testbed. The testbed to simulate water irrigation was built approximately twenty years ago [28], and has been checked and revised last year for [24]. It can simulate a channel of three pools (pool 1, 2, and 3) which get their water from a main stream, which will also be referred to as pool 0. Disturbances can be added by operating manual valves of intakes above the pool to simulate rain or rivers flowing into a pool, and off-takes in the pool to simulate farmers using water or rivers flowing out of this pool. The water from the main stream is also controlled by a manual valve. The testbed unit is a closed system, with a water tank below the pools with two electric pumps: one for the main stream and one for the intakes. Water from the off-takes, and the overflow of pool 3 runs back into this tank. The main stream, and pool 1, 2 and 3 are connected through undershot gates, and pool 3 also has an overshot gate at the end that can operate as an overflow. The gates are operated by Hitec servo motors [3]. The testbed also has Keller pressure sensors [20] at the begin and end of each pool, and one in pool 0 to measure the water level. For wireless control the sensors and servos have last year been connected to Firefly microcontrollers[34].

The sensor before and after a gate have been physically connected to the same Firefly as the servo operating that gate. The benefit of this is that it is possible to control the flow over a single gate using only local knowledge. Basic functionalities have all been tested last year, and technical details about the interconnections have been documented [24]. The layout and connections of the testbed in the lab have been visualised in Figure 3-1.

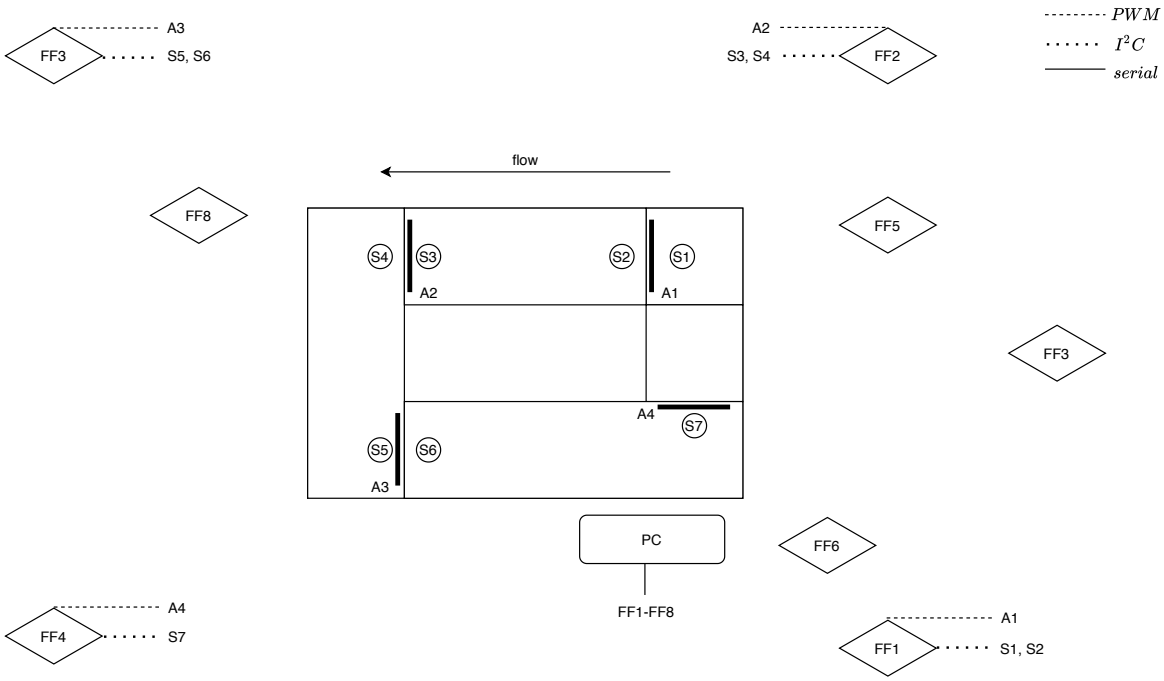


Figure 3-1: Schematic layout of the locations and connections of components of the testbed at TU Delft.

3-2 Sensors

To control the water levels it first is necessary to have knowledge about them. To measure the water levels Keller pressure sensors [20] have been placed under the pools (Figure C-9). Although the sensors can be read digitally, it was decided [24] to use an external AD-converter (ADSS15 [29]) on a custom made PCB called Fireboard, to allow for faster readings as suggested in [28]. The digitised values need to be converted to water levels for each sensor. Because the sensors are below the pool and not all at the same level, they need to be calibrated separately. Another reason for separate calibration is the fact that, although they are all of the same make and model according to their labelings, the sensors that were replaced last year have a higher accuracy. The sensors have linear characteristics so calibration can be done with only two measurements, for which the extrema of 5 cm and 31 cm were used for the most accurate result. The water level y_i in cm for a sensor value ($value_{s_i}$) can be calculated with the linear formula

$$y_i = a_i value_{s_i} + b_i,$$

from which the parameters a_i and b_i as calculated on 21-02-2021 can be found in Table 3-1.

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**
a_i	0.0188	0.0054	0.0181	0.0184	0.0187	0.0054	0.0054
b_i	-188.0213	-65.1942	-181.4417	-182.8977	-184.0834	-59.3130	-59.6097

Table 3-1: Parameters to calculate the water levels from ADS115 values for each sensor at an air pressure of 1012 mBar.

Note that because pressure sensors are used to measure the water levels, the outside air pressure influences the readings. When looking at the air pressures in the Netherlands of the last 50 years they are in the range of 995 to 1050 mBar [4], which can result in an offset of 55 cm when measuring water levels. This on the other hand is only important if we are interested in the actual water levels. If we are only interested in relative differences between pools, which is for example the case when estimating flow over a gate, this will not be a problem because all pools are affected by the same offset.

3-3 Gates

When we have knowledge about the water levels we need actuators to control them, which in this testbed are three undershot gates. The gates of the testbed are controlled by servos. Servos are motors for which the position can be controlled by changing the duty cycle (the ratio between on and off) of a block pulse signal. The pulse cycle is fixed at $20ms$ and the duty cycle has to vary between 0.9 ms and 2.1 ms. The maximum rotation of regular servos is 180 degrees, but the Hitec HS-725BB [3] used in the lab setup has a range of 1260 degrees. A gear is connected directly to the servo which shifts a pole connected to the gate up and down (Figure C-3). Note that the pole of the overshot gate ($gate_4$) is shorter, so its range has to be limited to prevent damaging the servo.

3-4 Driver optimization

Although basic tests were performed before [24], the drivers for the sensors and actuators needed to be adapted to be used on the nodes in a control network.

3-4-1 Sensors

The sensors were not fast enough, so the driver needed to be optimised. The ADS115 drivers to convert the current though the sensors to digital values did work correctly but were slow, because of a hard coded delay to wait for the conversion result, which resulted in a maximum reliable rate of the control network of 1-2Hz. Although this speed might be enough for control, a higher speed is preferred for measurements to be able to filter out the noise. Therefore the drivers were rewritten to poll the ADS115 over I^2C for the conversion ready bit. This results in the driver being able to operate at higher rates, depending on the SPS (samples per second) setting of the ADS115. Note that the driver is still busy waiting which can be approved on, but a modification of the Fireboard's PCB will be needed. When the ALERT/RDY pin of the ADS115 is connected to one of the GPIOs of the Firefly an interrupt can be used to handle the result, instead of a polling mechanism.

3-4-2 Actuators

The servos did not work when controlled over wireless control bus, so this had to be fixed. The servo drivers for the actuators of the gate were not compatible with Contiki setup (Section 5-4) used. The reason for this was found in the hard coded values for the length of the duty cycle of the PWM signal for the servos in clock ticks. Because in our setup the Fireflies run at their maximum clock rate of 32 Mhz, which is double the default, these values had to be doubled. To avoid future problems when changing the speed of the devices, these values are now shifted by the `SYS_CTRL_CONF_SYS_DIV` constant, which has been defined in Contiki especially for this purpose.

3-5 System Identification

To design controllers for a system it is necessary to have a model of it. System identification is the technique to create a model of a dynamical system from measured data. To create a model of the testbed unit each pool needed to be identified. To analyse the time domain data the MATLAB®'s System Identification Toolbox [5] was used, and the three stage process from [21] was followed for each of the pools:

1. Select the class of models.
2. Identify the unknown parameters.
3. Validate the result, to assess the quality of the model.

The scripts and data used for identification can be found via Appendix A-2.

3-5-1 Model selection

For simulation and design of the global controller (Section 2-1-4) the first order model will be used for which statistical identification techniques are not needed because it can be derived from the physical characteristics of the pools (α_i is the area of the pool), and the delays are neglected.

$$P_i(s) = \frac{1}{\alpha_i s}.$$

To justify this choice the local controller for a pool must have a bandwidth below the dominant wave characteristics (ϕ_{wave_i}) and delay (τ_i) of the pool (Section 2-1-3). To find these properties a third order model [10] combined with a time delay has been used for identification. This model is a combination of the first order model of the pool and a second order model to capture the wave dynamics. This results in the transfer function for pool i

$$P_i(s) = e^{-s\tau_i} \frac{\omega_{n,i}^2}{s^2 + 2\zeta_i\omega_{n,i}s + \omega_{n,i}^2} \frac{1}{\alpha_i s},$$

in which τ_i and $\omega_{n,i}$ are the delay and the natural frequency we need for the controller design, and ζ_i is the damping ratio.

3-5-2 Identification

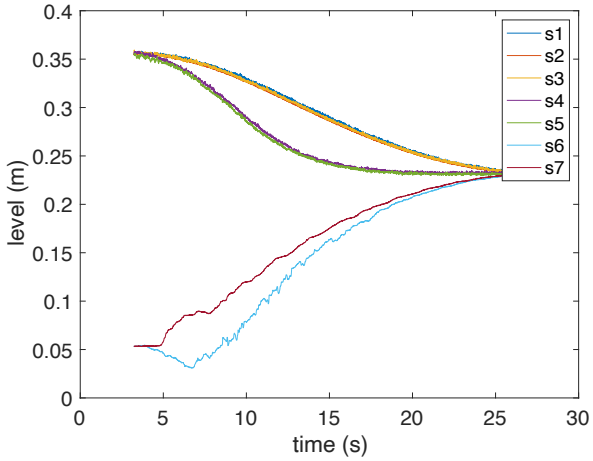


Figure 3-2: Time evaluation of the water levels for an identification experiment where the gate of pool 3 is fully opened.

To identify $pool_i$ only the gate to that pool and the pools after it ($\geq i$) were closed, and the pool itself was emptied to the minimum level. The pools before ($< i$) it were then filled to a level just below the top of the closed gate (i). Identification data has been collected after opening the gate (i) until the water levels were equal. On the testbed unit it is not possible to create a step response which is often used for identification, but this way we created a mix between a step and a (reverse) ramp which also contains all frequencies. To create a general model the experiment was repeated for each pool with three different gate settings for identification: fully open, about half open, and just a little open. An example of such an experiment is shown in Figure 3-2. The delay (τ_i) has been estimated using visual inspection of the delay in time plots, between opening the gate and the response of the pressure sensor at the other side of the pool. Because we have no actual knowledge of the flow over a gate, the flow had to be estimated. To estimate the flow, the water level change in the pools before the identified pool were used, because they are less disturbed by opening the gate than the pool which is identified itself. The area of the pool and the average water level change gives a volume change which can be converted to a flow. Because flow is essentially estimated based on a derivative this flow data is expected to be noisy, but because identification uses statistical methods and this noise is zero mean accurate results are still expected. From the sampling theorem of Nyquist and Shannon [15] we know it is impossible to capture frequencies above half the sampling rate, therefore the speed of the sensor drivers has been increased (Section 3-4). The data for identification has been collected at 128 Hz for more accurate identification of the dynamics.

3-5-3 Validation

To validate the models for each pool, the identification experiment has been repeated for the maximum gate setting. This setting was chosen because it has the most turbulence, and therefore the most wave dynamics.

Not only the fitting percentages of the experiments are very high, also validation is larger that 85 percent so we are confident that we can use the model parameters we obtained with the identification. An overview of the percentages can be found in Table 3-2.

	experiment 1	*experiment 2*	*experiment 3*	*validation*
pool 1	91.89	86.20	84.49	85.54
pool 2	96.66	97.81	91.41	90.67
pool 3	94.89	91.02	97.23	91.32

Table 3-2: Fitting percentage of the obtained model for all three experiments used to identify the pools, and fitting percentage of the the validation data.

3-5-4 Parameters

In Table 3-3 the physical parameters, the measured delays and the parameters from identification for each pool are shown. In a body plot, Figure 3-3, the spikes clearly show the dominant waves which are needed for the controller design.

	$\alpha_i(m^2)$	$\tau_i(s)$	ζ_i	$\omega_{n,i}$	$\phi_{wave_i}(rad/s)$
pool 1	0.1853	1.3	0.0255	0.2469	0.2468
pool 2	0.1187	0.7	1.9×10^{-11}	0.7760	0.7760
pool 3	0.2279	1.5	8.8×10^{-11}	0.1485	0.1485

Table 3-3: Physical and identified model parameters for each pool.

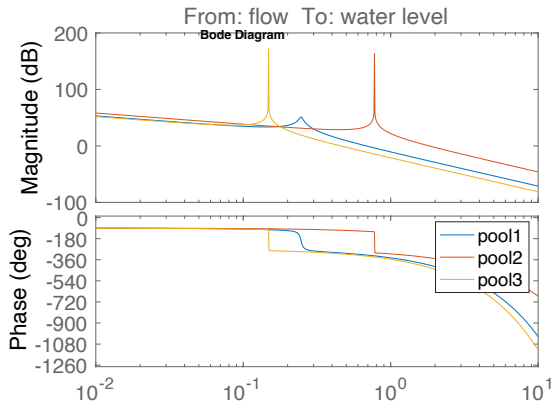


Figure 3-3: Bode plot of the three pools, showing the dominant wave frequencies.

3-6 Smart gates

The controller output will be a desired flow, but we can only actuate the opening of a gate. Therefore the gates have to be made smart enough to convert a flow to gate opening. The gates are controlled by servos, which can be rotated via the drivers with a single byte value

where 0 corresponds to closed and 255 is fully open. The flow over a gate depends on both the gate opening and the difference in water height on both sides of the gate, according to the model in Section 2-1-1. The gate opening is the servo setting, and because the pressure sensors on both sides of the gates are connected to the same Firefly, the water levels are also known. The only information missing from Equation 2-1 is a constant γ_i for the gate. To find this model from the noisy data from identification (Section 3-5-2) we optimise the sum of squared error between the model and the measurements, for which the script and data can be found via Appendix A-2). Unfortunately different gates, and different settings result in different 'constants' as shown in Figure 3-4.

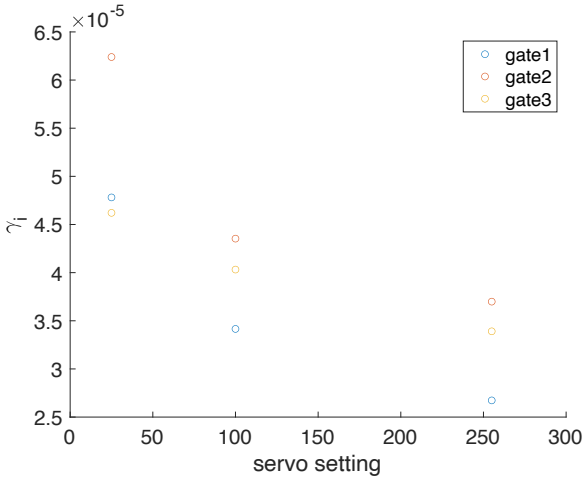


Figure 3-4: Estimated γ_i from measurements for different gate settings.

Because of inaccuracy of the gate settings and leakage, as described in Section 3-7, we do not believe it is possible to fully capture the dynamics of the gates. We will still use the model, but have to be aware of an extra input disturbance to the dynamics of the testbed. In Simulink we have added a model of the gates to the simulation, which shows the system is better in rejecting an overestimation of γ_i than an underestimation. Therefore we choose the largest $\gamma_i = 6.5 \times 10^{-5}$ for all gates.

Because we can only control the flow indirectly not all flows are always possible. For example it is not possible to make water flow if there is no difference in water level. To get an indication of the (maximum) flows in the system, the flows for different gate openings and water level differences are shown in Figure 3-5.

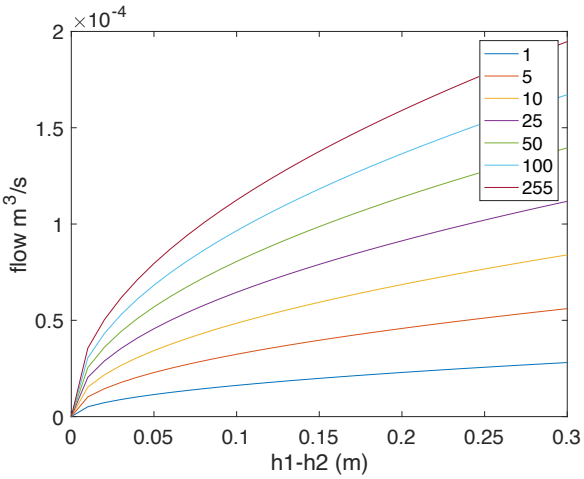


Figure 3-5: Flow over a gate for different gate openings and water levels with $\gamma_i = 6.5 \times 10^{-5}$.

3-7 Problems and improvements

During development on, and experiments with the testbed unit several problems were encountered. These will be summarised here for future development. Also (possible) solutions will be suggested.

Gates

The gates are controlled by a gear attached to the servo's axis (Figure C-3). This gear pulls a pole up and down. This pole is attached to the gate which slides through a slot of plexiglass (Figure C-6). This sliding has a lot of resistance which causes the gate to often move less than intended and being further open or closed than intended, depending if you are opening or closing the gate. This could be improved by redesigning the mechanism but also by adding sensors to get feedback on the actual height of the gate. Because of the resistance the gears also often came loose from the axis. This has been improved by DCSC-lab technician Wim Wien by using longer screws to fixate the gears to the servo. He also planned an upgrade to connect the gears via a tapped servo horn for even better fixation.

An important thing to notice is that although the pools in the testbed are 60 cm in height, this height cannot be used, because the gates are only 35 cm. To avoid spillage over the gates it is best not to use reference levels higher than 35 cm.

The most important problem of the gates is that they cannot close fully, so there is always a disturbance on the flow. It should be investigated if they can be made water tight. I would suggest to replace the rubber parts at the bottom of the gates with a more flexible version, so it is possible to apply some extra pressure to push them tight. A more drastic change from under- to overshot gates could be considered if the setup will be revised. This would make controlling the flow easier because for overshot gates the flow only depends on the head over gate (Section 2-1-1) which can be actively controlled, while for undershot gates the flow not only depends on the controlled gate opening, but also the difference in water level before and

after it. When the difference is large, even a very small gate opening will result in a large flow.

Static electricity

To avoid flooding the university the testbed has been placed in the basement, inside a plastic bin. Because of this, static electricity builds up inside you while walking around the testbed to operate it, depending on the weather and the clothes you wear. The testbed itself, and your computer are grounded, so if you touch one of these you risk a static shock. This is not only unpleasant, but also causes the serial ports of the Fireflies to disconnect. This could be solved by placing and anti-static mat (ESD) inside it.

Serial communication

Originally a Rasberry Pi was used to connect to the serial ports of the Fireflies [24]. This was both good looking and convenient, but unfortunately not reliable. The serial connections were often lost, and could only be revived by rebooting. We were not able to pinpoint exactly what the problem was, but the kernel logs seemed to indicate compatibility issues between the hardware of the Rasberry Pi and usb to UART used in the Fireflies. The Raspberry Pi has therefore been replaced by a regular PC running Ubuntu 18.04.5 and the problem has not occurred anymore.

Pressure sensors

Although being all of the same model according to their labels, the Keller pressure sensors [20] are not exactly the same because they are not from the same production year. The newer sensors installed have a higher accuracy, making it necessary to convert them to doubles, to get them on the same scale with enough accuracy. When all sensors are replaced by new higher precision sensors, this will not be necessary anymore and a faster integer implementation would be possible. The newer sensors also had problems starting up, probably caused by a small power drop of the power unit. This has been solved by DCSC-lab technician Will van Geest by replacing the power unit to one with an increased voltage. This power unit generates 15V which is the maximum supported by the voltage regulators inside the Fireboard. Because pressure sensors were used to measure the water level, the readings are influenced by the outside air pressure (Section 3-2). An improvement to compensate this would be to add an air pressure sensor, or connect to the internet to get weather data. Note that if a fixed offset is acceptable this will not be necessary, because the sensors are linear and only the water level difference is needed to calculate the flows.

Firebox

The Fireflies are connected to the ADS115 with the custom PCB called Firebox. When this PCB is revised it would be good to connect ALERT/RDY from the ADS115 to one of the GPIOs of the Firefly, as already suggested in Section 3-4-1, to make it possible to respond to an interrupt on conversion ready.

In- and off-takes

All in- and off-takes have to be controlled by hand. This can be hard if you need to set multiple valves at the same time. Because of this it is also impossible to do unmanned or remote experiments, which is a wish from the research group, and would also make it easier to do long running experiments. It would therefore be desirable to have electronically controlled valves on the in- and off-takes and electronic switches on the pumps so they can be controlled from the PC.

Pool 0

The level of the main stream (pool 0) is hard to keep stable because the pool is very small, and the inflow has to be operated by hand. This makes longer experiments infeasible. In [28] it was already suggested that a larger pool 0 would make it less sensitive to water demand of pool 1. Another solution could be to add an overflow to pool 0 to keep the water level stable at a fixed level when the inflow is fully opened. Pool 0 now already has an overflow at 60 cm, but this is too high because the water level then will be higher than the top of the gate, causing spillage over the gate. This could be resolved by adding an overflow at a lower level of 35 cm, which would not cause this extra disturbance.

Pump

Because the pumps are inside the water testbed unit, vibrations disturb the measurements of the pressure sensors. For the application in this thesis, this has been solved by adding a digital filter to remove these vibrations from the signal (Section 5-6-2).

