KTH Electrical Engineering

Wireless Water Tanks

Different scenarios and controllers

AITOR HERNÁNDEZ, JOAO FARIA AND JOSE ARAUJO

Stockholm July 20, 2011

TRITA-EE 2011:XXX
Version: 0.1

Contents

Acronyms

BO	Beacon Order. 17
CAP	Contention Access Period. 13, 17, 20
CFP	Contention-Free Period. 14, 20
CSMA-CA	Carrier Sense Multiple Access with Collision Avoidance. 9
GTS	Guaranteed Time Slot. 3, 14, 17
GUI	Graphical User Interface. 14–18, 21, 22
PAN	Personal Area Network. 15, 16
PCB	Printed Circuit Board. 4, 5
TDMA	Time Division Multiple Access. 20
UPM	Universal Power Module. 4

Chapter 1

Overview

1.1 Introduction

In the Automatic Control department a Coupled Water Tanks is extensible used for academic purposes, either in a Basic Control course or in the Advance Control course. But all the experiments are done using a wired topology. The Coupled Water Tanks is an excellent process to apply and try Wireless Sensor Actuator Networks due to the dynamics of the process. Hence, in this document, we present the different scenarios and controllers implemented.

For information regarding the modelling, and simple control laws we recommend the following references from Quanser [11, 12, 13]). This document is focused on the implementation and communication between the motes, not on the control perspective.

To implement the Wireless Sensor Actuator Network we use the motes from TmoteSky [4], Telosb [14] or MAXFOR [2] using the TinyOS [16] operating system. For the wireless communications we used the IEEE 802.15.4 standard protocol [10] for low power communications implemented by TU Berlin, called TKN15.4 [5]. For some experiments we extend the IEEE 802.15.4 implementation available to include the Guaranteed Time Slot (GTS) implementation. [9].

The code and software necessary to run the experiments explained below is available on the following URL:

http://code.google.com/p/kth-wsn/source/browse/trunk/kth-wsn/apps.water-tank/.

To compile the applications we recommend the use of the latest version of TinyOS that you could find on the following URL:

http://code.google.com/p/tinyos-main/

The different scenarios that were are going to present are based on a Centralized Controller topology where all the sensor data transmit from the sensor to the controller, and the controller sends the control voltage to the actuator node. Even though all the topologies are based on a Centralized Controller there are different controller approaches implemented: a simple controller implemented in a mote, self triggered control law implemented in the computer, or dynamic scheduling with sensor selection.

1.2 Setup

In this section we show how we need to setup the water tanks, Universal Power Module (UPM) and the new board where we need to plug our motes.

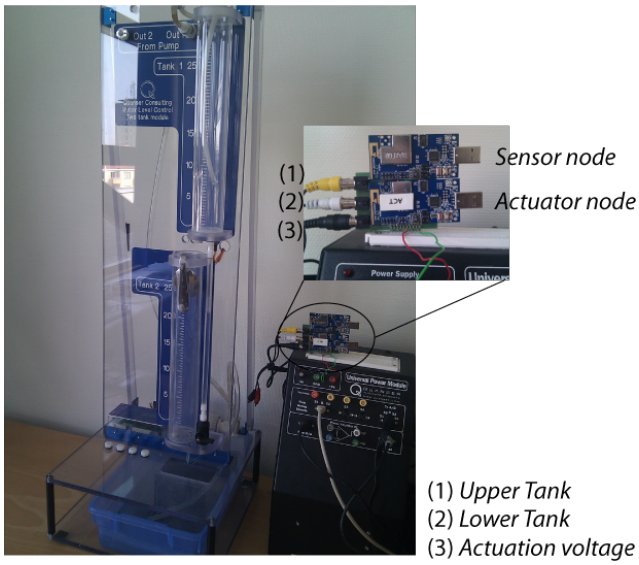

Figure 1.1: Water tank with the UPM and the PCB board connected

Figure 1.1 show a Coupled water tank with the UPM and a Printed Circuit Board (PCB) connected to it. The cables are connected following Quanser configuration specification in [13, p. 12 - 14]. However in our case instead of connect the sensor values to the Q8 Terminal Board, we connect them directly to our PCB board.

The PCB has connected three RCA cables: white cable is for the Upper tank level, yellow cable for the lower tank level and the black cable is for the actuator voltage. Moreover the PCB needs to be supplied by +12 V. For this reason, we connect the Ground of the UPM to the first PIN of the socket (starting from the right) and the 12 V (+Vs) to the PIN number 5.

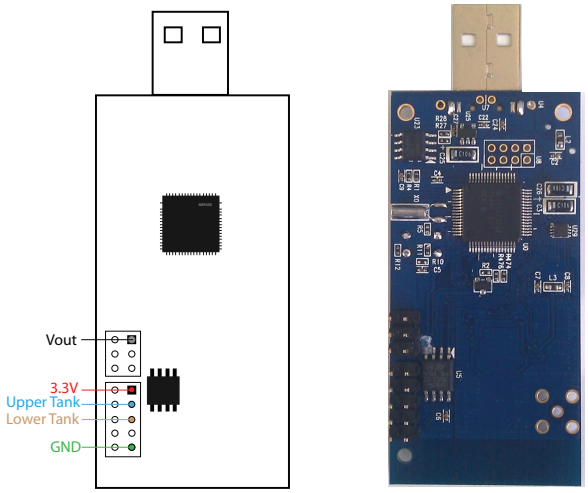

Figure 1.2: Motes connections. View from the bottom

Figure 1.2 shows the connection that the motes have with the PCB board. The PIN V_{out} is exclusive for the actuator mote.

Chapter 2

Scenarios

All the scenarios are based on a Centralized Controller were the sensor data is transmitted from the sensor to the controller (or base station), and from the controller to the actuator node.

2.1 Centralized Controller. General approach

This application consists on having a Centralized Controller for the water tanks using the IEEE 802.15.4 implementation from TU Berlin (TKN15.4) [5].

In the Figure 2.1 we show the scenario with three water tanks and the controller. Table 2.1 shows the different programs that are needed to deploy this scenario.

