Estimating the Safety Function Response Time for Wireless Control Systems

by

Victoria Pimentel

TR15-234, March 2015

This is an unaltered version of the author's MCS thesis
Supervisor: Bradford G. Nickerson

Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566
E-mail: fcs@unb.ca
http://www.cs.unb.ca

Copyright © 2015 Victoria Pimentel Guerra

Abstract

Safety function response time (SFRT) is a metric for safety-critical automation systems defined in the IEC 61784-3-3 standard for single input and single output systems communicating over wired technologies. This thesis proposes a model to estimate the SFRT for multiple input and multiple output feedback control systems communicating over the IEEE 802.15.4e wireless medium access control standard designed for process automation. The wireless SFRT model provides equations for the worst case delay time and watchdog timer of participating network entities, including wireless communication channels. Thirty-nine on board, wired and wireless control experiments using real devices were carried out to evaluate control performance, and the applicability of the wireless SFRT model. The estimated SFRT for the wired implementation is 38.2 ms. For the wireless experiments, the best SFRT obtained was 655.4 ms with no acceptable packet loss. The wireless implementation failed to provide successful control on 15 of the 21 experiments.

To Humberto Uribe

Acknowledgements

I would like to especially thank Dr. Bradford G. Nickerson for his support, supervision and guidance throughout the development of this thesis. His input, ideas and invaluable recommendations greatly improved this work. Dr. Nickerson has provided his guidance and advice during my entire studies at the University of New Brunswick. He has certainly contributed on my growth to become a better professional. I would also like to thank Dr. Thomas Watteyne for his patience, dedication and invaluable help during the development of the experimental validation of this thesis. Thanks to Michael Morton at the Electrical and Computer Engineering shop, his expertise and amazing skills made the experimental set-up of this thesis possible. Thanks to Universidad Simón Bolívar and its faculty, who provided me with the knowledge and skills required to succeed in my graduate studies. Thanks to my parents, who provided me with all kinds of support, and to the rest of my family and friends.

Table of Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

6LoWPAN	IPv6 over low-power wireless personal area networks
A	Power difference (corrective action) applied to motors
A_j	Action time of actuator j
ACK	Positive acknowledgement
AD	Action delay
ASN	Absolute slot number
C	Clockwise
c_1	IEC 61784-3-3 factor for the fail-safe watchdog timer of transmission delay entities
c_2	Factor for the worst case delay time of device entities
c_3	Factor for the watchdog timer of device entities
c_4	Number of acceptable consecutive packets lost
CC	Counter clockwise
CEN	European committee for standardization
CENELEC	European committee for electrotechnical standardization
CoAP	Constrained application protocol
CRC	Cyclic redundancy check
CSMA/CA	Carrier sense multiple access with collision avoidance
D	Derivative term of PID control (used only in Section 4.1.1)
D	Set of unique index numbers identifying transmission delay entities
d, $d(x)$	Function, given an input error x the corresponding distance in cm is calculated
DAG	Directed acyclic graph
DAT	Device acknowledgement time
ΔT_WD$_i$	Difference between the watchdog time of entity i and its worst case delay time
DSME	Deterministic and synchronous multi-channel extension
E	Set of network entities, $E = \{I, H, O, D\}$
e	Error, performance metric for line following
EB	Enhanced beacon
EEPROM	Electrically erasable programmable read-only memory
F	Time to complete one lap of line following course
F-Device	Fail-safe device entity
F-Host	Fail-safe host entity type

F_WD_Time_i	Fail-safe watchdog time of entity i
$\text{F\_WD\_Time}_{i,j}$	Fail-safe watchdog time of entity i where the sender initializing communication is entity j
GND	Ground
GTS	Guaranteed timeslots
H	Set of unique index numbers identifying fail-safe host entities
HAT	Host acknowledgement time
HTTP	Hypertext transfer protocol
I	Integral term of PID control (used only in Section 4.1.1)
I	Set of unique index numbers identifying input entities
I/O	Input/output
IE	Information element
IEC	International electrotechnical commission
IEEE	Institute of electrical and electronics engineers
IETF	Internet engineering task force
IP	Internet protocol
IPv6	Internet protocol version 6
ISA	International society of automation
ISO	International organization for standardization
K_D	Constant for the derivative term D
k_D	Number of transmission delay entities in the network
k_H	Number of fail-safe host entities in the network
K_I	Constant for the integral term I
k_I	Number of input entities in the network
k_O	Number of output entities in the network
K_P	Constant for the proportional term P
L	New left motor setting
L'	Previous left motor setting
LCD	Liquid crystal display
LLDN	Low latency deterministic network
LR-PAN	Low rate personal area network
L_{ts}	Total length of one timeslot
μ_{AD}	Average action delay
μ_{C}	Average results on the C direction
μ_{CC}	Average results on the CC direction
$\mu_{\text{C,CC}}$	Combined average for results on the C and CC directions
μ_e	Average error e
μ_{RL}	Average round trip latency
MAC	Media access control
$\max_e$	Maximum observed error e
MIMO	Multiple input and multiple output
n	Total number of network entities
N_a	Total number of actuators in the network
N_a^i	Number of actuators attached to output entity i

NACK	Negative acknowledgement
N_s	Total number of sensors in the network
N_s^i	Number of sensors attached to input entity i
N_{ts}	Number of timeslots in the slotframe
O	Set of unique index numbers identifying output entities
OFDT_i	One fault delay time of entity i
P	Proportional term of PID control
P'	Previous value of P
PAN	Personal area network
P_a^j	Worst case processing time of actuator j
P_i	Worst case processing time of entity i
P_s^j	Worst case processing time of sensor j
PDU	Protocol datagram unit
PHY	Physical
PID	Proportional integral derivative
PL	Packet loss percentage
R	New right motor setting
R'	Previous right motor setting
RAM	Random access memory
RFID	Radio frequency identification
RPL	IPv6 routing protocol for low-power and lossy networks
RX	UART receive channel
σ_{AD}	Standard deviation of action delay
σ_e	Standard deviation of error e
σ_{RL}	Standard deviation of round trip latency
SFRT	Safety function response time
T	Duration of test
TCP	Transmission control protocol
Tcy_i	Cycle time of entity i
TD	Transmission delay entity type
TDMA	Time division multiple access
TSCH	Timeslotted channel hopping
TX	UART transmit channel
U	Line position update interval
UART	Universal asynchronous receiver/transmitter
UDP	User datagram protocol
USB	Universal serial bus
VCN	Virtual consecutive number
WCDT_i	Worst case delay time of entity i
WDTime_i	Watchdog time of entity i
W_i	Longest waiting time of entity i
WirelessHART	Wireless highway addressable remote transducer protocol
WPAN	Wireless personal area network
WSN	Wireless sensor network

X Number of times a watchdog timer expired

Chapter 1

Introduction

Automation involves the interconnection of different devices to achieve tasks for industrial processes. Devices can work together to implement control systems, and for monitoring and supervising, among other tasks. When distance between devices is not a factor, closed-loop control systems are extensively employed to regulate the behaviour of a variable [13]. Figure 1.1 shows a block diagram representing a closed-loop control system. As explained by Frederick and Carlson [13], the plant is the part of the system that performs certain work that needs to be controlled. To implement this control, some of the plant's output variables are monitored by appropriate sensors. The sensors generate readings that are compared by the comparator with the desired reference values. The difference between a sensor reading and its reference value is communicated to the controller which dictates the control strategy, that is

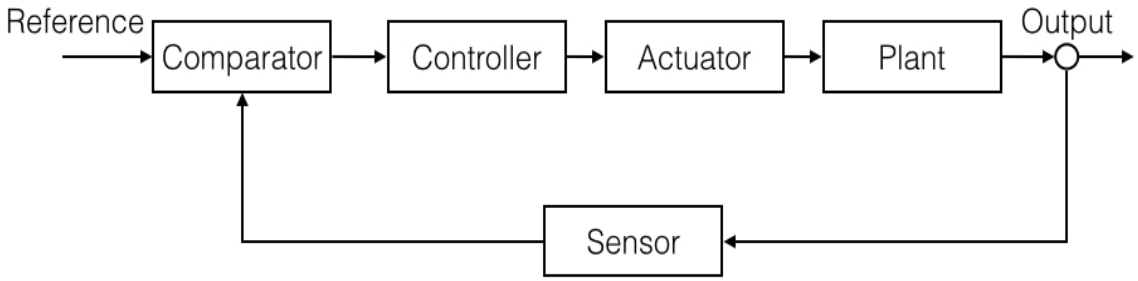


Figure 1.1: Elements of a closed-loop control system (from [13]).

Table 1.1: Wireless industrial control usage classes (adapted from [23]).

Type	Class	Description	Characteristic
Safety	0	Emergency action	Always critical
Control	1	Closed loop regulatory control	Often critical
	2	Closed loop supervisory control	Usually non-critical
	3	Open loop control	Human in the loop
Monitoring	4	Alerting	Short-term operational consequence
	5	Logging and downloading/uploading	No immediate operational consequence

the corrective action that should be applied to the plant. The actuator, which is a device that can perform actions to affect the operation of the system, implements or executes the control strategy. Then, the process is repeated and variations in the output resulting from changes in the forward path are fed back to the system for corrective action. This concept of feeding the system with the plant's output is the main characteristic of closed-loop or feedback control systems.

Some of the blocks in Figure 1.1 can be implemented in different devices in close physical proximity. Communication between the set of devices involved in the control system becomes an important factor to achieve the desired behaviour. Typically the communication occurring at this level is critical, some systems might be required to implement safety functions that are intended to achieve or maintain a safe state for the equipment under control [21]. Table 1.1 presents 6 usage classes for inter-device industrial wireless communication. Each class has different characteristics and different communication requirements.

Safety communication and integrity has been addressed by IEC standards, such as the IEC 61784-3 [20] and IEC 61784-3-3 [21]. One of the system requirements defined by the IEC 61784-3-3 is the safety function response time (SFRT) and is considered one of the most important metrics for safety-critical applications [3]. The estimation of the SFRT provides important metrics that study the performance of

a control network and evaluate if it is suitable for implementing safety and critical functions, which are widely used in automation. The SFRT was introduced and defined for wired fieldbus technologies, specifically for PROFIsafe, the safety profile that runs on top of PROFIBUS [24] .

The use of wireless technologies on automation can potentially lead to significant cost savings, e.g. reduce installation costs by a factor of 10 [28] [34]. Wiring can be expensive and the installation can be time consuming, especially in industrial environments where wires might require special coating, e.g. fire protection coating. Wireless technologies also provide ease of operation, installation, and maintenance. Wireless is more flexible and can provide communication in environments where wiring is not possible [29].

For an appropriate use of wireless technologies in industrial environments, concepts that were initially devised for wired technologies should be applied. This would provide the same safety metrics, and a comparison of process quality can be established. Due to the characteristics of the wireless medium, i.e. non-deterministic shared medium, however, applying the same concepts as wired technologies, e.g. estimating delays, is non-trivial. Deterministic communication is not possible for wireless channels due to interference and other unpredictable factors.

1.1 Problem Statement and Objectives

The safety function response time (SFRT) is considered one of the most important metrics for safety-critical applications. Currently, it is defined and modelled on the IEC 61784-3-3 standard for single input single output systems operating with wired technologies. The calculation of the SFRT requires the estimation of the worst case delay times and watchdog timers of the participating network entities. The IEC 61784-3-3 standard leaves the responsibility of providing the exact calculation

methods to manufacturers.

This thesis studies the application of the SFRT model to wireless networks. This thesis attempts to answer the following questions:

1. How can the SFRT of a wireless network be defined?
2. How can worst case delay times of wireless channels be defined and calculated?
3. Can the current definition of SFRT be extended to include other architectures, e.g. multiple input multiple output (MIMO) systems and multi-hop topologies?
4. Can existing industrial wireless communication protocols be incorporated in a model to estimate the SFRT?
5. How feasible is to apply the SFRT model on real wireless networks?
6. Can wireless technologies provide the same process quality on industrial applications as wired technologies?

IEC 61784-3-3 communication channels are not required to implement any safety measure, but the channel can be adapted to better accommodate the implementation of a safety function. Åkerberg et al. [3] demonstrated that changing the network schedule can reduce the SFRT. One of the network entities considered for calculating the SFRT is the transmission delay. The media and protocol of the communication channel directly affect the transmission delay, as demonstrated by Anand et al. in [4].

This thesis proposes to take into account the communication channel as a factor that affects the performance of the implementation of a safety function by a network, and thus the channel can be adapted to better accommodate such requirements. To achieve this, this thesis proposes a model to estimate the SFRT on a network implementing a feedback control loop, as control loops are extensively employed in control systems. The model takes into account factors that affect network delay, e.g.

medium access technique, communication protocol, number of devices and network topology. The model considers the delays which the network entities incur when implementing the blocks from the closed-loop control system while communicating with each other wirelessly. The worst case delay times and watchdog timers are modelled for all participating entities in the network, including wireless communication channels.

This thesis includes an experimental validation where the SFRT of a wireless feedback control system is estimated using the proposed model. The experimental validation evaluates the applicability of the model, and the feasibility of obtaining the input required by the model. The experimental validation also evaluates the performance of wireless communication when used in feedback control systems, where high speed reliable communication is required to maintain control of the system.

Chapter 2

Background

2.1 Control Networks

The fieldbus is a well known technology standardized in the IEC 61158 [17] standard and is commonly used for real-time control networks. Initially developed to avoid wiring problems that arise when working with some networks topologies such as point-to-point and star-like, the fieldbus provides two way communication between field devices in a shared medium. The fieldbus operates in a ring topology fashion where devices can communicate with each other through the bus, i.e. the shared medium. The idea of a shared bus for the communication of field devices can be traced back to 1970 when the roots of the modern fieldbus technology were conceived [41].

The International Electrotechnical Commission (IEC) has played an important role on the fieldbus standardization. Due to the many and different application fields of the fieldbus technology, different committees work on standardization activities inside and outside the IEC. Other organizations are the International Organization for Standardization (ISO), the European Committee for Standardization (CEN), and the European Committee for Electrotechnical Standardization (CENELEC).

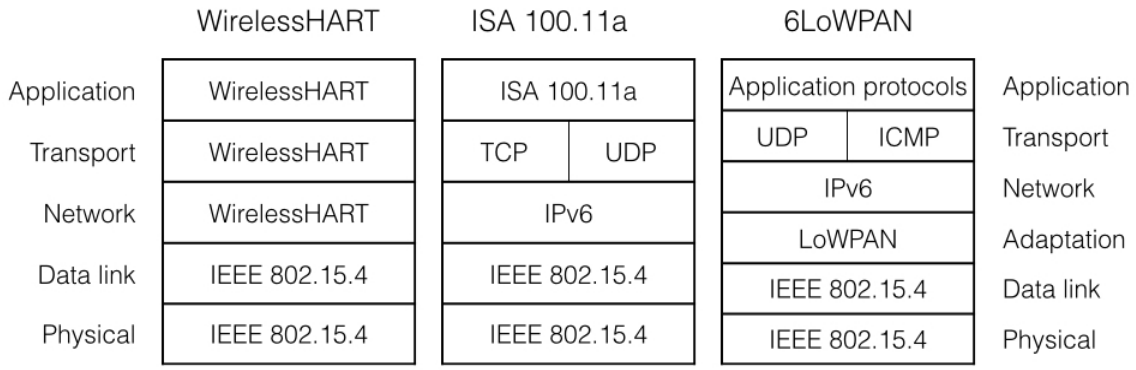


Figure 2.1: WirelessHART, ISA 100.11a and 6LoWPAN protocol stacks.

The fieldbus was designed as a wired technology, but the motivation for wireless gave way to the development of wireless communication protocols targeting real-time control networks and industrial applications. WirelessHART (Wireless Highway Addressable Remote Transducer protocol) and ISA 100.11a (International Society of Automation) are two examples.

6LoWPAN (IPv6 over Low-power Wireless Personal Area Networks) is a wireless communication protocol designed to enable efficient use of IPv6 (Internet Protocol version 6) on small devices, and plays an important role in the emerging Internet of Things [42] [5]. The range of applications for 6LoWPAN is very wide, and its use for industrial applications has been considered in [35]. 6LoWPAN implements an adaptation layer to define the specification for transmitting IPv6 packets (at least 1280 bytes [45]) over IEEE 802.15.4 (frames of 127 bytes [45]). The adaptation layer defines fragmentation, reassembly and header compression.

The protocol stacks of WirelessHART, ISA 100.11a and 6LoWPAN are shown in Figure 2.1. A detailed comparison between these three wireless communication protocols can be found in [36].

Wireless communication is carried in an open medium. As a consequence, wireless networks are especially vulnerable to two of the four classes of network attacks shown in [15]; fabrication and interruption attacks. Wireless communication is especially

vulnerable to continuous jamming at all frequencies and collisions attacks [8], during which data transmission is made impossible.

2.2 Safety Considerations for Network Control

Motivation for safety related communication networks initially arose for railway applications [22], and has been applied in other areas since. For example, Jiang et al. [25] present a design for vehicular safety communication. They propose a system that warns the driver or the vehicle system of potentially dangerous situations. They illustrate with an example where a vehicle that is stopping or moving slowly broadcasts its presence to other vehicles. Another vehicle approaching fast can be aware of the presence of the first vehicle and carry out the proper action (e.g. perform a quick maneuver) while broadcasting the message to other vehicles. This situation is an example where special communication is needed to respond or alert about an emergency or system state that can cause harm to humans. To achieve communication in this and other similar situations, Jiang et al. [25] define safety messages and safety communication protocols on top of the Dedicated Short Range Communication (5.9 GHz DSRC) standard.

Similarly, in industrial control networks there are many situations, e.g. extremely high or abnormal temperature or pressure values, where the safety of humans is directly involved. In these cases, systems are required to implement safety functions to keep risks at an acceptable level [1].

2.2.1 IEC 61784-3-3

Safety integrity can be claimed by systems that comply with the rules specified in the IEC 61508 [19] standard. For wired fieldbus, safety profiles were introduced in the IEC 61784-3 [20] standard, where the the black channel principle was defined.

The black channel principle simplifies the implementation of safety related measures by adding a safety layer on the device protocol stack. This safety layer comprises all measures to deterministically discover any fault that could be introduced by the communication channel. As a consequence, the communication channel does not implement any safety measures and only serves as the transmission medium for data packets and safety frames by non-safe and safe applications. The black channel principle provides interoperability, because the safety layer can be implemented regardless of the transmission medium.

The IEC 61784-3-3 [21] standard defines additional safety specifications for industrial process measurement, control and automation. One-hop communication channels are defined between a controller (fail-safe host, F-Host) and a device (fail-safe device, F-Device) for the exchange of safety packet datagram units (PDU).

The IEC 61784-3-3 standard also defines an important safety metric; the safety function response time (SFRT). The SFRT is defined as the worst case time elapsed from the actuation of a safety sensor connected to a fieldbus, until the safe state of the corresponding safety actuator is achieved in the system in the presence of errors in the channel. The IEC 61784-3-3 standard describes an architecture that consists of five network entities: (a) input, (b) transmission delay 1, (c) fail-safe host, (d) transmission delay 2, and (e) output. Delays from these five entities define the SFRT term, illustrated in Figure 2.2 and defined in the IEC 61784-3-3 standard with the following equation:

$$\mathrm{SFRT} = \sum_{i=1}^{n} \mathrm{WCDT}_i + \max_{i=1,2,\ldots,n} (\mathrm{WDTime}_i - \mathrm{WCDT}_i) \tag{2.1}$$

where:

- n is the number of network entities, $n = 5$ for the architecture described in the standard,

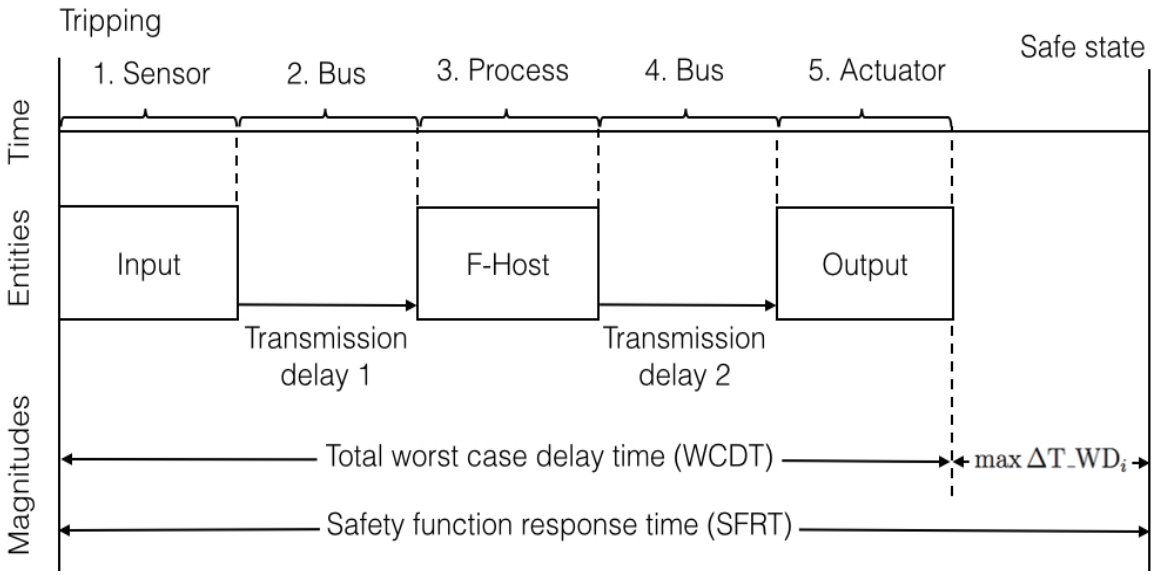


Figure 2.2: Safety function response time definition for the IEC 61784-3-3 model.

- WCDT_i is the worst case delay time of entity i. Thus, $\sum_{i=1}^{n} \mathrm{WCDT}_i$ represents the total worst case delay time for n entities,
- WDTime_i is the watchdog timer of entity i, which takes the necessary actions to activate the safe state whenever a failure or error occurs within entity i, and
- $\Delta\mathrm{T\_WD}_i = \mathrm{WDTime}_i - \mathrm{WCDT}_i$, where $\Delta\mathrm{T\_WD}_i$ is included as it appears in Figure 2.2.

The watchdog timer is implemented as a countdown timer in all participating network entities. Upon the expiration of the local timer, the entity abandons normal operation and activates its safe reaction to reach a safe state. Safe reactions are defined in the IEC 61784-3-3 standard for each device entity. For example, the safe reaction for the output entity consists of shutting down the outputs, and the automatic safe reaction of the actuator unit. The safe reaction of all the network entities constitute the fault reaction of the system. For each of the five network entities, the watchdog timer is defined in the IEC 61784-3-3 standard as follows:

1. For i = Input, F-Host, or Output,

$$\text{WDTime}_i = \text{OFDT}_i \tag{2.2}$$

2. For $i = \text{TD}_1$,

$$\text{WDTime}_i = \text{F\_WD\_Time}_1 + \text{WCDT}_{\text{TD}_1} + \text{Tcy}_{\text{F-Host}} \tag{2.3}$$

3. For $i = \text{TD}_2$,

$$\text{WDTime}_i = \text{F\_WD\_Time}_2 + \text{WCDT}_{\text{TD}_2} + \text{DAT}_{\text{Output}} \tag{2.4}$$

where:

- OFDT_i is the one fault delay time of entity i, i.e. worst case delay time in case of a fault within entity i,
- TD_i is the transmission delay entity i,
- F_WD_Time_i is the fail-safe watchdog timer of entity i. The minimum F_WD_Time is defined for a 1:1 safety protocol datagram unit (PDU) exchange using PROFIsafe, and is composed of four time delays; DAT (Device Acknowledgement Time), HAT (Host Acknowledgement Time), and two bus transmissions,
- WCDT_i is the worst case delay time of entity i, and
- $\text{Tcy}_{\text{F-Host}}$ is the F-Host entity cycle time.

DAT and HAT are two acknowledgement times measured by the device and the host, respectively. The acknowledgement time is the time the entity takes to process the PROFIsafe protocol and prepare a new safety PDU with the currently available process values.

The worst case of transmission delays are required to calculate the SFRT of the system. To get a good estimate of these magnitude, deterministic communication is ideal. Most of the time, however, deterministic communication is not possible for wireless channels due to interference and other unpredictable factors.

A wireless approach to estimate the SFRT was done by Åkerberg et al. [3] where they propose a framework for safe and secure communication regardless of the communication media and based on the black channel principle. One of the metrics of the proof-of-concept experiment is the SFRT that is calculated using Equation (2.1) with values provided by the authors. An approach to measure SFRT components was not included.

Bertocco et al. [6] propose an approach to estimate device delays, specifically access points, in hybrid wireless and wired real-time industrial networks.

2.2.2 IEEE 802.15.4e

IEEE 802.15.4 [32] is a standard that provides a framework for the physical and MAC layers of LR-PANs (low rate personal area networks). These networks have typically low complexity, low energy consumption and low cost [39].