Flow sensors

To improve control of the gates, and to acquire richer data from experiments, it would be beneficial to add flow sensors to the gates, and to the intakes and off-takes.

Chapter 4

Control design

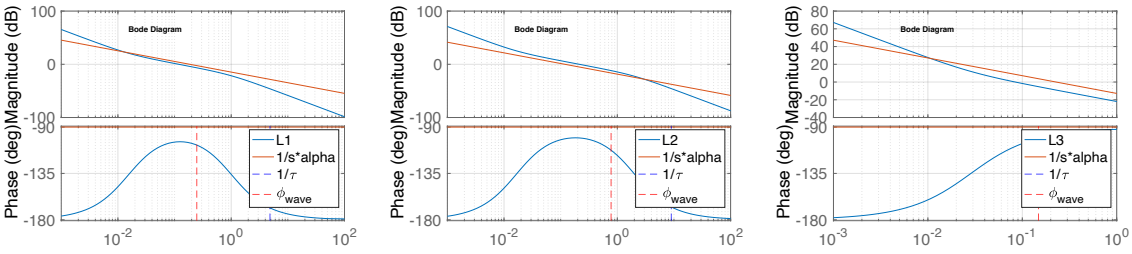
To control the water testbed the first thing we need is a controller. To design a controller for the water testbed unit, [24] and [23] were used, as described in the Preliminaries (Section 2-1). In Section 4-1 the local controllers for the pools are shaped, and in Section 4-2 global control for the system is described, but because this design was not fast enough a redesign is described in Section 4-3. Because these techniques result in a continuous controller, discretisation is explained in Section 4-4. We conclude this chapter with a description of the triggering condition, which will be used for event-triggered control in Section 4-5.

4-1 Local controllers

To use robust control to create the global controller, first we need to design the local compensators, that will be used as the weights as described in Section 2-1-4. As explained in Section 2-1-3 these controllers are of the form

$$W_i(s) = \frac{\kappa_i \left(1 + s\phi_i\right)}{s\left(1 + s\rho_i\right)},$$

and have to be shaped to have large loop-gain at low frequencies, and a bandwidth below both the dominant wave dynamics and time delay of the pool they control. The open-loop bode plots in Figure 4-1 show that our design meets these constraints. When comparing to the unshaped loop gains, they are higher at low frequencies for reference tracking and lower at high frequencies for disturbance rejection. Also the bandwidths of the shaped loop-gains are below $1/\tau_i$ and the dominant wave frequency.



(a) Pool 1 **(b)** Pool 2 **(c)** Pool 3

Figure 4-1: Bode plots to compare the open loop response of the shaped and unshaped system for each pool.

The parameters for the compensators can be found in Table 4-1.

	κ_i	ϕ_i	ρ_i
W_1	0.010	64	1
W_2	0.03	60	0.5
W_3	0.01	35	0.01

Table 4-1: Parameters for the local compensator of each pool, with a bandwidth below the dominant wave characteristics.

4-2 Global controller

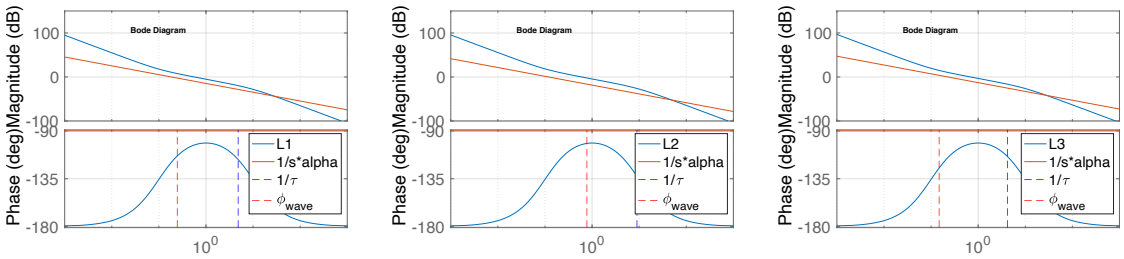
With the local controllers as weights it is possible the design the global controller using robust control techniques. The scripts to perform the optimisation and create the global controller [24] were originally written for five pools, so they had the be adapted for three. For future developments however we chose to make them more flexible to support a variable number of pools (≥ 2). These scripts can be found in Appendix A-2.

Simulation of this system shows the state of the global controller is close to zero. This is caused by the fact that the B-matrix is very small, and 12 orders of magnitude smaller than the D-matrix of the controller. Because the state of the global controller has no significant effect it has been chosen to remove it completely, and reduce it to a P-controller, which will both make implementation easier and execution faster on the cyber physical system.

4-3 Controller with increased bandwidth

Although the resulting system works correctly in simulation, it is very slow. Therefore we need to design a faster controller. Using the controller designed in Section 4-1 and 4-2, rejecting a disturbance like in our intended scenario for experiments (Section 6-1) takes 5 hours in simulation, which makes it not suitable to be used on a testbed with an intake that has to be controlled by hand. Therefore it was decided to increase bandwidth of the local controllers to go above the dominant wave dynamics which will make the systems respond faster. We

justify this by the fact that the main reason for this design constraint in [10], was the type of gates they used. The undershot gates in our testbed however, are less sensitive to height fluctuations than overshot gates, because the flow only depends on the square root of the height difference between two pools, while overshot gates depend on $\gamma^{\frac{3}{2}}$ of the head over gate (Section 2-1-1).



(a) Pool 1 **(b)** Pool 2 **(c)** Pool 3

Figure 4-2: Bode plots to compare the open loop response of the shaped and unshaped system for each pool after increasing the bandwidth.

	κ_i	ϕ_i	ρ_i
W_1	0.3	10	0.1
W_2	0.5	10	0.1
W_3	0.3	10	0.1

Table 4-2: Parameters for the local compensator of each pool, with a bandwidth above the dominant wave characteristics.

4-4 Discretising the controllers

In digital control, performance and stability not only depends on the controller parameters, but also on the sampling period and discretisation method, so these also needed to be chosen. When this project started we ran the control network at 1 Hz because this was the fastest the control network could run reliably, therefore this frequency was used to discretise the controllers. To further reduce network communication it could be investigated to lower this frequency, although this might degrade performance. A Tustin approximation was used to discretise the controllers, because it suffers less from phase lag than ZOH [27].

4-5 Event-triggering

For implementation of ETC, we need a triggering condition. In our setup we do not have access to the full state, therefore we cannot use the state-based triggering condition from Section 2-2-1. Instead we use the output of the plant to trigger. Because each node has the possibility to trigger (Section 2-3-5), triggering does not happen based on central knowledge, so we have to decentralise the triggering condition. Also we want to use a quadratic triggering

condition, so it will be easier to connect to the PSTC algorithm [11] later. The triggering condition for $pool_i$ at time t_k then becomes

$$\|y_i(t_k) - \hat{y}_i(t_k)\|^2 - \sigma^2 \|y_i(t_k)\|^2 > 0,$$

in which $y_j(t_k)$ is the actual water height, $\hat{y_j}(t_k)$ is the value kept from the previous trigger, and σ is the triggering parameter. Note that for the triggering condition we always perform a change of coordinates to put the reference set-point at the origin because ETC and STC literature make this assumption. As suggested in [24] and [9] and also used in [11] we will use a 'mixed triggering' condition by adding a dead band (ϵ) to avoid triggers caused by minor fluctuations, resulting in the triggering condition

$$\|y_i(t_k) - \hat{y}_i(t_k)\|^2 - \sigma^2 \|y_i(t_k)\|^2 > \epsilon.$$

Different values for σ and ϵ will be used in Chapter 6 to compare the resulting performance and communication reduction.

Chapter 5

Wireless control implementation

We start this chapter by describing two extensions to wireless control bus from Section 2-3, which add the capability to do aperiodic communication to WCB (Section 5-1 and 5-2). After that we will give an overview of the implementation of WCB (Section 5-3) and the Contiki operating system it runs on (Section 5-4). The architecture of the control application, how it works, and how it can be configured, can be found in Section 5-5 to 5-9. To try out controller designs for the application faster than real time, a full simulation has been created (Section 5-10). And to do experiments using the application on the real hardware, but without the real testbed, a hardware-in-the-loop simulation has also been added, which is described in Section 5-11.

References to the source code for the application, and scripts for communicating with it can be found in Appendix A.

5-1 WCB-P+

To be able to also do aperiodic communication to reduce radio on time over wireless control bus, we want to extend the protocol. WCB-P is intended for periodic control, and therefore both communication and control over it, have a fixed interval. If we make this sleeping interval variable, it will also be possible to use it for aperiodic control schemes, where the controller decides the sleeping periods. This would make it possible to use self-triggered control techniques, because in STC the controller defines the intervals, by predicting when the next event will occur based on the system state (Section 2-2-2). To distinguish this extension from normal WCB-P we will call it WCB-P+.

In this extension the controller node can send the number of periods the network has to go to sleep (default = 1) along with the CTRL command. By adding this to the control slot (CTRL) the chance of a node missing it can be reduced because the protocol already supports sending CTRL multiple times. The nodes will use the received CTRL to multiply the sleeping period. If a node misses the CTRL signal it will only sleep one period, so it will try to resynchronise each period until the network controller wakes up.

Note that for this to work the controller must have some sort of lookup table, so it can do immediate predictions. If the controller on the other hand needs more time for calculations, a protocol scheme similar to the one suggested in Section 5-2 could be used.

5-2 WCB-E+

Unfortunately this type of STC could not be used on our system, because it does not take disturbances into account. Therefore we have decided to create an aperiodic control scheme, based on a combination of a self-triggered prediction and event-triggered control over wireless control bus. For the prediction, preventive self-triggered control (Section 2-2-3) will be used. PSTC does take disturbances into account and chooses a conservative sleeping time, so the network wakes up at the first possible instant a trigger, caused by the maximum disturbance, could occur. After that a switch to ETC will be made, so the controller waits for the real event before applying control and putting the network to sleep again. Because this hybrid form of control, is basically an extension on ETC, we will call it ETC+. To implement this behaviour we extend WCB-E, which although it supports aperiodic control, does not support aperiodic communication yet. We will refer to it as WCB-E+.

Calculating the conservative sleeping times is computationally intractable on the Fireflies, therefore a PC capable of running the software for the prediction, needs to be connected to the network. The controller therefore cannot put the network to sleep in the same epoch that it updates the control signal, because communicating with this PC takes time. The controller sends a calculation request and the system information (water levels, flows and control signals) after an epoch, when the radio is off (`epoch_end`), and receives the result before the next epoch. To distribute the sleeping time over the network the event phase (EV) was given a payload with the number of additional sleeping periods. This way the controller can send the conservative sleeping periods at the beginning of an epoch so the network can sleep early. After this extra sleeping WCB acts like normal WCB-E where nodes will trigger an event (payload = 0). Because the controller can also send a zero payload it can force the network to wake up, which is used at startup during initialisation of the observer in PSTC. Because the estimates of PSTC are conservative it should not be possible to have conflicts between the controller and sensors triggering, as shown in Table 5-1. Note that a conservative estimation of 'one' is trivial, and will never be sent.

	Controller	Sensor	
Initial epochs	forces event	no event	Initial epoch runs and actuators are updated.
	forces event	event	Initial epoch runs and actuators are updated.
Normal epochs	waits for event	no event	No events, so sleep early and wait for next epoch. Actuators not updated.
	waits for event	event	Event detected, so run full epoch. Actuators updated.
Extra sleeping	asks for sleep	no event	Sleep early, wait multiple epochs. Actuators not updated.
	asks for sleep	event	Will lead to unexpected behaviour, but should not be possible!

Table 5-1: Overview of the possible intentions of the controller and sensor nodes during the EV-slot of each phase in WCB-E+, to verify there cannot be any conflicts.

5-3 WCB implementation

To build a control application on top of WCB it is necessary to know how the protocol has been implemented in [30]. For future reference, and to explain the extensions made to the protocol, a schematic overview will be presented in this section. The WCB protocol provides so called callback or hook functions. The moments when, and order in which they are called are shown in the lifecycle of the protocol in Figure 5-1. Note that because WCB was built as an extension of Crystal this is sometimes still reflected in the function names.

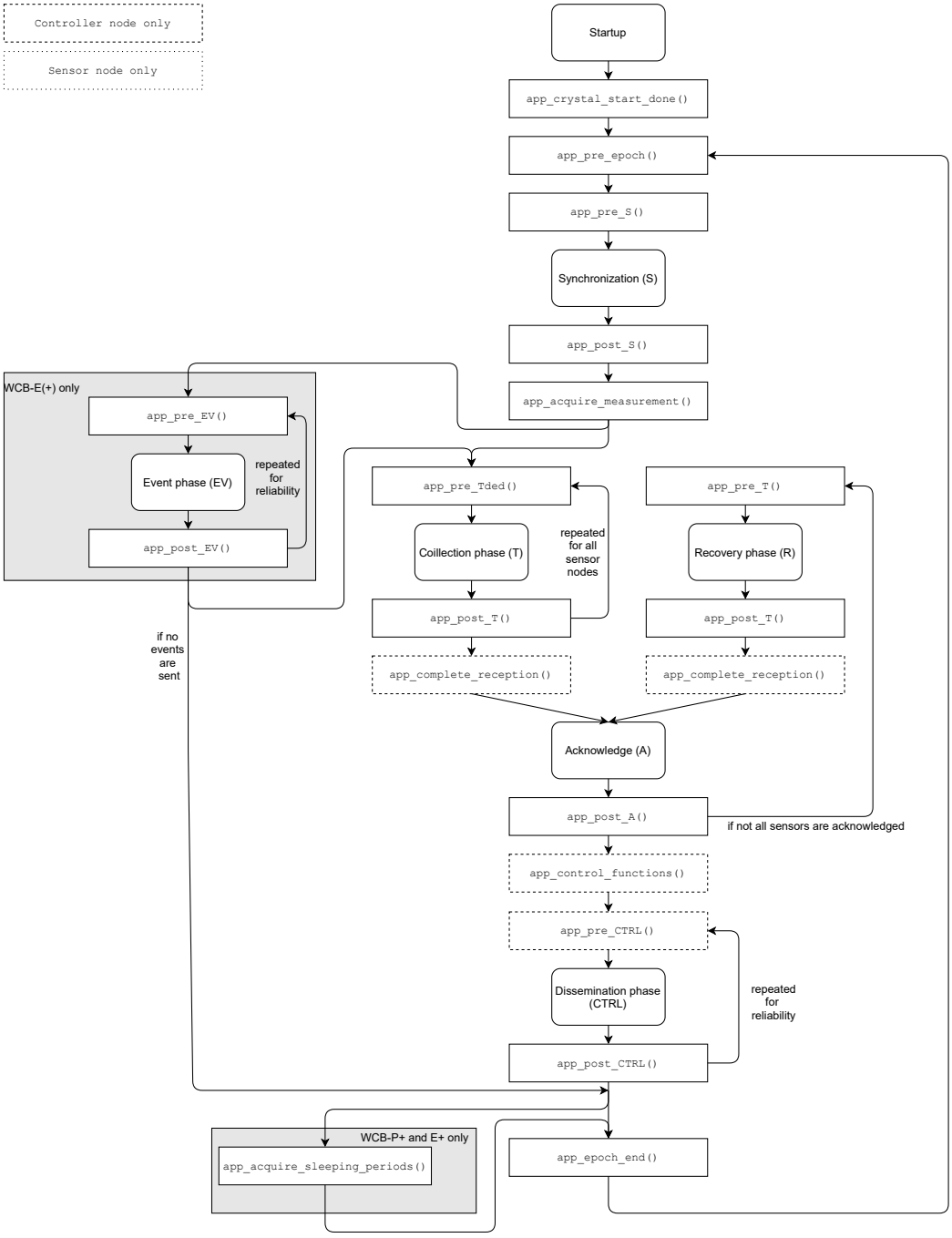


Figure 5-1: Lifecycle and hook functions of the WCB implementations. For clarity all function parameters have been omitted. The full signatures can be found inside the header file of the protocol[1]. Functions specific to the implementation on the testbed in Trento have also been omitted, but can be found in Appendix B.

[1] `/net/crystal/crystal.h`

`app_crystal_start_done()`	Initialisation of the system after starting (controller) or synchronising (other nodes) WCB.
`app_pre_epoch()`	Initialisation of the epoch.
`app_pre_S`	Called before sync (unused).
`app_post_S()`	Called after Sync. Used for analysis of synchronisation
`app_acquire_measurement()`	Get sensor readings.
`app_pre_EV()`	Prepare event-triggering payload (sensor nodes; controller node only in WCB-E+).
`app_post_EV()`	Process event-triggering payload to check if the network sleeps early (or set extra sleeping periods, WCB-E+ only).
`app_pre_Tded()`	Prepare sensor data payload (only used on sensor nodes).
`app_post_T()`	Update reception bitmap (only used on controller).
`app_complete_reception()`	Process sensor data (only used on controller).
`app_pre_T()`	Prepare sensor data payload (only used on sensor nodes).
`app_post_A()`	Check if controller received the data (only used on sensor nodes).
`app_control_functions()`	Calculate control signal.
`app_pre_CTRL()`	Prepare control data payload (including sleeping periods, WCB-P+ only).
`app_post_CTRL()`	Process control data (only used on sensor nodes).
`app_acquire_sleeping_periods()`	Return the number of sleeping periods.
`app_epoch_end()`	Called after the epoch when the network sleeps Sync. Used for analysis of the epoch.

5-4 Contiki

To run the application, a low power embedded OS capable of doing multitasking with a replaceable network stack is needed. The Contiki Operation System (Contiki) [2] meets these requirements. This older version of Contiki was used instead of the latest version Contiki-NG, because the newer version is not compatible with the WCB implementation. Contiki is an operating system, designed specific for resource-constrained, low memory and low-power microcontrollers and is open source. It does provide a wireless communication stack but this stack can be replaced by a custom stack, which has been done in this configuration with WCB. Contiki is event based, which means that processes are activated by events from the hardware, like serial messages, or software, like timers. It uses non-preemptive cooperative scheduling for processes [1]. So processes normally have to wait until other processes yield control back to the operating system, before they can run. Interrupts however can (and will) interrupt execution of a process. To support real-time tasks it is possible to run a process in interrupt context. When the rtimer library is used pre-emption of normal execution is possible to make sure the real-time task meets its timing constraints.

5-5 Application architecture

To understand how the control application was built on top of the WCB framework (Section 5-3), the general architecture will be explained here, and a graphical overview of the components and connections is shown in Figure 5-2.

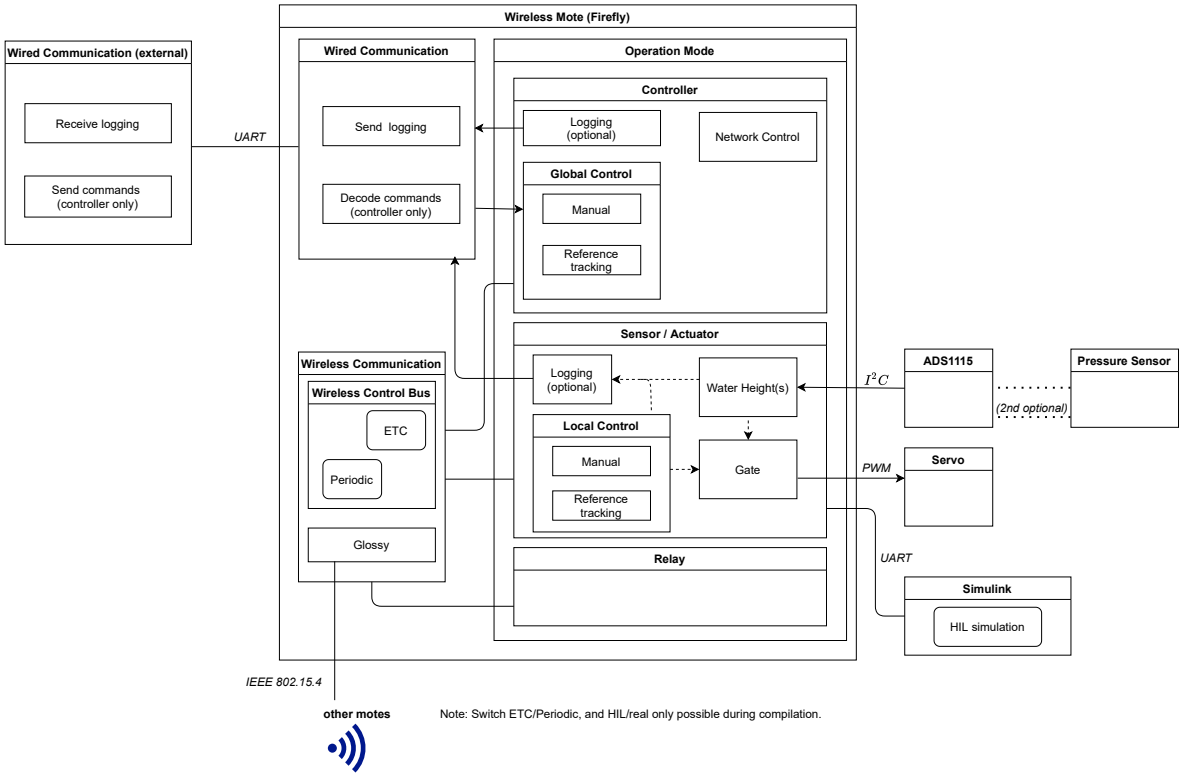


Figure 5-2: Architecture of the control application, showing all components and both internal and external connections.

Although there are three types of nodes, there is only one application (firmware). In this application the IEEE MAC-addresses and the roles of the Firefly nodes have been hardcoded. On startup the firmware selects the operation mode based on this configuration. The application can be compiled to use WCB-P(+) or WCB-E(+) with different levels of logging. It is also possible to compile the firmware for a hardware-in-the-loop simulation, instead of the real water testbed. Switching between manual control, in which the plant can be controlled over the serial connection via the controller node, and reference tracking can be done on runtime. To do this commands have to be sent over the serial connection to the controller node, which distributes the mode switch via WCB. The outer loop of the application used for global control and the epoch of WCB is hard real-time, to ensure synchronisation of the communication. The inner loop, which is only used on the sensor/actuators for local control and sensor measurements, therefore is soft real-time.