Application	Description
`Controller`	centralized controller
`SensorApp`	sensor connected to the PCB board in the water tank
`ActuatorApp`	actuator connected to the PCB board in the water tank

Table 2.1: Centralized Controller General applications

The `Controller` shall manage the N water tanks configured in the network, and set the upper tanks levels for a given a certain sequence (See 2.1.3). The `SensorApp` samples the water tank levels with a certain period (`DEFAULT_RATE`) and sends the values to the `Controller` using the IEEE 802.15.4 standard protocol. Once the `Controller` receives the packet with the sensor value, it computes the control voltage for the specific water tank and sends the packet back to the `ActuatorApp`.

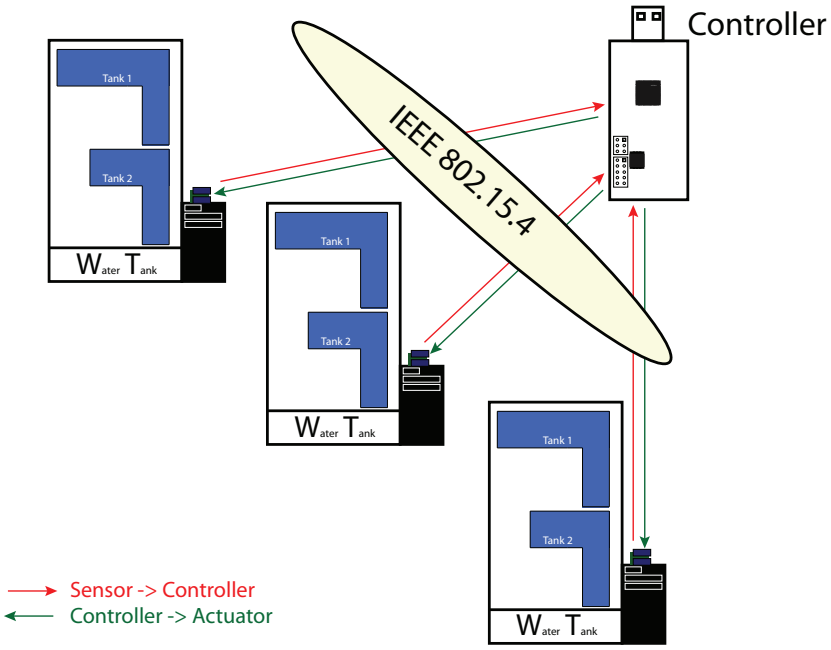

Figure 2.1: Scenario with a mote as a centralized controller and three water tanks

2.1.1 Requirements

Below we show a list of requirements that you need to run this example.

- N Quanser Coupled Water Tanks
- N PCB board to connect the motes with the water tanks
- $2 * N + 1$ Telosb or TmoteSky
- TinyOS properly installed [16]
- Source code for the motes `http://code.google.com/p/kth-wsn/source/browse/trunk/kth-wsn/apps.water-tank/CentralizedControllerGeneral`
- Matlab/Simulink (Optional) [6]
- Serial-Forwarder from TinyOS (Optional) [6]

2.1.2 Communication

In this scenario the we do not use any complex communication protocol, compared to the ones we see in the next sections. For this case, we use IEEE 802.15.4 standard protocol configured in the nonbeacon-enable mode.

In the nonbeacon-enabled mode, there is no coordinator managing the network or synchronizing the mote. To transmit they use the unslotted Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA).

2.1.3 Installation

We omit the installation methods assuming that the reader already knows how to program the motes. If not, please before continue with this document read the following references [15, 7]

Number of water tanks

The maximum number of water tanks is given by the constant `NUMBER_WT` founded in the `app_sensors_nbe.h`. This variable allocates memory to store the integrals of each node.

The controller assumes that each water tank is defined with the following IDs:

Sensor	$2 * WT\_ID - 1$
Acutator	$2 * WT\_ID$

An example for three water tanks would be:

Controller		`ID = 0`
Water Tank 1	Sensor	`ID = 1`
	Actuator	`ID = 2`
Water Tank 2	Sensor	`ID = 3`
	Actuator	`ID = 4`
Water Tank 3	Sensor	`ID = 5`
	Actuator	`ID = 6`

Sampling time

The sampling time is given by the constant `DEFAULT_RATE` defined in the header file `app_sensors_nbe.h`.

References levels

The references are given by the array `x_ref_vector` declared in the `ControllerC.nc` file. By default the sequence reference is set to: `5, 10, 15, 20, 15, 10, 2`.

Update interval

The references are changing every certain number of samples. This is given by the product of `DEFAULT_RATE * UPDATE_INTERVAL`.

Other configuration parameters could be found in `app_sensor_nbe.h`.

2.1.4 Options

Calibration mode

To run this experiments is mandatory to do a manual calibration of the water tanks in order to receive correct values from the sensor. For this purpose we have a routine that makes the calibration easier. First, you need to compile the `Controller` with the following precompiler directive:

```
CFLAGS += -DWT_CALIBRATION_PROCESS
```

You need the PrinfClient to read the printf messages from the serial port:

```
java net.tinyos.tools.PrintfClient -comm serial@/dev/ttyUSBXX:tmote
```

This directive will give you some instruction in the terminal output, as we see below.

1. Turn on the PrintfClient to see the printf's
2. Press the "UserButton" to start the calibration method. The terminal shall show the values for all the water tanks when it receives a message.
3. Disconnect the motor (disconnect the "From Load" cable)
4. Set both tanks to level $\sim 0cm$ by adjusting the `OFFSET` potentiometer
5. Fill the tank 1 (upper tank) to 25cm
6. Set tank 1 to level $\sim 25cm$ by adjusting the `GAIN` potentiometer
7. Transfer the water to tank 2
8. Set tank 2 to level $\sim 25cm$ by adjusting the `GAIN` potentiometer

User button

In some scenarios, it is useful to be able to control when we want to start the experiments, for this reason we have created this option. The `Controller` waits until the user has pressed the "UserButton", to send back to the `ActuatorApp` the voltage values.