The IEEE 802.15.4 standard can be applied with different configurations. For example, there could be non-beacon or beacon enabled PANs, slotted or unslotted medium access, implementation of the Carrier sense multiple access with collision avoidance (CSMA/CA) or Aloha protocols, star or peer-to-peer network formations, among other optional parameters.

Amendments and refinements of the IEEE 802.15.4 have been done over the past few years. Some of the most recent amendments are:

- IEEE 802.15.4g (2012): amendment to the physical layer for low data rate, wireless, smart, metering utility networks.

- IEEE 802.15.4e (2012): amendment to the MAC layer to support better the industrial market and permit compatibility with Chinese WPAN.

- IEEE 802.15.4f (2012): amendment to the physical layer that defines the Active Radio Frequency Identification (RFID) System for applications combining low cost, low energy consumption, reliable communication and precision on location.

- IEEE 802.15.4k (2013): amendment to the physical layer for low energy critical infrastructure monitoring networks.

- IEEE 802.15.4j (2013): amendment to the physical layer to support medical body area networks.

Specifically, the IEEE 802.15.4e [33] amendment arises to support specific and critical requirements of industrial applications, such as low latency, robustness and determinism, that are not adequately addressed by IEEE 802.15.4-2011. The IEEE 802.15.4e standard provides MAC amendments for specific industrial application domain requirements under the modes shown in Table 2.1; timeslotted channel hopping (TSCH), low latency deterministic network (LLDN), and deterministic and synchronous multi-channel extension (DSME).

Of interest to this thesis are industrial applications, specifically process automation. Some examples of process automation industries are oil, gas, pulp and paper [2]. The main characteristic of process automation, unlike discrete manufacturing, is the continuous nature of the production process. Åkerberg et al. [2] present three groups of sensor network applications and their requirements for the process automation domain. These groups are: monitoring and supervising, closed loop control, and interlocking and control. These groups match the wireless industrial control usage classes presented in Table 1.1. For monitoring and supervising, corresponding to

Table 2.1: MAC modes introduced in the IEEE 802.15.4e standard for specific industrial application domains.

Mode	Application	Major requirement	Topology	Medium access	Synchronization	Discovery
TSCH	Process automation	Network robustness	Any	CSMA-CA, guaranteed, channel hopping	Frames in defined timeslots	TSCH enhanced beacons
LLDN	Factory automation	Very low latency, high cyclic determinism	Star, many devices	TDMA, GTS	Beacons, superframes	Discovery state beacons
DSME	Industrial, commercial and healthcare	Deterministic latency, flexibility	Any, many devices	Multichannel, multisuperframe, GTS	Beacons from time synchronization parent	DSME enhanced beacons

usage classes 4 and 5, the required update interval of sensors ranges between seconds and days. For closed loop control, corresponding to usage classes 1 to 3, the required update interval of sensors ranges between 10 and 500 ms. For interlocking and control, also corresponding to usage classes 1 to 3, the required update interval of sensors ranges between 10 and 250 ms.

2.2.2.1 Timeslotted Channel Hopping

Typical process automation industries in which the timeslotted channel hopping (TSCH) mode could be used are: oil and gas, food and beverage products, chemical and pharmaceutical products, water and waste water treatment, green energy production, and climate control [33].

The IEEE 802.15.4e standard defines two main features of TSCH; time synchronized communication and channel hopping. Time synchronization is achieved by the

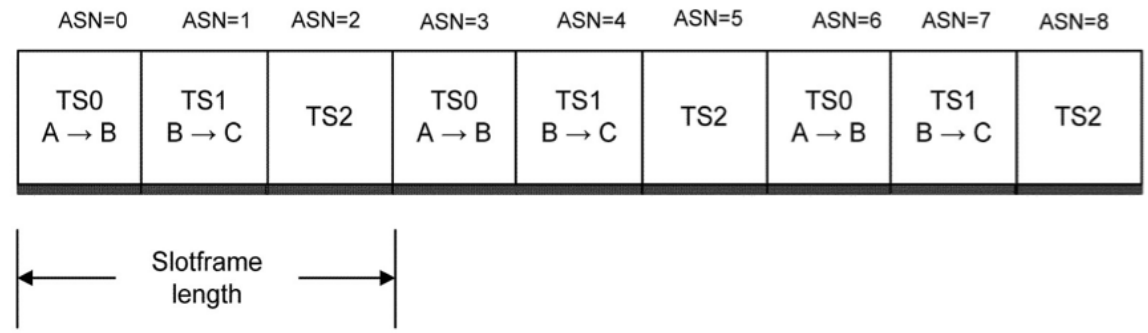
exchange of acknowledged frames and providing timing corrections via the ACK/-NACK Time Correction information element (IE), i.e. a frame or packet containing an ID, a length, and a data payload, used to specify synchronization information. Time synchronization information is represented with a signed 16 bit time correction number in the range of -2,048 μs to 2,047 μs (approximately -2 to 2 ms), where 2 bits are needed for specifying positive or negative ACK. TSCH is defined for star or full mesh topologies.

TSCH uses different types of IEs, not only to keep time synchronization in the network, but also to advertise the network to new devices. A TSCH personal area network (PAN) is formed when the PAN coordinator advertises the presence of the network using enhanced beacons (EB), a type of IE. The join priority of the PAN coordinator is zero, where lower join priority means that the device is the preferred one to connect to. EBs contain information about the network, such as the timeslot ID, the length of one timeslot, the minimum time to wait for the start of an acknowledgement, and transmission time to send the maximum length frame, among others.

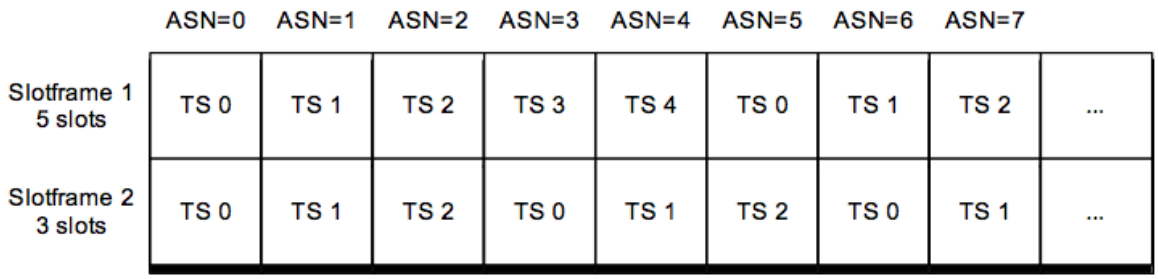
Timeslots are a very important unit of time on TSCH. The IEEE 802.15.4e standard defines a timeslot as a defined period of time during which a frame and an acknowledgement may be exchanged between devices. Slotframes are defined as a collection of timeslots repeating cyclically in time. Slotted communication reduces collisions and minimizes the need for retransmissions.

Access to transmission on a timeslot can be guaranteed or based on request. Guaranteed on dedicated timeslots, i.e. timeslots that are reserved for the communication of a specific pair of devices. For shared timeslots, transmission is based on request using CSMA/CA. Unlike for shared timeslots, there is no waiting for transmission on dedicated timeslots.

An example of a slotframe with three timeslots is shown in Figure 2.3a. Times-



(a) Slotframe with three timeslots.



(b) Multiple slotframes operating simultaneously.

Figure 2.3: Examples of slotframes in a network (from [33]).

lots 0 and 1 are dedicated timeslots reserved for the communication from device A to device B, and from device B to device C, respectively. Timeslot 2 is a shared timeslot and it is not reserved for any pair of devices. The absolute slot number (ASN) indicates the total number of timeslots that have elapsed since the start of the network or since an arbitrary time determined by the PAN coordinator.

TSCH also supports the use of multiple slotframes of different size operating simultaneously in one network. An example of multiple slotframes is shown in Figure 2.3b. Multiple slotframes, each with their own unique identifier slotframeHandle, provide multiple schedules for groups of devices that may need to communicate at different duty cycles. Not all devices need to participate in all slotframes, and one device can participate in multiple slotframes even at the same time. When one device has communication links in multiple slotframes at the same time, transmissions take precedence over receives, and a lower slotframeHandle identifier of the slotframe

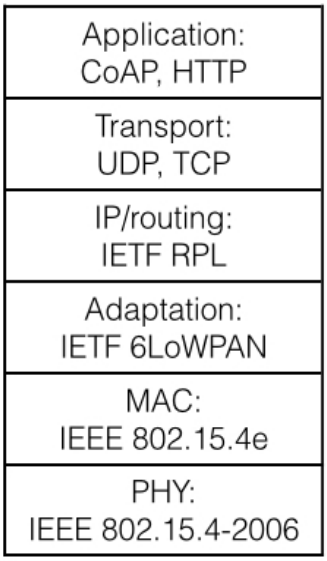


Figure 2.4: OpenWSN protocol stack.

takes precedence over higher slotframeHandle identifiers.

For one timeslot with the default length of 10 ms, after one transmission and if no ACK is received or a NACK is received, there is not enough time for a retransmission of the packet. As a consequence, the IEEE 802.15.4e standard defines that retransmissions, if required, will occur in the next available timeslot. Devices should implement an exponential backoff mechanism as described in the standard. Retransmission on a dedicated link may occur at any time. If an acknowledgement is still not received after macMaxFrameRetries retransmissions, the MAC sublayer assumes the transmission has failed and acts accordingly by notifying an upper layer. For example, for macMaxFrameRetries = 4, retransmission is assumed to be failed after 4 tries where there was no reply or a NACK was received.

OpenWSN [43] is a project that implements the open source standards-based protocol stack shown in Figure 2.4. OpenWSN implements the IEEE 802.15.4-2006 standard at the physical (PHY) layer, the TSCH mode of the IEEE 802.15.4e standard at the medium access (MAC) layer, the Internet Engineering Task Force (IETF) implementations of IPv6 over Low-power Wireless Personal Area Networks

(6LoWPAN) and the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) at the adaptation and Internet Protocol (IP)/routing layers, respectively, the User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) at the transport layer, and the Constrained Application Protocol (CoAP) and Hypertext Transfer Protocol (HTTP) at the application layer. To the author's best knowledge, OpenWSN is the only open source implementation of the TSCH mode of IEEE 802.15.4e.

OpenWSN consists of firmware and the OpenVisualizer software. The firmware is implemented in C and consists of the protocol stack that runs on small, i.e. low power and processing limited, devices also called motes. The OpenWSN firmware supports many hardware platforms, such as TelosB, GINA, and OpenMote, among others. The OpenVisualizer software is implemented in Python, and runs on a computer. OpenVisualizer interacts with the OpenWSN motes, and displays information about the OpenWSN network, such as routing structures, packet queues, network schedule, and error messages. OpenVisualizer also provides connectivity over a virtual interface between the OpenWSN network and the Internet. Remote programs can connect to the virtual interface in order to communicate with motes in the OpenWSN network.

The OpenWSN firmware includes the specification of the network schedule, which is a set of timeslots that can be of the following types:

- CELLTYPE_ADV: advertisement timeslot during which the PAN coordinator advertises the network, so other devices can join.
- CELLTYPE_TXRX: shared timeslot during which access can is requested using CSMA/CA, and can be assigned to any device in the network.
- CELLTYPE_TX: guaranteed transmission timeslot during which a specific device is known to have access to the medium for transmission.
- CELLTYPE_RX: guaranteed reception timeslot during which a specific device

is known to be the destination of a guaranteed transmission.

- CELLTYPE_SERIALRX: serial timeslot during which the mote is performing serial communication.

A recent (2013) MAC protocol suitable for time-critical applications was proposed by Zheng et al. [46]. The proposed MAC, WirArb, is based on user priority to guarantee that the most critical communication will occur first, ensuring timely delivery and real-time performance on critical wireless applications. The authors characterize the maximum packet delay based on the communication protocol required by WirArb.

2.3 Network Delay Models

Mathematical models can be built to estimate and study the behaviour of network parameters (see e.g. [27], [44] and [6]). Such models are built taking into account certain variables, like the network protocol and media access control (MAC), and based on assumptions, for example the number of devices in the network and the network topology.

Calculation of transmission or processing delays in time units is typically not included in models, because these delays are implementation dependent and difficult to estimate. Some models estimate network latency or service delay (e.g. [40]), but they are based on other more complicated approaches, such as Markov chains.

Cruz provides an extensive study [10] [11] in which the elements on a network and the traffic flow on those elements are modelled. Delays are calculated for each element based on traffic flow. The delays of a set of elements are added to calculate the delay from when a packet enters the network until the packet leaves the network. The model considers different network topologies and calculates upper bound delays.

Samaras and Hassapis [40] model the IEEE 802.15.4 unslotted CSMA/CA mechanism using an M/G/1 queue [12] combined with a discrete Markov chain. The paper describes the operation of unslotted CSMA/CA and based on this description the authors define the states in which a wireless device can be at any given time. These states are used to define the Markov chain model, from where probability equations are inferred.

To calculate latency on a wireless sensor network, Ghadimi et al. [14] built a model for a wireless network that consists of a number of stationary nodes sharing a common medium. The MAC protocol implemented by the nodes is IEEE 802.11 in the RTS/CTS mode. Packets arrive to the node with a known arrival rate distributed as a Poisson process and nodes are modeled as an M/G/1 queue. The mean message latency is defined in a closed form equation for both single hop and multihop networks. The variables needed to calculate mean message latency are derived from a Markov chain model built specifically for IEEE 802.11 by Bianchi [7]. The Markov chain model is used to analyze the probability of transmission at each node and derive channel access delay.

Table 2.2 compares the characteristics of some network delay models. To estimate the worst case delay time of transmission delays required to calculate the SFRT, the models in Table 2.2 that consider upper bounds are appropriate. These two models considering upper bounds do not incorporate medium access. In wireless networks, the medium access dictates the communication schedule, and should be taken in consideration when calculating the SFRT. The models by Samaras and Hassapis, and Ghadimi et al. consider medium access, but they are based on more complicated models, i.e. Markov chains. Note that the works in Table 2.2 did not include experiments with real equipment, but simulations were performed.

Table 2.2: Comparison of network delay models.

		Considered in model				Metrics estimated		
Authors	Year	Medium access	Propagation delay	Channel error	Upper bounds	Latency (ms)	Delay	Experimental evaluation
Cruz [10][11]	1991	No	No	No	Yes	No	Network flow delay	No
Zhang et al. [44]	2011	No	No	No	Yes	No	Buffer queue delay	Examples
Kleinrock [27]	1975	Yes	Yes	Yes	No	No	Number of re-transmissions	Simulator
Samaras and Hassapis [40]	2013	Yes	Yes	Yes	No	Yes	Mean service time	Simulator
Ghadimi et al. [14]	2011	Yes	Yes	Yes	No	Yes	Mean network delay	Simulator

Chapter 3

Wireless Safety Function Response Time Model

3.1 Formulation

The set $E = \{I, H, O, D\}$ defines the set of network entities divided into four subsets I, H, O, and D. Here, I, H, O, and D contain the unique index numbers $\in \{1...n\}$ identifying input, fail-safe host, output and transmission delay entities in the system, respectively. The safety function response time (SFRT) in Equation (2.1) is redefined for multiple input and multiple output (MIMO) systems as follows:

$$\text{SFRT} = \sum_{m \in E} \text{WCDTE}_m + \max_{m \in E}(\text{WDTimeE}_m - \text{WCDTE}_m) \tag{3.1}$$

where WCDTE_m is the worst case delay time of all entities in set m. Thus, $\sum_{m \in E} \text{WCDTE}_m$ represents the total worst case delay time of all the entities in the network, and WDTimeE_m is the watchdog timer of all entities in set m.

The n network entities are divided into $|I| = k_I$ input entities, $|H| = k_H$ fail-safe host entities, $|O| = k_O$ output entities, and $|D| = k_D$ transmission delay entities. Typically, $k_H = 1$ for a feedback control loop.

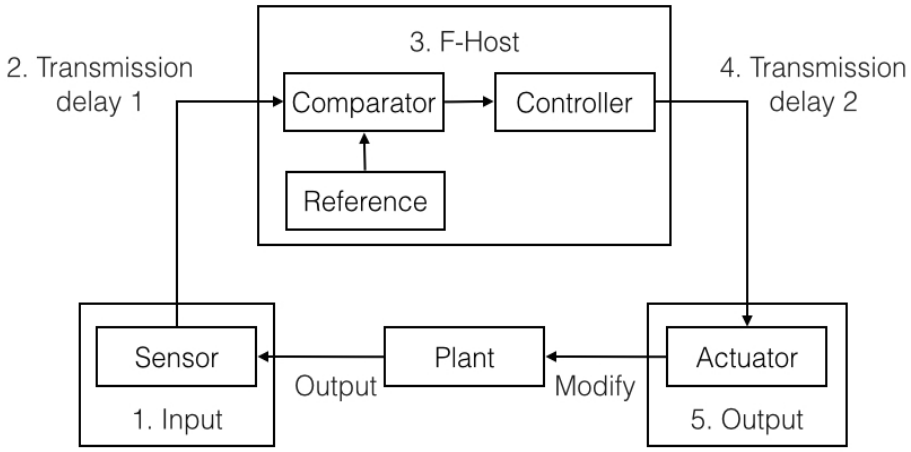


Figure 3.1: Network entities implementing the blocks of a feedback control system as defined in [13].

The wireless SFRT model for MIMO systems extends the IEC SFRT definition and model, as equation (3.1) becomes equation (2.1) in the case of a single input single output system; i.e. k_I, k_H and k_O are all 1, and $k_D = 2$.

An input entity is defined as the network element that implements the sensor block as shown in Figure 3.1. To achieve this, the entity has at least one attached sensor. The input entity builds a network packet with sensor reading(s) from the attached sensor(s) and requires access to the network to transmit the packet. An input entity i has N_s^i sensors attached, and there are k_I input entities in the network. Thus, the total number of sensors N_s in the network is defined as

$$N_s = \sum_{i \in I} N_s^i \tag{3.2}$$

The fail-safe host entity is defined as the network element that implements the comparator and controller blocks as shown in Figure 3.1. The fail-safe host entity requires access to the network to receive data from the input entities, and to send control strategies to the output entities.

An output entity is defined as the network element that implements the actuator block as shown in Figure 3.1. To achieve this, the entity has at least one attached

actuator. The output entity receives packets from the fail-safe host containing the corrective action to be implemented by the attached actuator(s). An output entity i has N_a^i actuators attached, and there are k_O output entities in the network. The total number of actuators N_a in the network is thus

$$N_a = \sum_{i \in O} N_a^i \tag{3.3}$$

3.2 Worst Case Delay Time

The IEC 61784-3-3 standard models the network entities in cycles composed of a waiting and processing time, but the standard does not define delays of entity cycles.

Each network entity in the control system has to perform a specific action. The execution of this action is triggered by a stimulus and the entity must respond accordingly. Table 3.1 shows the stimuli and responses for each entity of the feedback control system. The stimulus of an entity might be expected during the waiting time of its cycle, providing the necessary input for processing. During the processing time, the entity uses the most recent available input data to generate the appropriate response.

The worst case delay time of each entity is defined as the time elapsed since the beginning of the entity cycle time until the time when the corresponding response is achieved as stated in Table 3.1.

3.2.1 Input Entities

The role of the input entity in the feedback control loop is to provide the most recent information about the state of the variable that is being controlled by the system. To achieve this, every W_i time units the input entity performs a process that takes at most P_i time units. These 2 quantities W_i and P_i constitute the cycle of the

Table 3.1: Stimuli and responses for each network entity implementing a feedback control system.

Network entity	Stimulus	Response
Input	New sensor reading(s)	Packet with new sensor reading(s) generated
Transmission delay	New packet ready to be transmitted	Packet successfully received by destination
Fail-safe host	Packet with new sensor reading(s) successfully received	Packet with new corrective action(s) generated
Output	Packet with new corrective action(s) successfully received	Corrective action(s) implemented

input entity i, and vary for different sensors depending on the variable the sensor is measuring and on the physical device. For example, two different wind sensors from Gill Instruments Limited can have different update frequency rates varying from 1 to 10 Hz.

The worst case delay time of the input entity is observed at the start of the entity cycle, meaning the input entity must wait the longest possible time before processing readings from all the N_s^i attached sensors.

For the k_I input entities in the network, the worst case delay time to be used in Equation (3.1) is calculated by the following equation:

$$\text{WCDTE}_I = \max_{i \in I} \text{WCDT}_i \tag{3.4}$$

where WCDT_i is the worst case delay time of input entity i calculated as follows:

$$\text{WCDT}_i = W_i + P_i + \sum_{j=1}^{N_s^i} P_s^j, i \in I \tag{3.5}$$

where W_i is the longest waiting time of input entity i, P_i is the worst case processing time of input entity i and includes the time to read the available sensor value(s) and

create a packet for transmission, N_s^i is the number of sensors attached to input entity i, and P_s^j is the worst case processing time of sensor j (i.e. the time to obtain a new reading from sensor j). Note that individual sensor cycle times are not included in P_s^j. The focus for the wireless SFRT model is on the safety function response time, which always uses the latest sensor readings available.

3.2.2 Fail-safe Host Entity

The role of the fail-safe host is to compare sensor readings with the desired reference values and dictate the control strategy. Consistent with the IEC 61784-3-3 network entities, the wireless SFRT model assumes one fail-safe host per feedback control loop.

The fail-safe host cycle consists of a waiting time W_i, the processing time of the controller C_i, and the processing time to generate a packet with the corrective action P_i. After W_i the most recent available sensor readings are used as input by the controller to generate the corrective action. The controller is executed using a time triggered program (see [21]), which limits the waiting time to ensure that the fail-safe host will not wait indefinitely for input data. After executing the controller, the fail-safe host builds a packet containing the corrective action to be sent to the output entity.

The appropriate waiting time depends on the nature of the feedback control loop and on the variable(s) that are being monitored and controlled by the system. Similarly, processing times depend on the physical device in which the controller is implemented.

The worst case delay time of a fail-safe host entity is observed at the start of the entity cycle, meaning the fail-safe host must wait W_i before processing. Thus, the worst case delay time of the fail-safe host entity to be used in Equation (3.1) is

calculated with the following equation:

$$\text{WCDTE}_H = \text{WCDT}_i = W_i + C_i + P_i, i \in H \tag{3.6}$$

where $|H| = 1$, W_i is the longest waiting time of the fail-safe host entity, C_i is the worst case processing time of the fail-safe host entity to execute the controller, and P_i is the worst case processing time of the fail-safe host entity and includes the time to create a packet for transmission.

Fail-safe host processing and waiting times define the fail-safe host entity cycle time $\text{Tcy}_{\text{F-Host}}$ (see Equation (2.3)) as follows:

$$\text{Tcy}_{\text{F-Host}} = \text{WCDTE}_H \tag{3.7}$$

3.2.3 Output Entities

The role of the output entity is to implement the corrective action in the system as soon as possible. The action time A_j of actuator j is defined as the longest time it takes to apply the corrective action to the system. The time required by the actuator to perform such action depends on the characteristics of the system and on the actuator physical device. The output entity does not have to wait for the actuators' response. As each of the attached actuators receive the corrective action, they implement it in parallel.

The worst case delay time of the output entity is the longest action time observed in the N_a^i actuators attached to the output entity, plus any waiting and processing times within the output entity.

For the k_O output entities in the network, the worst case delay time to be used

in Equation (3.1) is calculated by the following equation:

$$\text{WCDTE}_O = \max_{i \in O} \text{WCDT}_i \tag{3.8}$$

where WCDT_i is the worst case delay time of the output entity i calculated as follows:

$$\text{WCDT}_i = W_i + P_i + \sum_{j=1}^{N_a^i} P_a^j + \max_{j=1}^{N_a^i} A_j, i \in O \tag{3.9}$$

where P_i is the worst case processing time of output entity i and includes the time to open and read the new packet recently received, N_a^i is the number of actuators attached to output entity i, P_a^j is the worst case processing time of actuator j that includes receiving the corrective action from the output entity, and A_j is the worst case action time of actuator j (i.e. the longest time actuator j takes to apply the corrective on the system).

If processing times within an output entity and all attached actuators is negligible, Equation (3.9) becomes

$$\text{WCDT}_i = W_i + \max_{j=1}^{N_a^i} A_j, i \in O \tag{3.10}$$

For the wireless SFRT model to take into consideration any kind of device entity, the processing and waiting times, i.e. P_i, W_i, P_j, A_j and C_i, are implementation dependent and are considered as input to the model.

3.2.4 Transmission Delay Entities

Transmission delays are observed in directed communication links that connect device network entities. The transmission delay of a directed communication link depends on the protocol implemented by the communication link.