5-6 Node types

Each node has its role in the network. These roles will be explained in detail in this section. The role for each node is configured in the firmware and selected based on the IEEE MAC-address during boot. Relay mode is default and selected when the address is not configured in the firmware. This makes it easy to add extra relay nodes to extend the network. The roles for the Fireflies connected to the testbed can be found in Table 5-2.

Name	Id	IEEE MAC-address	Role
FF1	201	0x00, 0x12, 0x4B, 0x00, 0x19, 0x4A, 0x51, 0x04	Sensor/Actuator
FF2	202	0x00, 0x12, 0x4B, 0x00, 0x19, 0x4A, 0x51, 0x6c	Sensor/Actuator
FF3	203	0x00, 0x12, 0x4B, 0x00, 0x19, 0x32, 0xe6, 0x91	Sensor/Actuator
FF4	204	0x00, 0x12, 0x4B, 0x00, 0x19, 0x4A, 0x52, 0x05	Sensor/Actuator
FF5	205	0x00, 0x12, 0x4B, 0x00, 0x19, 0x4A, 0x51, 0xa8	Controller
FF6	206	0x00, 0x12, 0x4B, 0x00, 0x19, 0x4A, 0x51, 0xd7	Relay
FF7	207	0x00, 0x12, 0x4B, 0x00, 0x19, 0x4A, 0x51, 0xcd	Relay
FF8	208	0x00, 0x12, 0x4B, 0x00, 0x19, 0x4A, 0x51, 0xb5	Relay

Table 5-2: Node configuration for each Firefly of the network at the water testbed.

5-6-1 Controller node

The controller node receives commands over serial port, does global control and synchronises the WCB network. So it is both a controller of the network protocol and a controller for the water irrigation system. When not clear from context which role is referred to, the controller will be called global controller when referring to plant control, and network controller for WCB synchronisation. When it starts the default mode is 'manual control'. In manual control both global and local controllers are disabled. The network controller just sends the desired actuator values to the sensor/actuator nodes. The desired value can be set by sending a (human readable) string with the id of the sensor/actuator and the servo value over the serial port to the network controller. The command '203 255' for example will fully open the gate connected to FF3. In automatic control mode for reference tracking the global and local controllers will try to reach a target level for pool 1, 2 and 3. These target levels are currently hardcoded in the firmware at 0.25, 0.20 and 0.15m. To switch between modes the commands '205 0' and '205 1' put the application in manual or automatic control mode. The scripts to calculate the parameters for the global controller, and write the C code for it, to include in the application can be found in Appendix A-2.

5-6-2 Sensor/actuator node

The sensor/actuator nodes read the sensors, handle global control and do local control to actuate the gates. Because the electric water pump is inside the water tank connected to the system, the pressure sensors are disturbed by it (Figure D-1a). When we perform a Fast Fourier transform in Matlab on the sensor data (Figure D-1) it shows frequency spikes at 22 and 50 Hz when the pump is operating which can be filtered out (Figure D-1b). The controller network operates at 1 Hz so it cannot respond to faster changes, therefore it will not affect the

controller's performance to add a lowpass filter at 1 Hz to the sensor signal. For a first order Butterworth filter (Equation 5-1) and a sampling rate of 128 Hz this results in a = [1.0000, -0.9521], b = [0.0240, 0.0240].

$$Y(z) = \frac{b(1) + b(2)z^{-1}}{1 + a(2)z^{-1}} X(z) \tag{5-1}$$

Because filtering happens in the fast inner loop of the application, the implementation must be as fast as possible. The Fireflies do not have floating point hardware [34] therefore a fixed point integer implementation has been chosen, in which only basic operations are used. With just minimal error the filter values can be relaxed to a = [1, $(2^{-5} - 2^{-7})$] and b = [$(1 - 2^{-5} - 2^{-6})$, $(1 - 2^{-5} - 2^{-6})$] which makes it possible to replace the floating point multiplications with basic additions and bit shifts. The pressure values are of type `uint16`, so on the 32-bit architecture of the Firefly 16-bit extra precision can be added for calculations and storing intermediate results without loss of performance.

When operating in automatic control mode the sensor/actuator nodes also perform local control which runs in sync with the epoch. Therefore the controllers (Section 4-1) do not need to be optimised like the sensor filter, and can be implemented using variables of type `double`. This not only makes implementation easier, but also development of the controller, because values from Matlab can directly be used. The sensor/actuator node also converts the desired flow to a servo setting for the gate as described in Section 3-6. This also happens in sync with the epoch to prevent flooding the servos with control signals. The scripts to calculate the parameters for the local controllers, and write the C code to include in the application can be found in Appendix A-2.

5-6-3 Relay node

The relay nodes do not implement any functionality, besides connecting to the WCB network. While connected they will automatically relay communication and can be used to increase reliability or extend the range of the network.

5-7 Scheduling

The Fireflies have to perform multiple tasks which are time critical. Therefore it is important that their task schedule can meet the constraints. The most important task on the Fireflies is communication, because the protocol has to stay in sync. To make sure this process meets its timing constraints, it has been implemented as a hard real-time process. Measuring the sensors, which only happens on the sensor/actuator nodes, is the most time consuming task, but has been implemented soft real-time, because a small delay would not hurt the overall performance, because the system is not able to respond to changes faster than 1 second, while measurements are being taken at 128 samples per second. A full time-analysis has not been done, but when increasing the number of measurements no deadlines where missed, until we got over 200 samples per second, so 128 which will be used in the application, is a safe choice.

5-8 Serial communication

The only way the Fireflies can communicate, besides WCB, is over a serial connection. Therefore this connection is used for debugging, remote controlling and remote communication. The different communication modes will be explained in this section. The Fireflies can communicate over UART which has been configured at the maximum speed of 460800 baud to minimise communication delays. The software can be compiled for two types of communication: data logging to perform experiments on the testbed, and development logging to view information about operation of the app for development. Both can be configured via the logging header[2].

5-8-1 Data logging

When data logging is enabled, development logging will be disabled automatic to prevent interference.

Different modes:

Mode	Description
NONE	No logging.
BEP	Log sensor and actuator values every epoch as csv on controller (only for use in dashboard of bachelor end project).
RAW	Log unfiltered sensor values on sensor node at full speed of the inner loop.
NORMAL	Log sensor data as used by the application in mm (controller and sensor node).
HIL	Send actuator data and receive sensor data in binary format for HIL simulation.

Table 5-3: Overview of the data logging modes over the serial connection for the control application.

5-8-2 Development logging

When data logging is disabled, development logging can be enabled by setting a log level. This log level can be any combination of:

Level	Description
NONE	No logging.
INFORMATION	General information about the application (eg. initialisation and flow).
DEBUG	Used to output debug messages. These should be removed after debugging or set to a more appropriate logging level.
COMMUNICATION	Information about communication over the WCB protocol.
ERROR	Information about errors.
ALL	Enables all logging levels.

Table 5-4: Overview of the development logging levels over the serial connection for the control application.

[2] `/app/wcb-lab/logging.h`

5-9 Visual feedback

As explained in Section 5-8 the only way to communicate with the Fireflies, is over the serial connection, but data communication conflicts with development logging. Therefore some visual feedback has been added using the RGB LEDs on the Fireflies to help debugging. Before an epoch start the LED switches off, therefore the pulse of the blinking LED shows the epoch and control period. The colour of the LEDs on the Fireflies give additional feedback, as shown in Table 5-5.

Node type	**Colour**	**Description**
Controller	GREEN	Epoch start.
Controller	RED	No event received, early sleep (WCB-E only).
Relay	YELLOW	In sync with controller.
Relay	BLUE	Missed synchronisation.
Relay	BLUE	Extra sleeping (WCB-E+ only, not blinking)
Relay	OFF	Early sleep (WCB-E only).
Sensor/Actuator	RED	In sync with controller, servo at 0 (closed).
Sensor/Actuator	YELLOW	In sync with controller, servo at 1-254.
Sensor/Actuator	GREEN	In sync with controller, servo at 255 (fully opened).
Sensor/Actuator	BLUE	Missed synchronisation.
Sensor/Actuator	BLUE	Extra sleeping (WCB-E+ only, not blinking)
Sensor/Actuator	OFF	Early sleep (WCB-E only).
Sensor/Actuator	MAGENTA	Simulation measurement corrupted (HIL only).

Table 5-5: Visual feedback on the Fireflies.

5-10 Simulation

To prepare experiments and test controller designs, it is convenient to have a full simulation, similar to the real testbed setup. The combined simulation [24], Section 2-1-5) was used as a starting point and reference, but to have a simulation more close to the real water testbed and controller implementation, a new simulation was created in Simulink (Appendix E-1). This simulation still uses the combined global controller because this will also be implemented on the hardware. The pools however were split into the local compensator (W_i) and a first order model of the plant (P_i), to mimic the the local compensator implementation on the hardware and the response of the pools of the testbed. The delay (τ_i) in the pools is simulated using a digital delay, and the inflow (u_i) a pool is negated from the previous pool P_{i-1} without delay. The code for the simulation can be found in the repository provided in Appendix A-2.

5-11 Hardware-in-the-loop

Controlling the testbed did not work as desired, as shown in Section 6-2 due to unmodelled disturbances. To be able to perform the experiments for this thesis on the real hardware, without the uncertainties of the real plant, a hardware-in-the-loop (HIL) simulation was created. The purpose of HIL is to create a simulation in such a way that the hardware cannot

distinguish between controlling the real or the simulated system [8]. Therefore adaptations to the application should be as minimal as possible. In the control application, the only difference between normal operation and HIL operation is the communication with the sensors and actuators, which has been replaced by serial communication. Note that logging over the serial port has to be disabled not to interfere. The simulation of the system has been implemented in Simulink, based on the full simulation to test the controller (Section 5-10), but adapted to be able to connect it to the Fireflies. The Simulink model can be found in Appendix E-2, and the code in the repository described in Appendix A-2. The flows over the gates are normally simulated, based on the water levels and servo setting (Section 5-11-2), but it is possible to bypass this, and use the desired flows calculated by the sensor/actuator nodes directly, which of course is less realistic, but more close to the full simulation which is helpful for comparison.

5-11-1 Communication

Simulink needs to send two simulated sensor values to each sensor node, which are 16 bit unsigned integers. The nodes only need to send an unsigned byte for the servo command to the simulation. Because the datatypes in serial communication with Simulink needs to be symmetric, 24 bytes can be used to send more information to Simulink, without increasing communication demand. These are used to send the radio on time in milliseconds as an unsigned byte which is typically in the range of 20-120ms, and the desired flow as a 16 bit unsigned integer. Because the flow is implemented as a `double` it has to be scaled from -0.01/0.05 to 0/65535. All communication happens in big-endian order because of the Firefly's architecture [34], as shown in Table 5-6.

	header	0	1	2	3
To Simulink	n/a	flow (msb)	flow (lsb)	radio	servo
From Simulink	'#'	sensor1 (msb)	sensor1 (lsb)	sensor2 (msb)	sensor2 (lsb)

Table 5-6: HIL serial communication byte order.

5-11-2 Gates

The flow over the gates has been simulated using the simulated water level on both sides, the servo setting and the model from Section 2-1-1. Also to make the simulation more realistic opening and closing the gates could be rate limited, using timings of the real gates of the water testbed, but we have removed this feature to reduce the disturbances in our experiments.

Chapter 6

Experiments

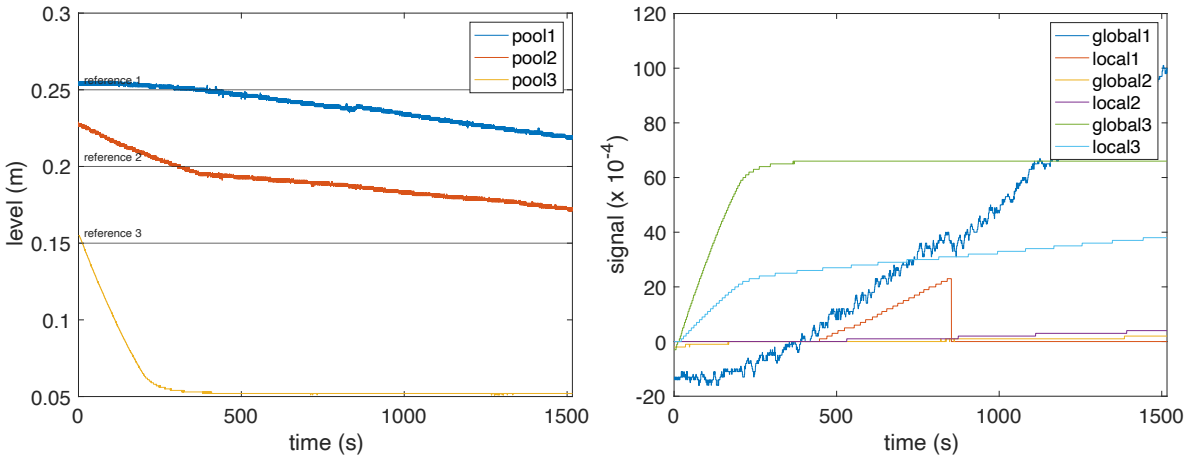
To test the control application and compare periodic and event-triggered control, a scenario (Section 6-1) for rejecting disturbance was run with different settings. First the scenario was tested on the real testbed (Section 6-2) but because this did not work as desired, a switch to hardware-in-the loop (Section 6-3) was made which showed ETC can greatly reduce communication without much performance loss. As a last contribution a setup was designed to do experiments with ETC+ (Section 6-4), which shows this hybrid approach can reduce communication times even further.

6-1 Scenario

For comparison of the results it is necessary to have a scenario for the experiments. In a water irrigation system, water levels are typically around the reference set point, until a farmer disturbs the system by using water to irrigate his crops. The scenario chosen for the experiments on the water testbed unit, is therefore to start with all pools at their reference levels of 0.25, 0.20 and 0.15 m. After that a step disturbance will be created in pool 3 by opening an out-take for 30 minutes (1800 seconds). The input disturbance created by the flow should then be rejected by the controller. Afterwards we can analyse performance of the system based on the reference error, and the performance of the wireless control network based on the radio on time.

6-2 The water testbed unit

One of the goals of this thesis was to implement a wireless control network on the testbed unit at TU Delft following up on the work of [24]. Therefore the first experiments were done using a controller, designed as described in Chapter 4.



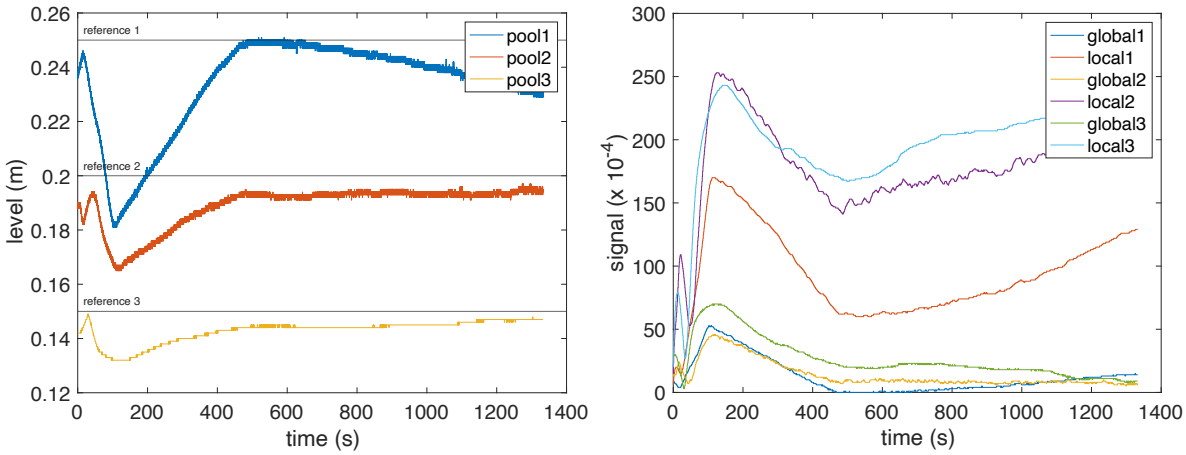
(a) Water levels.

(b) Control signals.

Figure 6-1: Time evolution of water levels and control signals after a step disturbance on the testbed using periodic wireless control with controller from Section 4-1 and 4-2. The controller fails to stabilise the references and the water levels slowly drift away. Notice there is a drop in the control signal because of a reset of the local compensator which cannot create the desired flow.

Unfortunately the first controller, described in Section 4-1 and 4-2, was not able to reject the disturbance, as demonstrated in Figure 6-1. To further investigate this problem a full simulation was run which showed that even without any restrictions (eg. water can flow up, and water levels can be negative) it took the controller 5 hours to reject the disturbance (Figure D-3). From this, the assumption was taken that the controller was too slow, and bandwidth needed to be increased.

This faster controller, designed in Section 4-3, did a much better job in simulation (Figure D-5) where it rejected the disturbance in 30 minutes. On the real testbed however, it was not able to do this, due to disturbances added by imperfections of the model especially caused by problems with the gates (Section 3-7) which cannot be positioned very accurate and always have some leakage. From the plots we can see that pool 2 and 3 are going in the right direction, but pool 1 is not. It might be possible that a longer experiment will also correct pool 1, but longer experiments are not feasible at the moment, because the water level in pool 0 representing the main stream has to be controlled by hand to keep it above 25 cm and below 35 cm to avoid overflow (Section 3-7).



(a) Water levels. **(b)** Control signals.

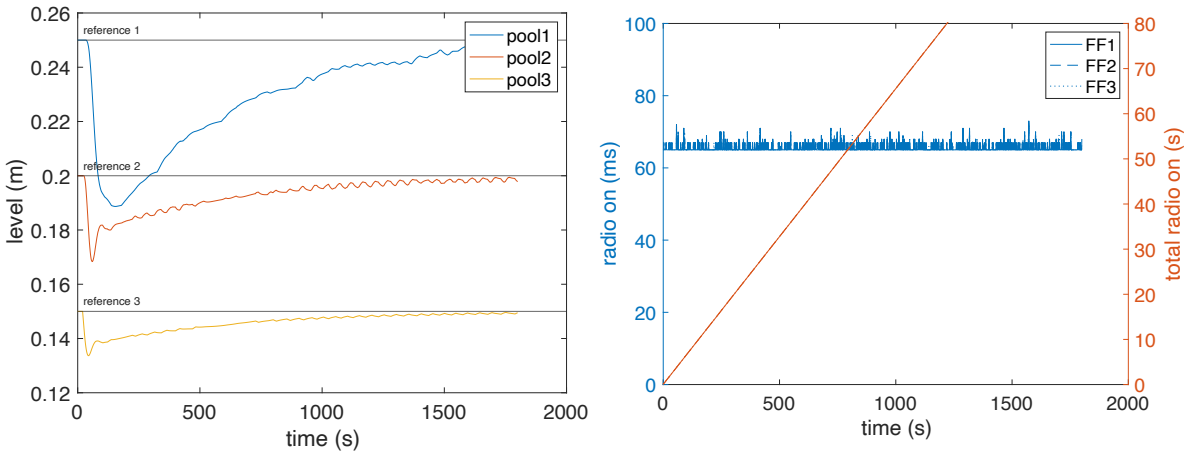
Figure 6-2: Time evolution of water levels and control signals after a step disturbance on the testbed using periodic wireless control with controller from Section 4-3. This controller is beginning to stabilise the references for pool 2 and 3, but pool 3 is still drifting away.

6-3 Hardware-in-the-loop

Because experiments longer than half an hour are infeasible, it was decided to switch to a hardware-in-the-loop simulation Section 5-11). This would give the benefit of a plant with a model closer to the model used for controller design, while still being able to use the real hardware for the control network. Because the real hardware will be used, a realistic comparison between radio on times for periodic and event-triggered control can be done. Because the controller designed in Section 4-3 performed best in rejecting the disturbance on both the real testbed and the full simulation, this controller will also be used for HIL simulation.

6-3-1 Periodic control

To be able to compare event-triggered control we need to have a baseline. Because periodic control is a proven technique, and expected to have the longest radio on time we choose this to be our baseline. The periodic controller runs over the WCB-P network, both running at 1 Hz. The result is shown in Figure 6-3. Although the response looks similar to the full simulation with flow restrictions (Figure D-6) it is less smooth because in the full simulation desired flows are used to control the system, while in the HIL simulation the gates are simulated based on the servo actuation commands. The flows therefore cannot be realised as precise as in the full simulation, because the servos have limited precision causing this less smooth behaviour.



(a) Water levels.

(b) Radio on.

Figure 6-3: Time evolution of water levels and control signals after a step disturbance in a HIL simulation using periodic wireless control with controller from Section 4-3. The HIL simulation is less smooth than the full simulation (Figure D-6) because the gates are also simulated.

The mean squared error for the water level in meters is 865 mm^2. The radio turns on every second during the 30 minutes experiment, resulting in a total on time of 118 s. In this WCB-P configuration the radio is on for less than 7% of the time which is a great reduction compared to always on, but now we want to know how much can be improved upon this, without too much performance loss using ETC.

6-3-2 ETC

To investigate communication reduction and performance we redo the same experiment multiple times over WCB-E with different triggering settings. In Table 6-1 the results for periodic control are shown, next to the ETC boundary cases, to never or always trigger an event, for comparison. The metrics used for comparison of the performance are the mean squared error used to compare overall performance, and the mean time squared error, which punishes errors at the end more to compare the steady-state error. They are defined as:

$$mse = \frac{\int_0^T e^2\,dt}{T} \quad \text{and} \quad mtse = \frac{\int_0^T e^2 t\,dt}{T},$$

where e is the error in mm, t the time in seconds and T the total time of the experiment.

description	**radio on (s)**	**trigger ratio**	**mse (mm^2)**	**mtse (mm^2s)**
Periodic control	118.0	1801	865	872
ETC (never trigger)	27.4	0	n/a	n/a
ETC (always trigger)	128.9	1801	867	874

Table 6-1: Overview of the results for the experiment from Section 6-1 on WCB-P and boundary cases for WCB-E.