To enable it, you should set the flag `CFLAGS += -DWT_USERBUTTON` in the `Makefile`. This precompiler directive provides the control of the water tanks through the "User-Button".

2.1.5 Results

For this example there are different options to show the results. In this case we only show two examples that are currently implemented in the programs. These methods are not useful for plotting purposes.

We recommend to read the document [6], there are different methods to obtain data from the network and/or send information to the motes.

From Controller

The controller flushes the data from all the sensors every time it receives a packet. This information is only for debugging purposes. But with few modifications is possible to have a column based output which easily could be easily imported in Matlab.

Below we have the output of the terminal that shows this:

```
[WT1−331] x_ref= 5.000 cm ; x1= 14.394 cm ; x2= 18.719 cm ; xi= 160.167 cm ; out= 0.005V
[WT1 − 332] x_ref= 5.000 cm ; x1= 14.497 cm ; x2= 18.925 cm ; xi= 162.067 cm ; out= 0.005V
[WT1 − 333] x_ref= 5.000 cm ; x1= 14.509 cm ; x2= 18.891 cm ; xi= 163.969 cm ; out= 0.005V
[WT1 − 334] x_ref= 5.000 cm ; x1= 14.509 cm ; x2= 18.891 cm ; xi= 165.870 cm ; out= 0.005V
[WT1 − 335] x_ref= 5.000 cm ; x1= 14.532 cm ; x2= 19.028 cm ; xi= 167.777 cm ; out= 0.005V
[WT1 − 336] x_ref= 5.000 cm ; x1= 14.509 cm ; x2= 18.891 cm ; xi= 169.679 cm ; out= 0.005V
[WT1 − 337] x_ref= 5.000 cm ; x1= 14.509 cm ; x2= 18.891 cm ; xi= 171.581 cm ; out= 0.005V
```

From Sensor

There is a Simulink model to read the values from one of the sensors placed in the `MatlabSimulink/` folder. The Simulink model listens the TCP/IP port 9002 from the localhost server, where we should have the `Serial-Forwarder` running [6]. Once we finish the experiments, we shall have the results in the workspace as two variables {`wt`, `pck`}, where `wt` contains the data of the water tank that we have connected to the computer, and the `pck` contains the information of packet success and packet total.

To plot, the results we have some simple functions: `plotResults` and `plotResultsAll`. Figure 2.2 is an example of the plots where we have two water tanks scenario, and we are monitoring one water tank with and without interferences.

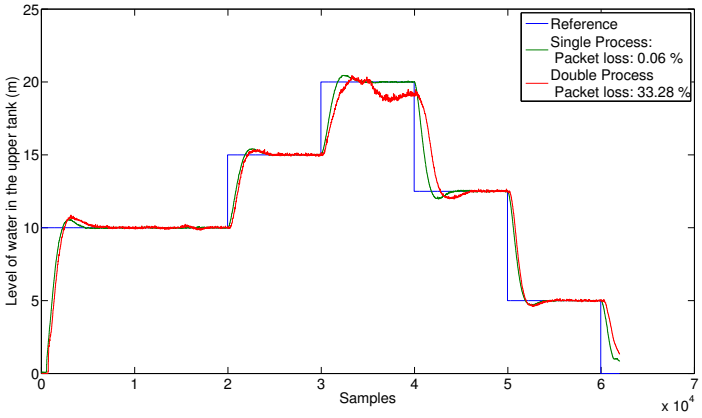

Figure 2.2: Reference tracking in a two water tank scenario. Data monitoring of one water tank with and without losses

2.2 Self-Triggered Controller

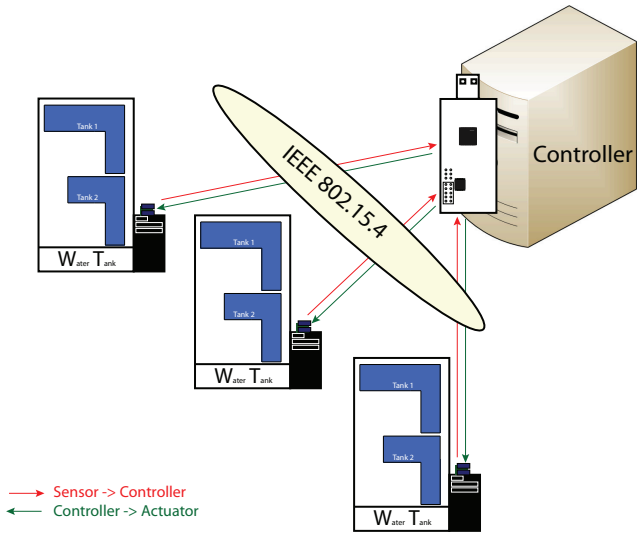

Figure 2.3: Scenario with a mote connected to the computer and working as a bridge between the controller and scheduler in the PC and the three water tanks

Figure 2.3 shows the scenario with three water tanks and the controller. In this case, we have a mote connected to the computer that acts as a bridge between the net of water tanks and the PC. The PC is the controller and scheduler. In the next sections we explain it with more detail.

This approach was presented in the 7th IEEE International Conference on Distributed Computing in Sensor Systems by Jose Araujo with the paper *Self-Triggered*

Control over Wireless Sensor and Actuator Networks [3].

2.2.1 Requirements

First of all, it is mandatory that at this point, you have read the following documents:

- *Getting started with TinyOS at the Automatic Control Lab* [7]
- *Communication between PC and motes in TinyOS* [6]

Below we show a list of requirements that you need to run this example.

- N Quanser Coupled Water Tanks
- N PCB board to connect the motes with the water tanks
- $2 * N + 1$ Telosb, TmoteSky or MAXFOR motes
- TinyOS properly installed [16]
- Source code for the GTS implementation [9] http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x-contrib/kth/tkn154-gts
- Source code for the modified version of the GTS implementation [8] http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x-contrib/kth/tkn154-gts-mod
- Source code for the motes http://code.google.com/p/kth-wsn/source/browse/trunk/kth-wsn/apps.water-tank/SelfTriggered
- Scripts to program the motes http://code.google.com/p/kth-wsn/source/browse/trunk/kth-wsn/apps.water-tank/SelfTriggered
- Matlab with TinyOS Java libraries compiled [6]
- Serial-Forwarder from TinyOS, C version. [6]

2.2.2 Communication

For this case, we use IEEE 802.15.4 standard protocol configured in the beacon-enable mode where the mote connected to the computer acts as the coordinator of the network.