Given a network that operates under the TSCH mode of IEEE 802.15.4e, the

worst case transmission delay of a directed communication link is observed when a packet has to wait the longest possible time before transmission. This happens when the packet is ready to be transmitted at the moment the sender's dedicated timeslot ends, meaning that the transmission has to wait for an entire slotframe. Thus, the worst case delay time of the transmission delay entity i is calculated as follows:

$$\mathrm{WCDT}_i = N_{\mathrm{ts}} L_{\mathrm{ts}}, i \in D \tag{3.11}$$

where N_{ts} is the number of timeslots in the slotframe[1], including one enhanced beacon frame for advertising, and L_{ts} is the total length of one timeslot[2], including any unused time after frame transmission and acknowledgement.

The worst case delay time of the k_D transmission delay entities to be used in Equation (3.1) is calculated by the following equation:

$$\mathrm{WCDTE}_D = k_D \mathrm{WCDT}_i, i \in D \tag{3.12}$$

where k_D is the total number of transmission delay entities in the network, and WCDT_i is the worst case delay time of the transmission delay entity i (see Equation (3.11)).

Equation (3.11) arises from the study of the TSCH mode of the IEEE 802.15.4e, and the analysis of the configuration parameters that are most suitable for a feedback control system. Equation (3.11) takes into account waiting and processing times of the directed communication link, and is based on the following assumptions:

1. The feedback control system is implemented in a network that operates under the IEEE 802.15.4e protocol in TSCH mode, with a fixed number of devices

[1]Corresponds to the variable macSlotframeSize defined in the IEEE 802.15.4e. The range for the variable is any integer between 0 and 65536.

[2]Corresponds to the variable macTsTimeslotLength defined in the IEEE 802.15.4e. The range for the variable is any integer between 0 and 65536. The default value is $10,000\mu$s (10 ms)

that have already successfully joined the network. There is one dedicated timeslot per directed communication link, and one timeslot to advertise the network. A directed communication link consists of a sender-destination pair of devices. Each directed communication link has access to one dedicated timeslot, meaning guaranteed access to the medium during the timeslot. Guaranteed access to the medium, however, does not imply that communication will be successful.

2. Consistent with the IEC 61784-3-3 standard, the network is assumed to operate in a star topology, where devices can be sensors, actuators or the host. The host is the central device with the role of PAN coordinator and has a one-to-one communication link with each device in the network. Sensors and actuators communicate only with the host, and not directly with each other. Sensors send readings to the host. The host processes readings, calculates the corrective actions and sends them to the actuators.

3. As a consequence of assuming a star topology, the host is active on every timeslot of the slotframe, either receiving a frame from a sensor or sending a frame to an actuator. Except for the device communicating with the host, the rest of the devices are not trying to access the medium. Thus, there is one slotframe in the network at any time, and only a pair of devices participate on each timeslot. If there were to be more than one slotframe on the network at the same time, the host will certainly be scheduled to participate in more than one timeslot simultaneously. Multiple slotframes will negatively affect the worst case delay time of the devices in the network because access to the medium will no longer be guaranteed. The wireless SFRT model does not include the possibility of multiple slotframes.

4. The order of timeslots in the slotframe will dictate the communication schedule of the network, i.e. the order of communication links in the slotframe. This

order directly affects delays because there will be devices that participate in the communication cycle earlier than others. Timeslots are assigned to devices as they join the network, so the order in which devices participate in the slotframe is not known beforehand.

5. In a feedback control system network, the next available timeslot for a retransmission is the next dedicated timeslot for the corresponding sender-destination pair. Due to the nature of automation, the data in the feedback control network is valid for a short period of time and propagating new data is preferred over guaranteed delivery [2]. Thus, the wireless SFRT model specifies macMaxFrameRetries = 0. This means that if the transmission of a frame fails, no retransmission is performed. In the next cycle, a new packet with the most recent information available will be used. If consecutive transmissions fail, the watchdog timer within the entity will expire and the entity will activate its safe reaction to reach a safe state. In the wireless SFRT model, acknowledgement frames are still necessary because they are used by the TSCH mode to perform timing corrections.

6. Channel hopping is an important feature of the TSCH mode as it can help mitigate the negative effects of multipath fading and interference. Transmissions hop over the entire channel space according to a calculated frequency channel sequence. Channel hopping does not affect the moment in which a device transmits a packet. Thus, it does not affect the worst case delay time of transmission delay entities.

3.3 Watchdog Timer

The IEC 61784-3-3 standard provides equations to model and calculate watchdog timers (see Equations (2.2) through (2.4)). The wireless SFRT model extends the

watchdog timer to consider MIMO systems, and introduces the watchdog timer for wireless communication links.

3.3.1 Device Entities

The watchdog timer of a device entity is defined as the one fault delay time (OFDT) of a device entity, and varies for different devices and implementations. As a consequence, OFDT of device entities are considered as input to the wireless SFRT model. The watchdog timer $\mathrm{WDTimeE}_m$ to be used in Equation (3.1) is calculated by the following equation:

$$\mathrm{WDTimeE}_m = \max_{i \in m} \mathrm{OFDT}_i \tag{3.13}$$

where m can be I, H, or O, and OFDT_i is the one fault delay time of entity i.

3.3.2 Transmission Delay Entities

DAT and HAT are defined in the IEC 61784-3-3 standard as the processing delay times of the PROFIsafe protocol for a received safety PDU and the preparation of a new safety PDU on the device and host entities. For the wireless SFRT model, the DAT corresponds to P_i for $i \in I$ (see Equation (3.5)) and HAT corresponds to P_i, for $i \in H$ (see Equation (3.6)).

These two delays DAT and HAT plus two bus transmissions compose the fail-safe watchdog time F_WD_Time of a 1:1 PROFIsafe communication link. The wireless SFRT model assumes that the transmission delay entities are implemented in communication links that operate under the TSCH mode of IEEE 802.15.4e, where in one timeslot there is enough time for the exchange of an acknowledged frame. As a consequence, when the input entity is the sender and the host is the destination, one timeslot includes the HAT and 2 transmissions. On the other hand, when the host is the sender and the output entity the destination, one timeslot includes the DAT

and 2 transmissions. This results in

$$\text{F\_WD\_Time}_{i,j} = c_1(P_j + L_{\text{ts}}) \tag{3.14}$$

where $i \in D$, $j \in I$ or $j \in H$, $\text{F\_WD\_Time}_{i,j}$ is the fail-safe watchdog time of transmission delay entity i where the sender initializing communication is entity j, P_j is the protocol processing time of sender entity j, L_{ts} is the total length of one timeslot, and c_1 is a constant. The IEC 61784-3-3 standard recommends c_1 values in the range $1 \leq c_1 \leq 1.3$.

The watchdog timer of transmission delay entity i is defined as

$$\text{WDTime}_i = \text{F\_WD\_Time}_{i,j} + \text{WCDT}_i, i \in D \tag{3.15}$$

where $\text{F\_WD\_Time}_{i,j}$ is the fail-safe watchdog time of transmission delay entity i and sender entity j, and WCDT_i is the worst case delay time of transmission delay entity i.

The watchdog timer to be used in Equation (3.1) is calculated by the following equation:

$$\text{WDTimeE}_D = \sum_{i \in D} \text{WDTime}_i \tag{3.16}$$

where WDTime_i is the watchdog timer of transmission delay entity i.

The wireless SFRT model has been applied to a climatic chamber system with 9 network entities implementing a dual control loop of temperature and humidity in [37].

Chapter 4

Experimental Validation

The safety function response time (SFRT) is the worst case delay time since the actuation of a sensor until a safe state is achieved in the system. The wireless SFRT model proposed in Chapter 3 provides a method for estimating the SFRT on a wireless network implementing a feedback control loop. The wireless SFRT model defines input, fail-safe host, output and transmission delay entities of the wireless network elements required to implement a feedback control loop. Based on the feedback control loop block that each entity implements, the wireless SFRT model provides equations to calculate the worst case delay time and watchdog timer value for each entity.

Equations in the model are based on the theory presented in standards, such as the IEC 61784-3-3 and the IEEE 802.15.4e, and on the analysis of the delays that the network entities incur when implementing feedback control loop blocks. The experimental validation provides a study of the application of these concepts in a real experiment with real devices and real communication links. The wireless SFRT model assumes some parameters are provided as input, e.g. waiting and processing times. In the experimental validation, the input parameters required by the model are measured experimentally. This provides an evaluation of the feasibility

of obtaining the input required by the model, and how these inputs can be measured in a real experiment.

The experimental validation also provides a set of results that can be used as a reference to determine important values that should be reported when implementing control loops, how to evaluate the response time of devices, and if response times are suitable for the control loop being implemented. The results from the experiments provide information that can help evaluate whether the achievable SFRT value is acceptable for the system under control.

4.1 Wireless Line Following

Line following robots are very popular in the robotics community and relatively easy to implement. The robot usually has a set of sensors that are able to detect the position of a black line on the floor and the motors change speed accordingly to keep the robot centered on the line. A line following robot can be implemented with a feedback control loop, where the variable being monitored is the position of the robot with respect to the line. The measured line position is provided as input to the controller. The controller generates the corrective action that is the speed at which the motors should be set in order to keep the robot centered on the line. The speed is applied to the motors, which changes the position of the robot with respect to the line. Then, the process is repeated and the new line position generated from the change in motor speed is fed back as input to the controller.

Successful line following requires a fast line position update interval, since the position of the line is constantly changing as the robot moves along the course. Also, a moving robot illustrates the motivation for using wireless technologies very well. A wired set-up would be inconvenient and would require special considerations when designing the course; for example the size of the course and length of wires, as well

as preventing the robot from tangling up the wires. Also, wireless line following experiments can be repeated on the same course as many times as required, which allows to evaluate the impact of different factors on the same experiment.

To implement a line following feedback control loop, the input, fail-safe host, and output entities are deployed in real hardware, and the feedback control loop blocks implemented in software.

4.1.1 Hardware

Figure 4.1 shows an architecture diagram with all the devices involved in the implementation of the wireless line following feedback control loop. The diagram shows the network entities and feedback control loop blocks implemented by each device, as well as the communication channels. The directed acyclic graph root (DAG root) TelosB mote is the personal area network (PAN) coordinator and gateway of the OpenWSN network, which has special functions such as advertising the network. The second TelosB, called the mobile mote, is serially connected to the 3pi. Appendix B illustrates the wireless line following experimental set-up.

The Pololu 3pi is a mobile robot with two micro metal gearmotors, five reflectance sensors, a liquid crystal display (LCD), a buzzer, and three user buttons. The 3pi is controlled by a C programmable ATmega328p processor with 32 KB of flash memory, 2 KB random access memory (RAM), 1 KB of persistent electrically erasable programmable read-only memory (EEPROM), and a maximum operating frequency of 20 MHz. The 3pi has a diameter of 9.5 cm, is powered by four AAA batteries, and is capable of reaching a speed of up to 100 cm per second. The Pololu Corporation provides a very complete C library with a collection of support functions for programming Pololu devices like the 3pi, among others. The Pololu 3pi and its components are shown in Figure 4.2.

The 3pi supports many applications and custom behaviours, but the 3pi was

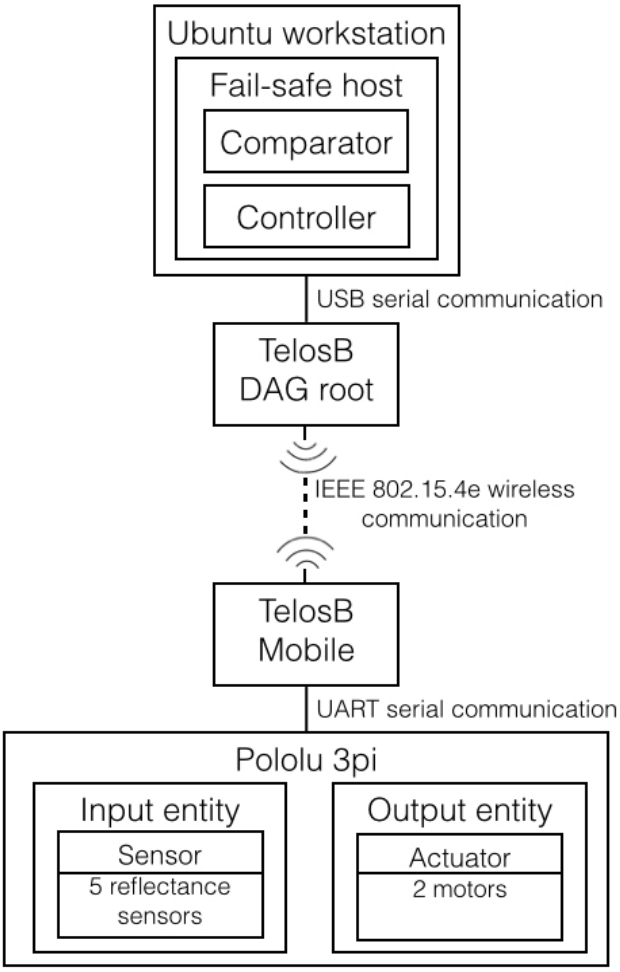
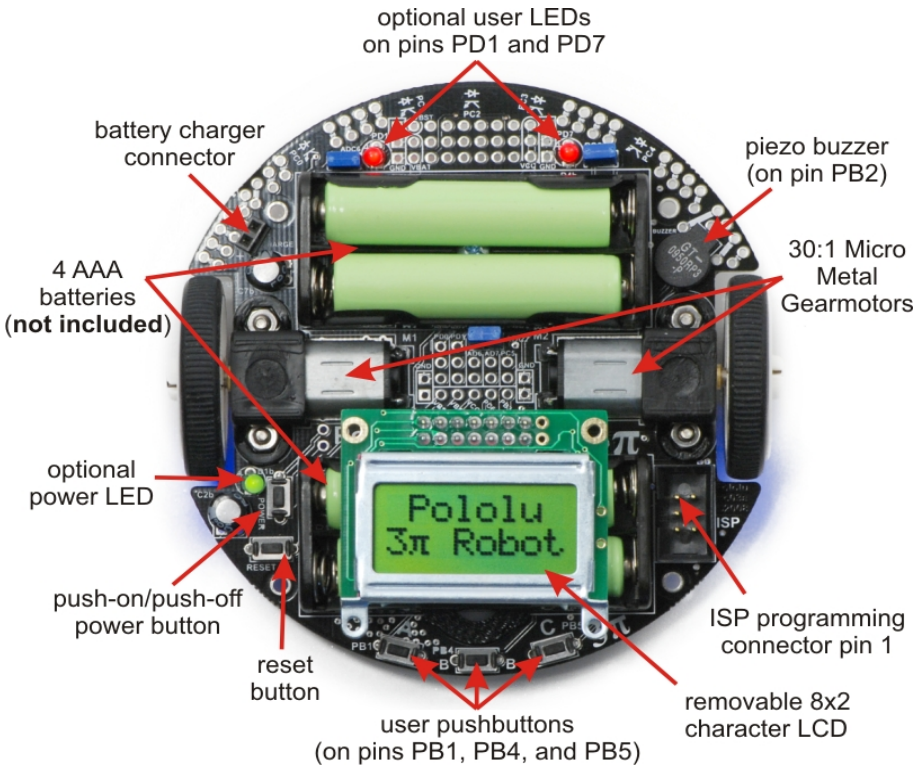
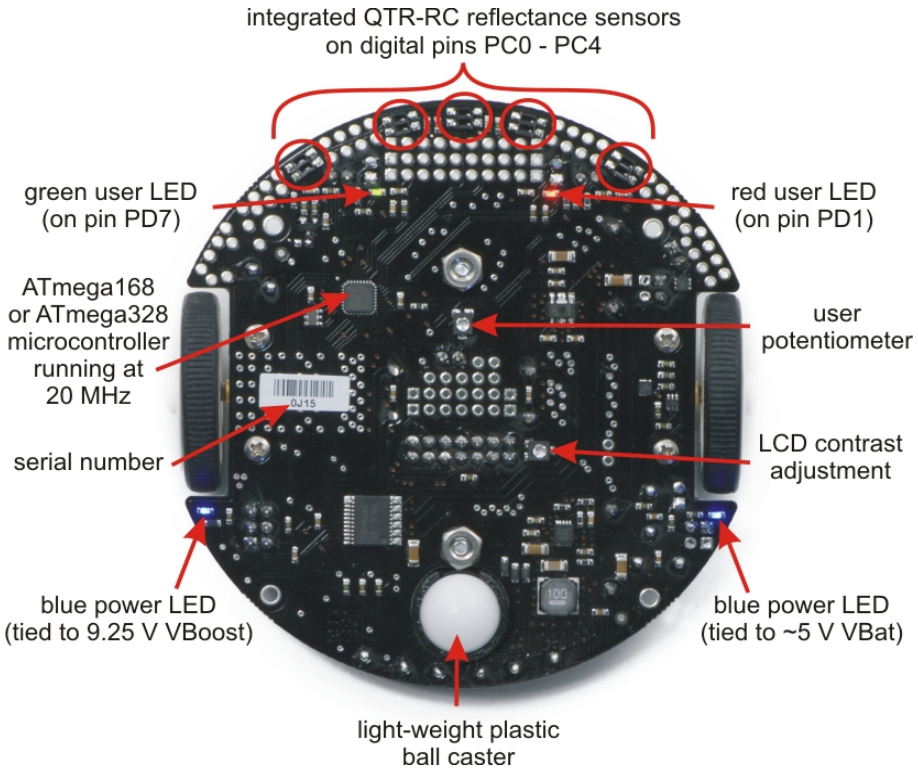


Figure 4.1: Devices implementing network entities and feedback control loop blocks to achieve wireless line following control.



(a) Pololu 3pi top view.



(b) Pololu 3pi bottom view.

Figure 4.2: Pololu 3pi with labeled components (from [9]).

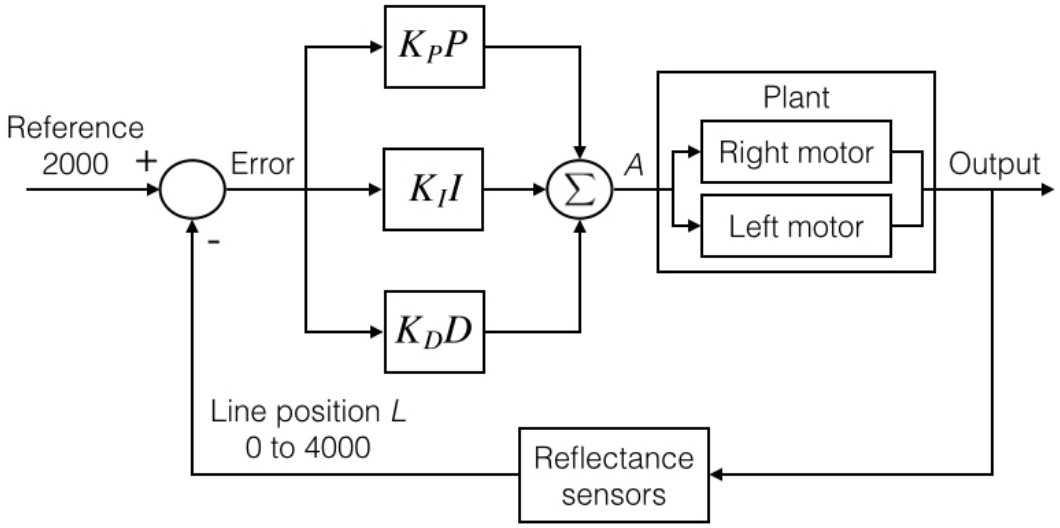


Figure 4.3: Line following feedback control loop.

designed to excel in line following, where the five reflectance sensors detect the position of a black line on the floor and the two micro metal gearmotors change the position of the 3pi. The Pololu Corporation C library provides an implementation of a PID controller for line following. In this implementation, the PID controller runs in a loop that reads the line position from the sensors, calculates the proportional, integral and derivative values, and applies the settings to each motor. This loop runs locally on the 3pi where one iteration of the loop takes less than 2 ms. This means that a new corrective action is calculated and applied to the motors approximately 500 times per second. The control model is shown in Figure 4.3 and is represented by the following equations:

$$\text{Error} = P = 2000 - L \tag{4.1}$$

$$D = P - P' \tag{4.2}$$

$$I = I + P \tag{4.3}$$

$$A = K_P P + K_I I + K_D D \tag{4.4}$$

where P represents the proportional term, P' represents the previous value of P, D represents the derivative term, I represents the integral term, A is the power difference (corrective action) applied to the motors and has values in the range $-60 \leq A \leq 60$, $K_P = -1/20$ and is the constant for the proportional term, $K_I = 1/10000$ and is the constant for the integral term, and $K_D = 3/2$ and is the constant for the derivative term. The actual power level transmitted to the left and right motors depends on a non-linear mapping of the corrective action A as follows:

$$L = \begin{cases} L' & \text{if } A \geq 0 \\ L' + A & \text{if } A < 0 \end{cases} \tag{4.5}$$

$$R = \begin{cases} R' & \text{if } A < 0 \\ R' - A & \text{if } A \geq 0 \end{cases} \tag{4.6}$$

where L is the new left motor setting (rpm), L' is the previous left motor setting (rpm), R is the new right motor setting (rpm), and R' is the previous right motor setting (rpm). The 3pi code that sets the power levels is given in Appendix A. The output of the plant is a change in the position of the 3pi.

Since the 3pi has access to the sensors and the motors, and has processing power, implementing the feedback control loop locally is possible, however this might not always be the case. A feedback control loop can be implemented in different devices of close proximity. In such cases, devices need to communicate with each other to achieve the desired control.

To apply the wireless SFRT model to PID control line following, the implementation provided by the Pololu Corporation is broken into the feedback control blocks, i.e. sensor, comparator, controller, and actuator blocks, and implemented on its corresponding network entity, i.e. input, fail-safe host, and output entities. The 3pi has access to the reflectance sensors (sensor block) and motors (actuator block), so

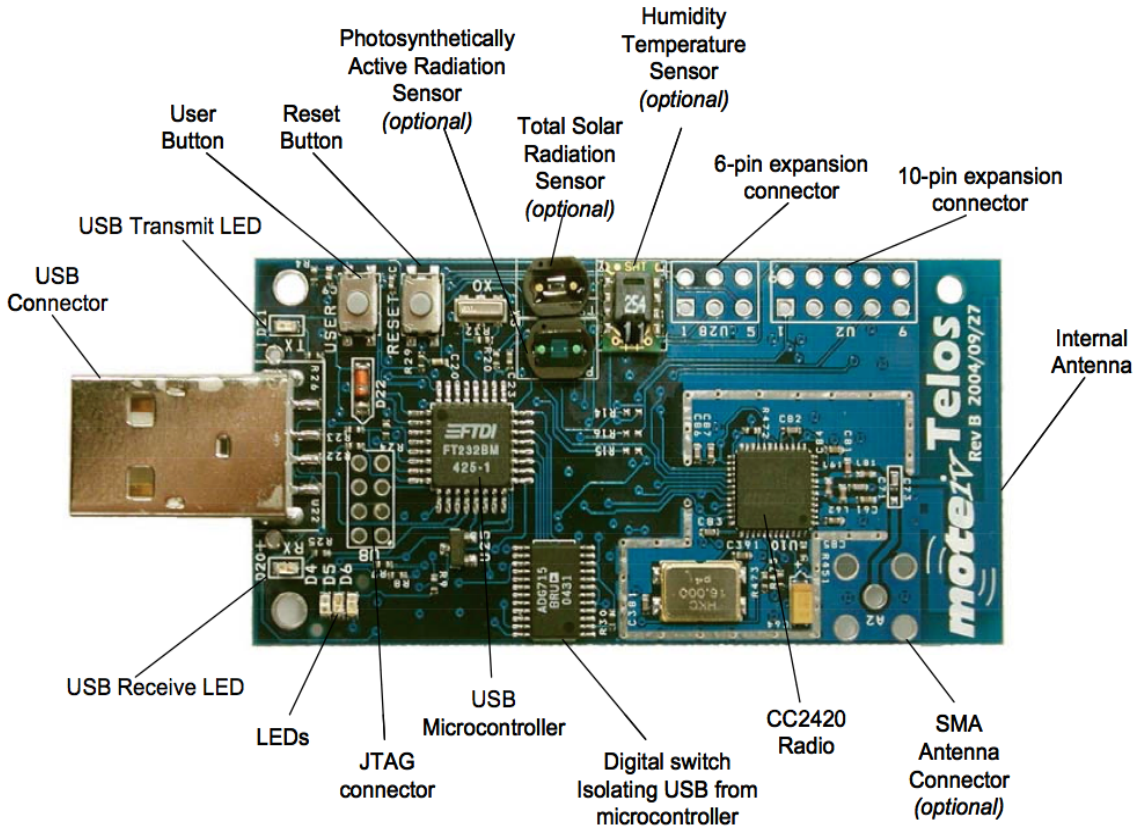
the 3pi implements both the input and output entities. A Ubuntu Linux workstations acts as the host device implementing the comparator and controller blocks. The fail-safe host and the 3pi need to communicate wirelessly with each other. As a consequence, the 3pi and the host must be given wireless networking capabilities, specifically for the IEEE 802.15.4e standard assumed by the wireless SFRT model.