Note that ETC always triggering, needs more communication than periodic control because there is some overhead in the protocol as described in Section 2-3-5. The minimal communica-

tion time is 27.4 s when there are no triggers at all. Looking at the errors for this experiment are not useful, because there is no control at all. The results for the regular experiments to show the influence of varying σ and ϵ are shown in Table 6-2. The time evaluations of the experiments can be found in Appendix D-3.

σ	ϵ (mm^2)	radio on (s)	triggers	mse (mm^2)	mtse (mm^2s)
0	0	103.8	1347	866	872
0.1	1	36.2	157	844	848
0.05	1	37.6	181	865	873
0.025	1	37.2	177	866	870
0.05	2	37.3	175	866	881
0.025	2	36.6	164	828	833
0.1	2	35.2	140	859	863
0.2	2	34.1	123	833	844
0.4	2	35.4	142	877	887
0.4	4	34.2	121	550	558
0.4	8	32.6	93	928	944
0.4	16	34.4	127	852	888
0.2	16	34.6	128	896	965

Table 6-2: Overview of the results for the experiment from Section 6-1 on WCB-E with different parameters.

Because the experiments take 30 minutes to run it was not feasible to repeat them multiple times with slightly changed starting conditions to have statistically solid results. Therefore we must be cautious taking conclusions from them.

We can however conclude that if σ and ϵ are kept small communication time can be reduced to about 35 seconds, which only a third compared to periodic control over WCB-P, with almost the same error, although the time evaluations (Section D-3) show more oscillations as σ and ϵ increase. It is good to notice that the mse and mtse do not differ very much for the experiments with small values, which means that they do reject the disturbance without much steady-state error.

When σ and ϵ get larger ($\sigma \geq 0.4, \epsilon \geq 8$) the errors increase and also the oscillations get so large that we might destabilise the system, while not even gaining that much communication reduction. That can be explained by the fact that we are already very close to the theoretical minimum of 27.4 seconds when there are no events at all.

A noticeable result is the case with $\sigma = 0$ and $\epsilon = 0$. At first glance one might expect this to be the same as the always trigger case, because with our triggering condition (Section 4-5) a trigger can only be absent when the difference between the current water level y and the kept value for it $\hat{y}$ is zero. The precision of sensors in the HIL simulation however is the same as on the real testbed, which makes it impossible to detect changes in pool 1, 2, and 3 smaller than 0.02 cm, 0.02 cm and 0.005 cm.

Another remarkable phenomenon is that one case has a significant smaller error than periodic control, with $\sigma = 0.4$ and $\epsilon = 4$. From the plot of this experiment (Figure D-19) this looks like a lucky coincidence, were events are timed such that pool 2 asks so much water that the

integrator of pool 1 has time to wind up, which then gives it a boost to correct faster than normal, when pool 2 does not ask for water.

6-4 ETC+

ETC+ did not work as desired on the HIL simulation, created for the experiments with periodic control and ETC. Therefore a new simulation was created, and the control application was adapted for it. Despite problems with ETC+ caused by communication problems of the control network, the experiments that succeeded show ETC+ has potential to reduce communication time compared to ETC.

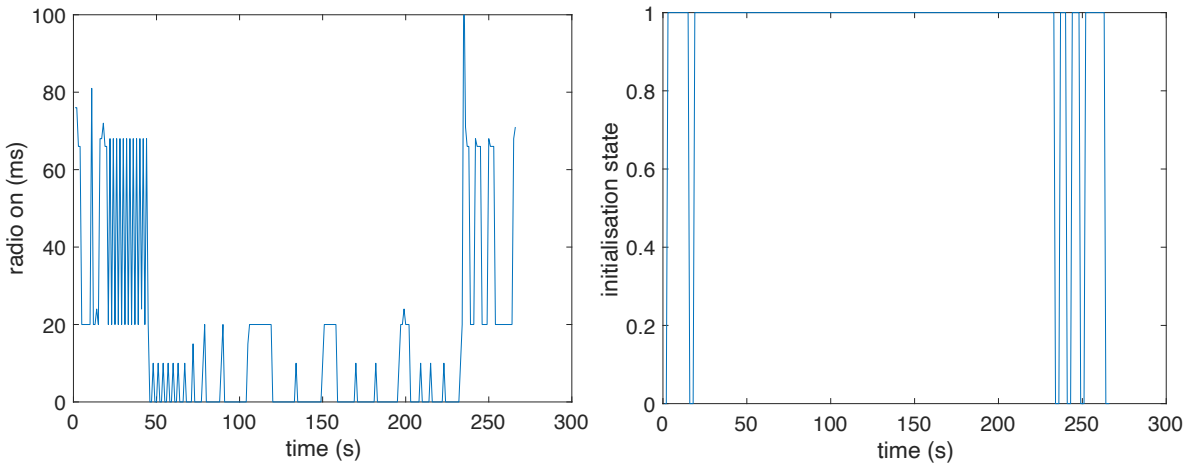
6-4-1 Setup changes

Mismatches between the state of the control system and the prediction lead to non trivial predictions of a single sleeping period, therefore the setup was re-created to keep them as close as possible. Although the PSTC algorithm [11] we are using to calculate conservative sleeping times, takes bounded disturbances into account, we need to know these bounds for al states. Small extra disturbances caused by delays or modelling differences, resulted in too conservative predictions of 'one' sleeping period. Therefore a new HIL simulation was created in Python, Appendix A-3 which is computationally less intensive than the Simulink version, so it is less likely to cause delays on our PC. Also the model used to simulate the pools has been changed to use the Padé approximation, so we could use the same model in the simulation as in the PSTC algorithm. To make sure the simulation runs in sync with WCB, a version of the control application was created which communicates directly with the simulation from the different WCB phases for synchronisation. Because the PSTC algorithm assumes a heartbeat (a maximum number of periods between two events) to do its offline calculations, this has been added to the control application. PSTC assumes the control action is not updated in between events, so this has also been changed in the control application. For more reliability of receiving events, the number of transmissions (NTX) for the EV-slot has been increased from 2 to 3. Note that this new version of the software is only intended to be used for these experiments, and not for future development, because it can not be connected to the real hardware anymore.

6-4-2 Initialisation problem

Unfortunately after the changes made in Section 6-4-1 there were still problems, with the initialisation state of the PSTC algorithm. Although we tried to make disturbances for this setup as predictable as possible, we still encountered problems. Because the PSTC algorithm is observer based it needs to initialise, for which the network controller enforces two events. After that it should only re-initialise when the disturbance changes. This happens, and works correctly in the ETC+ setup. Unfortunately when there are problems with communication in the WCB network the states of the control network and the PSTC algorithm get out of sync, causing it the have to re-initialise at every event. Our assumption is that the problem is a mismatch between the controller states, but we have not been able to verify this. A problems always start when there is a spike in the radio on time, as shown in Figure 6-4 at 234 seconds,

but not every spike causes a problem. The loggings show all Fireflies have their radio on for over hundred milliseconds, except FF2 which is only on 21 ms. This indicates that FF2 did not enter the collection phase, and the network controller tries to recover because it does not get the T-slots from it. This could mean that it did not receive both EV-slots, although this seems unlikely given the high reliability of Glossy (Section 2-3-1) and the frequency this happens. It could also mean that there is a bug in the control application, but we have not been able to find it yet.



(a) Radio **(b)** Initialisation state

Figure 6-4: Time evolution of a failed experiment for ETC+. When the experiment starts initialisation is needed, and again when the disturbance changes after 20 seconds, after that no re-initialisation should be needed. After a communication problem at 234 seconds however, re-initialisation happens at every event.

6-4-3 Results

Despite the communication problems, we were able to show ETC+ can reduce communication time for a few experiments. Because of the large difference between this setup and the one used for ETC in Section 6-3-2, we cannot compare the results. Therefore we will also redo ETC experiments on this new setup. The new ETC experiments and the ETC+ experiments to compare them to can be found in Table 6-2. The time evaluations for the ETC+ experiments can be found in Appendix D-4.

	σ	ϵ (mm^2)	radio on (s)	triggers	mse (mm^2)	mtse (mm^2s)
ETC	no	trigger	42.0	69	n/a	n/a
ETC	0.1	1	48.9	182	808	818
ETC+	0.1	1	46.6	197	812	822
ETC	0.2	2	48.8	189	826	838
ETC+	0.2	2	42.2	195	862	873

Table 6-3: Overview of the results for the experiment from Section 6-1 on WCB-E+ with different parameters.

The ETC+ experiment with $\sigma = 0.1$ and $\epsilon = 1$ unfortunately had a communication problem after 1748 seconds of the 1800 second experiment, but despite this its radio on time is 1.7 seconds shorter than normal ETC. The ETC+ experiment with $\sigma = 0.2$ and $\epsilon = 2$ did not have communication problems. It does have a few extra triggers, compared to ETC, caused by initialisations of the algorithm in the beginning, but its radio on time is almost six seconds shorter with 42.2 ms. This is almost the same radio on time for the case where the only triggers are caused by the heartbeat of the system, which shows the potential of the hybrid approach. The performance with this setting however is worse than normal event-triggered control, caused by oscillations at the beginning during extra sleeping (Figure D-24a). Sleeping periods are large in this area (Figure D-24c), so this could be solved by making the predictions more conservative, at the cost of more communication time, if desired.

Chapter 7

Discussion and conclusions

7-1 Conclusion

As intended we implemented a wireless control network on the water testbed unit at TU Delft using the protocols WCB-P and WCB-E. There were several mostly mechanical problems with the testbed which prevented to run the experiments we had in mind to compare the performance of the different control methods. The Contiki-application on the Firefly network however worked well. therefore a switch to a hardware-in-the-loop simulation was made.

Using the HIL simulation it was possible to show all parts of the application worked, and compare the protocols. WCB-P can do control with only 7% radio on time, but WCB-E only needs 2% with almost the same performance to reject a disturbance.

With an extension on WCB we were able to implement an aperiodic hybrid approach of ETC and predictive self-triggered control, which call ETC+. Although this approach does not work well when there are communication errors, the succeeded experiments show that ETC+ can reduce communication, compared to normal ETC. Although it needs some extra triggers for initialisation, the extra sleeping periods reduce communication enough to compensate for this.

7-2 Future work

7-2-1 Simulation

Because one of the goals was to implement an application to control the water testbed unit, this is where we started instead of a simulation. A hardware-in-the-loop simulation on the other hand gives much more control, and could even be used to do normal STC because you can control everything in simulation. For this thesis the HIL simulation was created to run in Simulink, which made it possible to do more from home during the Covid-measures, but my advice would be for future development to run the simulation on Speedgoats, which can run much faster and are therefore more accurate.

7-2-2 Control the real water testbed

Because we were not able to stabilise the real water testbed, this should be looked at. Many improvements to the setup are explained in Section 3-7, but most important to fix is the leakage of the gates. My advice would then be not to try to control all 3 pools but start with one or two and use pool 1 as extra water reservoir to make it easier to regulate the inflow of the system.

There is also an option to slow down the system by placing barriers with holes in the pools. This will make it easier to control the system and more similar to a real water irrigation system. Unfortunately we did find out about this option late in the project. A small experiment with the barriers showed that the holes should be made smaller to create a significant slow down.

I would also recommend to design a fully centralised controller, which can take advantage of knowledge of the full system, so known flows from one pool to the other do not have to be treated like disturbances. This will make the overall system more stable because errors will not propagate through the system. This centralised controller could even be run on the external PC, in a setup similar to the setup used for ETC+, when an extra period delay is modelled into the plant, which will make designing the controller much easier.

7-2-3 Control application

The reference values and controller parameters are now hardcoded in the firmware of the application. To make control design easier it would be good to make them configurable. The mechanism to switch from manual to automatic mode could be used to distribute these from the network controller to the other nodes. These are the reference values and the controller parameters. Development of WCB-P and WCB-E, which were used for this thesis, has been done in two separate development branches which are not in sync anymore. Merging them again seems very difficult, but might be worth looking at to speed up future development. Another thing to look at is integrator windup, if we can get a better estimation or a measurement of the flows over a gate.

7-2-4 Protocol

Because we were mainly interested in a comparison between the protocols we have not tuned them, we only made sure that WCB-P and WCB-E use the same settings for a fair comparison. Especially because we ran into communication problems during the ETC+ experiments, it might be a good idea to investigate the reliability of WCB in a practical setup, and find the optimal balance between speed reliability, for both single and multi-hop networks.

Also we used 1 Hz for periodic control because this was the maximum speed the first version of our application could run. Evaluating this control period, and reducing the frequency is an easy way to reduce communication. Of course this reduction might degrade performance.

7-2-5 ETC+

Because the experiments show ETC+ can reduce communications, it is promising to look further into. The main problem we encountered when using PSTC to do the conservative predictions, is that it cannot re-initialise after a communication problem. This happens when one or more Fireflies do not enter the collection phase. It should be looked into why this happens. It could be a wireless communication problem, but it could also be a bug in the code.

Another thing to look at, and which might be a cause of the re-initialisation problem, is the update of the controller state. The algorithm now updates the state itself, based on the feedback. Because of this 'open loop' update, it cannot recover when something goes wrong. When the setup would be changed to send the controller state from the control network to the algorithm, this could make it more robust.

It can be very challenging to find all bounds on the disturbances, which need to be known for the PSTC algorithm to work. When a bound is violated the algorithm now just re-initialises, but it might be interesting to investigate if it is possible to detect what disturbance caused this violation and automatically adapt the bounds for it.

The sleeping times predicted by PSTC decrease when the reference error gets smaller. A small improvement might therefore be to switch between ETC and ETC+ based on the current error.

Appendix A

Codebase

All code for this thesis has been developed under version control, and shared via Gitlab and Github. The control application has been written in C (A-1). Experiments and simulations have been performed in Matlab (A-2), which was also used to do PSTC predictions. Python was mainly used to create helper scripts to collect data from the Fireflies (A-3), but also does an alternative HIL simulation for the ETC+ experiments.

A-1 Control application

The control application has been written in C for the Contiki operating system. Because it it has been written on top of the Wireless Control Bus protocols, which were developed at the University of Trento [30], the code cannot be made public, yet. The repository is maintained by Gabriel Gleizer of the SENTIENT (Scheduling of Event-Triggered Control Tasks) research group at the TU Delft. It contains five branches:

- periodic
- etc
- dev-periodic
- dev-etc
- dev-etcp

Periodic and etc are the branches with the original code for WCB-P and WCB-E. Dev-periodic and dev-etc contain the extensions and application created for this thesis. Dev-etcp is a copy of dev-etc, and is only intended to do some ETC+ comparisions, not for future development.

https://gitlab.tudelft.nl/ggleizer/wireless-control-bus

The code is not public yet because it contains code from the D3S Research Group of the University of Trento who will make their code public after [30] has been approved. When it has been approved their code will be made public, and my application built on it can also be made public.

A-2 Identification and simulation

The scripts to identify the water testbed, create the controllers, do the simulations and create the plots are written for Matlab and Simulink (R2019b Update 8) and CVX (Version 2.2, Build 1148) with SDPT3 (4.0). Also the hardware-in-the-loop simulation in Simulink, and the scripts to do the PSTC predictions to be used by WCB-E+ can be found here.

https://github.com/basboot/WIS-sim

A-3 Logging and simulation

The control application can send logging data over the serial connection. To make it easier to save data from multiple Fireflies and combine them into a single csv-file Pyhon scripts have been written for this task. Also a less realistic HIL simulation to do ETC+ experiments can be found here. The version of Python used is 3.8, with additional packages Numpy (1.21.1), PySerial (3.4) and Scipy (1.7.1).

https://github.com/basboot/WIS-com

Appendix B

WCB setup in Trento

Wireless Control Bus (WCB) was originally developed by the D3S Research Group for use on the testbed at the University of Trento. In this appendix their setup will be explained (Appendix B-1) and the lifecycle functions of the protocol omitted in Section 5-3 because they were not needed in our setup, will be documented (Appendix B-2).

B-1 Testbed in Trento

The testbed at the University of Trento is a network-in-the-loop setup to test performance of the WCB protocols [30]. It has a network of Fireflies which are not connected to a real water irrigation setup, but are connected to a Simulink simulation like in our hardware-in-the-loop simulation (Section 5-11). The difference with our HIL simulation is that the controllers also run in Simulink and not on the Fireflies. Therefore it is not needed to send actual control data over WCB. Instead it is only checked if measurements and control messages are received in the correct epoch, while the actual data comes from Simulink.

B-2 Lifecycle and hook functions

To check network reliability and do communication with Simulink some extra hook functions were added to the protocol which are specific for this setup in Trento, and are not part of the protocol itself. A full overview of the implementation used, including these extra functions and without the functions added for this thesis, is shown in Figure B-1.

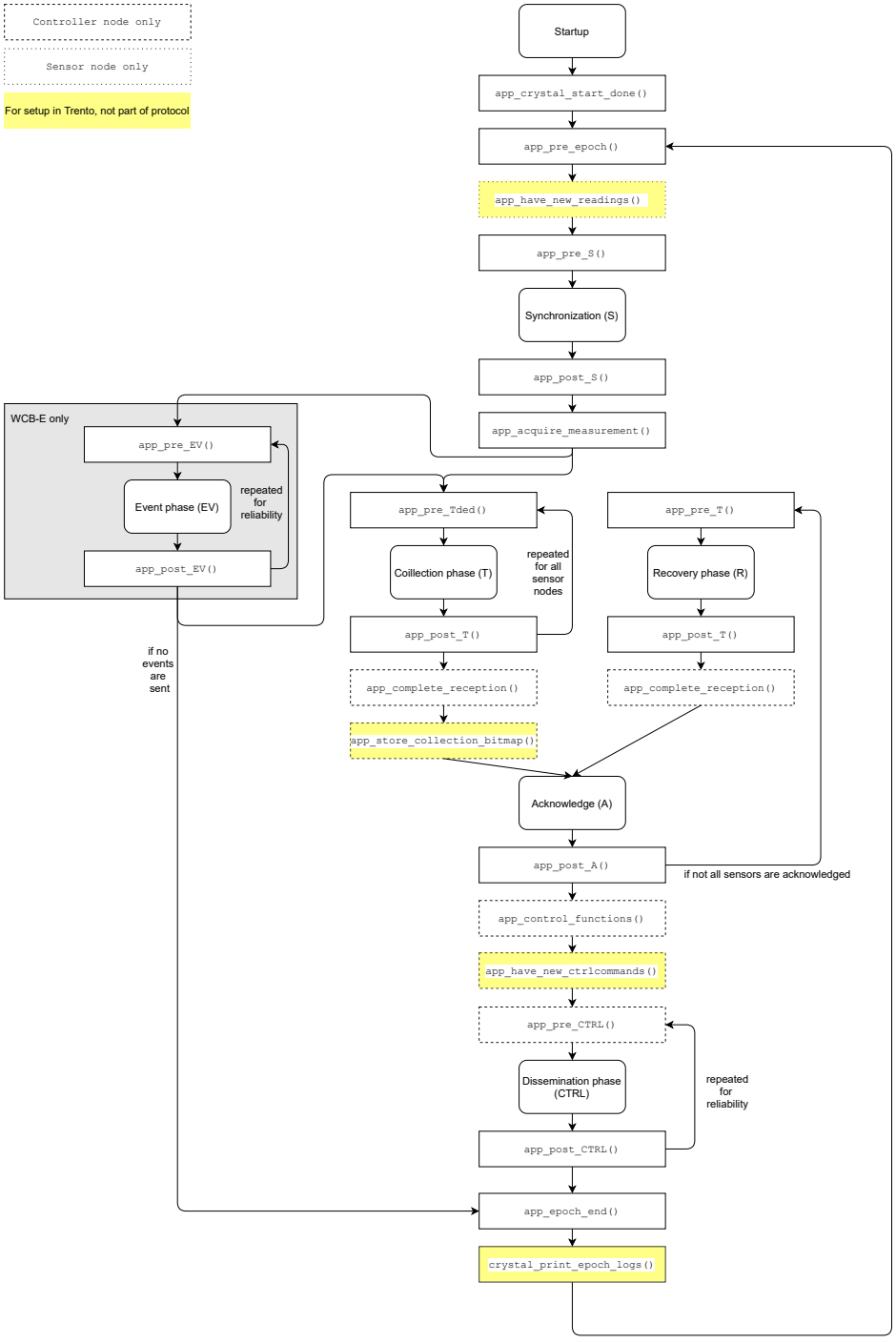


Figure B-1: Lifecycle and hook functions of the WCB implementation, as used for the testbed at the university of Trento . For clarity all function parameters have been omitted. The full signatures can be found inside the header file of the protocol.[1]

A full explanation of the protocol and lifecycle can be found in Chapter 2-3. The extra functions for the setup (in yellow) are described below.

`app_have_new_readings()`	Checks if Simulink sensor readings have been received in the correct epoch.
`app_store_collection_bitmap()`	For analysis purpose. Stores the bitmap at the end of the Collection Phase.
`app_have_new_ctrlcommands()`	Checks if Simulink control commands have been received in the correct epoch.
`crystal_print_epoch_logs()`	For analysis purpose. Logs statistics for this epoch.

[1] `/net/crystal/crystal.h`

Appendix C

Water testbed images

In this appendix, images of all parts of the water testbed unit at TU Delft, as described in Chapter 3, are shown.