In the beacon-enabled mode, the coordinator sends beacons periodically and the motes are synchronized between each other. With this mode, we have two ways

of transmission, we could transmit during the Contention Access Period (CAP) or Contention-Free Period (CFP). The approach that we have been presenting in [3] is focused on the CFP, where every mote transmits and receives within a certain slot assigned by the scheduler implemented in the computer.

In the IEEE 802.15.4 standard protocol the slots in the CFP that we could allocate is limited to 7 GTS. So, we have extended the protocol to be able to use more slots. The controller will always allocate a fix number of slots. Figure 2.4 shows the superframe structure when the Self-Triggered Controller is running. More details about the GTS modification could be found at [8].

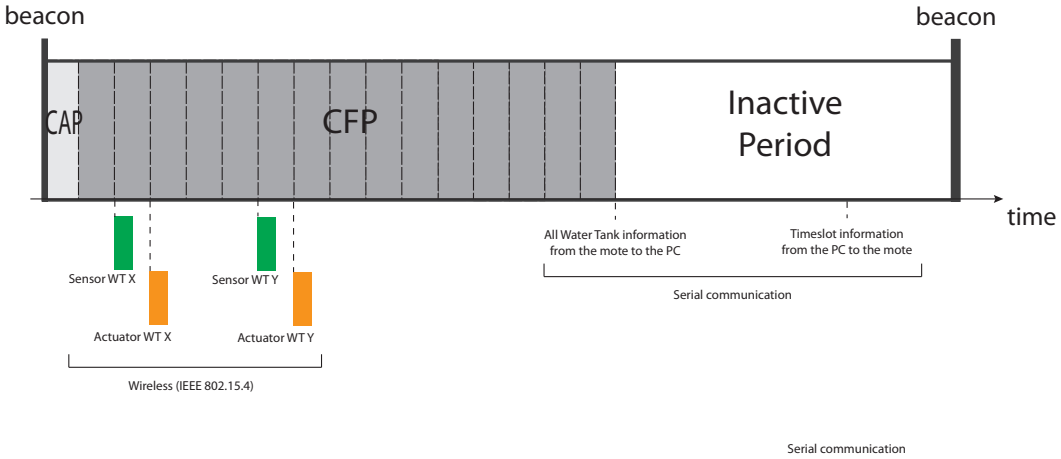

Figure 2.4: Superframe structure with $aNumSuperframeSlots - 1$ GTS slots

2.2.3 Installation

Once we have all the requirements for this setup, we need to configure the scripts and tools. The scripts and tools are optionals, but once you know how they work and how to run them, they simplify the compilation methods. The configuration involves the Scripts and the Graphical User Interface.

Scripts

To have an automatic process for configuration and logging data, we use the following PHP/Bash scripts *i*) `moteallClass` *ii*) `motelistSelfTriggered.php` *iii*) `motereinstall` *iv*) `setSelfTriggeredVariables`. Below we describe the scripts that we need to modify in other to adapt them for our requirements and motes.

- `motelistSelfTriggered.php` This script will read the data from the `motelist`, compile the apps and program the motes that are in the *sensorList*, *actua-*

torList, *snifferList*, *dummyList* and *baseStationList* with their applications. With this, we only will need to modify the list cited before and the path where the applications are, *projectFolder*.

Graphical User Interface

We have developed a Graphical User Interface (GUI) to make the installation and the configuration of the scenarios easy. In the document [6]. we describe how to setup the computer to be able to use the GUI.

2.2.4 Usage

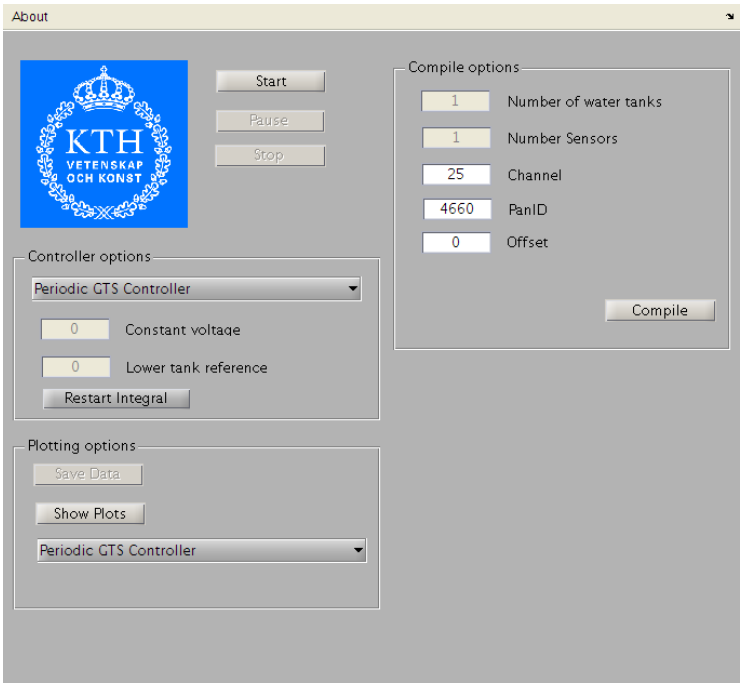

Figure 2.5: Screenshot of the GUI for the Water Tanks

Figure 2.5 shows an screenshot of how the GUI looks like. In the upper right corner, we have the *Compile options* panel where we can configure the desired scenario modifying the channel, Personal Area Network (PAN) identifier and offset. In the middle, we have the *Controller options* panel where we can configure the controller that we want to use. In the bottom left corner, we have the *Plotting options* panel, useful to show the logged data that we store during the experiment

Figure 2.6 show the basic example, with all the step that we need to do, in order to compile/program and run the experiment logging the data. First of all, we need