The TelosB is a low power wireless module with a Texas Instruments MSP430F1611 microcontroller with 10 KB of RAM and 48 KB of flash memory. The TelosB has an integrated antenna and an IEEE 802.15.4 wireless transceiver that can operate at 250 kbps on the 2.4 GHz band. A TelosB can be powered by two AA batteries and has many components as shown in Figure 4.4, such as temperature, humidity and solar radiation sensors, two user buttons, and pin expansion support among others.

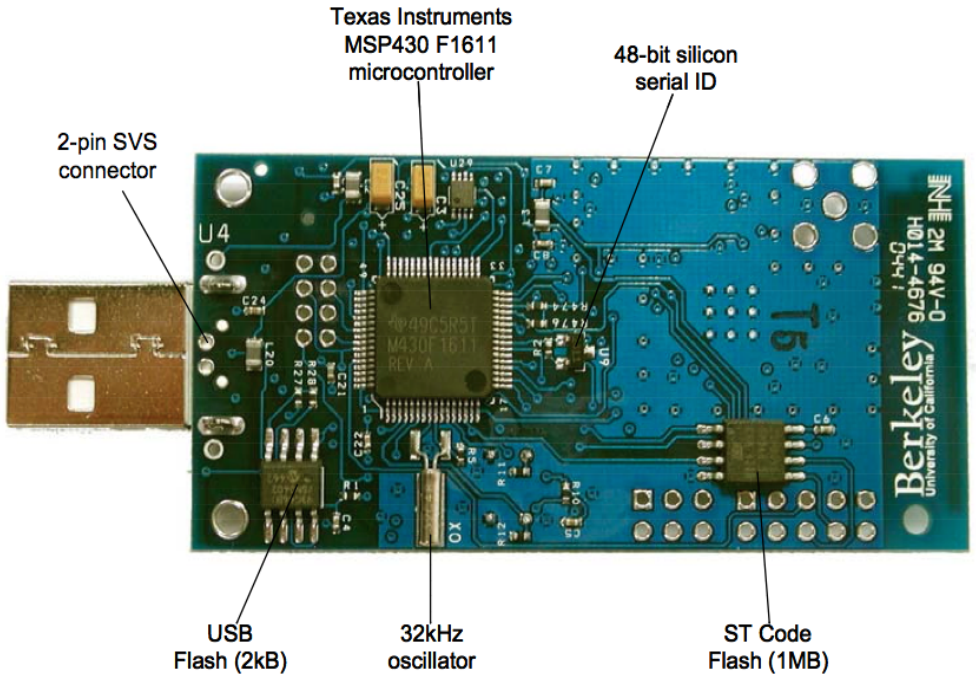
The TelosB is fully supported by OpenWSN [43], the open source implementation of a complete protocol stack based on the Internet of Things standards that includes the IEEE 802.15.4e standard. A TelosB running the OpenWSN firmware works together with the Pololu 3pi and the Ubuntu workstation in order to provide IEEE 802.15.4e wireless networking capabilities. To achieve this, the TelosB keeps constant communication with the 3pi and the Ubuntu workstation. For example, the 3pi generates a line position that is transmitted wirelessly by the TelosB to the Ubuntu workstation.

The TelosB has serial communication lines, two universal asynchronous receiver/-transmitter channels (UART0 and UART1) and a universal serial bus (USB), that the TelosB can use to communicate with other devices. The USB male connector incorporated on the TelosB provides the required communication link between the TelosB and the Ubuntu workstation. The TelosB running the OpenWSN firmware is easily plugged into one of the female USB serial connectors available at the Ubuntu workstation to provide IEEE 802.15.4e wireless networking capabilities.

The Pololu 3pi does not have a female USB connector and one cannot be simply



(a) TelosB top view.



(b) TelosB bottom view.

Figure 4.4: TelosB wireless module with labeled components (from [31]).

added, as a USB controller is required. The 3pi has UART TX and RX channels which can be used to communicate with the UART RX and TX channels on the TelosB. Access to the UART0 lines on the TelosB is provided on the 10 pin expansion connector shown in Figure 4.4a. UART0, however, uses the same resources on the TelosB as the wireless radio. As a consequence, special low level resource arbitration needs to be implemented in order to use both the UART0 and radio components on the TelosB. Fortunately, there is the second UART1 line which does not have conflicts with other components, but is not easily accessed via an expansion connector. The UART1 line is used by the USB interface component which uses an I/O buffer to translate UART to USB signals. These translated signals are then transmitted over the USB male connector on the TelosB. This is the serial line used by OpenWSN to transmit information regarding the state of the TelosB mote. The only way to access the UART1 line is to intercept its pins directly on the microcontroller. To achieve this, three cables were soldered directly on the GND (ground), UART1TX, and UART1RX pins on the MSP430F1611 microcontroller on the TelosB as shown in Figure 4.5. The cables accessing the GND, UART1TX, and UART1RX on the TelosB are connected to the GND, UARTRX, and UARTTX, respectively, on the 3pi. These connections allow for communication between the TelosB and the 3pi, making it possible for the TelosB running OpenWSN firmware to provide IEEE 802.15.4e wireless networking capabilities to the 3pi.

4.1.2 Software

The software components involved in the implementation of the wireless line following feedback control loop are shown in Figure 4.6.

OpenWSN consists of firmware, i.e. the protocol stack running on TelosB motes, and on the OpenVisualizer software, i.e. the program running on a computer where at least the DAG root mote is connected on a serial port. OpenVisualizer commu-

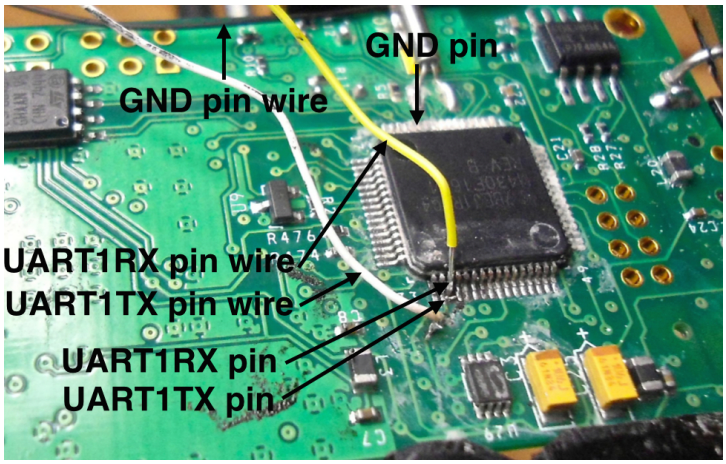


Figure 4.5: TelosB MSP430F1611 microcontroller with wires soldered into the GND, UART1TX, and UART1RX pins.

nicates with the DAG root mote serially to receive information from the OpenWSN network, such as routing structures, packets addressed to the Internet, and error messages. OpenVisualizer provides connectivity between the OpenWSN network and the Internet over a virtual interface. Other programs running locally or remotely can connect to the virtual interface in order to communicate with motes in the OpenWSN network. This functionality of connecting the OpenWSN network to the Internet over a virtual interface is provided by default in OpenVisualizer.

As shown in Figure 4.1, the TelosB mote that is connected serially to the Ubuntu workstation assumes the role of the DAG root. The DAG root mote is running the OpenWSN firmware and is communicating serially with OpenVisualizer, which is running on the Ubuntu workstation, as illustrated in Figure 4.6.

The second TelosB mote shown in Figure 4.1 is called the mobile mote; it is serially connected to the Pololu 3pi. The purpose of the mobile mote is to wirelessly connect the 3pi with the DAG root mote. To achieve this, the mobile mote runs two applications that serve as a bridge between serial and radio communication. These two applications are called udpprint and udpinject as illustrated in Figure 4.6.

The udpprint applications runs on the mobile mote and is accessed on port 2189 of the mobile mote. When the mobile mote receives a UDP packet on port 2189

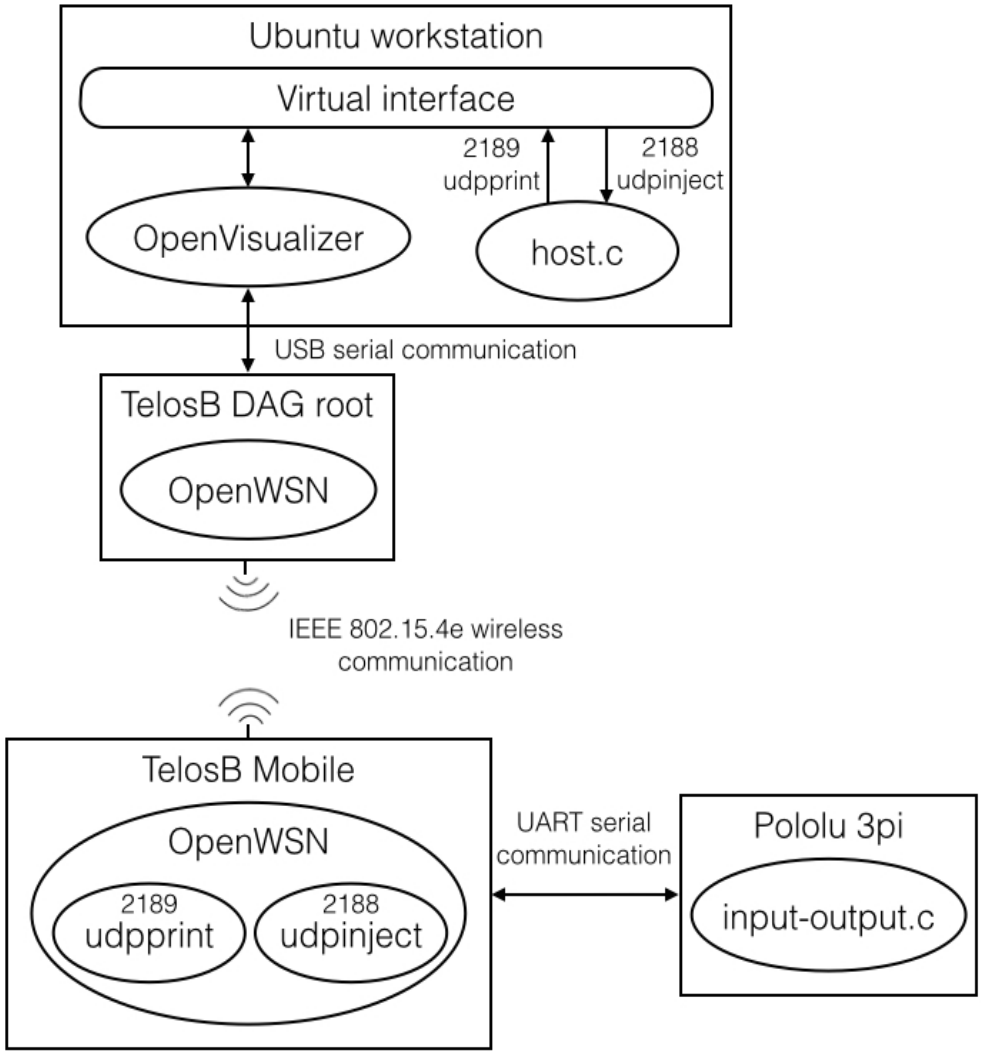


Figure 4.6: Software components implementing the wireless line following feedback control loop.

addressed to itself, the packet moves up the stack reaching the application layer. The data payload of the packet is then handled by the udpprint application with the `udpprint_receive` method. This method prints a data frame containing the packet payload over the TelosB UART1 serial pins, where the 3pi is connected and constantly reading. The udpprint application receives packets transmitted over the radio and forwards them to the serial pins. The udpprint application is implemented as shown in Appendix D.

The second application running on the mobile mote receives packets transmitted serially by the 3pi and forwards them over the radio to the DAG root mote. This udpinject application is accessed on port 2188 of the mobile mote. When the mobile mote receives a serial frame starting with the character 'U', the method `udpinject_trigger` is called. This method generates a UDP packet and adds to the payload of that packet the data received on the serial frame.

The mobile mote is constantly transmitting serial frames to the 3pi. As a consequence, the 3pi is running the program input-output.c that constantly reads bytes from the serial pins of the 3pi, builds a frame, checks the CRC, and acts according to the type of frame received. OpenWSN only allows for serial communication from the 3pi to the mobile mote to occur upon request from the mobile mote. This request is sent from the mobile mote using a request serial frame. Once the 3pi receives a complete request frame, the 3pi reads the line position and generates a udpinject frame that is transmitted serially to the mobile mote. The udpinject application is implemented as shown in Appendix E.

The 3pi can also receive from the mobile mote a data frame (generated by the udpprint application), which will contain a corrective action to be applied to the motors. When the 3pi receives a data frame and the frame passes the CRC check, the 3pi extracts the corrective action and applies the new speed to the motors. If the frame does not pass the CRC check, the frame is discarded.

The mobile mote can also transmit error frames to the 3pi, which can indicate that the mobile mote got desynchronized, that there was a CRC check failure, or other errors. The 3pi does not perform any action when an error frame is transmitted, but it keeps a counter of the number of times this happened. Besides the error, data and request frames, the 3pi should also be receiving status frames, which contain information about the OpenWSN network. The 3pi does not take action to change the status of the OpenWSN network, so status frames are ignored by the 3pi.

The 3pi reads the line position and applies the corrective action, but the controller is the component that calculates the corrective action based on the line position. The controller is implemented on the host, which is a C program running at the Ubuntu workstation. The virtual interface raised by OpenVisualizer provides the host with access to the mobile mote and the 3pi. To achieve this, the host generates two UDP sockets with the destination address of the mobile mote and the ports of the udpprint and udpinject applications, as illustrated on Figure 4.6. The socket connecting to the udpprint application will communicate with the 3pi and the host in the direction from the host to the 3pi. The socket connecting to the udpinject application will communicate in the direction from the 3pi to the host. Using two sockets for different communication directions and on different ports of the same destination address, is appropriate since the communication between the host and the 3pi is asynchronous. The 3pi does not wait for a corrective action from the host before reading a line position. This means that the 3pi can send a line position while the host sends a corrective action.

The process to achieve a wireless implementation of line following via feedback control using OpenWSN is illustrated with the sequence diagram in Figure 4.7. The 3pi receives a request frame from the mobile mote once every Δ ms. After the 3pi receives the request frame, the 3pi reads from its reflectance sensors and generates a line position, which is represented as an integer in the range of -2000 to 2000.

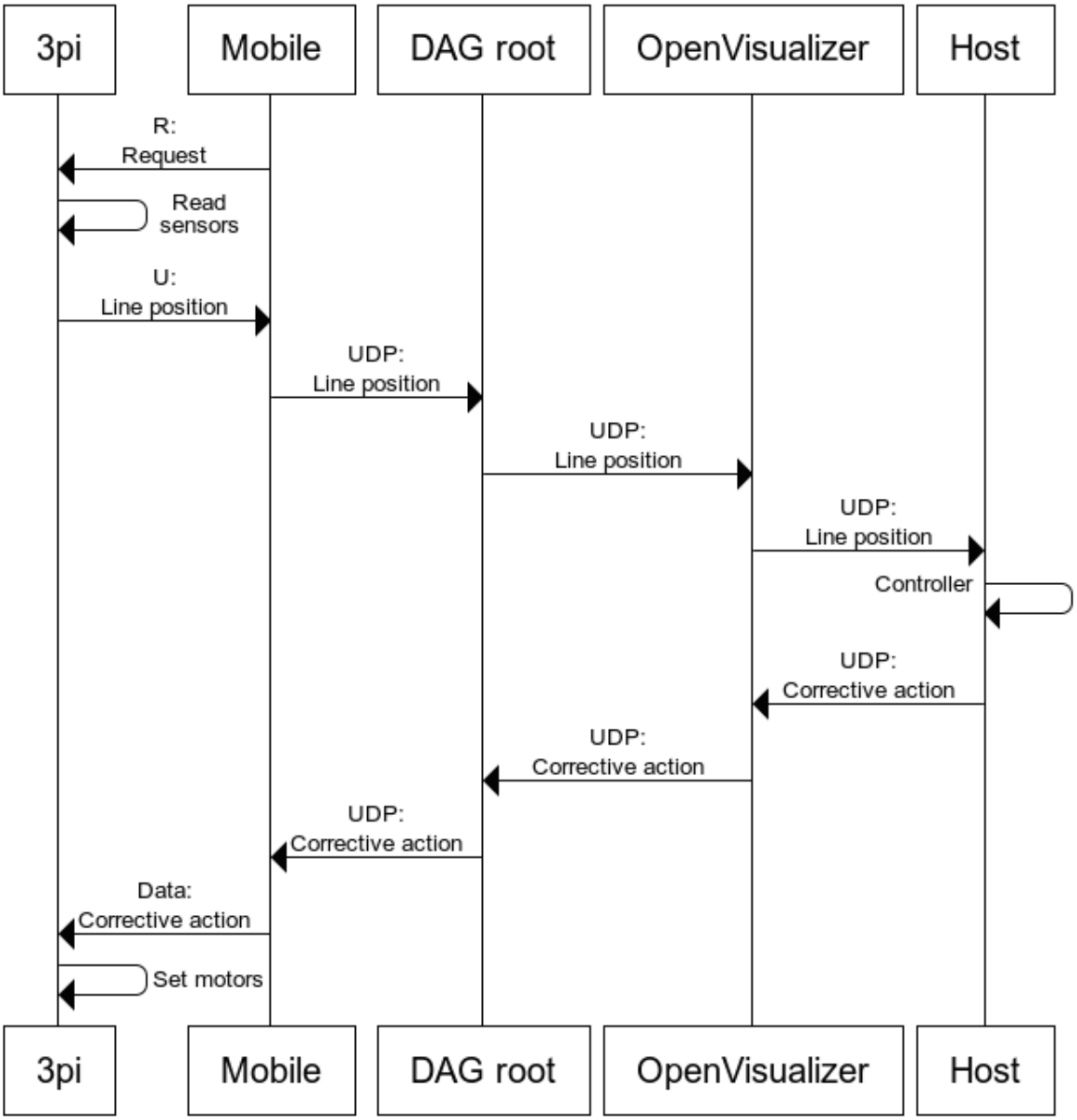


Figure 4.7: Sequence diagram for one iteration of the wireless line following feedback control loop.

The line position is transmitted serially to the mobile mote, which generates a UDP packet with the line position and injects the packet into the OpenWSN network using the udpinject application. This UDP packet containing the line position is addressed to the host. As a consequence, when the DAG root receives the UDP packet, it is forwarded serially to OpenVisualizer. OpenVisualizer forwards the UDP packet over the virtual interface to the socket connected at the udpinject port on the host. After a line position is successfully received at the host, the host runs the controller and calculates the corrective action. The corrective action is the power difference that should be applied to the motors, and is represented with an integer in the range of -40 to 40. A UDP packet containing the corrective action is created. Note that even though there is only one integer transmitted, there are two corrective actions applied, since each motor gets a different setting based on the same power difference number. The UDP packet is then transmitted back over the udpprint socket connected to OpenVisualizer on the virtual interface. OpenVisualizer then forwards the UDP packet to the DAG root mote, which injects the packet on the OpenWSN network. The packet is then received by the mobile mote and printed serially to the 3pi. Once the 3pi successfully receives a corrective action, the 3pi applies the new speed setting to its motors.

4.2 Model Application

The five network entities implementing wireless line following are shown in Figure 4.8. The SFRT of this system can be estimated by applying Equation (3.1) of the wireless SFRT model, which defines the SFRT as follows:

$$\text{SFRT} = \sum_{m \in E} \text{WCDTE}_m + \max_{m \in E}(\text{WDTimeE}_m - \text{WCDTE}_m) \tag{4.7}$$

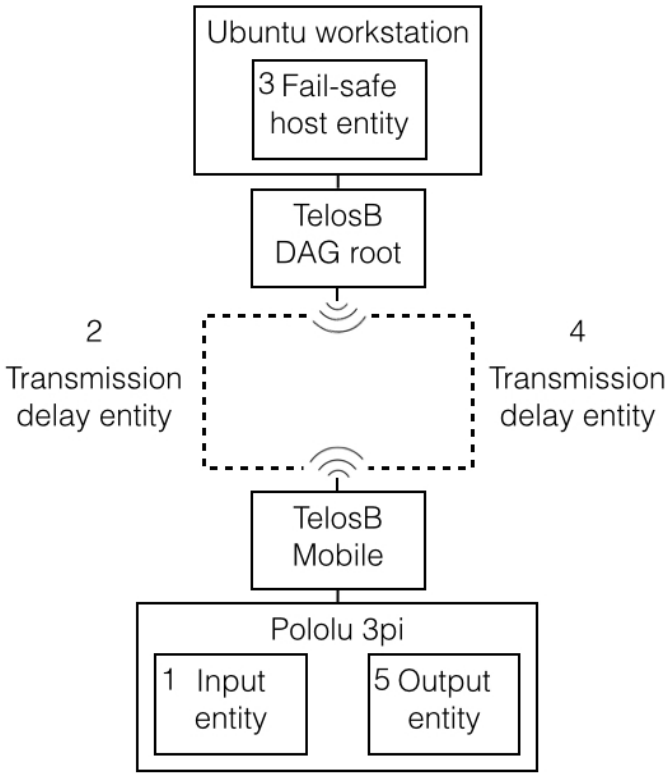


Figure 4.8: Five network entities implementing wireless line following.

For the system in Figure 4.8, $E = \{I, H, O, D\}$, $I = \{1\}$, $H = \{3\}$, $O = \{5\}$ and $D = \{2, 4\}$. Given that I, H and O have only one element, $\text{WCDTE}_m = \text{WCDT}_i$, when m is I, H, or O, and $i \in I$, $i \in H$, or $i \in O$, respectively. Equation (4.7) can be simplified for the case of single input single output systems as follows:

$$\text{SFRT} = \sum_{i=1}^{5} \text{WCDT}_i + \max_{i=1}^{5}(\text{WDTime}_i - \text{WCDT}_i) \tag{4.8}$$

where WCDT_i is the worst case delay time of entity i, WDTime_i is the watchdog timer of entity i, and $i = 1$ for the input entity (with 5 reflectance sensors attached), $i = 2$ for the transmission delay entity where the input entity is the sender and the host the destination, $i = 3$ for the fail-safe host entity, $i = 4$ for the transmission delay entity where the host is the sender and the output entity the destination, and $i = 5$ for the output entity (with two micro metal gearmotors attached).

To calculate the SFRT of the system in Figure 4.8, the components of the worst case delay times and watchdog timers of each participating entity are determined, and entities should meet the assumptions made by the wireless SFRT model.

4.2.1 Worst Case Delay Times

The wireless SFRT model provides Equations (3.4) to (3.12) to estimate the worst case delay times of the network entities implementing a wireless feedback control loop. The worst case delay times are modelled as the sum of the processing and waiting times incurred by the entities.

The wireless SFRT model assumes that processing and waiting times are input to the model. In a real experimental set-up, processing and waiting times can be precisely defined and experimentally measured.

4.2.1.1 Input Entity

The wireless SFRT model defines the worst case delay time of input entity i in Equation (3.5) as follows:

$$\mathrm{WCDT}_i = W_i + P_i + \sum_{j=1}^{N_s^i} P_s^j, i \in I \tag{4.9}$$

where W_i is the longest waiting time of input entity i, P_i is the worst case processing time of input entity i (including the time to read the available sensor value(s) and create a packet for transmission), N_s^i is the number of sensors attached to input entity i, and P_s^j is the worst case processing time of sensor j.

In the wireless line following experiment, entity 1 (input), shown in Figure 4.8, reads the line position in the 3pi. The 3pi only sends a new line position after a request from the mobile mote is received. This means that the waiting time at the input entity is defined as the time between two received request frames. According

to the OpenWSN schedule implementation, this happens at the beginning of every CELLTYPE_SERIALRX timeslot. Based on the slotframe with 8 timeslots each with a duration of 15 ms, a CELLTYPE_SERIALRX timeslot occurs every 120 ms. This means that the 3pi waits 120 ms before generating a new line position, giving $W_1 = 120$ ms for entity 1 (input) in the wireless line following system. This waiting time also provides the update interval of the feedback control loop.

After the 3pi receives a request frame from the mobile mote, the 3pi calls the method `hdlcRes_request` which performs the necessary operations to respond to the request frame received. This includes reading the line position and building the serial frame to be transmitted back to the mobile mote. This serial frame contains the line position, the sequence number and logging information, and should be built using the corresponding High-Level Data Link Control (HDLC) serial communication implementation. The processing time P_1 corresponds to the processing time of the `hdlcRes_request` method. The implementation of the `hdlcRes_request` method is shown in Appendix F.

Even though there are 5 reflectance sensors attached to the 3pi, the Pololu Corporation C library provides an abstraction that allows the line position to be read with one call to the method `read_line`. This method reads from all the reflectance sensors and returns an integer from -2000 to 2000 that represents the line position. Calling this method directly unifies the processing times from the 5 reflectance sensors, making $N_s^1 = 1$ and P_s^1 the processing time for the `read_line` method. This `read_line` method is called by the `hdlcRes_request` method. As a consequence, for the wireless line following experiment, the processing time P_1 includes the processing time P_s^1. Processing time P_1 was experimentally measured as reported in Section 5.3.