C-1 Images

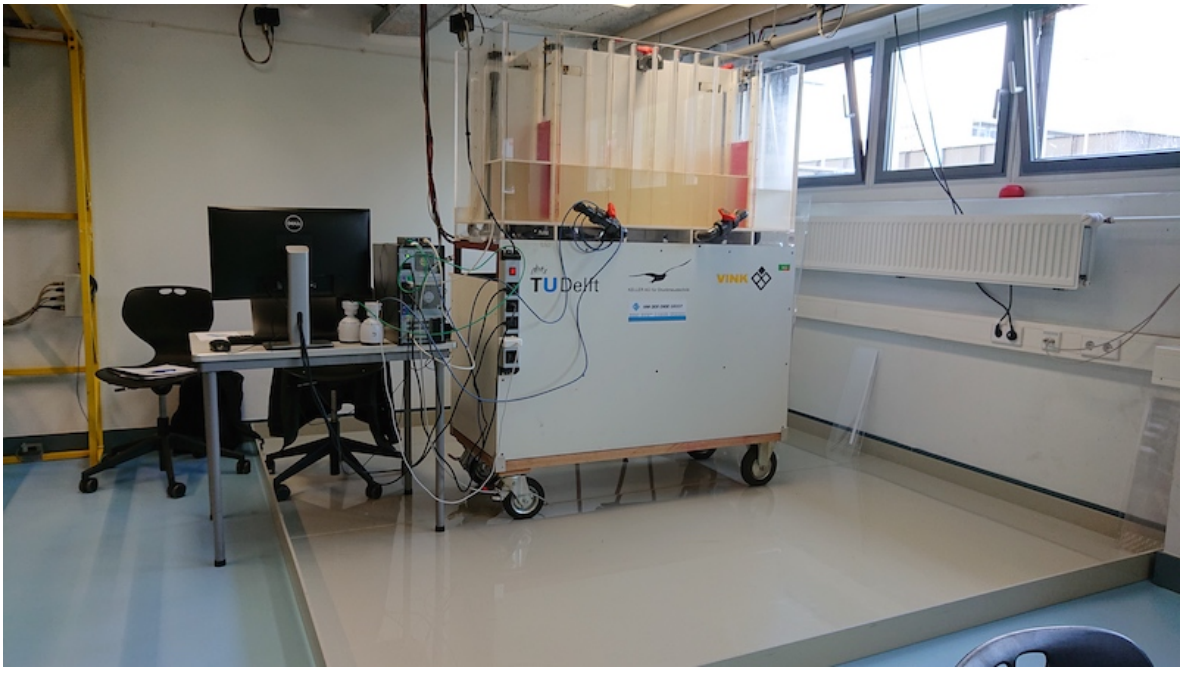


Figure C-1: The testbed is located in the basement of TU Delft and placed inside a bin to prevent flooding.

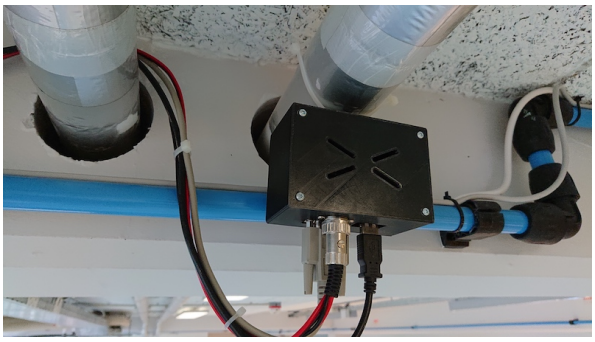
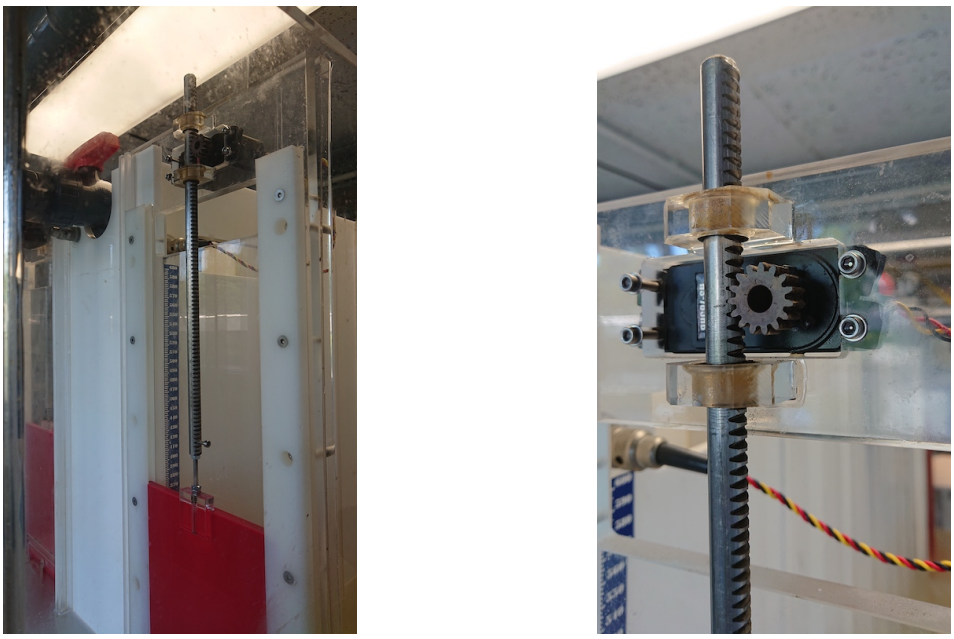


Figure C-2: Firebox. In the lab there are 8 Fireboxes containing the Fireflies.



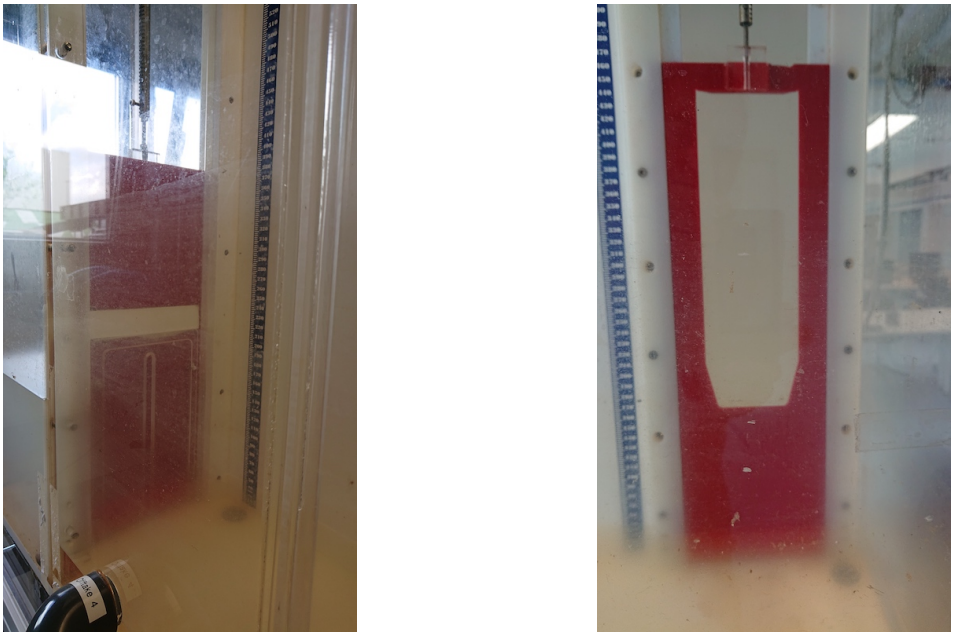
(a) Gate mechanism. **(b)** Servo.

Figure C-3: Gate. The gates are operated by the servos which pull them up and down with gears connected directly to them.

Figure C-4: Main valve. The main valve leads to $pool_0$ and has to be controlled by hand.

(a) In-take. **(b)** Off-take.

Figure C-5: In- and off-takes to simulate disturbances. The in- and off-takes for $pool_1$ to $pool_3$ are controlled by hand.



(a) Undershot gate. **(b)** Overshot gate.

Figure C-6: Gate 3 and 4. Undershot gates are used for $gate_1$ to $gate_3$. The last gate $gate_4$ is an overshot gate.

Figure C-7: Power supply for the testbed. There are no switches for the pumps, so they are operated by plugging them in. The plug on the left is used to connect the testbed to the ground.

Figure C-8: Water pump. The pumps are located inside water basin under the testbed.

Figure C-9: Pressure sensor. The sensors are located under the pools.

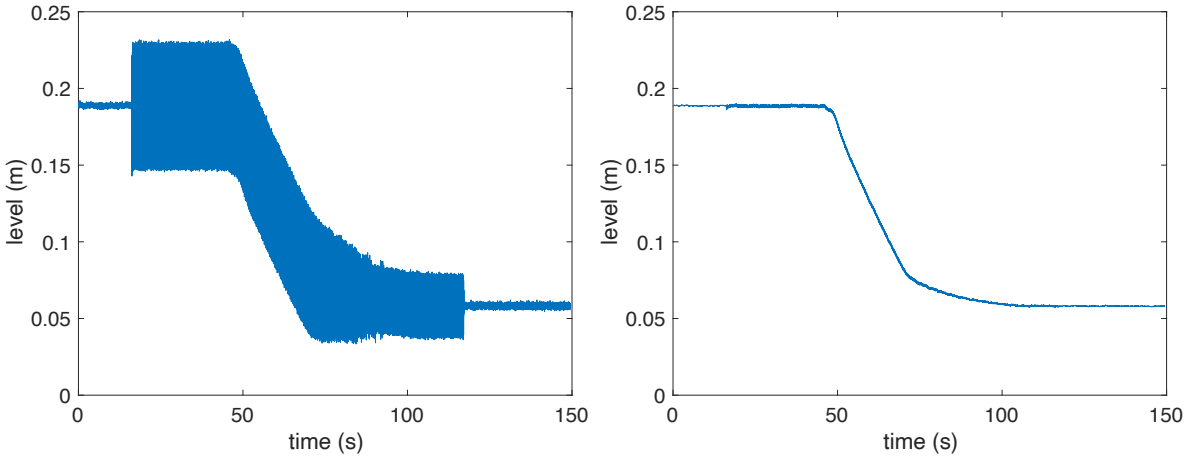
Figure C-10: Powered usb-hubs. The Fireflies are connected to powered usb-hubs in the following order (left to right): FF4, FF3, FF6, FF8, FF5, FF7, FF2, FF1.

Appendix D

Plots

In this appendix, extra plots are shown which did not fit into the main text, and were not absolutely necessary for the story. In Appendix D-1 the disturbance of the pumps and effect of the digital filter can be found, in Appendix D-2 the full simulations referred to in Section 6-1 are shown and in Appendix D-3 the time evaluations of the hardware-in-the-loop simulations for periodic and event-triggered experiments summarised in Table 6-2 are shown. The time evaluations along with the predicted sleeping times for ETC+ are shown in Appendix D-4.

D-1 Filtering



(a) Unfiltered **(b)** Filtered

Figure D-1: Comparison of the filtered and unfiltered sensor data for an experiment where the pump is turned on in the middle.

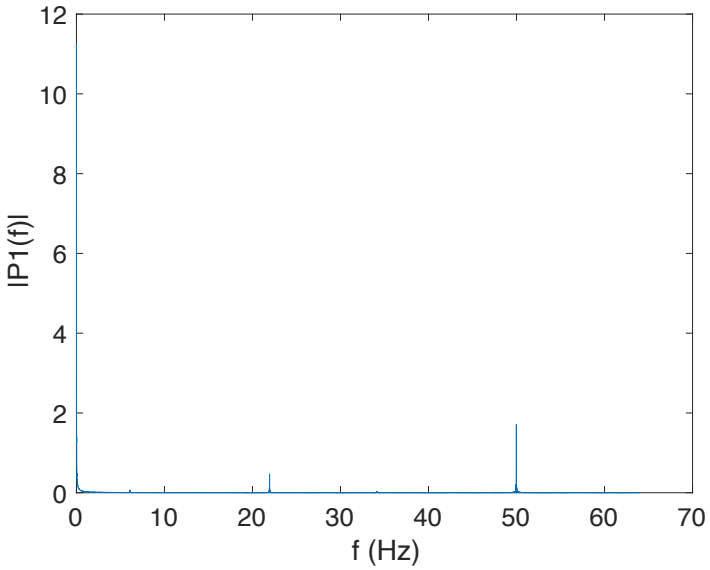
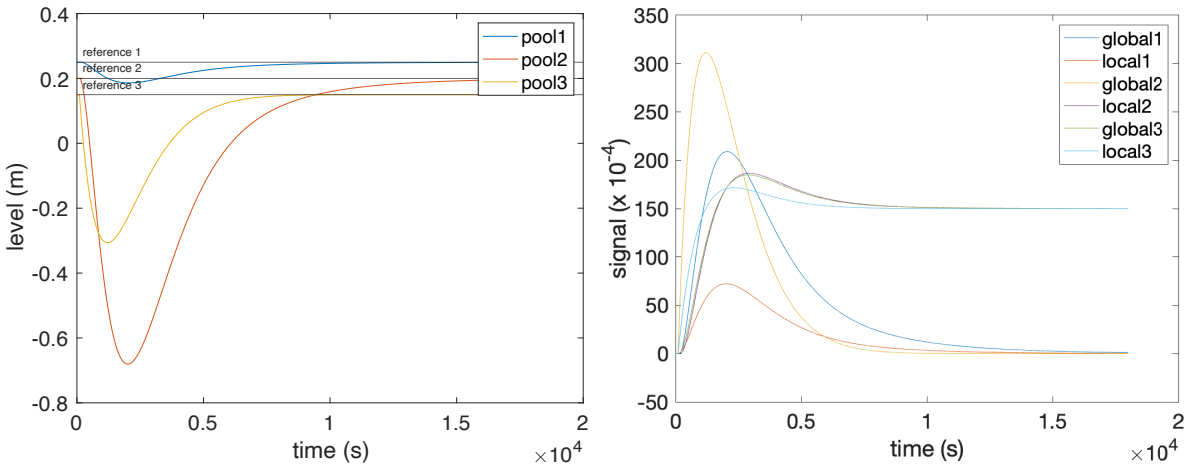


Figure D-2: Single-Sided Amplitude Spectrum of the unfiltered sensor data from Figure D-1a.

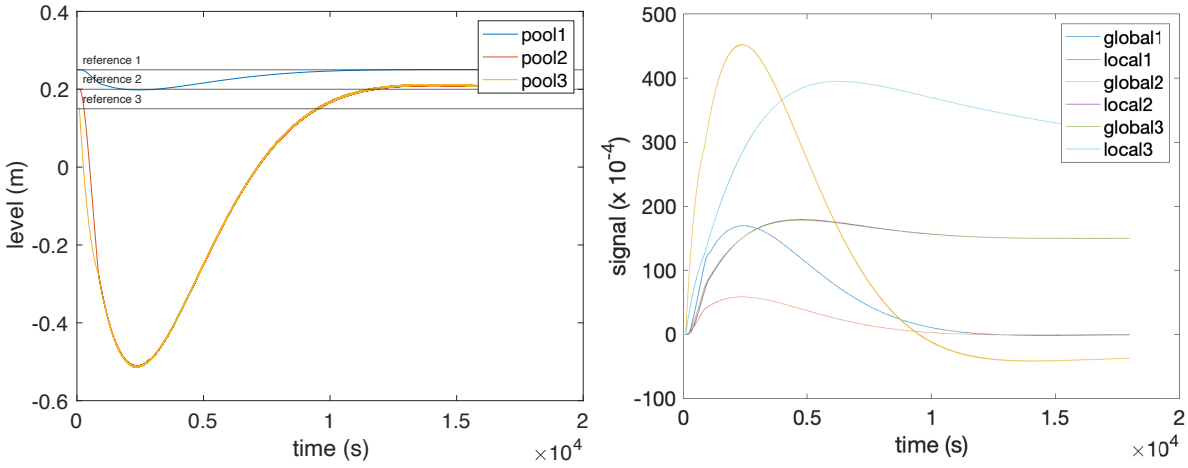
D-2 Simulated experiments



(a) Water levels

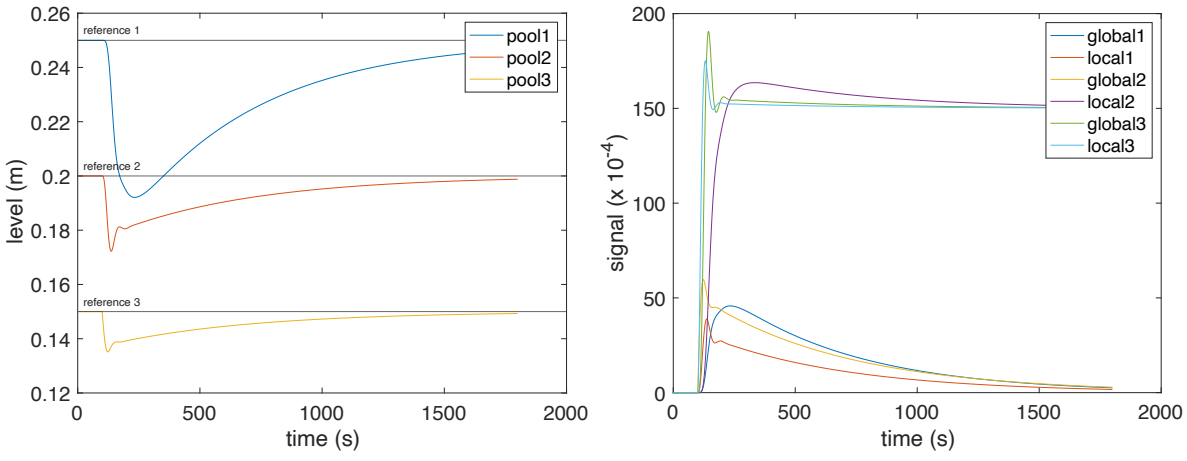
(b) Control signals

Figure D-3: Time evolution of water levels and control signals after a step disturbance on the unrestricted simulation using periodic wireless control with controller from Section 4-1 and 4-2. The controller takes 5 hours the reject the disturbance the water reaches negative levels and streams up from pool 2 to pool 3.



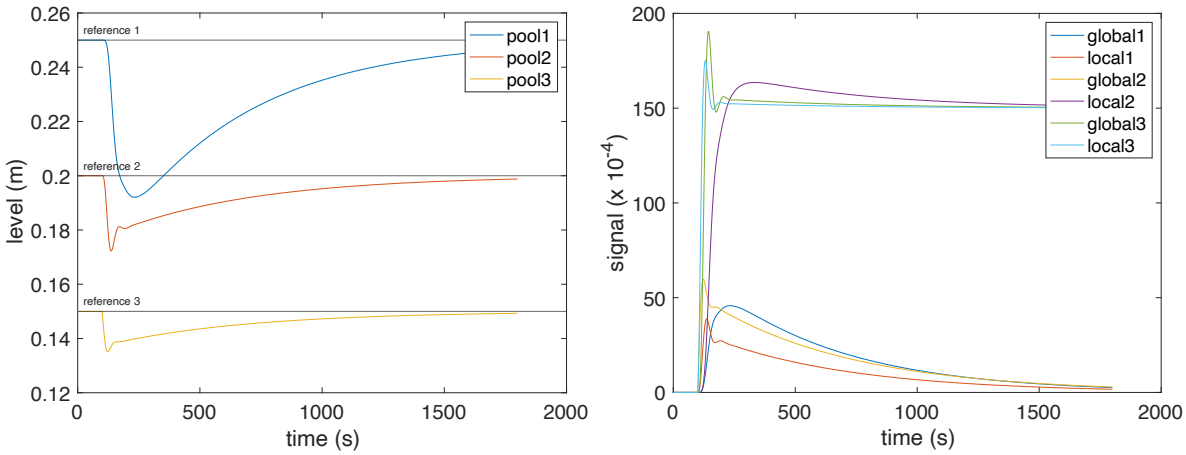
(a) Water levels **(b)** Control signals

Figure D-4: Time evolution of water levels and control signals after a step disturbance on the flow restricted simulation using periodic wireless control with controller from Section 4-1 and 4-2. Results are similar to Figure D-3, but water does not flow up anymore.



(a) Water levels **(b)** Control signals

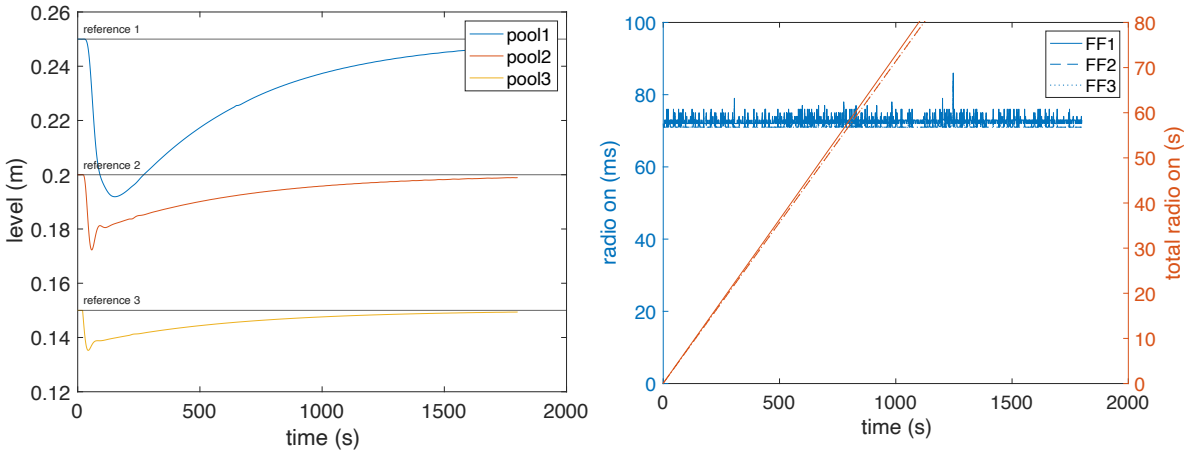
Figure D-5: Time evolution of water levels and control signals after a step disturbance on the unrestricted simulation using periodic wireless control with controller from Section 4-3. In simulation it can reject the disturbance in half an hour without exceeding the minimum water levels.