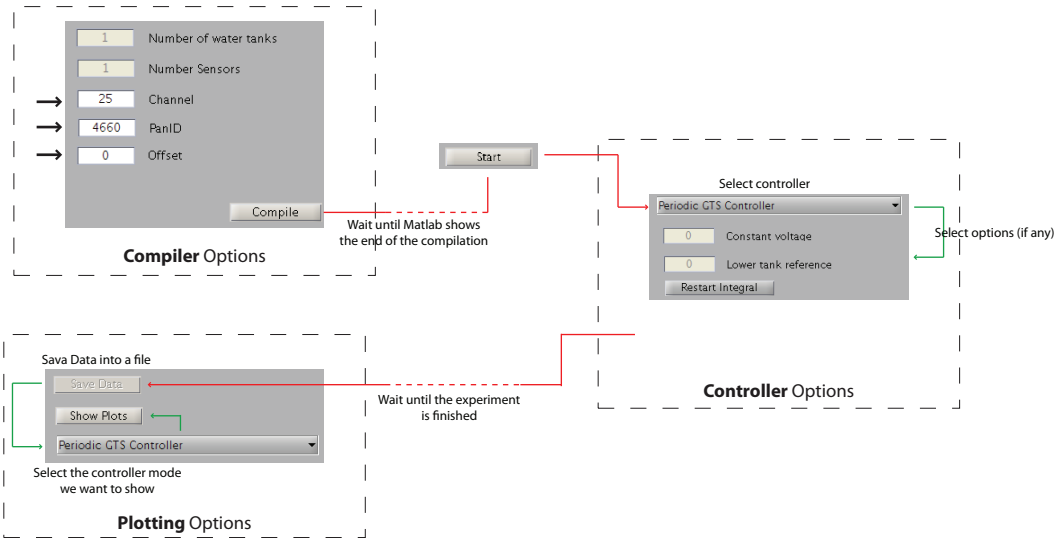

Figure 2.6: Flow diagram with the necessary steps to run the experiments

to configure the network with the channel and PAN identifier that we want. Then, we can compile and program the motes by pressing the *Compile* button. It will call the PHP script in order to update the list of motes. Once, the process finishes, we can press the *Start* button to run the experiment. At this point, the controller could be selected. Finally, to store the data we need to press the *Save Data* button. It creates a file with the current controller selected in the *Controller options* panel. To see, the last data stored for a certain controller, we select it in the pop-up menu in the *Plotting options* panel, and press *Show Plots*

In the following subsections we describe the panels more carefully to understand how it works and why some parameters are not available.

Compile options There are some parameters that we could modify from the GUI regarding the compilation. These are *Channel*, *PanID* and *Offset*.

- **Number of water tanks and sensors** This options is disabled by default. If we change the values the header file is modified and the motes will be programmed with the new configuration. But, for this experiment, we need to Java messages and they depend on the number of water tanks. Moreover the `javaclasspath` in Matlab must be reloaded to use the new Java classes.
- **Channel** We change the radio channel.
- **PanId** If we change the channel it is not necessary to change the PAN identifier because both networks do not collide, but it is useful to detect them

easily.

- **Offset** This field refers to the order that we have in the `motelistSelfTriggered.php`. By default we will program all the motes that we have connected to the PC.

Controller options Although the GUI has been designed specifically for the Self-Triggered controller, in order to test it and debug, the controller, modelling, manual controller, we have different controllers. Once we press the *Start* button, we can change the controller by selecting one option of the pop-up menu.

Periodic GTS Controller This is a periodic controller. It sets a fixed GTS slot for each node (sensor and actuator), and then they are able to control the water tanks with a sampling period, depending on the Beacon Order (BO).

Self-Triggered Controller Based on the model of the system and the current state, the scheduler assigns slots to the Water tanks and/or dummy sensors. The scheduler indicates who has to transmit in the currect superframe.

- Modelling Tanks - This option provides an automatic modelling method to get the parameters k, β and τ.

Constant controller It just sends a constant voltage to the motors. It is useful to turn them off quickly. The voltage value is set in the white box called *Voltage reference*.

Track reference Not implemented.

Null entry This entry is the same as the *Reset integral* command.

Event-Triggered Controller With this controller, is the mote itself who decides when he needs to send a packet to the coordinator, to update the controller. For this case, we completely remove the CAP period, in order to have all the superframe with GTS slots. The slots of one water tank have a separation of ΔD_{max} between each other, doing this we could afford a periodic sampling of ΔD_{max} using GTS slots.

This controller requires other modifications. To enable this controller, we need to compile the mote with $BO = SO$, in order to remove the inactive period.

Plotting options With this GUI, we log all the data from the Base Station with includes all the water tanks levels, voltage in the motors, integrals,.... On the other hand, we are also logging the communications performance if the sniffer mote is connected to the computer.

By pressing the button *Save data*, the logged data is saved in a file with a path `mat/data_modeX_TTtanks_YYmmdd_HHMMSS` where X is the mode that corresponds

with the controller used at the moment that we press the button *Save data*. Remark that to save the file we used the **controller** instead of the pop-up in the *Plotting panel*. Also, when we press the *Start* button we start logging the data from the sniffer mote, which will give us information about the communication.

For showing the results, first we need to set the mode that we want to show by changing the pop-up in the *Plotting panel*. When we press the *Show plots* button it uses the latest file found for this mode and it shows the results.

Figure 2.7 shows a diagram with the steps explained above.