4.2.1.2 Fail-safe Host Entity

The wireless SFRT model defines the worst case delay time of the fail-safe host entity i in Equation (3.6) as follows:

$$\text{WCDTE}_H = \text{WCDT}_i = W_i + C_i + P_i, i \in H \tag{4.10}$$

where W_i is the longest waiting time of the fail-safe host entity, C_i is the worst case processing time of the fail-safe host entity to execute the controller, and P_i is the worst case processing time of the fail-safe host entity and includes the time to create a packet for transmission.

In the wireless line following experiment, entity 3 (fail-safe host), shown in Figure 4.8, is implemented in the host running on the Ubuntu workstation. The waiting time at the host is defined as the time between two received line positions. Since the update interval provided by the 3pi is 120 ms, the host should wait 120 ms between each new line position. This means $W_3 = 120$ ms for entity 3 (fail-safe host) in the wireless line following system.

Every time the host receives a new line position, the host runs the controller which is implemented in the `control_calculate_action(line_position)` method (see Appendix A). This method computes the derivative, integral and proportional values based on the line position received from the 3pi, and returns the calculated corrective action. The processing time of the `control_calculate_action` method corresponds to C_3 and is measured experimentally as reported in Section 5.3.

The corrective action generated by the `control_calculate_action` method is added to a message along with the corresponding sequence number by the `act` method. The message should use the same sequence number received with the line position that was used to calculate such corrective action. The `act` method builds and transmits the message back to the 3pi, and is shown in Appendix G. The pro-

cessing time of the `act` method corresponds to P_3 and is measured experimentally as reported in Section 5.3.

4.2.1.3 Output Entity

The wireless SFRT model defines the worst case delay time of output entity i in Equation (3.9) as follows:

$$\text{WCDT}_i = W_i + P_i + \sum_{j=1}^{N_a^i} P_a^j + \max_{j=1}^{N_a^i} A_j, i \in O \tag{4.11}$$

where W_i is longest waiting time of output entity i, P_i is the worst case processing time of output entity i and includes the time to open and read the new packet recently received, N_a^i is the number of actuators attached to output entity i, P_a^j is the worst case processing time of actuator j that includes receiving the corrective action from the output entity, and A_j is the worst case action time of actuator j (i.e. the longest time actuator j takes to apply the corrective on the system).

In the wireless line following experiment, entity 5 (output), shown in Figure 4.8, is implemented in the 3pi, which has two attached motors, thus $N_a^5 = 2$. The output entity implements corrective actions sent by the host. This means that the waiting time at the output entity is defined as the time between two received data frames with a new corrective action. Since the waiting time at the host is 120 ms, the host can only generate a new corrective action at most once every 120 ms. As a consequence, $W_5 \geq 120$ ms for entity 5 (output) in the wireless line following system, and depends on the processing time at the host.

When the 3pi receives a corrective action, the `hdlcRecv_data` method is called to handle the recently received data frame. This method performs the required HDLC operations and checks for the CRC. If the CRC is correct, the corrective action is extracted from the frame and applied to both of the motors. The processing time P_5

corresponds to the processing time of the `hdlcRecv_data` method and is measured experimentally as reported in Section 5.3. The implementation of the `hdlcRecv_data` method is shown in Appendix H.

The motors are set using the `set_motors(speed1, speed2)` method, which includes P_a^j and A_j for both actuators. The method `set_motors` is called at the `hdlcRecv_data` method. For the wireless line following experiment, a simplifying assumption is made to include the processing times P_a^j and A_j, for $j = 1, 2$, in processing time P_5.

4.2.1.4 Transmission Delay Entities

Wireless transmission delays are observed in the wireless communication links that connect the devices in the system. These wireless communication links are accessed by the TelosB motes, which operate as part of an OpenWSN network. OpenWSN implements the TSCH mode of the IEEE 802.15.4e standard at the MAC layer. As a consequence, there is an IEEE 802.15.4e network operating in TSCH mode, as assumed by the wireless SFRT model. The TSCH mode operates with a schedule that is defined as a slotframe repeating cyclically in time. A slotframe is a collection of timeslots, where one timeslot is a defined period of time during which a pair of devices can exchange a frame and an acknowledgement.

An assumption made by the wireless SFRT model is that the network operates in a star topology. In the experimental set-up, there are only two devices, i.e. the DAG root mote and a mobile mote, accessing the IEEE 802.15.4e network. The PAN coordinator role is assumed by the DAG root mote and there is one communication hop between the PAN coordinator and the mobile mote.

Another assumption made by the wireless SFRT model is that propagating new data is preferred over guaranteed delivery. As a consequence, there are no retransmissions and each packet is only given one try to be successfully transmitted. OpenWSN

allows configuration of the number of attempts to transmit a packet with the variable `TXRETRIES`. This variable is set to 1 to allow for only one attempt of transmission and no retransmissions.

One of the most important assumptions made by the wireless SFRT model concerns the network schedule. The model defines a directed communication link as a sender-destination pair of devices, and the model assigns one dedicated or guaranteed timeslot to each directed communication link. There is one directed communication link from the mobile mote to the DAG root (from the 3pi to the host), and a second directed communication link from the DAG root to the mobile (from the host to the 3pi). Each of these directed communication links requires one guaranteed timeslot.

In OpenWSN the communication schedule is defined and processed when starting the network. The schedule is included in the OpenWSN firmware that runs in each of the TelosB devices where a timeslot is added using the method `schedule_addActiveSlot (slotOffset, type, shared, channelOffset, neighbor)`. The first argument indicates the number of the timeslot being added, and the second argument indicates the type of the timeslot, i.e. advertisement, shared, guaranteed, or serial. The third argument `shared` is a boolean that indicates if the timeslot is shared or not. The fourth argument `channelOffset` indicates the frequency channel offset for the timeslot, and is used for channel hopping. The last argument is used for guaranteed timeslots to indicate the IP address of the second device participating in the timeslot.

Based on the definition and implementation of the network schedule provided by OpenWSN, for a given value of the `slotOffset` argument, a guaranteed timeslot is added at the sender of the directed communication link by specifying type CELLTYPE_TX on the second argument of the `schedule_addActiveSlot` method. The `shared` boolean on the third argument should be set to false, and the last argument should indicate the IP address of the destination. The fourth argument `channelOffset` is set to the default value 0, since the frequency channel for a trans-

mission is calculated dynamically by OpenWSN at the beginning of each timeslot. For the same value of the `slotOffset` argument, the destination should have a corresponding CELLTYPE_RX timeslot, with the `shared` boolean set to false, and the last argument specifying the IP address of the sender.

Using the `schedule_addActiveSlot` method with the correct arguments, two guaranteed timeslots are added to the schedule; one where the sender is the mobile mote and the destination is the DAG root mote, and a second timeslot where the sender is the DAG root mote and the destination is the mobile mote. An advertisement timeslot is also added to the schedule using the CELLTYPE_ADV on the `type` argument of the `schedule_addActiveSlot` method. The resulting schedule meets the wireless SFRT model assumption. OpenWSN, however, has its own set of schedule requirements that should be met in order to get a proper OpenWSN network.

OpenWSN uses shared timeslots to transmit important network information such as routing structures. As a consequence, a requirement of OpenWSN is that the slotframe defining the schedule of the network should have at least one shared timeslot. A second requirement is that the slotframe should have at least one off timeslot, where an off timeslot is a timeslot that has not been allocated using the `schedule_addActiveSlot` method. A third requirement is regarding serial communication. OpenWSN does not perform serial communication in the background and requires reserved time to deal with serial activity. This is done during serial timeslots, which are defined with the `schedule_addActiveSlot` method where the second argument `type` has the value of CELLTYPE_SERIALRX. One timeslot with type CELLTYPE_SERIALRX has a length of three times a regular timeslot, which translates to three serial timeslots.

Based on the schedule requirements of OpenWSN and considering that the wireless SFRT model requires two guaranteed timeslots, the minimum number of times-

lots in one slotframe required to implement the wireless SFRT model using OpenWSN is eight; 1 advertisement timeslot, 1 shared timeslot, 2 guaranteed timeslots, 3 serial timeslots, and 1 off timeslot. By default, each timeslot in OpenWSN has a duration of 15 ms. This means that the minimal slotframe of eight timeslots has a total duration of 120 ms.

In OpenWSN the order of the timeslots in the schedule is known beforehand because the `schedule_addActiveSlot` method is called sequentially and the schedule is processed before starting the network. In the wireless line following experiment, the line position from the 3pi is required to calculate the corrective action at the host. As a consequence, assigning the first guaranteed timeslot to the communication link from the 3pi to the host makes sense.

Even though the order of the timeslots influences the moment at which a device accesses the wireless channel, the order of the timeslots does not affect the worst case delay time of wireless transmissions, since the worst case delay time is the same for all transmission delays regardless of the communication link and timeslot in which they are observed.

The worst case delay time of a transmission delay entity i is defined by the wireless SFRT model in Equation (3.11) as follows:

$$\mathrm{WCDT}_i = N_{\mathrm{ts}} L_{\mathrm{ts}}, i \in D \tag{4.12}$$

where N_{ts} is the number of timeslots in the slotframe, and L_{ts} is the total duration in time of one timeslot.

The wireless SFRT model defines the worst case delay time of the k_D transmission delay entities in Equation (3.12) as follows:

$$\mathrm{WCDTE}_D = k_D \mathrm{WCDT}_i, i \in D \tag{4.13}$$

where k_D is the total number of transmission delay entities in the network, and WCDT_i is the worst case delay time of the transmission delay entity i.

Considering the OpenWSN network in the wireless line following experiment, $k_D = 2$, $N_{\text{ts}} = 8$, and $L_{\text{ts}} = 15$ ms. Using these values in the wireless SFRT model equations, for transmission delay entity i, where $i \in D$, the following estimates are obtained:

$$\text{WCDT}_i = N_{\text{ts}} L_{\text{ts}} = 8 \times 15 = 120 \text{ ms} \tag{4.14}$$

$$\text{WCDTE}_D = k_D \text{WCDT}_i = 2 \times 120 = 240 \text{ ms} \tag{4.15}$$

Equation (4.14) indicates that $\text{WCDT}_2 = \text{WCDT}_4 = 120$ ms for entities 2 and 4 (transmission delays) in the wireless line following system. This means that in the worst case, a wireless transmission will be completed after 120 ms since the moment the packet was ready to be transmitted. This applies for successful transmissions, since the wireless SFRT model specifies no retransmissions. In the wireless line following experiment, a transmission from the 3pi to the host will take at most 120 ms, and similarly a transmission from the host to the 3pi will take at most 120 ms.

Equation (4.15) indicates the total worst case delay time of all the wireless transmission delays. This 240 ms will contribute to the total worst case delay time of the system, which is required for the calculation of the SFRT as shown in Equation (3.1).

4.2.2 Watchdog Timers

Watchdog timers are implemented as a superposed countdown timer in all participating network entities. Upon the expiration of the local timer, the entity abandons normal operation and activates its safe reaction to reach a safe state. The watchdog timer is a design decision that should be taken based on the study of how critical and fast the control loop needs to be.

4.2.2.1 Device Entities

Consistent with the IEC 61784-3-3 standard, the wireless SFRT model defines the watchdog timers of devices, i.e. input, fail-safe host and output, entities as the one fault delay time OFDT, i.e. the worst case delay time in case of a fault within the entity (see Equation (3.13)). The OFDT is implementation dependent and is considered input to the model.

The idea of the watchdog timer is to have some supervision over the operation of the entity. If the entity is not responding as expected, the watchdog timer expires and drives the entity into a safe state. For example, if the output entity does not apply a corrective action within a certain amount of time, control of the system can be lost. The output entity might be unresponsive for different reasons, e.g. the corrective action was never received or the entity received a damaged packet and could not apply the corrective action. Instead of losing control of the system, the watchdog timer forces the output entity to reach a safe state. For the wireless line following experiment, a safe state for the output entity could be setting the motors to 0 rpm.

Considering the idea of a watchdog timer that supervises the entire operation of the entity, the watchdog timer is set to expire on each entity at the worst case delay time of the entity plus a certain factor for flexibility. The watchdog timer will expire if the point of reset has not been reached by the entity during the expiration time of the watchdog timer. Each entity has a reset point on which the watchdog timer is reset, which happens when the entity is responding as expected. The reset should occur every time the entity successfully finishes one cycle.

For the input entity one cycle includes the waiting for a request frame, reading a line position, and transmitting the packet. Based on this cycle, the watchdog timer is reset on the input entity every time a new line position is sent. At the host, the cycle includes waiting for a line position, and calculating and transmitting the

corrective action. Considering this, the watchdog timer is reset every time a new corrective action is sent. The output entity cycle includes the waiting, processing and application of the corrective action, so the watchdog timer is reset every time a new corrective action is applied.

The time between two line positions being sent at the input entity, the time between two corrective actions sent at the host, and the time between two corrective actions applied at the output entity is measured experimentally (see Section 5.3) to obtain what the expiration values of the watchdog timers should be.

4.2.2.2 Transmission Delay Entities

The wireless SFRT model defines the watchdog timer of a transmission delay entity i in Equations (3.15) and (3.14) as follows:

$$\text{WDTime}_i = \text{F\_WD\_Time}_{i,j} + \text{WCDT}_i \tag{4.16}$$

and

$$\text{F\_WD\_Time}_{i,j} = c_1(P_j + L_{\text{ts}}) \tag{4.17}$$

where $i \in D$, $j \in I$ or $j \in H$, $\text{F\_WD\_Time}_{i,j}$ is the fail-safe watchdog timer of transmission delay entity i where the sender initializing communication is entity j, P_j is the protocol processing time of sender entity j, L_{ts} is the total length of one timeslot, $c_1 \geq 1$ is a constant, and WCDT_i is the worst case delay time of transmission delay entity i.

For the wireless line following experiment, for entity 2 (transmission delay), the sender initializing communication is the input entity, and for entity 4 (transmission delay), the sender is the fail-safe host entity. Processing P_j for the input and fail-safe host entities are determined experimentally, and $L_{\text{ts}} = 15$ ms. The IEC 61784-3-3 standard recommends c_1 values in the range $1 \leq c_1 \leq 1.3$. The higher the value

of this constant, the longer the watchdog timer expiration time is, which allows for more flexibility. The worst case delay time WCDT_i for transmission delay entity i, where $i \in D$, is determined for the wireless line following experiment in Equation (4.14), and is 120 ms.

Wireless transmission delays are observed in the wireless communication links. Since the wireless communication links are abstract and cannot perform any action, the fail-safe watchdog timer of transmission delay entity F_WD_Time is implemented by the sender of the communication link. The F_WD_Time supervises the operation of the communication link only and is reset every time an acknowledgement is received after a new packet has been sent.

In the TSCH mode of IEEE 802.15.4e, one timeslot includes the exchange of an acknowledged frame. This means that the successful transmission of one frame and the reception of the corresponding acknowledgement is known to occur during one timeslot. As a consequence, there is no need to experimentally measure the time since a frame has been sent until the corresponding acknowledgement has been received.

If a transmission fails, and the watchdog timer at the destination does not account for failed transmissions, the watchdog timer at the destination will expire. This means that the watchdog timer at the device entities are monitoring the communication channel. Note that this is consistent with the idea of a watchdog timer that supervises the entire operation of device entities, as it includes the operation of the communication channel and waiting times. As a consequence, the fail-safe watchdog timer of transmission delay entities is not directly implemented nor measured in the wireless line following experiment.

4.3 Experimental Design

The main objective of the experimental validation is to apply the wireless SFRT model in a real experiment to study the usability and accuracy of the model. A wireless implementation of a real feedback control loop also provides a good opportunity to evaluate the performance of wireless communication, specifically OpenWSN, when fast reliable communication is required to maintain control of a system.

The experimental validation is divided in two parts. The first part evaluates the performance of wireless communication when implementing the line following feedback control loop. The second part is the application of the wireless SFRT model to the wireless line following feedback control loop to estimate the SFRT of the system.

4.3.1 Performance Metrics

The objective of the line following feedback control loop is to keep the 3pi centered on the line. The performance metric of this activity should measure how well or how badly the 3pi follows the line.

The proportional term P of the PID line following controller can have values in the range $-2000 \leq P \leq 2000$. When the 3pi is centered on the line, $P = 0$. When the line is on the left of the 3pi, $P < 0$. When the line is on the right of the 3pi, $P > 0$. The performance metric measuring the error of the line following activity is defined as $e = |P|$, where P is the error communicated to the controller in Figure 4.3. The error of the line following activity e can have values in the range of $0 \leq e \leq 2000$. The error e is calculated at the host every time a new line position is received.

Since the terms P and e do not have units, function $d(e)$ is defined to calculate the corresponding distance in cm from an input error e based in the distance between

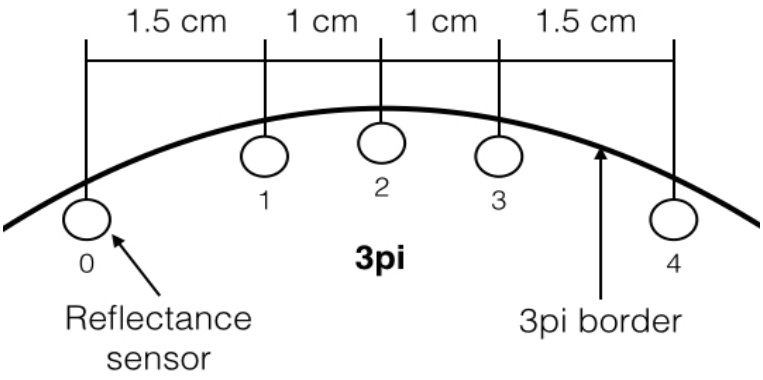


Figure 4.9: Position of the 5 reflectance sensors on the Pololu 3pi.

the 5 reflectance sensors in the 3pi, which span a total of 5 cm from sensor 0 to sensor 4, as illustrated in Figure 4.9. Function d is defined as follows:

$$d(e) = \begin{cases} 0.001e & \text{if } e \leq 1000 \\ 1 + 0.0015(e - 1000) & \text{if } e > 1000 \end{cases} \tag{4.18}$$

The center of the 3pi is sensor 2, where $e = 0$ and $d(e) = 0$ cm. If the line is under sensor 1, $e = 1000$ and $d(e) = 1$ cm, which indicates the line is 1 cm away from the center of the 3pi.

The performance measures of the line following feedback control loop are defined as the average, standard deviation, and maximum value of the error e. The smaller the error, the better the 3pi performs at line following. These performance measures are calculated at the host and reported at the end of a test in a log file.

To evaluate if an obtained value of e is acceptable, a comparison should be established with other values of e. One goal for wireless communication is to achieve the same level of communication and performance as achieved by wired technologies. As a consequence, the wireless line following experiment is compared with a wired line following experiment.

PROFIBUS [24] is a well-known wired communication fieldbus protocol. PROFIBUS operates on a master/slave architecture, where the slave sends information

to the master upon request, and the master sends commands to the slave. The wired line following implementation simulates the master/slave communication and architecture present in the PROFIBUS communication protocol. In the wired line following experiment, the host assumes the role of the master device, and the 3pi is the slave. Every certain period of time, the host requests a line position from the 3pi. The 3pi reads the line position and sends it to the host. Once the host receives the line position, the host calculates the corrective action and sends a command to the 3pi with the corrective action. Then, the 3pi implements the corrective action. Wired communication between the 3pi and the host is achieved with a USB to mini-USB serial cable. The mini-USB end is connected to the Pololu USB AVR programmer that translates USB signals to the UART RX and TX lines on the 3pi. Appendix C illustrates the wired line following experimental set-up.

Another line following experiment that is worth studying is on board control as provided by the Pololu Corporation C library. Since on board control does not include any kind of communication channel because all entities are implemented in the same device, on board control is not comparable with either the wired or wireless implementations. On board control can, however, be used as an indicator of the best achievable performance.

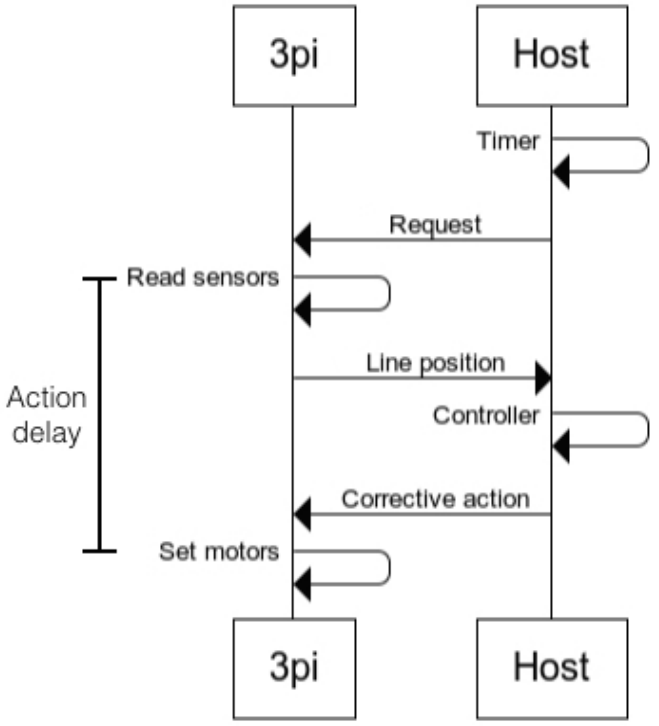
Besides measuring the line following performance metric e, evaluating other communication metrics is important, since they affect the performance of the line following activity. To evaluate the performance of the communication channel, the packet loss PL and the action delay AD are also measured and logged.

The packet loss PL is calculated for both communication links; from the 3pi to the host, and from the host to the 3pi. For the first communication link, the number of packets lost is calculated by subtracting the number of line positions received at the host from the number of line positions sent by the 3pi. The packet loss percentage is then calculated based on the total number of line positions sent from

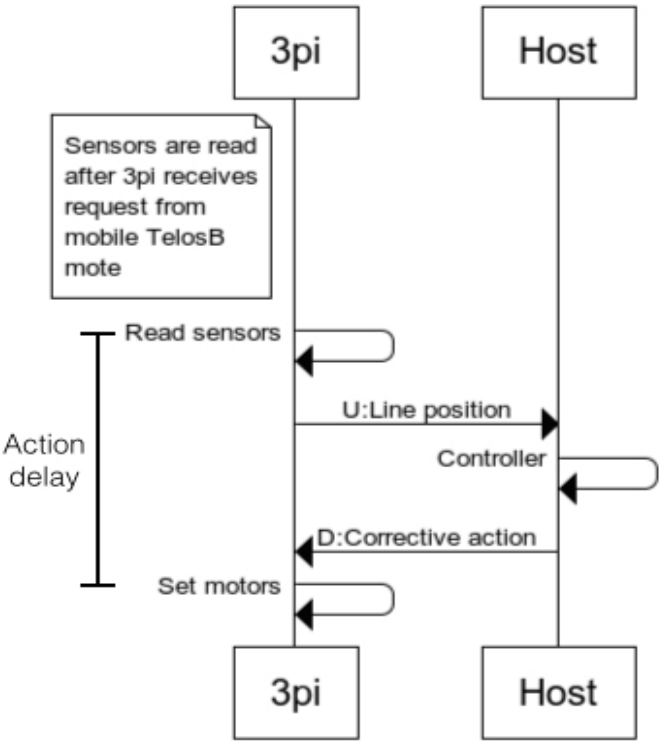
the 3pi. Similarly, for the second communication link, the number of packets lost is calculated by subtracting the number of corrective actions received at the 3pi from the number of corrective actions sent by the host. The packet loss percentage is then calculated based on the total number of corrective actions sent from the host. At the end of one test, the 3pi sends the total number of line positions sent to the host, and the total number of corrective actions received from the host. The host also keeps its own counter for the number of line positions received and the number of corrective actions sent. After the host receives the counters from the 3pi, the host performs the required operations to calculate the packet loss for each of the communication links.

The action delay AD is defined as the time since a line position is read until the time its corresponding corrective action is applied. The line position is read at the 3pi, and the corrective action is also applied at the 3pi. As a consequence, the 3pi can measure the action delays by taking a timestamp when a line position is read, and taking a second timestamp when the corrective action corresponding to such line position is applied. Subtracting both timestamps will result in the time elapsed since the line position was read until the corresponding corrective action was applied. The sequence diagrams in Figure 4.10 show the interactions between the 3pi and the host in the wired and wireless implementations of the line following experiment. The diagram also indicates the time frame that is measured by the action delay in both implementations.

On every test, there are many action delays that need to be reported. The 3pi has limited resources and cannot store in memory all the action delays measured in one test. As a consequence, the 3pi needs to send the action delays as they are calculated. To achieve this, every time the 3pi sends a line position message to the host, and there is an action delay to be reported, the 3pi appends the action delay to the end of the line position message. When the host receives a message that contains more data than one line position and a sequence number, the host parses the rest of



(a) Wired implementation.