(a) Water levels **(b)** Control signals

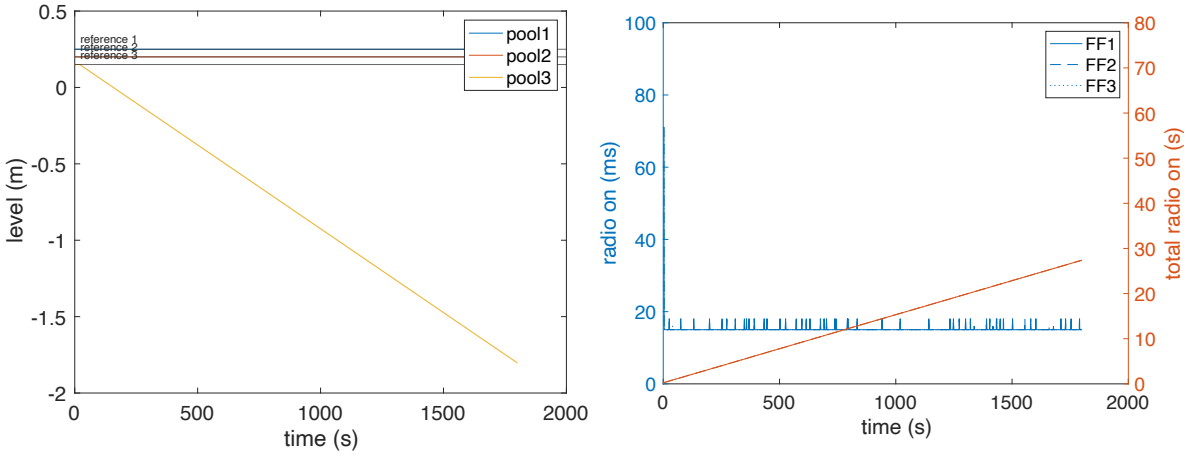
Figure D-6: Time evolution of water levels and control signals after a step disturbance on the flow restricted simulation using periodic wireless control with controller from Section 4-3. The result is the same as Figure D-5 because the controller does not ask for impossible flows.

D-3 HIL ETC experiments



(a) Water levels **(b)** Radio

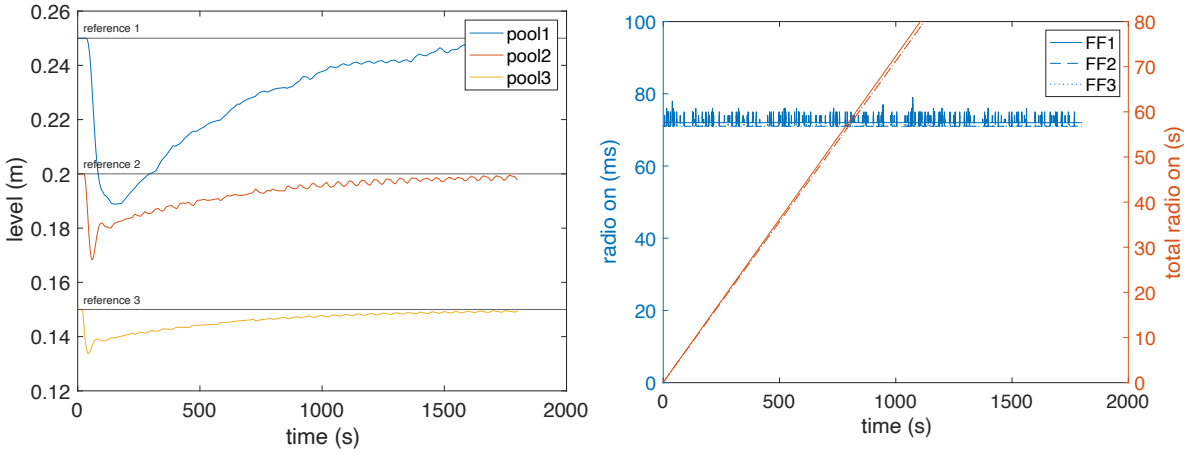
Figure D-7: Time evolution of water levels and radio on time after a step disturbance on the HIL simulation using periodic wireless control with controller from Section 4-3.



(a) Water levels

(b) Radio

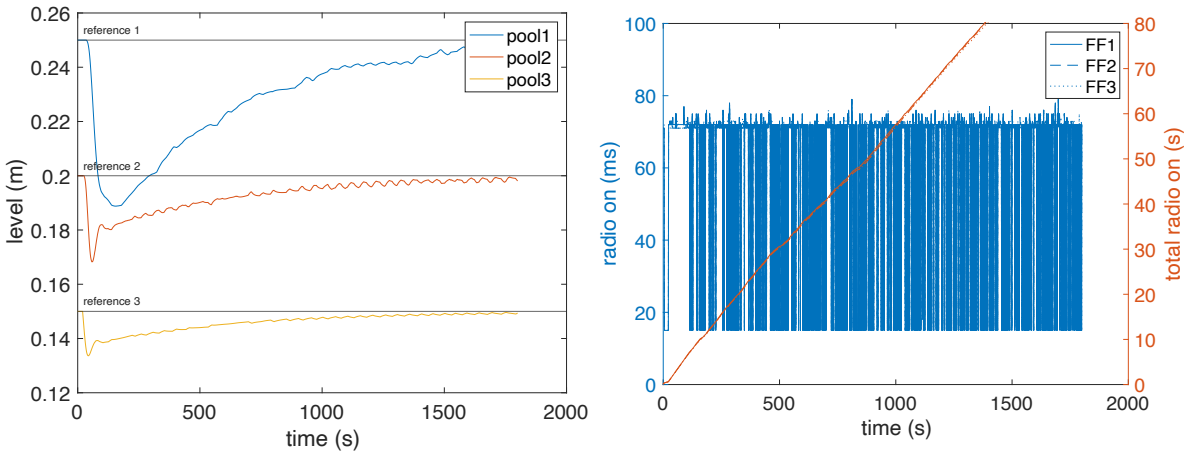
Figure D-8: Time evolution of water levels and radio on time after a step disturbance on the HIL simulation using event-triggered wireless control with controller from Section 4-3 (never triggering).



(a) Water levels

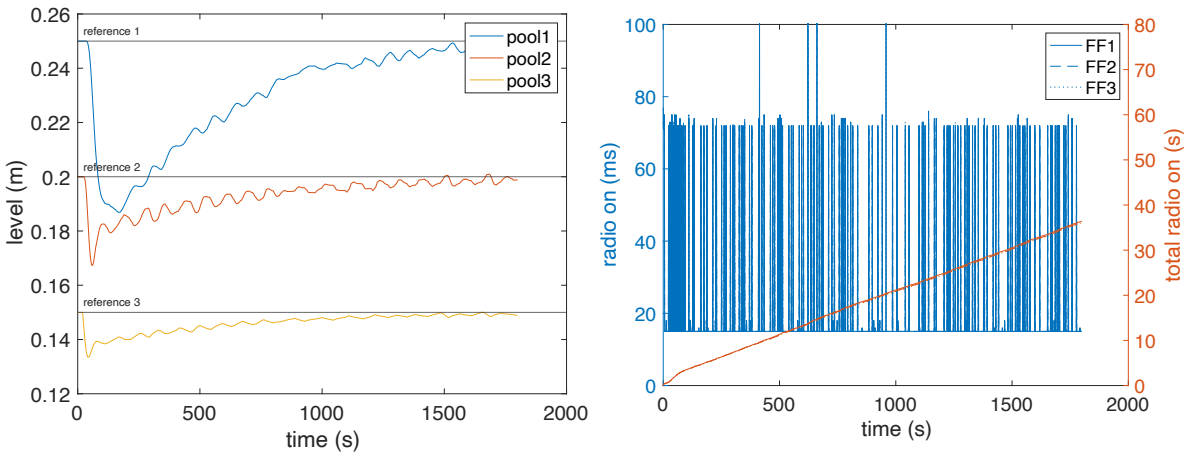
(b) Radio

Figure D-9: Time evolution of water levels and radio on time after a step disturbance on the HIL simulation using event-triggered wireless control with controller from Section 4-3 (forcing a trigger every period).



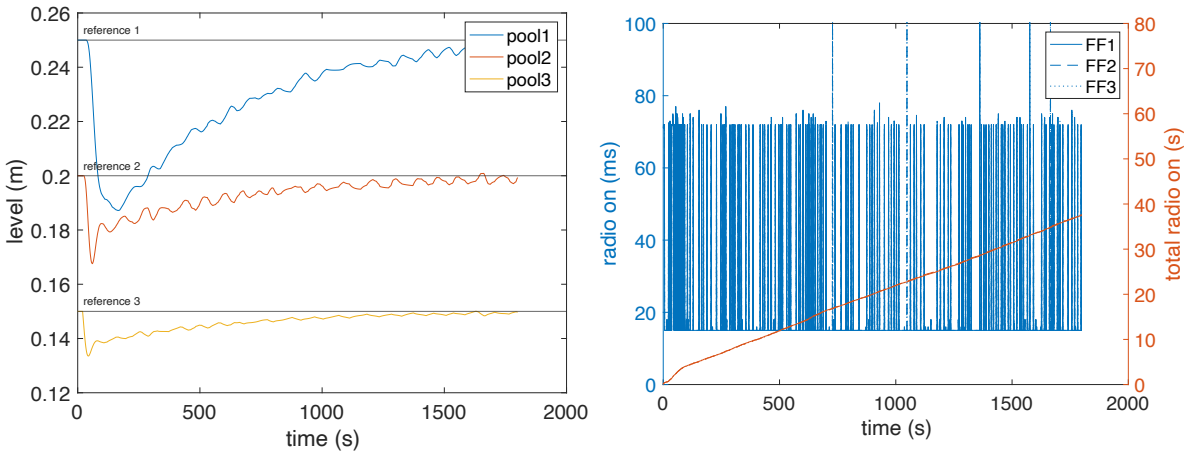
(a) Water levels **(b)** Radio

Figure D-10: Time evolution of water levels and radio on time after a step disturbance on the HIL simulation using event-triggered wireless control with controller from Section 4-3 ($\sigma = 0, \epsilon = 0$).



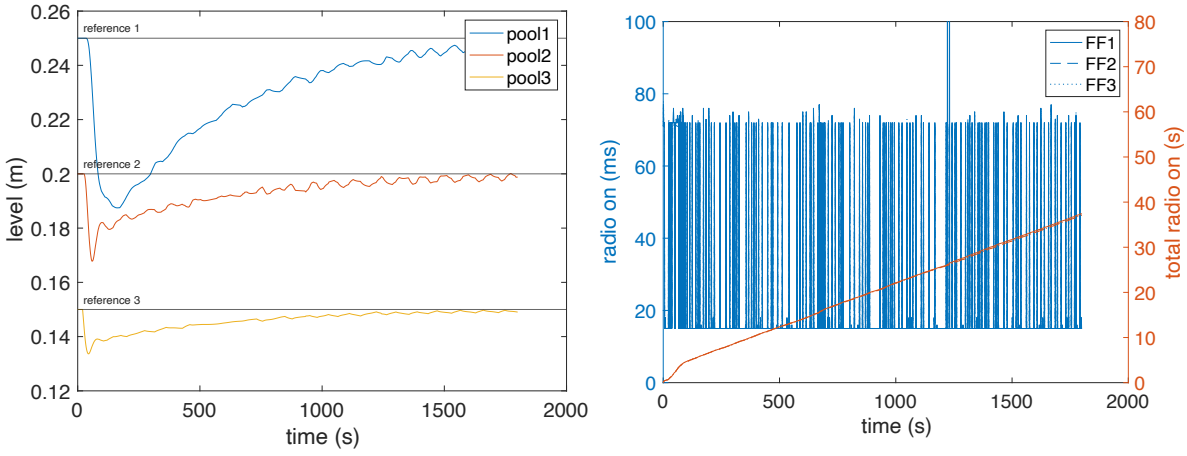
(a) Water levels **(b)** Radio

Figure D-11: Time evolution of water levels and radio on time after a step disturbance on the HIL simulation using event-triggered wireless control with controller from Section 4-3 ($\sigma = 0.1, \epsilon = 1$).



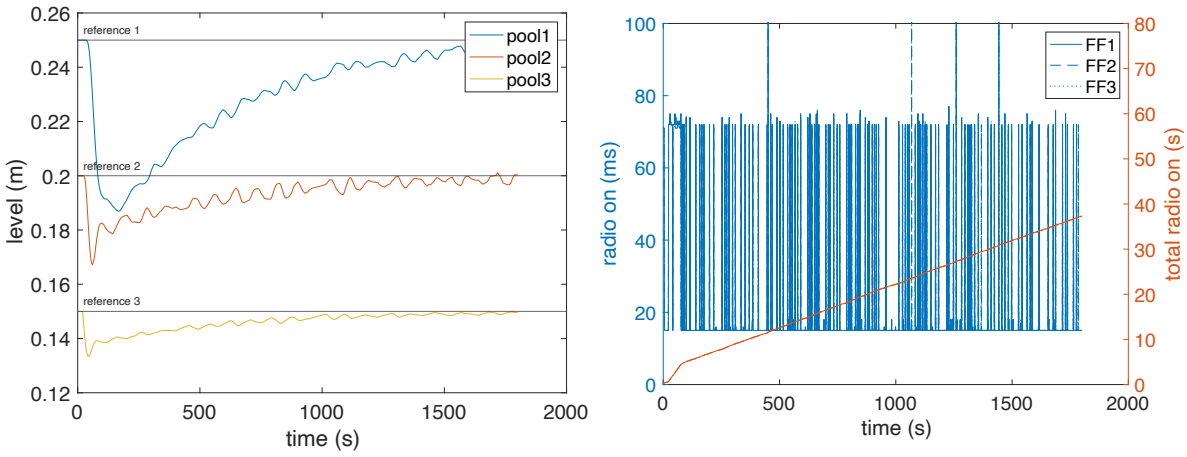
(a) Water levels **(b)** Radio

Figure D-12: Time evolution of water levels and radio on time after a step disturbance on the HIL simulation using event-triggered wireless control with controller from Section 4-3 ($\sigma = 0.05, \epsilon = 1$).



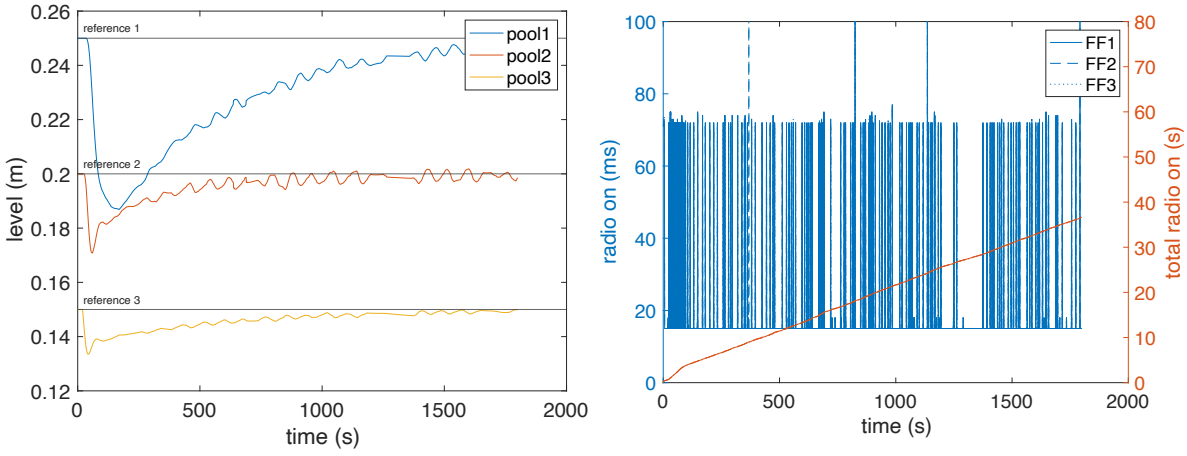
(a) Water levels **(b)** Radio

Figure D-13: Time evolution of water levels and radio on time after a step disturbance on the HIL simulation using event-triggered wireless control with controller from Section 4-3 ($\sigma = 0.025, \epsilon = 1$).



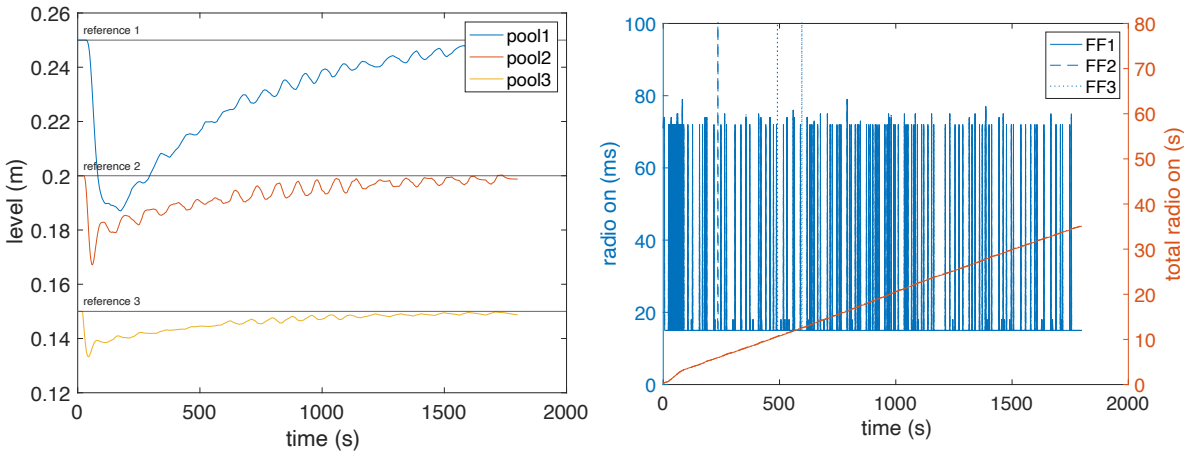
(a) Water levels **(b)** Radio

Figure D-14: Time evolution of water levels and radio on time after a step disturbance on the HIL simulation using event-triggered wireless control with controller from Section 4-3 ($\sigma = 0.05, \epsilon = 2$).



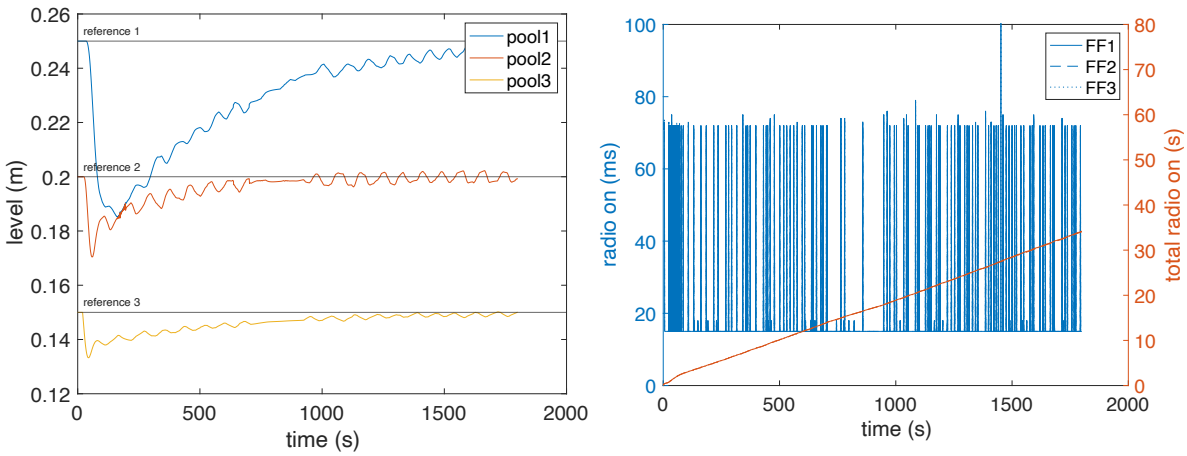
(a) Water levels **(b)** Radio

Figure D-15: Time evolution of water levels and radio on time after a step disturbance on the HIL simulation using event-triggered wireless control with controller from Section 4-3 ($\sigma = 0.025, \epsilon = 2$).



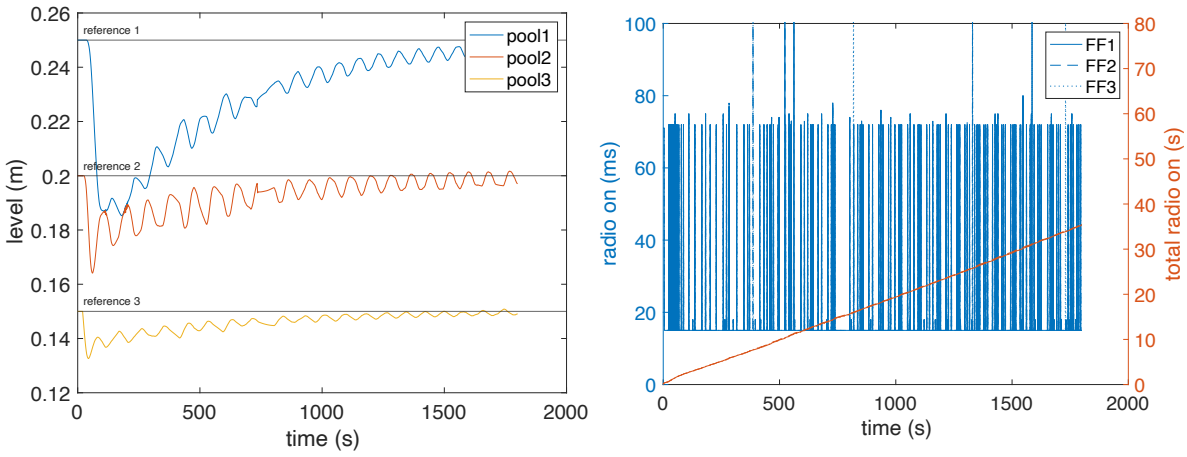
(a) Water levels **(b)** Radio

Figure D-16: Time evolution of water levels and radio on time after a step disturbance on the HIL simulation using event-triggered wireless control with controller from Section 4-3 ($\sigma = 0.1, \epsilon = 2$).