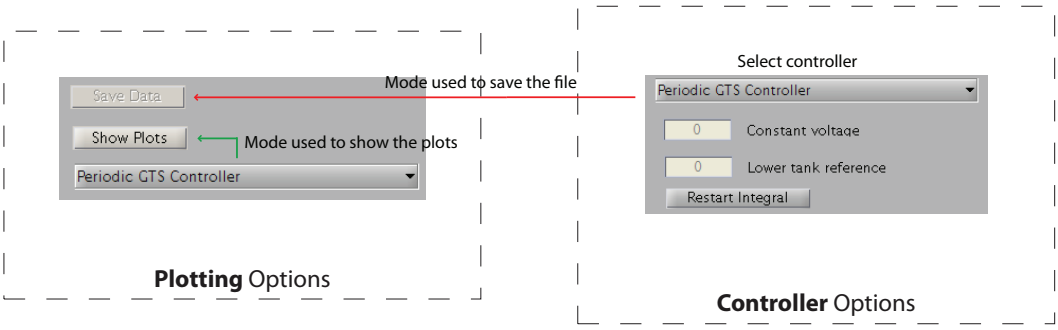

Figure 2.7: Flow diagram for the plotting functions

2.2.5 Libraries

The purpose of this document is not to have a extensive explanation about the code implemented and libraries. However, it is important to give an guidance where to start looking at. Below we have a table with the main files that are involved on this implementation. Other functions or toold could be used.

File	Description
`waterTanksSelfTriggered.m`	The files includes the core of the water tanks experiment. The GUI uses the function from this file.
`functions/readFromBaseStation.m`	This is the callback function for receiving a packet through the `sf`. The functionality changes depending on the selected controller
`functions/updateTimeslot.m`	We use this function when the Self-Triggered Controller is used. We assign the slots depending on the scheduler implemented
`gui/waterTanksSelfTriggered_gui.m`	The functionality of the GUI is implemented on this file.
`gui/waterTanksSelfTriggered_gui.fig`	It is easy to modify the GUI by using this figure. You could easily redistribute the panels and buttons.

2.2.6 Results

As we have seen in Section 2.2.3, to show the results we just need to click the *Show plots* button. It will show plots for the slots assigned, inter-event times for the Self-triggered controller, tanks levels, ... Figure 2.8 shows some examples.

2.3 Dynamic Wireless Sensor Scheduling

In the Figure 2.9 we show the scenario with three water tanks and the controller. In this case, we have a mote connected to the computer that acts as a bridge between the networks of water tanks and the PC. Again, the PC is the controller and scheduler but there are some differences between the Self-Triggered approach in Section 2.2 and the current one. In the next sections we explain with more detail what scheduler means in this scenario.

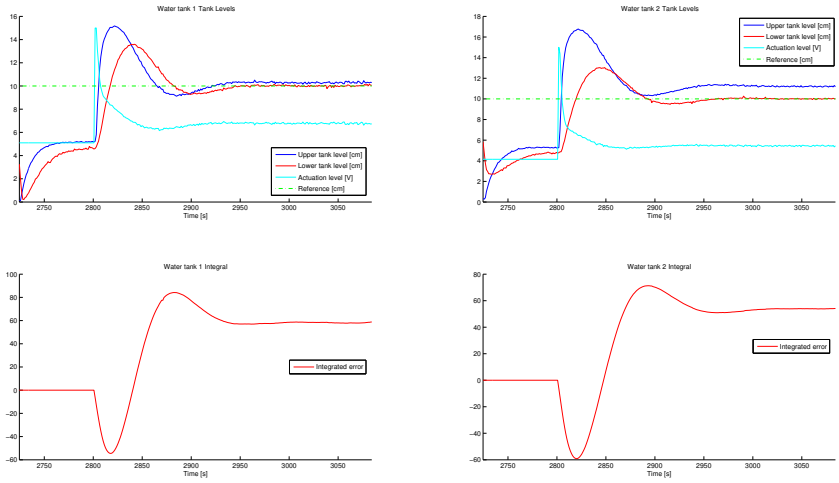

Figure 2.8: Example of the water tank levels and integrals for two water tanks

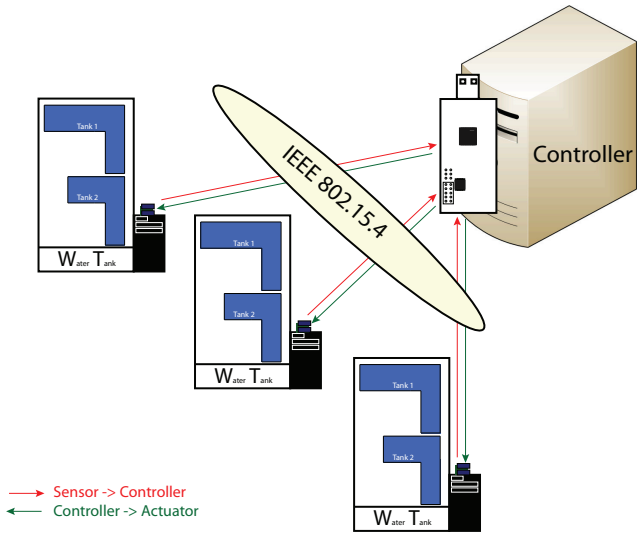

Figure 2.9: Scenario with a mote connected to the computer and working as a bridge between the controller and scheduler in the PC and the three water tanks

This approach is presented in the paper *Periodic Constraint-based Control using Dynamic Wireless Sensor Scheduling* submitted to IEEE Conference on Decision and Control (CDC) [1].

2.3.1 Requirements

Below we show a list of requirements that you need to run this example.

- N Quanser Coupled Water Tanks
- N PCB board to connect the motes with the water tanks
- $2 * N + 1$ Telosb or TmoteSky
- TinyOS properly installed [16]
- Source code for the motes http://code.google.com/p/kth-wsn/source/browse/trunk/kth-wsn/apps.water-tank/SlotsDistribution
- Matlab with TinyOS Java libraries compiled [6]
- CVX Matlab Toolbox http://cvxr.com/cvx/
- Serial-Forwarder from TinyOS, C version. [6]

2.3.2 Communication

In this case, we use IEEE 802.15.4 standard protocol configured in the beacon-enable mode where the mote connected to the computer acts as the coordinator of the network as well.

In the beacon-enabled mode, the coordinator send beacons periodically and the motes are synchronized between each other. With this mode, we have two ways of transmission, we could transmit during the CAP or the CFP. This approach is focused on the CAP, but implementing a virtual Time Division Multiple Access (TDMA) scheme inside the CAP.

A possible extension is the use of the modification presented in [8] which assures a perfect slots division and no interference between slots.