(b) Wireless implementation.

Figure 4.10: Sequence diagrams illustrate the interactions between the 3pi and the host in two implementations of the line following experiment.

Table 4.1: Worst case delay times and watchdog timers for the 5 network entities implementing the line following experiment.

i	Entity	F_WD_Time$_{i,j}$	WCDT$_i$	WDTime$_i$
1	Input	NA	$c_2(W_1 + P_1)$	OFDT$_1 = c_3(W_1 + P_1)$
2	TD	$c_1(P_1 + L_{\text{ts}})$	$N_{\text{ts}}L_{\text{ts}}$	F_WD_Time$_{2,1}$ + WCDT$_2$
3	F-Host	NA	$c_2((c_4 + 1)W_3 + P_3)$	OFDT$_3 = c_3((c_4 + 1)W_3 + P_3)$
4	TD	$c_1(P_3 + L_{\text{ts}})$	$N_{\text{ts}}L_{\text{ts}}$	F_WD_Time$_{4,3}$ + WCDT$_4$
5	Output	NA	$c_2((c_4 + 1)W_5 + P_5)$	OFDT$_5 = c_3((c_4 + 1)W_5 + P_5)$

the message as action delays. At the end of the test, the host has all reported action delays, and calculates the average and standard deviation. These two values are also reported in the log.

4.3.2 Model Validation

The experimental validation of the wireless SFRT model consists of two sets of experiments. The first is a set of calibration experiments. During these experiments the waiting and the maximum processing times are measured. These waiting and processing times are used as shown in the expressions in Table 4.1 to calculate the worst case delay time WCDT$_i$ and watchdog timer WDTime$_i$ of entity i, where $1 \leq i \leq 5$. For MIMO systems with n entities, the expressions in Table 4.1 are still applicable, with the difference that there would be n rows and $1 \leq i \leq n$. All input entities share the same expressions and this also applies for transmission delays and fail-safe host entities.

Processing times are subject to changes in the state of the processes and other tasks carried out by processors. To accommodate for variations in the processing times incurred by device entities, constant $c_2 \geq 1$ is introduced as a factor that multiplies the worst case delay time estimation provided by the wireless SFRT model (see Table 4.1). Similarly to constant c_1 introduced by the IEC 61784-3-3 standard, c_2 values are recommended in the range of $1 \leq c_2 \leq 1.3$.

Watchdog timers should reflect how critical the control in the system is. There

might be systems where one failed entity cycle (where the cycle consists of waiting and processing) is acceptable, but two failed cycles are not acceptable. This depends on the type of control that the system is trying to achieve. To evaluate different watchdog timers, constants c_3 and c_4 are introduced. As shown in Table 4.1, constant c_3 multiplies the watchdog timer of device entities to allow for variations in processing times. The recommended range of values for c_3 is again $1 \leq c_3 \leq 1.3$. Since the watchdog timer should not expire if the entity is responding within its known worst case delay time, the condition $c_3 > c_2$ is imposed.

Constant c_4 is defined as the number of acceptable consecutive packets lost, and multiplies the waiting time of device entities, as shown in Table 4.1. One lost packet implies that the destination entity did not receive the required input, and counts as a failed cycle. Values for c_4 are positive integers in the range $c_4 \geq 0$. Control loops operating reliably with a high number of failed cycles, e.g. $c_4 \geq 10$, are unusual.

The set of calibration experiments provides estimates for the processing times incurred by device entities. These estimates are used to calculate worst case delay times, which in turn are used to calculate watchdog timer values. The calculated watchdog timer values are used and implemented on the second set of experiments where 6 watchdog timers are implemented with values of c_4 varying from 0 to 5. Different values for the c_4 constant are implemented to study how the control loop behaves with different numbers of acceptable failed cycles, and also to illustrate the use of constant c_4.

When a watchdog timer expires, the entity should be driven to a safe state. To avoid interrupting the execution of a test, when a watchdog timer expires in the line following experiment, a counter for this watchdog timer is increased. At the end of the test, the number of times each watchdog timer expired is reported in the log.

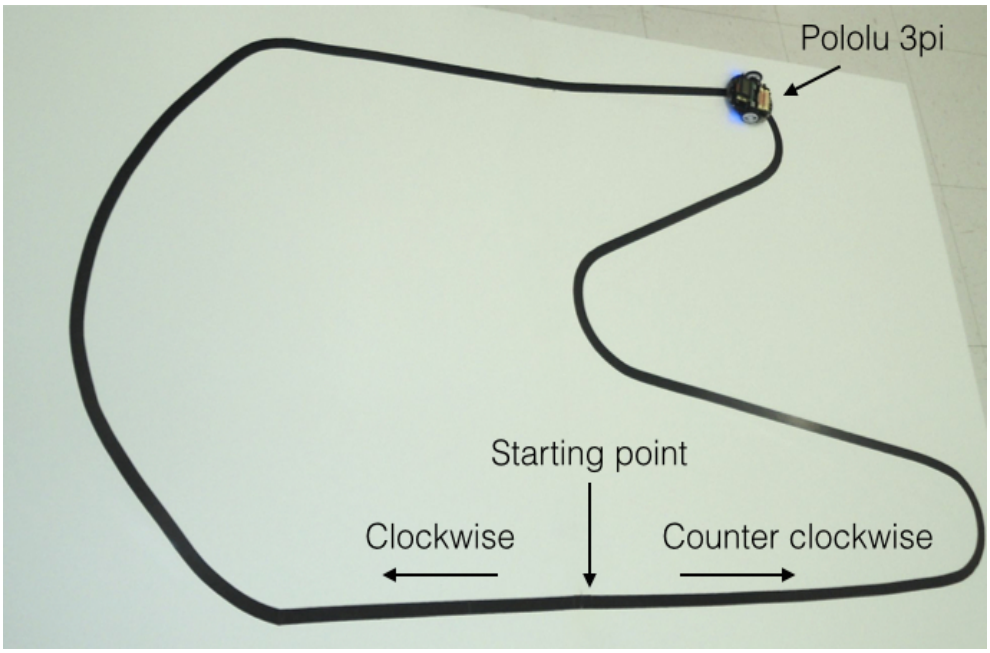


Figure 4.11: Line following course used in experiments.

4.3.3 Experiments

The line following course used in the experiments is shown in Figure 4.11. The course has an approximate length of 4.532 m, and the width of the line is 1.7 cm. This course provides different elements to test line following, such as sharp and wide turns, and straight lines. One test consists of the completion of one lap of the course from the starting point indicated in Figure 4.11. After one test is completed, the host has created and saved a log file with all the information relevant to the test. The results reported in the log consist of the number of packets sent and received at the host and 3pi, packet loss percentages, the average and standard deviation of the action delays, the average and standard deviation of the error e, the maximum value of error e, maximum processing and waiting times at the host, input and output entities, and the number of times each watchdog timer expired.

The 3pi is not aware of positioning and does not know where the course starts or ends, so the 3pi is placed at the starting point at the beginning of each test. After the test time duration has elapsed, the 3pi stops its motors, and sends the log

messages to the host. After all the expected messages are received at the host, the host calculates and reports the results and the total duration of the test, including the logging time.

The duration of one test is set to approximately the time the 3pi takes to finish one lap of the course, as measured in the calibration tests. Reporting the exact time the 3pi took to complete one lap is also an indicator of performance. To report this, the time taken by the 3pi to finish one lap is measured manually with a stopwatch and added to the set of results reported.

One test is repeated 3 times from the course starting point for the clockwise and counter clockwise directions, for a total of 6 tests. Considering clockwise and counter clockwise directions could help reduce the effect of physical errors in the 3pi, for example power and gear differences in the motors. These 6 tests comprise one complete experiment, which is done for each of the line following implementations; i.e. on board, wired, and wireless.

Chapter 5

Results

This chapter presents the results of worst case delay time and reliability measures for the on board, wired and wireless line following control for the experiments described in Chapter 4.

5.1 On Board Control

The line following implementation provided by the Pololu Corporation C library is used to run the on board control experiment. The Pololu Corporation line following implementation has the following default values for the proportional integral derivative (PID) control parameters; $K_P = -1/20$, $K_I = 1/10000$, and $K_D = 3/2$. The maximum corrective action is 60, and the maximum motor speed 120 rpm.

On board control means that the input, host and output entities are implemented in the same device; the 3pi. This means the communication channels and transmission delay entities are entirely contained within the 3pi board shown in Figure 4.2. Given that the 3pi has limited resources, a serial communication link was established from the 3pi to the host for logging purposes only.

Results for the on board control experiments are shown in Table 5.1, where each of the 3 tests in the clockwise (C) direction and in the counter clockwise (CC) direction

Table 5.1: Experimental results using on board control to achieve line following; $N \approx 12645$ for each of the six experiments.

Dir.	μ_e	d (cm)	σ_e	$\max_e$	μ_{AD} (ms)	T (s)	F (s)
C	190	0.2	168.4	828	1.9	24	23.2
C	183.3	0.2	169	814	1.9	24	21.6
C	194.6	0.2	174.9	898	1.9	24	23.2
μ_{C}	189.3	0.2	170.8	846.7	1.9	24	22.7
CC	202.5	0.2	173.2	849	1.9	24	23.1
CC	216.1	0.2	184	946	1.9	24	23.2
CC	208.8	0.2	178.8	974	1.9	24	23.2
μ_{CC}	209.1	0.2	178.6	923	1.9	24	23.2
$\mu_{\text{C,CC}}$	199.2	0.2	174.7	884.8	1.9	24	22.9

are included. Table 5.1 reports the average error μ_e, the average error expressed in cm d (using the conversion function $d(e)$ defined in Equation (4.18)), the error standard deviation σ_e, the maximum observed error $\max_e$, the average action delay μ_{AD} (ms), the duration of the test T (s), and the time to complete one lap of the course F (s). The number of samples $N \approx 12645$. These results are reported for each test in the C and CC directions. The average results for the C direction μ_{C}, the CC direction μ_{CC}, and the combined directions $\mu_{\text{C,CC}}$ are also reported.

All entities are implemented in the same device and communication channels involved are hard-wired into the 3pi board, so the control loop is running in the 3pi as fast as possible. The results provided using on board control constitute the best line following performance that can be achieved by the 3pi. Further tests revealed that this performance could be improved even more if serial logging was removed, reducing the average error by approximately 10%. Removing serial communication does not allow for any logging at the host and the standard deviation σ_e could not be reported. Considering that wired and wireless implementations keep a log at the host, using serial communication for logging on board control response provides a good base for comparison.

The action delay is defined as the time since a line position is read until the time its corresponding corrective action is applied. This indicates the delay between the

input and output entities. Input and output entities are implemented in the same device, so the action delay is measured as one iteration of the control loop where a new line position is read, and the corresponding corrective action is calculated and applied. For on board control, the average action delay reported in Table 5.1 corresponds to the duration of the test divided by the number of iterations performed. This provides the average time for one iteration. As shown in Table 5.1, one iteration takes approximately 2 ms. This also means that the update interval is approximately 2 ms, which means that there is a new line position read every 2 ms.

Applying function d to the average error $\mu_{\mathrm{C,CC}} = 199.2$ results in an average position of 0.2 cm. This means that on average the center of the 3pi was only 0.2 cm away from the line.

An average distance of 0.2 cm from the center of the 3pi to the position of the line is the best performance the 3pi can achieve. The behaviour observed under these conditions is a very smooth line following where the 3pi never loses sight of the line. This is reflected in the short average time of 23 s that the 3pi took to finish the course at a maximum motor speed of 120 rpm.

Even though the same course is used in all the tests, the 3pi can react differently for each test based on many factors. For example, even though the starting point is fixed, the exact angle and place in which the 3pi is placed at the beginning of the test might change slightly. Also, the sensors are calibrated on each test based on the current lighting of the room, and the movement of the motors can be affected by dust or other small particles accumulated on the course. Considering all these factors, it is very unlikely that the 3pi will perform exactly the same for 2 tests in the same direction of the same course. This explains the differences in the results reported by the 3 runs of the test in each direction. These factors could also explain differences between the results reported in the C and CC directions. On average, the 3pi performed better in the C direction, which is reflected in smaller values for

the average error μ_e, the average standard deviation, and the maximum error $\max_e$ for the C direction. In the eyes of the 3pi, the C and CC directions correspond to two different courses. The first one starts with a wide curve, and the second with a sharper turn. Even though there are the same number of curves and straight lines on one lap of the course regardless of the direction, the position on which the 3pi starts each segment of the course will affect the line positions readings from that point forward. The position at which the 3pi starts each segment of the course is not the same and not symmetric for both directions.

5.2 Wired Implementation

On board control provides a theoretical best performance because the communication channels involved are all on the same circuit board. To obtain more realistic results, the wired implementation of line following via feedback control consists on the 3pi, where the input and output entities are deployed, and the Ubuntu workstation, where the fail-safe host entity is deployed. The 3pi and the Ubuntu workstation communicate serially in a master/slave architecture, where the Ubuntu workstation acts as the master and the 3pi as the slave.

The wireless SFRT model can be applied to wired technologies, with the exception of equations for the worst case delay time and watchdog timer of transmission delay entities, where the TSCH mode of IEEE 802.15.4e is assumed. For the wired transmission delay entities operating over serial communication, the worst case delay time was estimated as the latency observed in the network. An experiment with 100 samples reports the average round trip latency $\mu_{\mathrm{RL}} = 3.5$ ms, with a standard deviation $\sigma_{\mathrm{RL}} = 0.06$ ms, and an average one way latency $\mu_{\mathrm{RL}}/2 = 1.7$ ms. The standard deviation σ_{RL} is small (less than 0.1 ms), so the worst case delay time of the transmission delay entity can be estimated as the average one way latency multiplied by

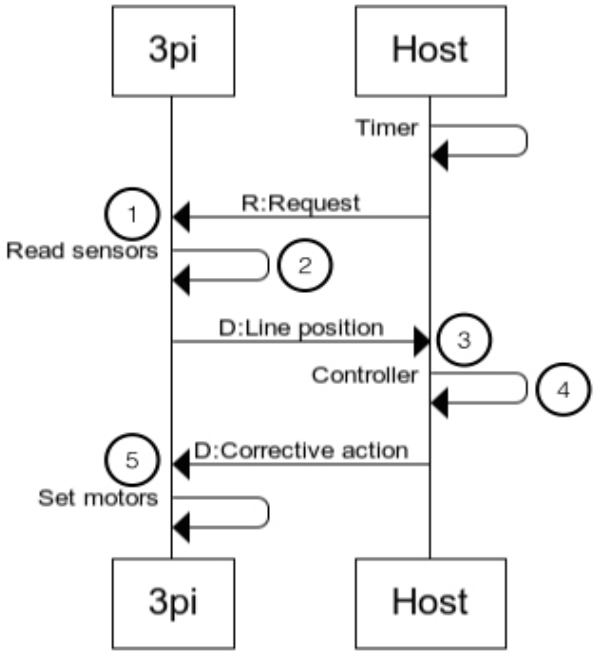


Figure 5.1: Sequence diagram for the wired implementation of line following control. Numbers indicate the steps that should be accounted for the timer at the host.

constant c_2. A similar approach is used when estimating the watchdog timer of the wired transmission delay, which is defined as the average one way latency multiplied by constant c_3.

In the wired implementation, since the 3pi only sends a line position upon a request from the host, the host dictates the update interval of the control loop. The update interval U is defined as the frequency at which the host requests a new line position from the 3pi, and is the time at which the timer in Figure 5.1 is set. Experimentally it was determined that for an update interval less than or equal to 6 ms, the host would send two or more requests to the 3pi before receiving the first line position. Ideally, the host will send a request and wait for the response from the 3pi before sending a second request. This allows for the next frame sent to the 3pi to be the corresponding corrective action. To achieve this, the update interval should account for the time since the request is sent to the 3pi, until the time the corrective action is sent. This process is indicated in steps 1 to 5 in Figure 5.1. Steps

Table 5.2: Expressions to calculate worst case delay times and watchdog timers for network entities in the wired implementation.

i	Entity	WCDT_i	WDTime_i
1	Input	$c_2((c_4+1)8+4)$	$\text{OFDT}_1 = c_3((c_4+1)8+4)$
2	TD	$c_2(1.7)$	$\text{OFDT}_2 = c_3(1.7)$
3	F-Host	$c_2((c_4+1)8+0.1)$	$\text{OFDT}_3 = c_3((c_4+1)8+0.1)$
4	TD	$c_2(1.7)$	$\text{OFDT}_4 = c_3(1.7)$
5	Output	$c_2((c_4+1)8+2)$	$\text{OFDT}_5 = c_3((c_4+1)8+2)$

1, 3 and 5 correspond to three one way transmissions that were measured to take approximately 1.7 ms each. Steps 1, 3 and 5 sum to a total of 5.1 ms. Steps 2 and 4 are the processing time at the input and host entities, respectively. Calibration tests revealed that the maximum processing time at the input and host entities are 4 and 0.1 ms, respectively. Thus, steps 2 and 4 sum to at most 4.1 ms. The total time from steps 1 to 5 is estimated to be 9.2 ms. This means that the update interval, i.e. the host entity timer, should be set to around 9 ms.

Experimentally, it was determined that with an 8 ms update interval the second request was sent from the host after the first corrective action. An update interval of 8 ms, however, is less than the calculated 9 ms. This calculated 9 ms included estimates for transmission delays, i.e. latency, which includes the processing time to receive, open, and check the cyclic redundancy check (CRC) for one packet. This processing time was also included in the maximum processing times observed at the input and host entities. This means that the real time from steps 1 to 5 could be less than the calculated 9 ms. An update interval of 8 ms, only 1 ms less than the calculated value, was used for the wired implementation.

An update interval of 8 ms is reflected in the waiting times of the device entities. With known waiting and processing times, the worst case delay times and watchdog timers are calculated with the expressions shown in Table 5.2.

The values for constants c_2 and c_3 are set to 1.05 and 1.3, respectively. Constant c_4 represents the number of acceptable consecutive lost packets. For constant c_4,

Table 5.3: Safety function response time results for the wired implementation. All units are ms.

(a) Results for $c_2 = 1.05$, $c_3 = 1.3$ and $c_4 = 0$.

i	Entity	WCDT_i	WDTime_i	$\Delta\text{T\_WD}_i$
1	Input	12.6	15.6	3
2	TD	1.8	2.3	0.4
3	F-Host	8.5	10.5	2
4	TD	1.8	2.3	0.4
5	Output	10.5	13	2.5

(b) Results for $c_2 = 1.05$, $c_3 = 1.3$ and $c_4 = 1$.

i	Entity	WCDT_i	WDTime_i	$\Delta\text{T\_WD}_i$
1	Input	21	26	5
2	TD	1.8	2.3	0.4
3	F-Host	16.9	20.9	4
4	TD	1.8	2.3	0.4
5	Output	18.9	23.4	4.5

two values of 0 and 1 are used. Using these values for constants c_2, c_3 and c_4 in the expressions in Table 5.2, the calculated worst case delay time WCDT_i, watchdog timer WDTime_i, and the maximum difference between an entity's watchdog timer and its worst case delay time $\Delta\text{T\_WD}_i$ are shown in Table 5.3. This results in

$$\text{SFRT} = \sum_{i=1}^{5} \text{WCDT}_i + \max_{i \in \{1,\ldots,5\}} \Delta\text{T\_WD}_i = 38.2 \text{ ms} \tag{5.1}$$

when WCDT_i and $\Delta\text{T\_WD}_i$ are taken from Table 5.3a, and

$$\text{SFRT} = \sum_{i=1}^{5} \text{WCDT}_i + \max_{i \in \{1,\ldots,5\}} \Delta\text{T\_WD}_i = 65.4\text{ms} \tag{5.2}$$

when WCDT_i and $\Delta\text{T\_WD}_i$ are taken from Table 5.3b.

Table 5.3 shows the difference in the worst case delay time, watchdog timer, and safety function response time values caused by c_4. When $c_4 = 0$, no packet lost is acceptable. If one packet was to be lost in the $c_4 = 0$ implementation, watchdog timers would expire and the system would be driven to a safe state. This describes

very strict control where worst case delay times and watchdog timers permit no packet loss, and where the five participating network entities can be driven to a safe state in at most 38.2 ms.

When $c_4 = 1$, one packet lost per communication link is acceptable. To accommodate this, worst case delay times and watchdog timers are greater than those when $c_4 = 0$. Greater values for worst case delay times and watchdog timers account for one lost packet. If one packet is lost, this is considered normal operation and watchdog timers would not expire, unless a second consecutive packet is lost. This less strict control is reflected in a larger value for the safety function response time, where the five participating network entities can be driven to a safe state in at most 65.4 ms.

Note that the worst case delay time of wired transmission delay entities TD is only 1.8 ms. The main difference between on board control and the wired implementation is the increase in the update interval from 1.9 ms to 8 ms. With an update interval of 8 ms, the wired implementation of line following can provide 125 new line positions in 1 second.

An update interval U = 8 ms is reasonable for the line following activity as shown by the results in Table 5.4. The six metrics μ_e, d, σ_e, $\max_e$, T and F reported in Table 5.4 are the same six metrics reported in Table 5.1 for on board control. The control parameters K_P, K_I, K_D and A have also the same values. The only difference is the that number of samples N used to obtain average values changed to $N \approx 5042$.

Table 5.4 reports that on average the 3pi was able to complete one lap of the course in 32 s, with a maximum speed of 120 rpm. The average error reported is 0.2 cm, which means that on average the center of the 3pi was within 0.2 cm of the line.

The maximum value of the performance metric e is 2000. Error e can reach a value of 2000 when the reflectance sensors detecting the line are sensors 0 or 4 (see Figure 4.9), or when the line is not being currently detected by the 3pi at all. For

Table 5.4: Experimental results using wired communication to achieve line following control; $K_P = -1/20$, $K_I = 1/10000$, $K_D = 3/2$, $-60 \leq A \leq 60$, U = 8 ms, and $N \approx 5042$ for each of the six experiments.

Dir.	μ_e	d (cm)	σ_e	$\max_e$	T (s)	F (s)
C	246.8	0.2	230.4	918	40.3	32.2
C	221.2	0.2	224.6	1000	40.9	31.3
C	232.1	0.2	232.2	941	40	32.6
μ_C	233.3	0.2	229.1	953	40.4	32
CC	272.5	0.3	246.9	995	40.9	32.8
CC	246	0.2	233.4	924	40.7	31.2
CC	263	0.3	241.5	901	40	31.7
μ_{CC}	260.5	0.3	240.6	940	40.6	31.9
$\mu_{C,CC}$	247	0.2	234.8	946.5	40.5	32

the results in Table 5.4, the maximum observed value of error e is 1000, which means that the 3pi never lost sight of the line. A maximum error e of 1000 means that the line was always within sensors 1 and 3 (see Figure 4.9).

The behaviour observed when running the tests was a successful line following. When compared to on board control, however, the wired implementation provides a less smooth line following, and the 3pi experiences vibrations.

As discussed in Section 5.3, experiments revealed that the 3pi is unable to follow the line in the wireless implementation with values of $K_P = -1/20$, $K_I = 1/10000$, $K_D = 3/2$, and when $-60 \leq A \leq 60$. The power difference A (calculated in Equation (4.4)) determines the magnitude of the corrective action, which is the speed to be applied to the motors. The higher the corrective action, the sharper the turns. To obtain a more gradual and gentle control, A should be smaller. To achieve this, the control parameters were changed to $K_P = -1/40$, $K_I = 1/20000$ and $K_D = 3/4$. Using these control parameters, two tests were done with two different maximum values of A: 60 and 40. Tests with a maximum A of 60 reported a higher error e than when a maximum A of 40 was used. The reason for this increase in e when only the control parameters are changed but the maximum value of A is not, is that the same speed used for the default control is being used on a more gentle

Table 5.5: Experimental results using wired communication to achieve line following control; $K_P = -1/40$, $K_I = 1/20000$, $K_D = 3/4$, $-40 \leq A \leq 40$, U = 8 ms, and $N \approx 5034$ for each of the six experiments.

Dir.	μ_e	d (cm)	σ_e	$\max_e$	T (s)	F (s)	μ_{AD} (ms)	σ_{AD} (ms)
C	288.3	0.3	264.7	984	40.3	37.6	4.5	0.5
C	296	0.3	266.3	986	40.5	39.4	4.8	0.4
C	285.7	0.3	263.3	961	40.2	38.9	4.6	0.5
μ_{C}	290	0.3	264.8	977	40.3	38.6	4.6	0.4
CC	271.2	0.3	258.4	983	41	38.1	4.9	0.2
CC	278.1	0.3	251	1002	40.4	38	5.2	0.4
CC	279.6	0.3	257.8	1074	40.2	37.9	4.9	0.3
μ_{CC}	276.3	0.3	255.8	1019.7	40.5	38	5	0.3
$\mu_{\text{C,CC}}$	283.1	0.3	260.3	998.3	40.4	38.3	4.8	0.4

control loop. This means the same speed is being applied to more gentle corrective actions. Considering this, values of A were limited to the range $-40 \leq A \leq 40$, which means that the maximum corrective action is now 40 rpm instead of 60 rpm, for $K_P = -1/40$, $K_I = 1/20000$ and $K_D = 3/4$. Experimentally it was determined that these values generated more gentle corrective actions and slowed down the 3pi enough so that it could follow the line when using the wireless implementation.