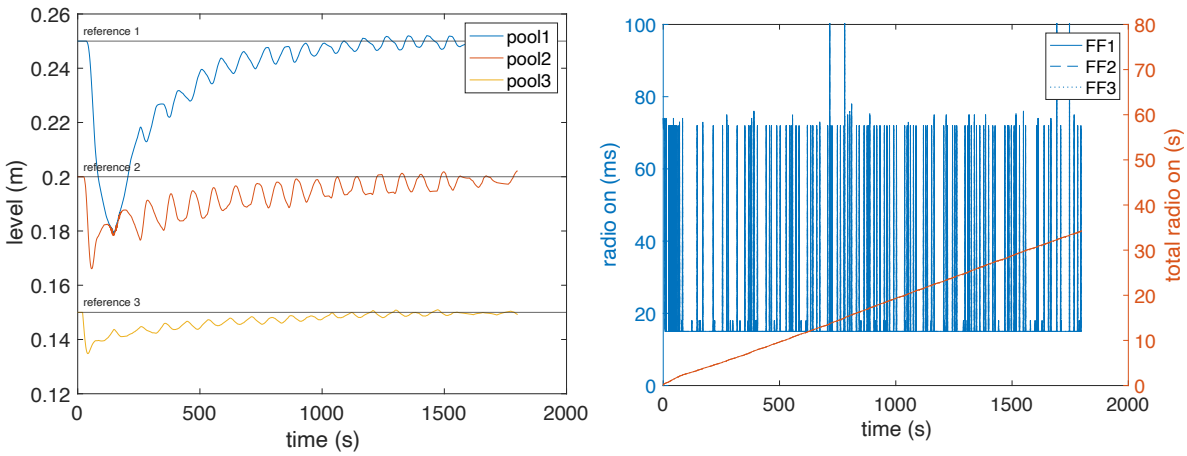
(a) Water levels **(b)** Radio

Figure D-17: Time evolution of water levels and radio on time after a step disturbance on the HIL simulation using event-triggered wireless control with controller from Section 4-3 ($\sigma = 0.2, \epsilon = 2$).



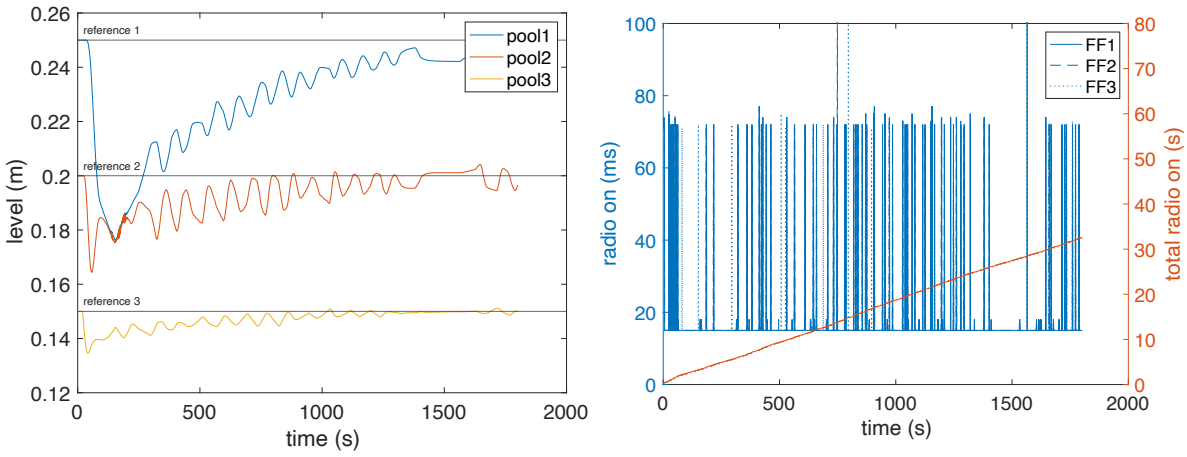
(a) Water levels **(b)** Radio

Figure D-18: Time evolution of water levels and radio on time after a step disturbance on the HIL simulation using event-triggered wireless control with controller from Section 4-3 ($\sigma = 0.4, \epsilon = 2$).



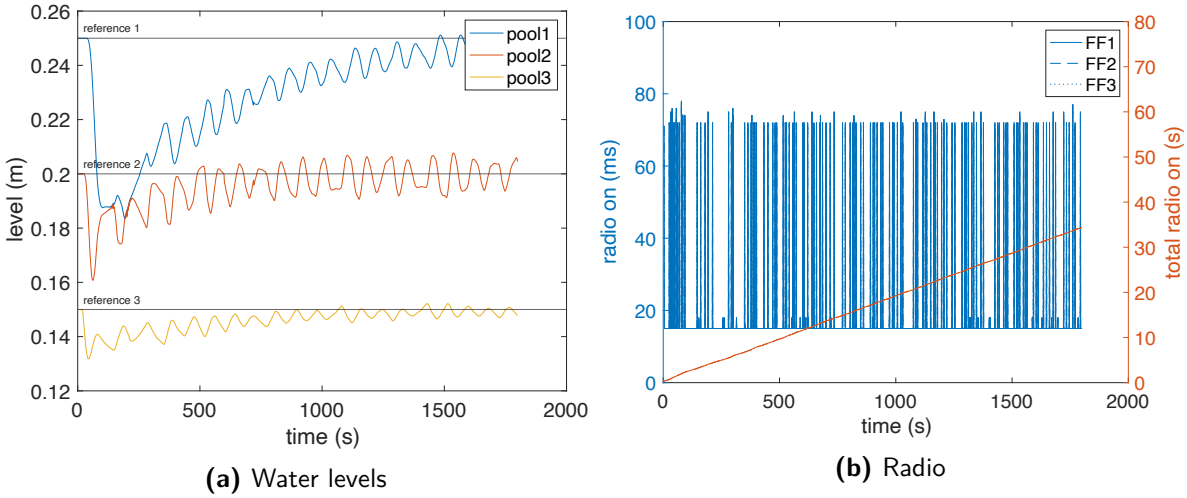
(a) Water levels **(b)** Radio

Figure D-19: Time evolution of water levels and radio on time after a step disturbance on the HIL simulation using event-triggered wireless control with controller from Section 4-3 ($\sigma = 0.4, \epsilon = 4$).



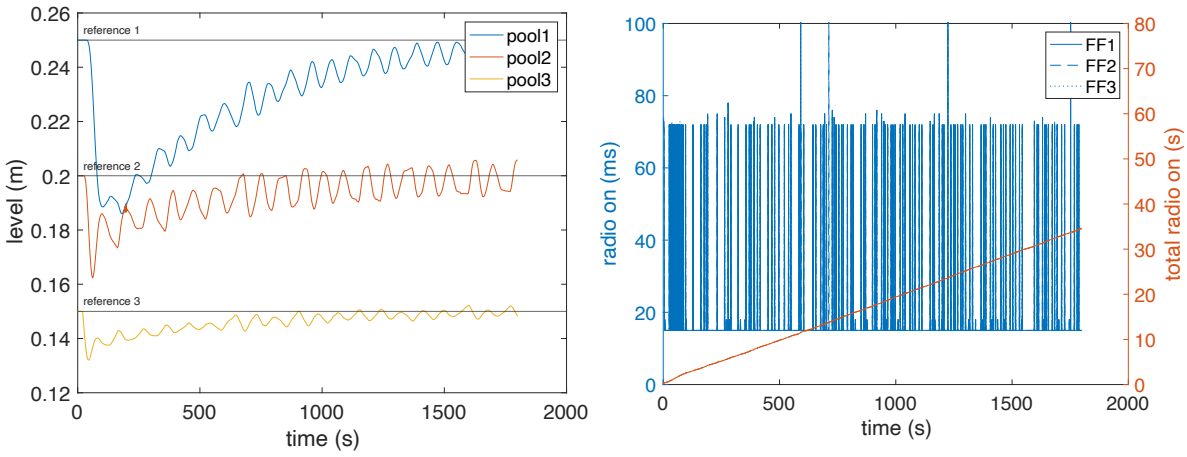
(a) Water levels **(b)** Radio

Figure D-20: Time evolution of water levels and radio on time after a step disturbance on the HIL simulation using event-triggered wireless control with controller from Section 4-3 ($\sigma = 0.4, \epsilon = 8$).



(a) Water levels **(b)** Radio

Figure D-21: Time evolution of water levels and radio on time after a step disturbance on the HIL simulation using event-triggered wireless control with controller from Section 4-3 ($\sigma = 0.4, \epsilon = 16$).

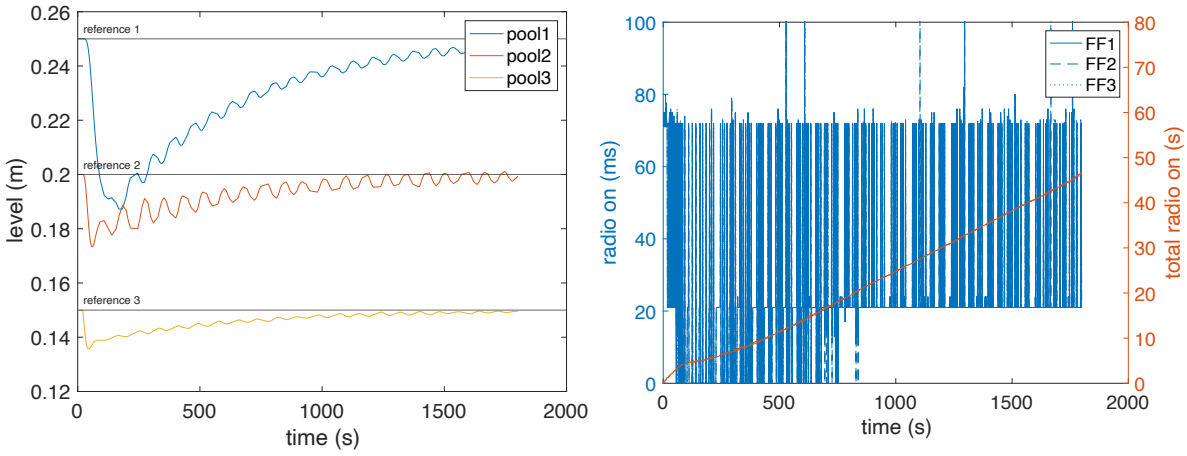


(a) Water levels

(b) Radio

Figure D-22: Time evolution of water levels and radio on time after a step disturbance on the HIL simulation using event-triggered wireless control with controller from Section 4-3 ($\sigma = 0.2, \epsilon = 16$).

D-4 HIL ETC+ experiments



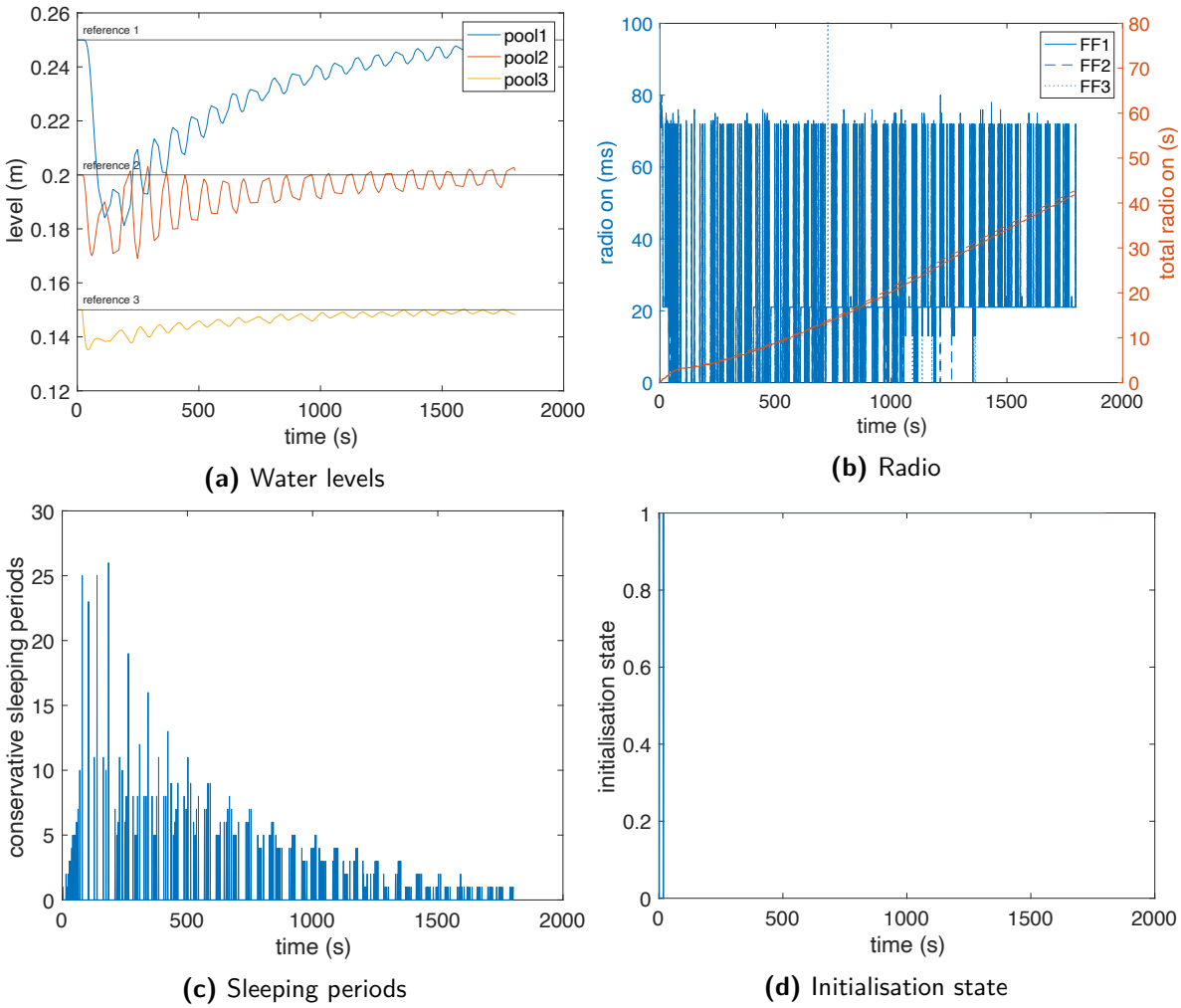
(a) Water levels

(b) Radio

(c) Sleeping periods

(d) Initialisation state

Figure D-23: Time evolution of water levels, radio on time, calculated sleeping periods and initialisation state after a step disturbance on the HIL simulation using ETC+ with controller from Section 4-3 ($\sigma = 0.1, \epsilon = 1$).



(a) Water levels

(b) Radio

(c) Sleeping periods

(d) Initialisation state

Figure D-24: Time evolution of water levels, radio on time, calculated sleeping periods and initialisation state after a step disturbance on the HIL simulation using ETC+ with controller from Section 4-3 ($\sigma = 0.2, \epsilon = 2$).

Appendix E

Simulink models

This appendix contains images of the most important parts of the full simulation (Appendix E-1) and the hardware-in-the-loop simulation (Appendix E-2). The code for the models can be found in Appendix A-2.

E-1 Full simulation

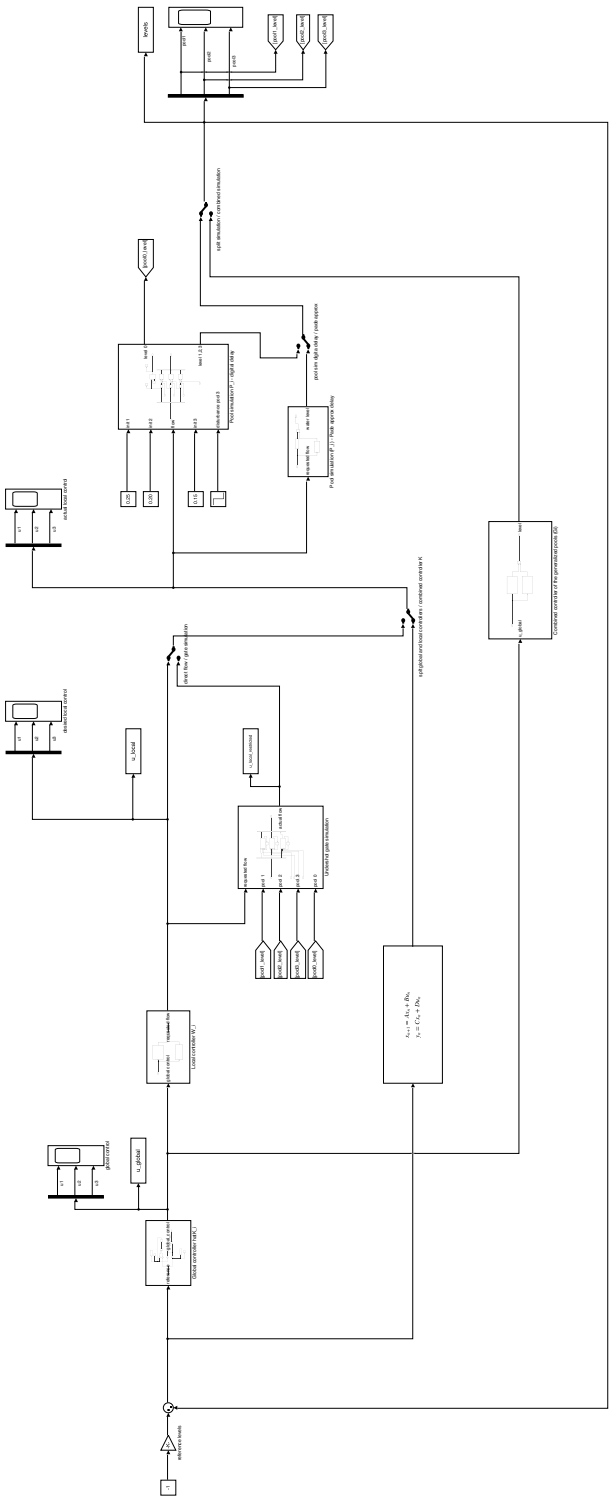


Figure E-1: Simulink model for the simulation of the water testbed unit.

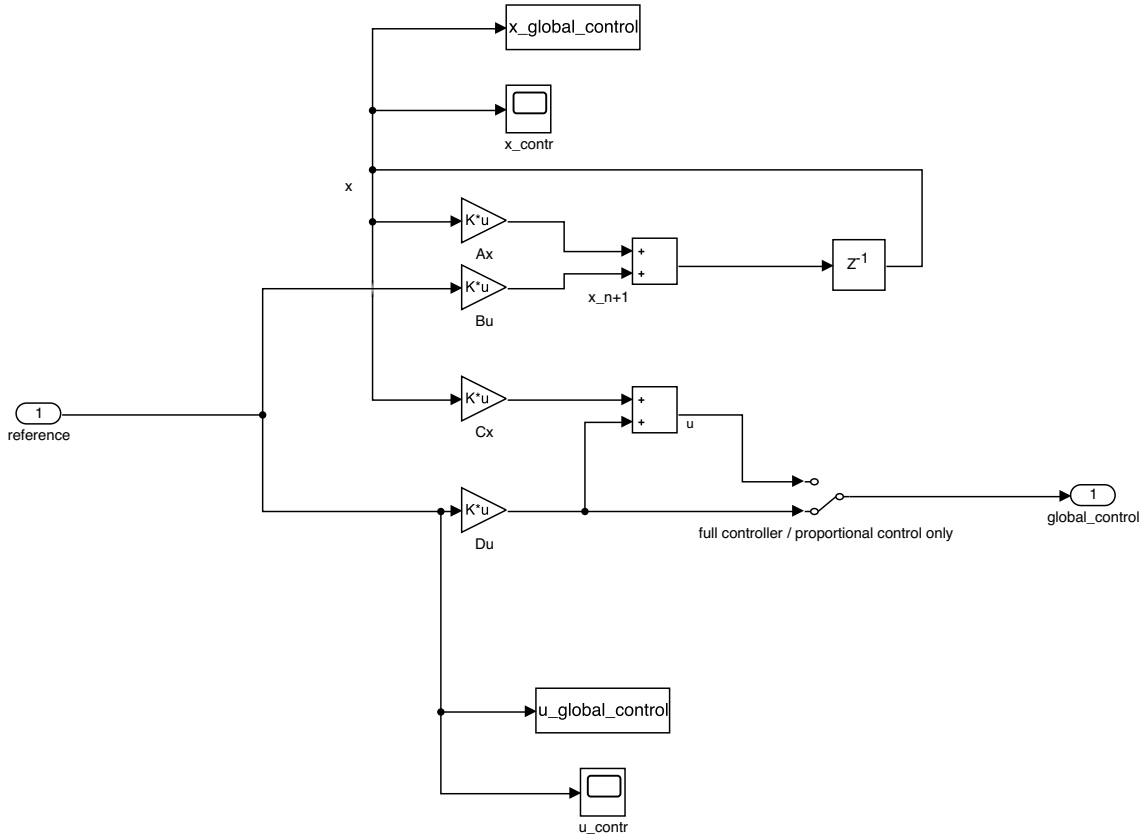


Figure E-2: Simulink model for the global controller.

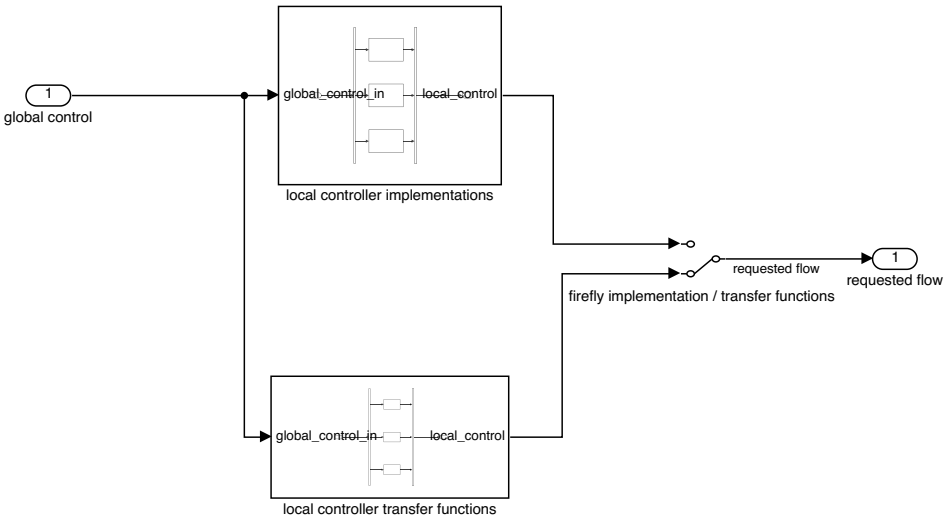


Figure E-3: Simulink model for the local controller.