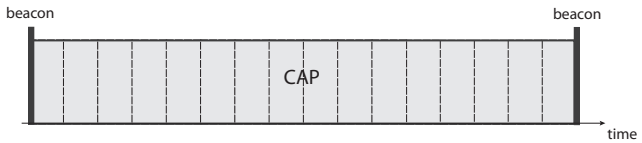

Figure 2.10: Superframe structure $IEEE154\_aNumSuperframeSlots$ slots

All the superframe is used for the CAP period, and we divide the superframe in $(NUMBER\_WT * 2 + 1)$ slots. With this structure we simulate the transmission in the CFP period, but without the reduction of energy that it provides.

2.3.3 Installation

Once we have all the requirements for this setup, we need to configure the scripts and tools. The scripts and tools are optionals, but once you know how they work and how to run them, they simplify the compilation methods. The configuration involves the Scripts and the Graphical User Interface.

Scripts

In this example, we do not have all the compilation/logging completely automatic. We only use one simple script that compiles and program the motes, based on the port ID and the desired ID for each mote. *i*) `motewatertanks` *ii*) `motereinstall`. Below we describe the scripts that we need to modify ourself in other to adapt them for our requirements and motes.

- `motewatertanks` This bash script compiles the applications and programs the motes that are in the *WT_SENSOR* array and *WT_ACTUATOR* array. So we need to setup the motes that we want for each application, and the folder where the applications are `FOLDER`. With this example, we assume that the Base Station is always the port and ID 0.

Graphical User Interface

We have developed a GUI to make the installation and the configuration of the scenarios easier. In the document [6]. we describe how to configure the setup the computer to be able to use this GUI.

Figure 2.11 shows an screenshot of how the GUI looks like. In the upper right corner, we have the *controller options* panel where we can configure the controller that we want to use. In the bottom, we have the *Plots* panel, useful to show the log data, that we store during the experiment.

2.3.4 Libraries

The purpose of this document is not to have a extensive explanation about the code implemented and libraries. However, it is important to give an guidance where to start looking at. Table **??** shows the main files that are involved on this implementation. Other functions or toold could be used.

File	Description
`Kvalue_watertanks_JIM.mat`	The LQR Controller matrix is saved in this mat file and loaded at the beginning of the application.
`final_files`	This folder contains all the Dynamic Scheduler controller matrices. We have the code implemented for the controller using the dynamic scheduler.
`apps/waterTankApp`	The files includes the core of the water tanks experiment. The GUI uses the function from this file. All the controller initialization are in the `init` function. Most of the paths are absolute, so before running this experiment check that all the paths are set for your computer.
`apps/computeControl.m`	We have the controllers algorithms implemented: { *null*, *constant*, *LQR*, *Dynamic scheduler*, *calibration* }
`gui/waterTankApp_gui.m`	The functionality of the GUI is implemented on this file.
`gui/waterTankApp_gui.fig`	It is easy to modify the GUI by using this figure. You could easily redistribute the panels and buttons.

Table 2.2: Main files involved in the Dynamic Scheduler implementation

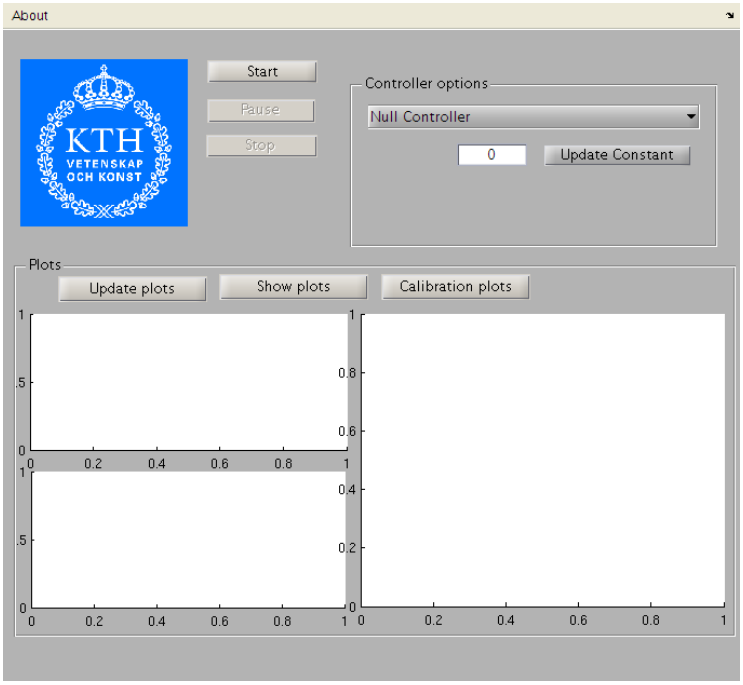

Figure 2.11: Screenshot of the GUI for the Water Tanks

2.3.5 Options

As we have seen there are different options that we could configure to fit our conditions ans scenario:

NUMBER_WT The number of water tanks will modify the number of slots allocated in the superframe. It is important to know that the number of slots is limited. We need to have enough time during the slot to transmit the packet.

UPDATE_INTERVAL This value indicates the periodicity of the samples. The sensors may sample their values every $UPDATE\_INTERVAL$ superframe. The controller manages this value and sends the counter in the superframe payload.

2.3.6 Results

In this scenario there are two ways to show the results. One of them is in the GUI with the small figure, and another is to show a complete analysis of the controller and scheduler with external plots. Figure 2.12 shows the different functions that each button does.