Tests were repeated for the wired implementation using the more gentle control. Results are reported in Table 5.5 and they include the standard deviation σ_{AD} for the average action delay μ_{AD}. The number of samples used to obtain average values is $N \approx 5034$.

Table 5.5 reports an increase in the average error e of approximately 15% compared to Table 5.4, where the default control parameters and maximum speed are used. This 15% increase only signifies a change of 1 mm, which means that the average distance from the center of the 3pi to the line increased in 1 mm.

The behaviour observed when running the tests with the more gentle control was a successful line following. The 3pi only experienced some minor vibrations, and the line following was smoother than when using the default control parameters and speed.

Applying function d to the average error $\mu_{\mathrm{C,CC}} = 283.1$ results in an average position error of 0.3 cm. This means that on average, the center of the 3pi was only 0.3 cm away from the line.

Table 5.5 shows that the average time to complete the course was approximately 38 s. This corresponds to 15 s more than on board control, but the maximum speed used during the tests for the wired implementation was 80 rpm, 40 rpm less than the maximum 120 rpm used for on board control.

Table 5.5 shows that the maximum error, $\max_e$, was on average approximately 998. This is almost 1000, which corresponds to half the maximum value of error e. An average $\max_e$ of 998 is higher than the average $\max_e$ of approximately 885 observed during on board control tests. This could be a consequence of the longer update interval of 8 ms, which gives the 3pi more time to move before sending a new line position. The system still remains in control because the average action delay is 4.8 ms (μ_{AD} reported in Table 5.5), which means that a new corrective action is received 4.8 ms after the line position was read.

The results obtained for the wired implementation provide a more realistic best performance when off-board communication channels are involved. Results reported consistent 0% packet lost in every test. The high performance observed for wired communication is also reliable, as demonstrated by the small (less than 1 ms) standard deviation of action delays.

The experiments used to obtain the results reported in Table 5.5 implemented the watchdog timers calculated in Table 5.3. For all the tests, both watchdog timers implemented on each entity expired 0 times. This is expected with a reported packet loss of 0%.

5.3 Wireless Implementation

In the wireless implementation, when using the default control parameters $K_P = -1/20$, $K_I = 1/10000$ and $K_D = 3/2$, and corrective action values in the range $-60 \leq A \leq 60$, experiments showed that the 3pi is unable to follow the line. The 3pi loses sight of the line within 1 second of the test. In the best observed performance, the 3pi was able to follow a segment of the straight line in the course, while stopping and rotating on its center a couple of times. The results obtained when running this experiments cannot be used, since most of the times the 3pi fell off the poster board track.

To achieve line following feedback control using wireless communication, a more gentle control loop was used in the experiments. The new values for the control parameters are $K_P = -1/40$, $K_I = 1/20000$, $K_D = 3/4$, and the corrective action has values in the range $-40 \leq A \leq 40$.

Wireless calibration tests revealed that the maximum processing time for the input, host and output entities are $P_1 = 4$ ms, $P_3 = 0.1$ ms, and $P_5 = 2$ ms, respectively. These maximum processing times correspond to the same values reported for the wired implementation, as indicated in Section 5.2. Since the wired and wireless implementations share the same sensor, comparator, controller and actuator blocks, it is reasonable that measured maximum processing times are the same.

Given that the input entity can only provide an updated line position upon a serial request from the mobile TelosB mote running OpenWSN, the update interval of the line position is the time interval at which the mobile TelosB mote sends a request frame to the input entity (3pi). Considering that the OpenWSN network schedule has 8 timeslots with a duration of 15 ms each, the mobile mote can send a serial request frame as fast as 120 ms, giving $W_1 = W_3 = 120$ ms for entity 1 (input) and entity 3 (fail-safe host). The waiting time W_5 for entity 5 (output) is defined as the interval at which the host entity can provide a new corrective action to the

Table 5.6: Expressions to calculate worst case delay times and watchdog timers for network entities in the wireless implementation.

i	Entity	F_WD_Time$_{i,j}$	WCDT$_i$	WDTime$_i$
1	Input	NA	$c_2(120+4)$	OFDT$_1 = c_3(120+4)$
2	TD	$c_1(4+15)$	8×15	F_WD_Time$_{2,1}$ + WCDT$_2$
3	F-Host	NA	$c_2((c_4+1)120+0.1)$	OFDT$_3 = c_3((c_4+1)120+0.1)$
4	TD	$c_1(0.1+15)$	8×15	F_WD_Time$_{4,3}$ + WCDT$_4$
5	Output	NA	$c_2((c_4+1)120+2)$	OFDT$_5 = c_3((c_4+1)120+2)$

output entity. This interval is the waiting time at the host plus the processing time to generate a new corrective action. Since the maximum processing time at the host $P_3 = 0.1$ ms, the simplifying assumption that $W_5 = 120$ ms is made.

Using the corresponding waiting and processing times in the expressions in Table 4.1, the worst case delay times WCDT$_i$, fail-safe watchdog timers F_WD_Time$_{i,j}$, one fault delay time (OFDT) values, and watchdog timers WDTime$_i$ are calculated for the wireless implementation as shown in Table 5.6.

The values for constants c_1, c_2 and c_3 are set to 1.3, 1.05 and 1.3, respectively. Using these in the expressions in Table 5.6, the calculated worst case delay time WCDT$_i$, fail-safe watchdog timer F_WD_Time$_{i,j}$, watchdog timer WDTime$_i$, and the maximum difference between an entity's watchdog timer and its worst case delay time ΔT_WD$_i$ are shown in Table 5.7a for entities 1 (input), 2 and 4 (transmission delays).

The host and output entities are the only two possible destinations in the two communication links present in the line following system. As a consequence, the host and output entities are influenced by the behaviour of the communication links. Constant c_4 represents the number of acceptable consecutive lost packets. The value of c_4 determines the worst case delay times and watchdog timer values for the host and output entities. For the wireless implementation, values of c_4 from 0 to 5 are used. Each possible value of c_4 results in different values for the worst case delay time WCDT$_i$ and watchdog timer WDTime$_i$ of entities $i = 3$ (fail-safe host) and

Table 5.7: Safety function response time results for the wireless implementation. All units are ms.

(a) Results for $c_1 = 1.3$, $c_2 = 1.05$ and $c_3 = 1.3$.

i	Entity	WCDT_i	$\text{F\_WD\_Time}_{i,j}$	WDTime_i	$\Delta\text{T\_WD}_i$
1	Input	130.2	NA	161.2	31
2	TD	120	24.7	144.7	24.7
4	TD	120	19.63	139.6	19.6

(b) Results for $c_1 = 1.3$, $c_2 = 1.05$, $c_3 = 1.3$, and c_4 values in the range $0 \leq c_4 \leq 5$.

c_4	WCDT_3	WDTime_3	$\Delta\text{T\_WD}_3$	WCDT_5	WDTime_5	$\Delta\text{T\_WD}_5$	SFRT
0	126.1	156.1	30	128.1	158.6	30.5	655.4
1	252.1	312.1	60	254.1	314.6	60.5	936.9
2	378.1	468.1	90	380.1	470.6	90.5	1218.9
3	504.1	624.1	120	506.1	626.6	120.5	1500.9
4	630.1	780.1	150	632.1	782.6	150.5	1782.9
5	756.1	936.1	180	758.1	938.6	180.5	2064.9

$i = 5$ (output). For each of the six values of c_4, the safety function response time (SFRT) is calculated as shown in Table 5.7b.

As shown in Table 5.7b, the minimum SFRT value for the wireless implementation of line following is 655 ms, when $c_4 = 0$. The minimum SFRT of 655 ms is about 17 times the minimum SFRT for the wired implementation. This increase in the SFRT is caused by the much larger update interval of 120 ms for the wireless implementation, compared with the 8 ms for the wired implementation.

Another significant difference between the wired and wireless implementations is the worst case delay time of transmission delay entities. It is estimated to be 1.8 ms for the wired implementation, and 120 ms for the wireless implementation.

As shown in Table 5.7b, different values of c_4 result in different values of the SFRT. If $c_4 > 0$, worst case delay times and watchdog timers should account for a number of acceptable consecutive lost packets. Entities are only driven to a safe state when the number of acceptable consecutive lost packets is exceeded. The higher the value of c_4, the more failed cycles are considered acceptable, resulting in higher values of the system SFRT.

Table 5.8: Packet loss values for different OpenWSN network schedules; $N = 200$ for each of the nine experiments.

Set	GTS	N_{ts}	U (ms)	PL_2 (%)	PL_4 (%)	T (s)
1	Y	8	120	13	19.7	29.2
	Y	8	240	29	42.6	48.1
	Y	8	360	3.5	3.6	72.2
2	N	8	120	41	70.1	24.1
	N	8	240	47	89.5	48.1
	N	8	360	2.5	6.7	72.2
3	N	9	135	42.5	77.2	27.1
	N	10	150	44.5	83.6	30.1
	N	11	165	45.5	80.6	33.1

The minimum SFRT value for the wireless implementation is achieved when $c_4 = 0$. This means that the acceptable number of lost packets is 0. If one packet is lost, the watchdog timers will expire and the system will be driven to a safe state.

A packet loss of 0% is a strong assumption for wireless communication, especially for the wireless line following experiments where tests revealed that packet loss values can be up to 80% under certain conditions. Table 5.8 shows measured packet loss percentages for three sets of tests, where GTS indicates if the network schedule has guaranteed timeslots allocated for each communication link in the system, N_{ts} is the number of timeslots in the network schedule, U is the update interval (ms) at which input was sent from the input entity, PL_2 represents the packet loss percentage observed in entity 2 (transmission delay, where the input entity is the sender and the fail-safe host the destination), PL_4 represents the packet loss percentage observed in entity 4 (transmission delay, where the fail-safe host is the sender and the input entity the destination), and T is the duration of the test (s).

As shown in Table 5.8, in set #1 the network schedule has 8 timeslots: 1 advertisement timeslot, 1 shared timeslot, 2 guaranteed timeslots (one for each communication link), 3 serial timeslots, and 1 off timeslot. This is the network schedule assumed by the wireless SFRT model. If the 3pi answers every serial request frame from the mobile TelosB mote with a new line position, the update interval U = 120

ms. If the 3pi answers every 2nd serial request frame, U = 240 ms, and if the 3pi answers every 3rd serial request frame, U = 360 ms. The difference between sets #1 and #2 is that the network schedule used in set #2 has 3 shared timeslots and no guaranteed timeslots. During these packet loss tests (Table 5.8), only packet loss values were measured, and the 3pi was stationary on the table next to the DAG root mote.

Consistently in sets #1 and #2, the smallest packet lost percentage is observed with the slowest update interval of 360 ms. Except for one packet loss value when U = 360 ms, smaller packet loss values are observed in set #1, where guaranteed timeslots are used. This provides evidence that the use of guaranteed timeslots can increase the reliability of the network.

Set #3 uses a network schedule consisting of more than 8 timeslots, where there is 1 advertisement timeslot, 3 serial timeslots, 1 off timeslot, and the rest are shared timeslots. Event though set #3 has more timeslots than sets #1 and #2, on every request frame received by the 3pi a new line position is transmitted. As a consequence, update intervals are less than the 240 ms and 360 ms implemented in sets #1 and #2.

It seems reasonable that with a shorter update interval, more packets are lost, since the network might not be able to keep up. Results in Table 5.8, however, show that a longer update interval can result in a higher packet lost. For example, with U = 240, set #1 reports $PL_4 = 42.6\%$. With a shorter update interval of U = 120 ms, set #1 reports $PL_4 = 19.7\%$. This inconsistency can indicate a timing problem with the OpenWSN network schedule.

Table 5.8 also shows that PL_4 is always greater than PL_2, even though entities 2 and 4 belong to the same network and operate under the same conditions. The value of PL_4 can exceed PL_2 up to 1.9 times.

Based on the results shown in Table 5.8, the best packet loss performance for

Table 5.9: Experimental results using wireless communication to achieve line following control; $K_P = -1/40$, $K_I = 1/20000$, $K_D = 3/4$, $-40 \leq A \leq 40$, U = 120 ms, and $N \approx 634$ for each of the six experiments.

Dir.	μ_e	d (cm)	σ_e	$\max_e$	T (s)	F (s)	μ_{AD} (ms)	σ_{AD} (ms)
C	1437.7	1.7	658.8	2000	92.3	75.4	411.5	26.5
C	1494.8	1.7	652.8	2000	92.3	67.8	410.8	24.9
C	1503.6	1.8	644.7	2000	92.3	67	413.5	29.9
μ_{C}	1478.5	1.7	652.1	2000	92.3	70.1	411.9	27.1
CC	1440.3	1.7	657.5	2000	92.3	66.5	409.8	23
CC	1404.7	1.6	685.3	2000	92.3	65.9	410.6	24.4
CC	1429.7	1.6	665.4	2000	92.3	67.3	411.5	26.7
μ_{CC}	1425.1	1.6	669.4	2000	92.3	66.6	410.6	24.7
$\mu_{\text{C,CC}}$	1451.8	1.7	660.8	2000	92.3	68.3	411.3	25.9

line following implemented with OpenWSN is achieved with the network schedule used in set #1 with an update interval of 360 ms. For line following, however, a longer update interval affects the control of the system, and higher values of error e were observed (i.e. $\mu_e = 1487$ for U = 120 ms, $N = 232$ compared to $\mu_e = 1691$ for U = 360 ms, $N = 94$) for a line following test of roughly 30 seconds.

Further tests showed that when U = 120 ms and guaranteed timeslots are used, even with PL_2 and PL_4 packet loss values of around 13% and 20%, respectively, the best value for error e is obtained. The schedule used in set #1 of Table 5.8 with an update interval of 120 ms was used to run the tests reported in Table 5.9.

Table 5.9 shows values of the average error e higher than those obtained with the wired implementation (see Table 5.5). These higher values of the average e are a consequence of the longer update interval used in the wireless implementation. This means that in the wireless implementation input to the feedback control loop is provided less often and will take more time to be transmitted, making the control of the system more challenging.

The average error e for the wireless implementation is 1451.8, which means that on average the center of the 3pi was 1.7 cm away from the line. Sensors 0 and 4 (see Figure 4.9) are 2.5 cm away from the center of the 3pi. This means that on average

the 3pi was less than 1 cm away from losing sight of the line.

Results in Table 5.9 show that $\max_e = 2000$ for all the tests. This provides evidence that the line reached sensors 0 and 4, and that the 3pi might have lost sight of the line momentarily. The 3pi was still able to successfully complete the course in an average time of approximately 1 minute and 8 seconds, 30 seconds more than the average time to complete the course in the wired implementation.

When the error e reaches the value of 2000, the controller calculates the maximum corrective action, which causes the 3pi to rotate on its center at the maximum speed. If the line is close enough, the 3pi will be able to find the line and resume the line following. If the line is too far from the 3pi, i.e. is not under the area covered by the 3pi, the 3pi cannot find the line and gets stuck rotating on its center.

To report the 6 successful completions of one lap of the course shown in Table 5.9, tests had to be repeated 11 times on the C direction, and 10 times on the CC direction. The 15 tests where the 3pi did not finish one lap of the course in 90 seconds include tests where the 3pi fell off the poster board track, could not find the line and was stuck rotating, or rotated about 180 degrees and continued following the line in the opposite direction. Results for these tests reported higher μ_e and σ_e values. For example, one test reported $\mu_e = 1520.2$ and $\sigma_e = 638.4$. This test reported similar packet loss and action delay values as those reported in the tests were the 3pi finished one lap of the course, suggesting that the test was not influenced by particular network conditions.

The behaviour observed when running the tests was a wobbly 3pi, where the 3pi followed the line while oscillating from side to side, and performing wide turns that sometimes caused the 3pi to lose sight of the line.

The average action delay for the wireless implementation reported in Table 5.9 is 411.3 ms. This means that on average the 3pi receives a corrective action 411 ms after the line position was read. During this time, the 3pi is applying the previous

Table 5.10: Packet loss percentages and the number of times each watchdog timer expired X for the wireless implementation tests in Table 5.9.

			WDTime_3 (ms)			WDTime_5 (ms)					
	PL (%)		156	312	468	159	315	471	627	783	939
Dir.	PL_2	PL_4	X = number of times WDTime_i expired								
C	14.3	18.4	94	13	1	163	35	11	3	1	1
C	15.6	21.5	109	7	0	186	46	8	4	1	0
C	16	19.9	109	11	0	174	44	13	1	0	0
μ_{C}	15.3	20	104	10.3	0.3	174.3	41.7	10.7	2.7	0.7	0.3
CC	14.4	20	93	14	2	172	37	13	4	1	1
CC	16.3	20.8	105	16	2	166	41	18	5	3	1
CC	15.5	23.7	100	14	2	154	44	16	5	3	1
μ_{CC}	15.4	21.5	99.3	14.7	2	164	40.7	15.7	4.7	2.3	1
$\mu_{\text{C,CC}}$	15.4	20.7	101.7	12.5	1.2	169.2	41.2	13.2	3.7	1.5	0.7

corrective action. An average action delay of 411.3 ms corresponds to about 86 times the average action delay reported for the wired implementation.

For the results shown in Table 5.9, the packet loss (%) of entities 2 and 4 (transmission delays), PL_2 and PL_4, are reported in Table 5.10. These packet loss values result in the expiration of the watchdog timers calculated in Table 5.7.

The watchdog timer of input entity WDTime_1 set to 161 ms (see Table 5.7a) did not expire in any of the tests. This means that the input entity is reliably providing a new line position once every 120 ms, as expected.

The watchdog timer WDTime_3 of entity 3 (fail-safe host) expired at least once when set to 156, 312 or 468 ms, as reported in Table 5.10. These watchdog timer values correspond to values of c_4 of 0, 1 and 2, respectively. This means that up to 3 consecutive packets were lost in entity 2 (transmission delay). This is consistent with the observed maximum waiting time on entity 3 (fail-safe host) of 481.3 ms, which corresponds to 3 failed cycles with a waiting time of 120 ms each.

At the fail-safe host entity, 3 consecutive packets lost is the less frequent number of consecutive packets lost, and was only observed up to 2 times in one test. The watchdog timer of 156 ms (when $c_4 = 0$), however, expired approximately 102 times.

Given the packet loss percentages shown in Table 5.10, and considering that when $c_4 = 0$ no packet loss is acceptable, such high expiration times of the watchdog timer of 156 ms is expected.

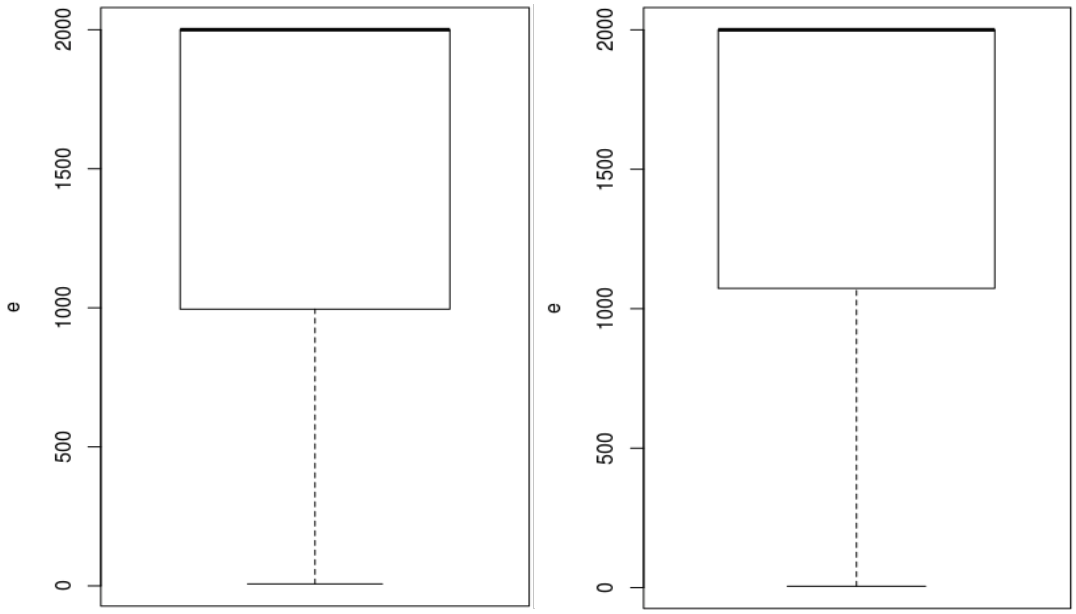
For entity 5 (output), all the calculated watchdog timers in Table 5.7b expired at least once, as shown in Table 5.10. When $c_4 = 5$, $\text{WDTime}_5 = 939$ ms. This value of the watchdog timer accounts for up to 5 consecutive packets lost, and expired once on 4 of the 6 tests.

As shown in Table 5.10, values of PL_4 are higher than values of PL_2, which means that the second communication link, where the output entity is the destination, is less reliable than the first communication link. As a consequence, the watchdog timers at the output entity expired more times than the watchdog timers of the host entity.

5.4 Discussion

One unsuccessful wireless line following experiment, during which the 3pi did not finish one lap of the course in 90 s, reported 328 samples with a value of 2000. This corresponds to 52% of the samples, where the 3pi was detecting the line with sensors 0 or 4 (see Figure 4.9), or the 3pi had lost sight of the line. Even with 52% of the samples having a value of 2000, the average reported for this test is $\mu_e = 1520.2$, 79.9 higher than the average error for a test in which the 3pi successfully completed one lap of the course.

Box plots are a standardized way of displaying a data set, as explained in Appendix I. In this section, box plots are used to study the distribution of the error samples obtained during the wired and wireless experiments in order to determine the characteristics of the error samples in the experiments where the 3pi lost sight of the line.



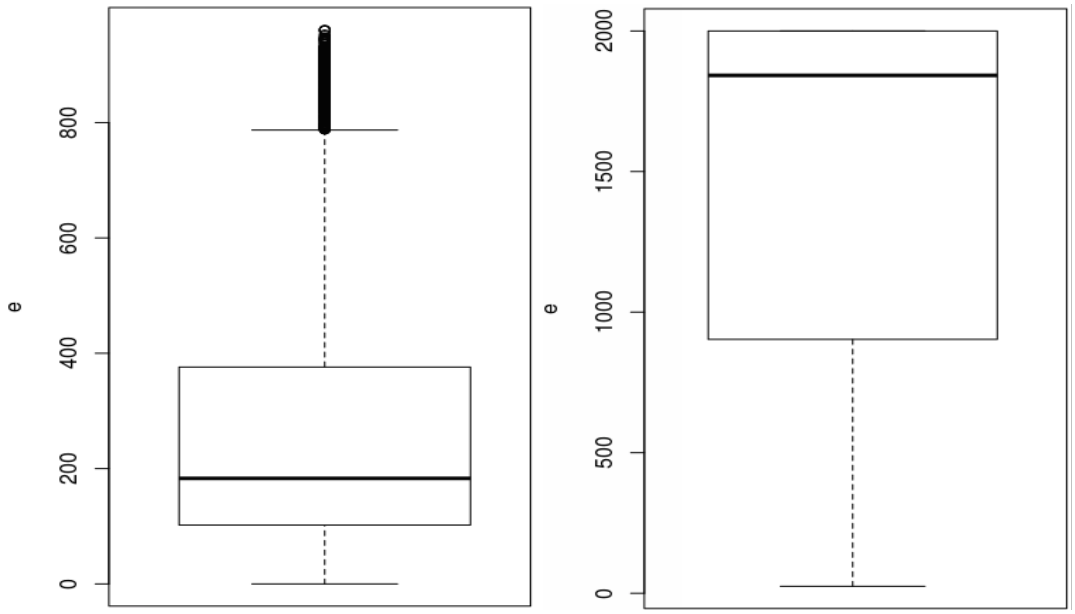
(a) 3pi lost sight of the line.

(b) 3pi rotated on its center.

Figure 5.2: Box plots for samples of error e on unsuccessful wireless line following experiments.

The box plot for error samples from a test where the 3pi lost sight of the line and did not finish one lap of the course in 90 s is shown in Figure 5.2a. The median, third quartile and upper extreme have all a value of 2000, which indicates there is a high number of samples with a value of 2000. Given that the median for this data set is 2000, 50% of the samples have values above 2000, and 50% of the samples have values below 2000. Error e, however, has a maximum value of 2000, which indicates that approximately 50% of the samples have a value of 2000. The graph also shows a widely dispersed data set, where the range is 1993. The first quartile has a value of 995, indicating that 75% of the samples have values greater that 995, which corresponds to an error of almost 1 cm.