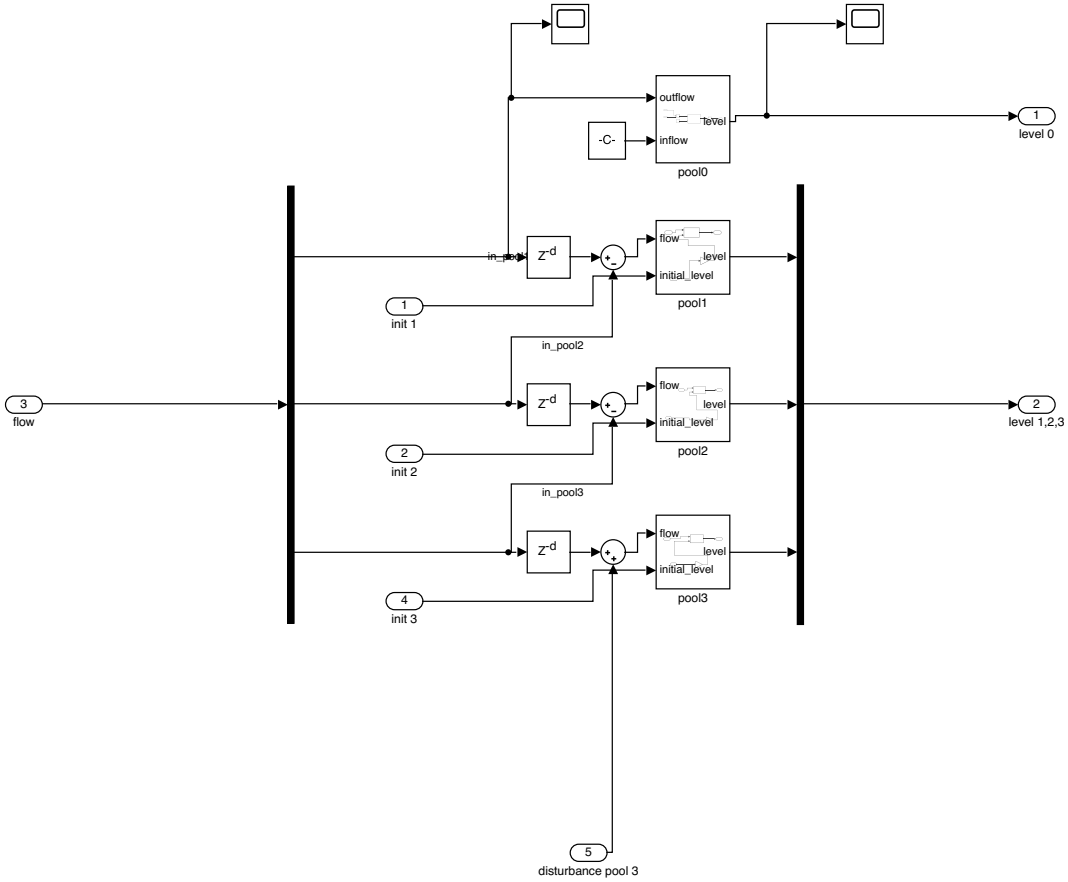


Figure E-4: Simulink model for the pools.

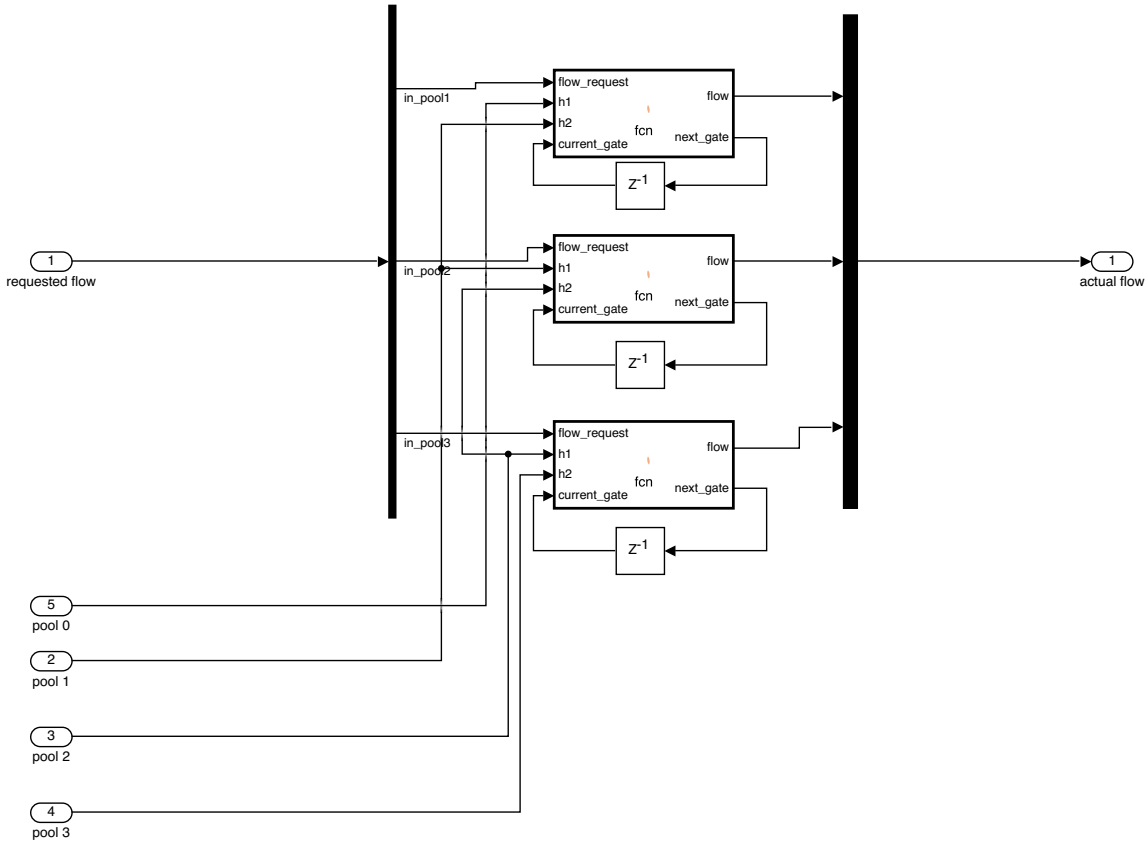


Figure E-5: Simulink model for the undershot gates.

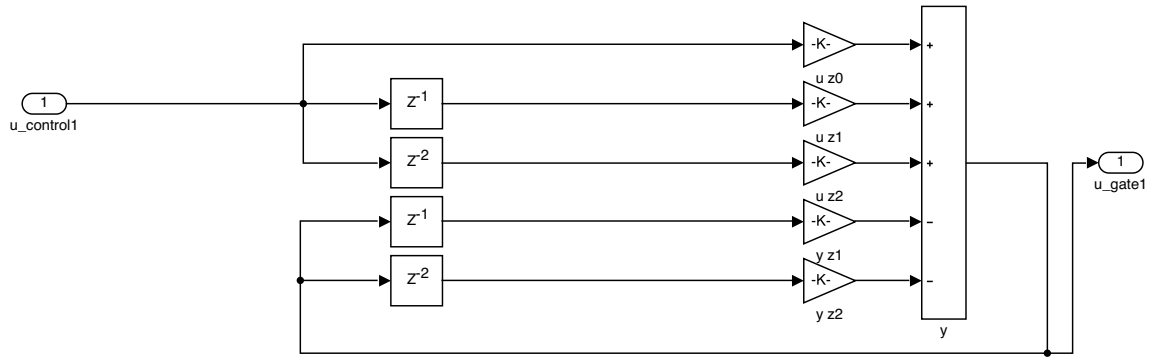


Figure E-6: Simulink model of the local controller implementation on the Fireflies.

E-2 Hardware-in-the-loop

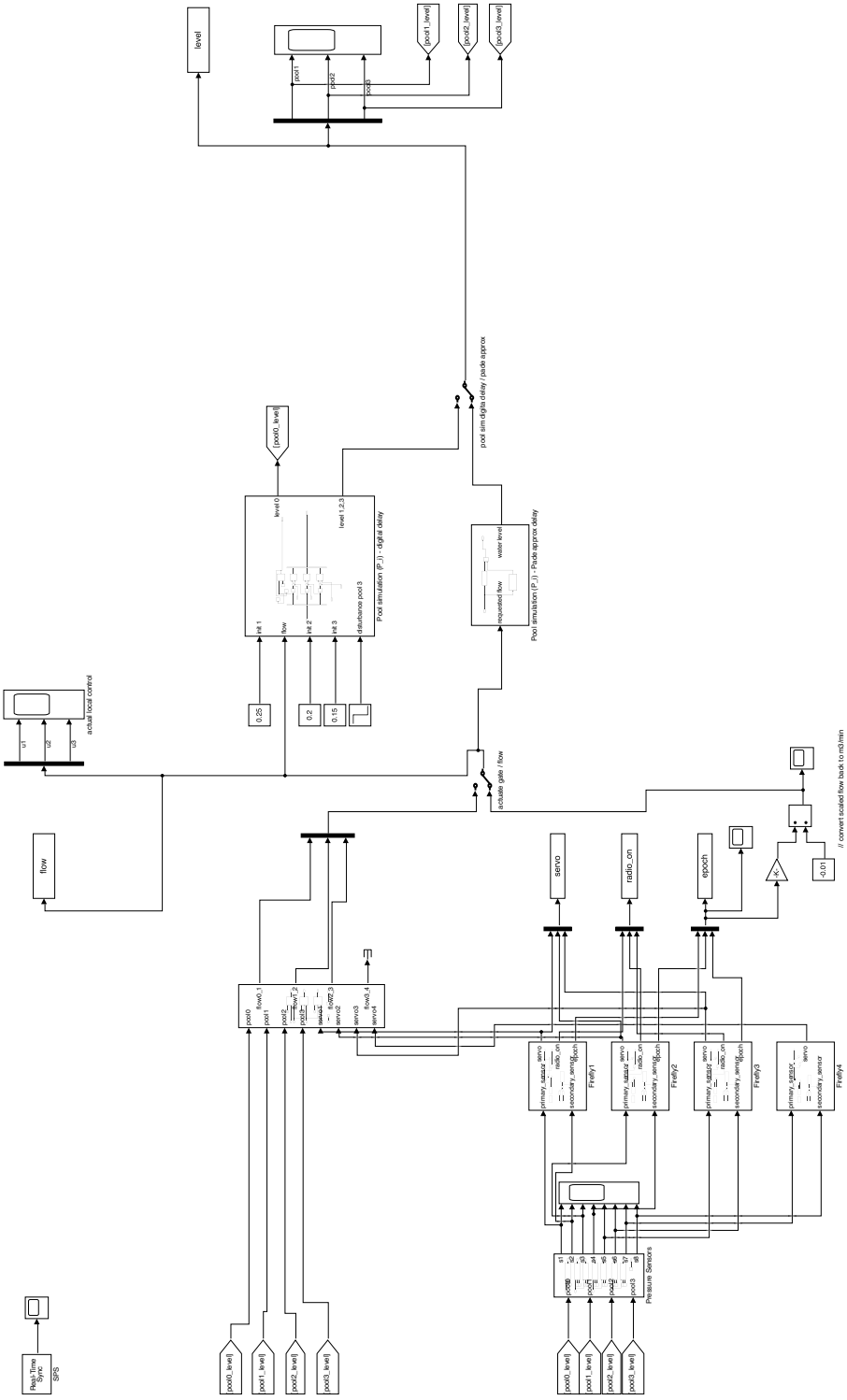


Figure E-7: Simulink model for the hardware-in-the-loop simulation of the water testbed unit.

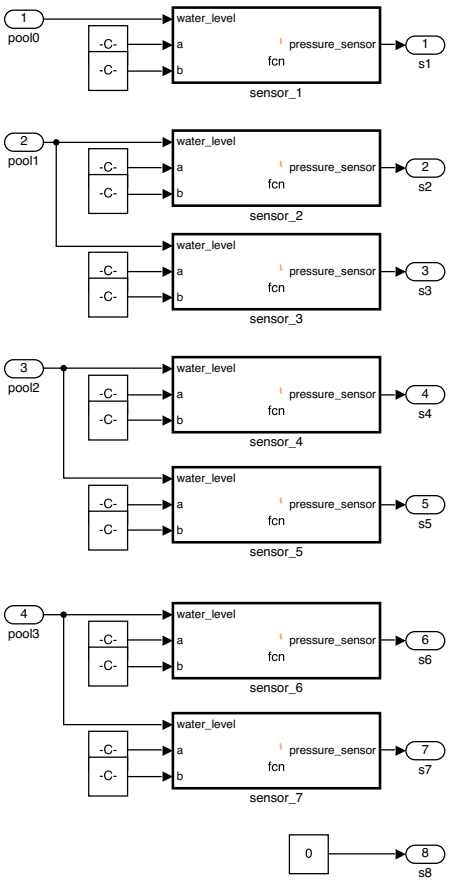


Figure E-8: Simulink model to convert water levels to pressure values.

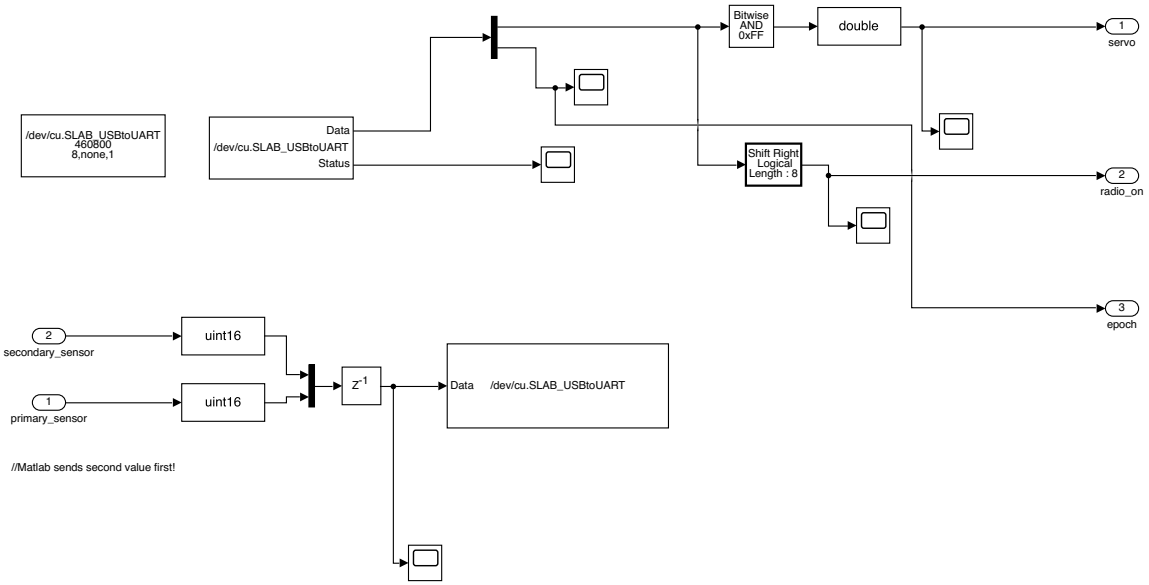


Figure E-9: Simulink model to do the serial communication with a Firefly.

Bibliography

[1] Contiki-os/contiki wiki on github. https://github.com/contiki-os/contiki/wiki. (Accessed on 11/07/2020).

[2] Contiki: The open source operating system for the internet of things. http://www.contiki-os.org/. (Accessed on 09/01/2020).

[3] Hitec hs-725bb - 3.5 turn sail drum servo. https://servodatabase.com/servo/hitec/hs-725bb. (Accessed on 04/28/2021).

[4] Knmi - daggegevens van het weer in nederland. https://www.knmi.nl/nederland-nu/klimatologie/daggegevens. (Accessed on 07/19/2021).

[5] System identification toolbox - matlab. https://www.mathworks.com/products/sysid.html. (Accessed on 05/04/2021).

[6] Ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area network–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 5: Preassociation discovery. *IEEE Std 802.11aq-2018 (Amendment to IEEE Std 802.11-2016 as amended by IEEE Std 802.11ai-2016, IEEE Std 802.11ah-2016, IEEE Std 802.11aj-2018, and IEEE Std 802.11ak-2018)*, pages 1–69, 2018.

[7] José Araújo, Adolfo Anta, Manuel Mazo, João Faria, Aitor Hernandez, Paulo Tabuada, and Karl H. Johansson. Self-triggered control over wireless sensor and actuator networks. In *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*, pages 1–9, 2011.

[8] M. Bacic. On hardware-in-the-loop simulation. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 3194–3198, 2005.

[9] D. P. Borgers and W. P. M. H. Heemels. Event-separation properties of event-triggered control systems. *IEEE Transactions on Automatic Control*, 59(10):2644–2656, 2014.

[10] Michael Cantoni, Erik Weyer, Yuping Li, Su Ki Ooi, Iven Mareels, and Matthew Ryan. Control of large-scale irrigation networks. *Proceedings of the IEEE*, 95(1):75–91, 2007.

[11] Gabriel de Albuquerque Gleizer and Manuel Mazo. Self-triggered output-feedback control of lti systems subject to disturbances and noise. *Automatica*, 120:109129, 2020.

[12] M.C.F. Donkers and W.P.M.H. Heemels. Output-based event-triggered control with guaranteed $\mathcal{L}\infty$-gain and improved event-triggering. In *49th IEEE Conference on Decision and Control (CDC)*, pages 3246–3251, 2010.

[13] Federico Ferrari, Marco Zimmerling, Luca Mottola, and Lothar Thiele. Low-power wireless bus. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, pages 1–14, New York, NY, USA, 2012. Association for Computing Machinery.

[14] Federico Ferrari, Marco Zimmerling, Lothar Thiele, and Olga Saukh. Efficient network flooding and time synchronization with glossy. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 73–84, 2011.

[15] Gene F. Franklin, J. Da Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*. Prentice Hall Press, USA, 7th edition, 2014.

[16] W. P. M. H. Heemels, M. C. F. Donkers, and Andrew R. Teel. Periodic event-triggered control for linear systems. *IEEE Transactions on Automatic Control*, 58(4):847–861, 2013.

[17] W.P.M.H. Heemels, K.H. Johansson, and P. Tabuada. An introduction to event-triggered and self-triggered control. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 3270–3285, 2012.

[18] Joo P. Hespanha, Payam Naghshtabrizi, and Yonggang Xu. A survey of recent results in networked control systems. *Proceedings of the IEEE*, 95(1):138–162, 2007.

[19] Timofei Istomin, Amy L. Murphy, Gian Pietro Picco, and Usman Raza. Data prediction + synchronous transmissions = ultra-low power wireless sensor networks. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, SenSys '16, pages 83–95, New York, NY, USA, 2016. Association for Computing Machinery.

[20] Keller. *Highly Precise (0,01%) Pressure Transmitters*, 6 2020.

[21] F. Kozin and H.G. Natke. System identification techniques. *Structural Safety*, 3(3):269–316, 1986.

[22] K. Leentvaar and J. Flint. The capture effect in fm receivers. *IEEE Transactions on Communications*, 24(5):531–539, 1976.

[23] Yuping Li and Michael Cantoni. Distributed controller design for open water channels. *IFAC Proceedings Volumes*, 41(2):10033–10038, 2008. 17th IFAC World Congress.

[24] Jacob Jan Lont. Wireless event-triggered control for water irrigation systems, 2020.

[25] Manuel Mazo, Adolfo Anta, and Paulo Tabuada. An iss self-triggered implementation of linear controllers. *Automatica*, 46(8):1310–1314, 2010.

[26] Luis Santos Pereira, Theib Oweis, and Abdelaziz Zairi. Irrigation management under water scarcity. *Agricultural Water Management*, 57(3):175–206, 2002.

[27] Karl J. Åström and Björn Wittenmark. *Computer-Controlled Systems (3rd Ed.)*. Prentice-Hall, Inc., USA, 1997.

[28] J. M. Schram. Handleiding watermodel, 2001.

[29] Texas Instruments. *ADS111x Ultra-Small, Low-Power, I2C-Compatible, 860-SPS, 16-Bit ADCs With Internal Reference, Oscillator, and Programmable Comparator*, 1 2018.

[30] Matteo Trobinger, Gabriel de Albuquerque Gleizer, Timofei Istomin, Manuel Mazo Jr. au2, Amy L. Murphy, and Gian Pietro Picco. The wireless control bus: Enabling efficient multi-hop event-triggered control with concurrent transmissions, 2021.

[31] Manel Velasco, Josep Fuertes, and Pau Marti. The self triggered task model for real-time control systems. In *Work-in-Progress Session of the 24th IEEE Real-Time Systems Symposium (RTSS03)*, volume 384, 2003.

[32] Erik Weyer. System identification of an open water channel. *IFAC Proceedings Volumes*, 33(15):265–270, 2000. 12th IFAC Symposium on System Identification (SYSID 2000), Santa Barbara, CA, USA, 21-23 June 2000.

[33] Jerry Zhao and Ramesh Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, page 1âĂŞ13, New York, NY, USA, 2003. Association for Computing Machinery.

[34] Zolertia. *Zolertia Firefly Revision A2 Internet of Things hardware development platform, for 2.4-GHz and 863-950MHz IEEE 802.15.4, 6LoWPAN and ZigBee®Applications*, 2017.

Glossary

List of Acronyms

AD	analog to digital
CETC	continuous event-triggered control
D3S	dynamic distributed decentralized systems group
DCSC	Delft center for systems and control
ESD	electrostatic discharge
ETC	event-triggered control
FF	firefly
GPIO	general purpose input/output
HIL	hardware-in-the-loop
I^2C	inter-integrated circuit
IEEE	institute of electrical and electronics engineers
LTI	linear time-invariant
LWB	low power wireless bus
MAC	media access control
MSE	mean squared error
MTSE	mean timed squared error
NTX	number of retransmissions
OS	operating system
PCB	printed circuit board
PETC	periodic event-triggered control
PSTC	Preventive self-triggered control
PWM	pulse width modulation
SENTIENT	scheduling of event-triggered control tasks research group
SPS	samples per second

STC	self-triggered control
UART	universal asynchronous receiver-transmitter
USB	universal serial bus
WCB	wireless control bus
WCN	wireless control network
WIS	water irrigation system
ZOH	zero order hold