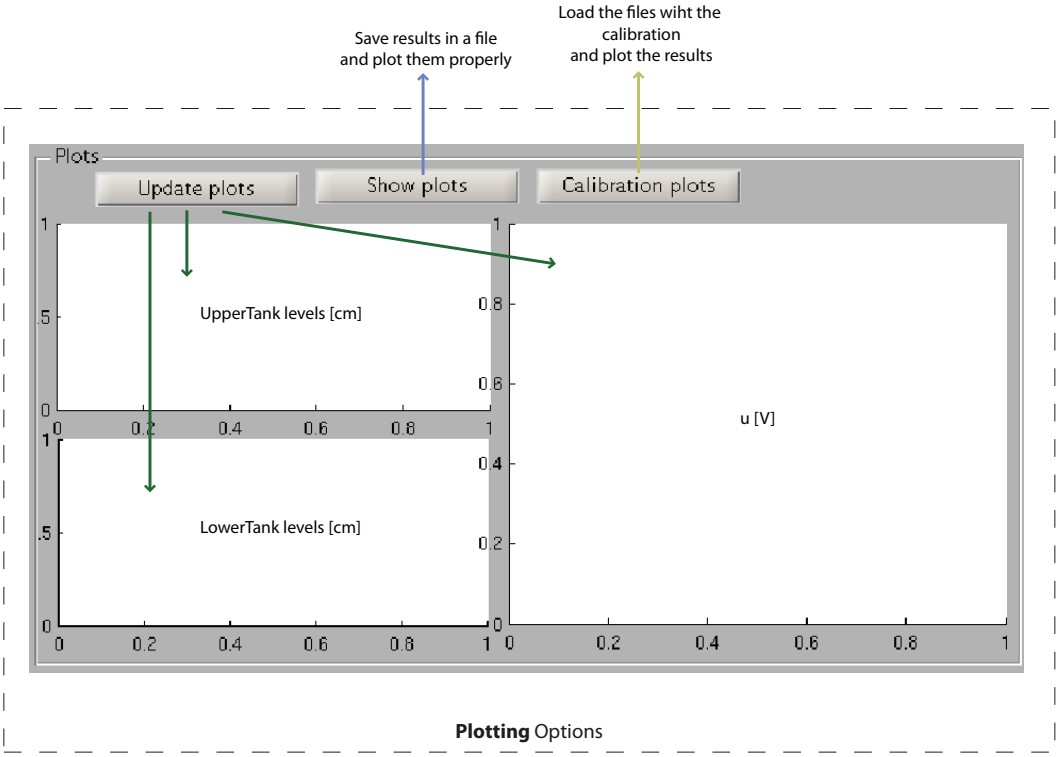

Figure 2.12: Screenshot of the plotting options and button with their description

Figure 2.13 shows an example of eight water tanks tracking a reference with the dynamic scheduler.

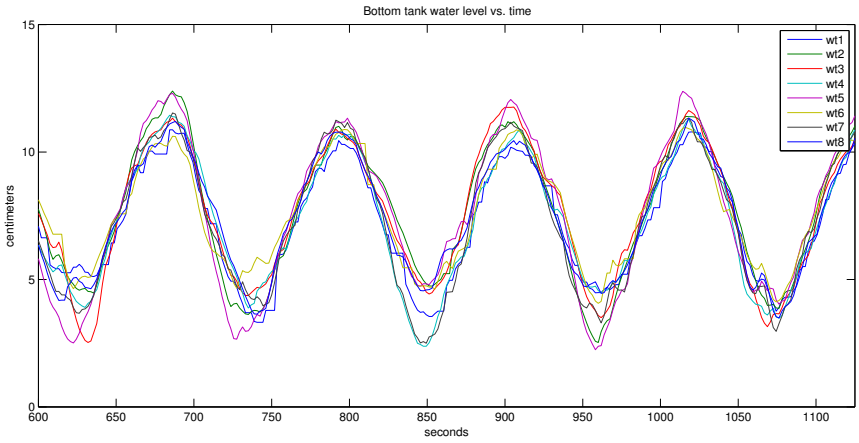

Figure 2.13: Example of eight water tanks connected in cascade

References

[1] *Periodic constraint-based control using dynamic wireless sensor scheduling.* IEEE Conference on Decision and Control (CDC â11), December 2011.

[2] Advantic. Maxfor data sheet. Technical report, Madrid, 2010. URL http://www.advanticsys.com/files/CM5000.pdf.

[3] Jose Araujo, Adolfo Anta, M. Mazo Jr., Joao Faria, Aitor Hernandez, P. Tabuada, and Karl H. Johansson. Self-triggered control over wireless sensor and actuator networks. 7th IEEE International Conference on Distributed Computing in Sensor Systems, jun. 2011.

[4] Moteiv Corporation. Tmote sky data sheet. Technical report, San Francisco, June 2006. URL http://www.bandwavetech.com/download/tmote-sky-datasheet.pdf.

[5] Jan-Hinrich Hauer and Adam Wolisz. TKN15.4: An IEEE 802.15.4 MAC Implementation for TinyOS 2. Technical report, Technical University Berlin - Telecommunication Networks Group, March 2009. URL http://www.tkn.tu-berlin.de/publications/papers/TKN154.pdf.

[6] Aitor Hernandez. Communication between pc and motes in tinyos. Technical report, Royal Institute of Technology (KTH), July 2011.

[7] Aitor Hernandez. Getting started with tinyos at the automatic control lab. Technical report, Royal Institute of Technology (KTH), July 2011.

[8] Aitor Hernandez. Ieee 802.15.4 implementation based on tkn15.4 using tinyos. gts modification. Technical report, Royal Institute of Technology (KTH), July 2011.

[9] Aitor Hernandez. IEEE 802.15.4 implementation for TinyOs based on TKN15.4. GTS implementation. Technical report, TinyOS Contribution, January 2011. URL http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x-contrib/kth/tkn154-gts/doc/pdf/gts_implementation.pdf.

[10] IEEE Std 802.15.4, Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs) , September 2006. URL http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf.

[11] Quanser. Coupled water tanks. instructor manual. Technical report, Quanser.

[12] Quanser. Coupled water tanks. student handout. Technical report.

[13] Quanser. Coupled water tanks. user manual. Technical report, Quanser.

[14] Crossbow Technology. Crossbow telosb. Technical report, San Jose, California. URL http://www.willow.co.uk/TelosB_Datasheet.pdf.

[15] TinyOS. Tinyos tutorials. Technical report. URL http://docs.tinyos.net/tinywiki/index.php/TinyOS_Tutorials.

[16] TinyOS. Tinyos. Technical report, 2010. URL http://www.tinyos.net/.