A second box plot is shown in Figure 5.2b for error samples from a test in which the 3pi did not finish one lap of the course in 80 s. The 3pi was unable to follow the line and rotated on its center several times. Similar to the box plot in Figure 5.2a, the median, third quartile and upper extreme have all a value of 2000. Data is



(a) Wired implementation. (b) Wireless implementation.

Figure 5.3: Box plots for samples of error e on successful line following experiments.

distributed in a range of 1995 and the first quartile occurs at 1073.5.

The data sets shown in Figures 5.2a and 5.2b suggest that when the 3pi is unable to follow the line, even with an average error of approximately 1500, the median of the error samples is 2000.

Figure 5.3a shows the box plot for error samples from the third test in the C direction using the wired implementation. Results for this test are shown in Table 5.5. The graph shows a much more concentrated data set distributed in a range of 961. The median for this data set is 185, and the first and third quartiles are 102 and 376, respectively. This means that 75% of the data has values below 376, which corresponds approximately to an error of only 0.3 cm. The maximum error in this data set has a value of 961 and was observed only once. From a total of 5005 samples in the data set, 505 are considered outliers between the upper extreme value of 787 to the maximum value of the data set 961. The R software [38] by default extends the upper extreme to the most extreme data point which is no more than 1.5 times the inter-quartile range from the box (third quartile). Values greater than the upper

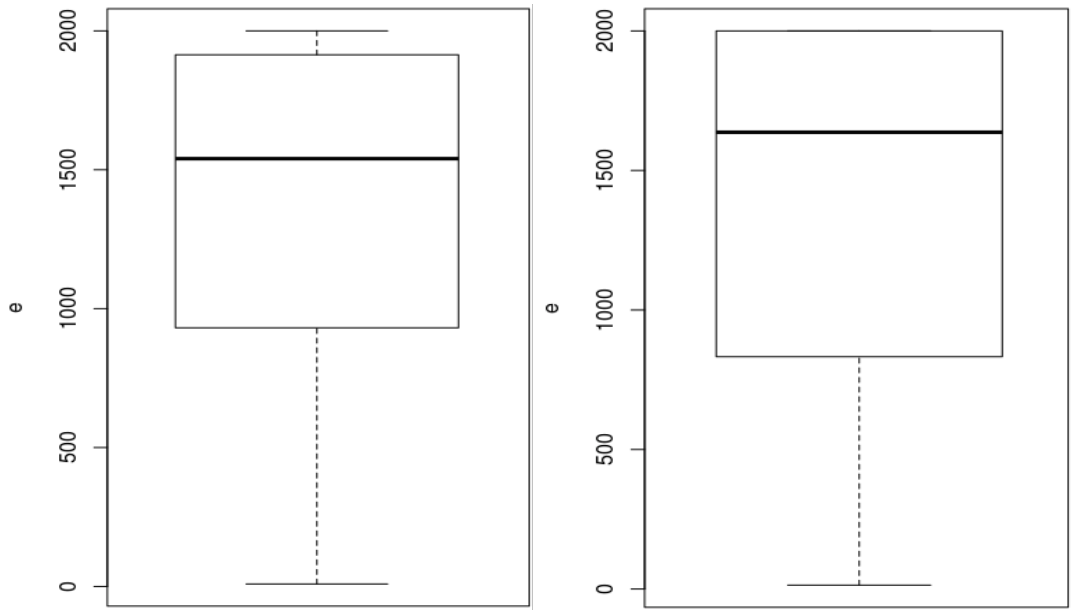
extreme are considered outliers. A high concentration of outliers shows up in a box plot as a vertical thick line resulting from many overlapping circles.

Figure 5.3b shows the box plot for error samples from the first test in the CC direction using the wireless implementation. Results for this test are shown in Table 5.9. Figure 5.3b shows that the third quartile and upper extreme have values of 2000. The median for this data set is 1440.3, almost 8 times the median for the wired implementation shown in Figure 5.3a. The third quartile with a value of 2000, indicates that at least 25% of the samples are 2000. From the 641 samples in the test, 278 had a value of 2000, giving approximately 43%. Samples are dispersed in a range of 1975.

Figure 5.3b shows that the wireless implementation generates a less stable data set, where 50% of the samples have values greater or equal to 1842, and at least 25% of the samples have a value of 2000. Even under these conditions, the 3pi was able to finish one lap of the course and successfully followed the line. This successful line following is reflected by the median of 1440, which is less than the median of 2000 observed in the tests where the 3pi lost sight of the line.

Section 5.3 reported that an important difference between the wired and wireless implementation is the update interval. In the wireless implementation, the update interval is 120 ms, while in the wired implementation the update interval is 8 ms. Figures 5.4a and 5.4b show box plots for error samples using a wired implementation with an update interval of 120 ms and 240 ms, respectively. These graphs show the impact of an increase in the update interval. Compared to Figure 5.3a, the range and median have increased. Network conditions remained the same, no packet loss was observed and watchdog timers did not expire.

For the test shown in Figure 5.4a, the median is 1540, and the first and third quartiles are 931 and 1914, respectively. The maximum error in the sample has a value of 2000, and is reported in 109 samples in the data set, from a total of 757



(a) Update interval of 120 ms. (b) Update interval of 240 ms.

Figure 5.4: Box plots for samples of error e on successful wired line following experiments.

samples.

Increasing the update interval to 240 ms can further negatively affect the performance, as shown in Figure 5.4b. The median for this data set is 1637, and the third quartile is 2000. Of the 379 samples in the data set, 121 samples have a value of 2000.

With an update interval of 120 ms, the wireless implementation has an average error $\mu_e = 1440.3$, and the wired implementation reports an average error $\mu_e = 1361.4$. The wireless implementation reports an increase in the average error of only 78.9, corresponding to an increase in the average distance between the center of the 3pi and the line of less than 0.1 cm. The median, however, experienced a higher increase of 302 in the wireless implementation, corresponding to an increase in the median distance of approximately 0.3 cm.

5.4.1 OpenWSN for Wireless Control

OpenVisualizer is an essential part of OpenWSN and needs to be running and communicating serially with the DAG root mote, so the OpenWSN network can operate correctly. An attempt to implement the line following feedback control loop using OpenWSN without OpenVisualizer was made. The host communicated serially with the DAG root mote, providing a more direct communication where no sockets are required. The mobile mote addressed udpinject frames with control data to the DAG root mote. When the DAG root mote received a frame from the mobile mote, the frame was forwarded serially and received by the host. Transmissions from the DAG root to the mobile mote, however, were not possible since the application layer at the DAG root is implemented by OpenVisualizer.

If transmissions from the DAG root to the mobile mote and viceversa were possible without the need of OpenVisualizer, communication between the host and the mobile mote would be more direct, less components would be involved, and this could simplify the implementation and reduce transmission times. OpenVisualizer is a mandatory layer that provides interesting features, but also limits the way in which OpenWSN can be used, and in which the OpenWSN network can be accessed.

The fail-safe watchdog timer, F_WD_Time, of a transmission delay entity supervises a one-to-one communication link and monitors the correct operation of the communication channel. The F_WD_Time is started every time a new packet is sent, and is reset when the corresponding acknowledgement is received.

The F_WD_Time needs to be reset by the `sendDone` method, which is the method in the OpenWSN implementation that reports successful and unsuccessful transmissions. On the DAG root mote, the `sendDone` method runs locally, and OpenVisualizer is not notified. As a consequence, when the host sends a UDP frame to the mobile mote, the host is not notified when the corresponding acknowledgement is received. To resolve this, when the DAG root receives an acknowledgement, the DAG

root should serially forward the frame to OpenVisualizer. OpenVisualizer could then send a notification to the host over the socket. This would provide support for the F_WD_Time, which cannot be implemented with the current OpenWSN implementation.

Experimental results discussed in Section 5.3, reported from 2.5% to 89.5% packet loss for different OpenWSN network schedules. To investigate this behaviour, a packet analyzer (sniffer) was configured as explained in Appendix J. The sniffer was used to capture packets being transmitted over the air between the DAG root and mobile motes. The Wireshark software was used to display the contents of the packets captured by the sniffer. This set-up revealed that packets are sent successfully from the mobile mote, but they are not received by the DAG root. As a consequence, the DAG root does not reply with the corresponding acknowledgements, and the mobile mote reports these failed transmissions. The reason why the DAG root is not receiving the frames is suspected to be an OpenWSN synchronization problem at the DAG root, as reported in Appendix K.

To reduce the minimum update interval provided by OpenWSN of 120 ms, and thus the minimum SFRT achievable, the number of required timeslots in a slotframe should be reduced. One way of achieving this is to perform serial activity in the background. This would eliminate the requirement of having three reserved timeslots for serial activity. OpenWSN should provide more reliability, so it can be used when implementing critical control loops with fewer acceptable failed cycles. To achieve this, the synchronization problem at the DAG root needs to be solved, which would significantly reduce the packet loss.

5.4.2 Acceptable Safety Function Response Time

The IEC 61511 standard [18] defines the process safety time as the time between a failure in a system and a hazardous event. A safe system state should be reached

within the process safety time, to guarantee that a hazardous event will not occur. For the line following experiment, a hazardous event is defined as the 3pi losing sight of the line to the point where the 3pi is unable to find the line again. This means that the line is no longer in the area covered by the dimensions of the 3pi, which is an area of 69.3 cm^2. For the line following experiment, a safe state is reached when the 3pi stops before moving so far that the area covered by the 3pi no longer contains some part of the line. For the line following experiment, the SFRT should be short enough to ensure that the 3pi reaches its safe state in time.

The length of the course used in the experimental validation is 4.532 m. For the wired implementation, the 3pi was able to complete one lap of the course in 38.3 s, as shown in Table 6.1. This provides an estimate of the speed at which the 3pi moves in the course of 11.83 cm/s. Given that the calculated SFRT for the wired implementation is 38.2 ms (see Table 6.1), it can be calculated that during 38.2 ms the 3pi moves 0.45 cm. For the wired implementation, the center of the 3pi was at most 1.11 cm from the line. From the worst case initial position of the center of the 3pi being 1.11 cm away from the line, if the 3pi moves 0.45 cm more before stopping, the area of the 3pi is guaranteed to still cover some part of the line. Based on this analysis, an SFRT of 38.2 ms is acceptable for the wired implementation of line following.

Similarly, for the wireless implementation, the estimate of the speed at which the 3pi moves in the course is 6.63 cm/s. The SFRT for the wireless implementation is estimated to be 655.4 ms, as shown in Table 6.1. During this time, the 3pi moves 4.34 cm. For the wireless implementation, the worst case initial position is 2.5 cm from the center of the 3pi to the line. If the 3pi moves 4.3 cm from this initial position, it cannot be guaranteed that the area of the 3pi will still cover some part of the line, since it depends in the angle, speed and corrective action being applied at the moment.

In the wireless implementation of the line following experiment, waiting times are the main contributor for the SFRT. To reduce the SFRT, the minimum update interval of 120 ms provided by OpenWSN needs to be reduced.

Chapter 6

Conclusions

This thesis defined a model to estimate the safety function response time (SFRT) on a wireless network implementing a feedback control loop with multiple inputs and multiple outputs. The model extends the definition of the SFRT provided by the IEC 61784-3-3 standard to consider multiple input and multiple output (MIMO) systems. The model treats the wireless transmission medium as an entity that can be adapted to accommodate the implementation of safety functions. To achieve this, the model considers the transmission medium to operate under the timeslotted channel hopping (TSCH) mode of the IEEE 802.15.4e wireless communication protocol, suitable for industrial applications and designed for process automation.

An experimental validation was performed in which the wireless SFRT model was applied to a real feedback control loop implemented in a single-hop wireless network. Processing and waiting times assumed as input by the wireless SFRT model were experimentally measured, and the wireless SFRT model equations were applied.

Three line following feedback control experiments were completed to evaluate the wireless SFRT model. Table 6.1 shows the average results of these tests for the C and CC directions, where the update interval was 8 ms for the wired implementation, and 120 ms for the wireless implementation. More detailed conclusions for each thesis

Table 6.1: Summary of the experimental results for the on board, wired and wireless experiments. *Corresponds to the minimum SFRT achievable when no packet loss is acceptable ($c_1 = 1.3$, $c_2 = 1.05$, $c_3 = 1.3$, and $c_4 = 0$).

Metric	On board	Wired	Wireless
μ_e	199.2	283.1	1451.8
d (cm)	0.2	0.3	1.7
σ_e	174.7	260.3	660.8
$\max_e$	974	1074	2000
μ_{AD} (ms)	1.9	4.8	411.3
F (s)	22.9	38.3	68.3
SFRT* (ms)	NA	38.2	655.4

activity are below.

6.1 Wireless Safety Function Response Time Model Summary

The wireless SFRT model applies the communication protocol defined in the IEEE 802.15.4e standard to the architecture defined in the IEC 61784-3-3 standard, considering that IEEE 802.15.4e packets can carry out the functions specified for safety protocol datagram units (PDUs), i.e. keep the network securely synchronized and deliver process data over one-hop communication links.

The wireless SFRT model considers periodic uplink and downlink transmissions, that is demonstrated to reduce the SFRT by a factor of 4 [3]. The model is applicable to any other timeslotted wireless communication protocol that can accommodate the model assumptions, such as the IEEE 802.15.4 (with a limit of up to 7 timeslots), and the LLDN and DSME modes of IEEE 802.15.4e. These two modes can be configured based on the assumptions made by the model, but the LLDN and DSME modes were designed for different requirements, including many devices in the network. For the wireless SFRT model, this results in many timeslots in a slotframe, and would increase the worst case delay time of transmission delay entities.

The performance of the wireless medium is non-deterministic due to many possible unpredictable communication errors, such as corruption and packet loss, among others. An important contribution from this thesis is to model the wireless communication channel to provide an equation that estimates the worst case delay time of wireless channels. By including in the model the concept of watchdog timers introduced by the IEC 61784-3-3 standard, then when wireless communication fails, the system will be guaranteed to be able to reach a safe state.

As shown in Table 4.1, mathematical constants c_2, c_3 and c_4 were introduced to the wireless SFRT model to complement the IEC 61784-3-3 constant $c_1 \in [1, 1.3]$. Constants c_2, c_3 and c_4 take into consideration variations in the response time of real devices, and the packet loss observed in wireless communications. These constants make the wireless SFRT model adaptable to different types of control.

6.2 Experimental Validation Conclusions

Experimental validation was performed on a wireless network composed of two TelosB motes running OpenWSN (see Section 4.1.2). OpenWSN is suitable for control where an update interval greater or equal to 120 ms is appropriate, and where several failed cycles are acceptable. The wireless line following experiment was compared with on board control and a wired implementation.

The 3pi was able to follow the line successfully in the wired implementation using the default control parameters provided by the Pololu Corporation. With the wireless implementation, however, the 3pi was unable to follow the line using the same control parameters. To achieve control in the wireless implementation, the parameters were changed to develop a more gradual and gentle control. Using the more gentle control, tests for the wireless implementation of the line following experiment had to be repeated 21 times to achieve 6 tests in which the 3pi was able

to follow the line to complete one lap of the course. The wireless implementation experienced an increase in the average error of 1.4 cm, or about five times worse than the wired implementation. There are two main reasons for the decline of control performance on the wireless implementation. The first is the update interval and the second is network conditions.

The update interval for the wired implementation is 8 ms, compared to 120 ms for the wireless implementation. For the wireless implementation, the update interval is dictated by the network schedule and the moment in which the mobile TelosB mote is scheduled to communicate serially with the 3pi. Serial activity is not handled in the background, so serial communication is scheduled to occur once every slotframe. Given that the minimum slotframe with two guaranteed timeslots has a duration of 120 ms, the fastest frequency for serial communication between the 3pi and the mobile mote is once every 120 ms.

The impact of an update interval of 120 ms is demonstrated by the wired implementation. When changing the update interval from 8 ms to 120 ms using the wired implementation, the average error increases from 0.3 cm to 1.5 cm. Even with perfect network conditions, i.e. no packet loss and high reliability, provided by the wired implementation, the control performance drops with a longer update interval. When comparing the wired and wireless implementations where both use an update interval of 120 ms, the increase in the average error is only 0.2 cm for the wireless implementation. This increase is due to the second reason for decline in control performance: network conditions. The average packet loss in the wireless implementation is 16%, which provides an increase in the average error of 0.2 cm compared to the wired implementation. This provides evidence that the main problem with the wireless implementation is the high minimum update interval of 120 ms.

The wireless implementation can report median error values close or equal to 2000. The wireless implementation does not provide reliable behaviour, and control

of the 3pi cannot be guaranteed, but can be kept at a safe level with the right watchdog timers and SFRT.

6.3 Future Work

The wireless SFRT model assumes a wireless network operating on a star topology with one-hop communication links between the host (PAN coordinator) and the the rest of the devices in the network. The host participates in all communication links, and there is only one slotframe to guarantee medium access. On a multi-hop network, the host will not be scheduled to participate in all the timeslots, and multiple slotframes can be used to enhance the performance of the network, since two pairs of devices could be communicating simultaneously in different frequencies without affecting each other. Future work could consider multi-hop networks, where the calculation of the optimal schedule and number of slotframes is important.

A software tool can be implemented to automatically calculate the SFRT of a given wireless network, provided that the waiting and processing times, and the values for constants are read from the user as input. This tool could help configure a given wireless network to achieve a desired SFRT value.

Future experimental validations could be done using other protocol stacks that implement timeslotted and guaranteed medium access. For example, TinyOS version 2.1.2 [16] implements the IEEE 802.15.4-2006 standard and supports allocation of guaranteed timeslots.

Implementing wireless communication from the input sensor to the host, and wired communication from the host to the output actuator constitutes a valid architecture for industrial control [29]. This architecture merges the advantages of wired and wireless technologies. Estimating the SFRT for this architecture would involve combining the worst case delay time of both wireless and wired transmission delay

entities.

To reduce the minimum SFRT achievable by wireless implementations, the waiting time restricting the update interval should be reduced. For OpenWSN, this waiting time is determined by the number of timeslots in a slotframe.

References

[1] Johan Åkerberg, *On security in safety-critical process control*, 2009, Licentiate thesis.

[2] Johan Åkerberg, Mikael Gidlund, and Mats Björkman, *Future research challenges in wireless sensor and actuator networks targeting industrial automation*, The 9th IEEE International Conference on Industrial Informatics (INDIN), IEEE Computer Society, July 26-29 2011, Caparica, Lisbon, Portugal, pp. 410–415.

[3] Johan Åkerberg, Mikael Gidlund, Tomas Lennvall, Jonas Neander, and Mats Björkman, *Efficient integration of secure and safety critical industrial wireless sensor networks*, EURASIP Journal on Wireless Communications and Networking (2011), 100.

[4] Dhananjay M. Anand, James R. Moyne, and Dawn M. Tilbury, *Performance evaluation of wireless networks for factory automation applications*, IEEE Conference on Automation Science and Engineering (CASE), August 22-25 2009, Bangalore, India, pp. 340–346.

[5] Luigi Atzori, Antonio Iera, and Giacomo Morabito, *The internet of things: A survey*, Computer Networks **54** (2010), no. 15, 2787–2805.

[6] M. Bertocco, C. Narduzzi, and F. Tramarin, *Estimation of the delay of network devices in hybrid wired/wireless real-time industrial communication systems*, Instrumentation and Measurement Technology Conference (I2MTC), 2012 IEEE International, May 2012, pp. 2016–2021.

[7] Giuseppe Bianchi, *Performance analysis of the IEEE 802.11 distributed coordination function*, IEEE Journal on Selected Areas in Communications **18** (2000), no. 3, 535–547.

[8] Delphine Christin, Parag S. Mogre, and Matthias Hollick, *Survey on wireless sensor network technologies for industrial automation: The security and quality of service perspectives*, Future Internet (2010), 96–125, `www.mdpi.com/journal/futureinternet`.

[9] Pololu Corportaion, *Pololu 3pi robot user's guide*, 2014, `http://www.pololu.com/docs/pdf/0J21/3pi.pdf`.

[10] Rene L. Cruz, *A calculus for network delay, part i: Network elements in isolation*, IEEE Transactions on Information Theory **37** (1991), no. 1, 114–131.

[11] ———, *A calculus for network delay, part ii: Network analysis*, IEEE Transactions on Information Theory **37** (1991), no. 1, 132–141.

[12] John N. Daigle, *The basic M/G/1 queueing system*, ch. 5 of Queueing Theory with Applications to Packet Telecommunication, pp. 159–223, Springer US, 2005.

[13] Dean K. Frederick and A. Bruce Carlson, *Linear systems in communication and control*, John Wiley and Sons, Inc, 1971.

[14] E. Ghadimi, A. Khonsari, A. Diyanat, M. Farmani, and N. Yazdani, *An analytical model of delay in multi-hop wireless ad-hoc networks*, Wireless Networks **17** (2011), no. 7, 1679–1697.

[15] Wolfgang Granzer and Albert Treytl, *Security in industrial communication systems*, 2nd ed., ch. 22 of The Industrial Electronics Handbook: Industrial Communication Systems, CRC Press, 2011, edited by Bogdan M. Wilamowski and J. David Irwin.

[16] Jan-Hinrich Hauer, *TKN15.4: An IEEE 802.15.4 MAC implementation for TinyOS 2*, Tech. Report TKN-08-003, Telecommunication Networks Group, Technical University of Berlin, March 2009.

[17] International Electrotechnical Commission (IEC), *IEC 61158 - Digital data communications for measurement and control - Fieldbus for use in industrial control systems*, 2003.

[18] International Electrotechnical Commission (IEC), *IEC 61511 - Functional safety - Safety instrumented systems for the process industry sector*, 2004.

[19] International Electrotechnical Commission (IEC), *IEC 61508 - Functional safety of electrical/electronic/programmable electronic safety-related systems*, 2005.

[20] International Electrotechnical Commission (IEC), *IEC 61784-3 - Industrial communication networks - Profiles - Part 3: Functional safety fieldbuses - General rules and profile definitions*, 2010.

[21] International Electrotechnical Commission (IEC), *IEC 61784-3-3 - Industrial communication networks - Profiles - Part 3-3: Functional safety fieldbuses - Additional specifications for CPF 3*, 2010.

[22] International Electrotechnical Commission (IEC), *IEC 62280 - Railway applications - Communication, signalling and processing systems - Safety related communication in transmission systems*, 2014.

[23] International Society of Automation (ISA), *ISA-100.11a-2011 Wireless systems for industrial automation: Process control and related applications*, May 2011.

[24] Ulrich Jecht, Wolfgang Stripf, and Peter Wenzel, *PROFIBUS: Open solutions for the world of automation*, ch. 14 of Integration Technologies for Industrial Automated Systems, CRC Press, 2006, edited by Richard Zurawski.

[25] Daniel Jiang, Vikas Taliwal, Andreas Meier, Wieland Holfelder, and Ralf Herrtwich, *Design of 5.9 GHz DSRC-based vehicular safety communication*, IEEE Wireless Communications (2006), 36–43.

[26] T.W. Kirkman, *Statistics to use*, 1996, `http://www.physics.csbsju.edu/stats/`.

[27] Leonard Kleinrock, *Packet switching in radio channels: Part I-Carrier sense multiple-access modes and their throughput-delay characteristics*, IEEE Transactions on Communications **Com-23** (1975), no. 12, 1400–1416.

[28] Wallace Lueders, *Wireless sensors for environmental compliance*, Chemical Engineering (2005), 40–43, Feature report `http://www.che.com`.

[29] Raul K. Mansano, Eduardo P. Godoy, and Arthur J. V. Porto, *The benefits of soft sensor and multi-rate control for the implementation of wireless networked control systems*, Sensors **14** (2014), no. 12, 24441–24461.

[30] Robert McGill, John W. Tukey, and Wayne A. Larsen, *Variations of box plots*, The American Statistician **32** (1978), no. 1, 12–16.

[31] Moteiv Corporation, *Telos Rev B (low power wireless sensor module)*, 2004.

[32] Institute of Electrical and Electronics Engineers (IEEE) Computer Society, *Part 15.4: Low-rate wireless personal area networks (LR-WPANs)*, IEEE Standards Association, June 2011.

[33] _____, *Part 15.4: Low-rate wireless personal area networks (LR-WPANs) - Amendment 1: MAC sublayer*, IEEE Standards Association, April 2012.

[34] José E. G. Oliveira, Sónia M. V . Semedo, Duarte M. G. Raposo, and Francisco J. A. Cardoso, *Place-&-play industrial router addressing potential explosive atmospheres*, 40th Annual Conference of IEEE Industrial Electronics Society, October 29-November 1 2014, Dallas, USA, pp. 3914–3918.

[35] M.R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L.A. Grieco, G. Boggia, and M. Dohler, *Standardized protocol stack for the internet of (important) things*, Communications Surveys Tutorials, IEEE **15** (2013), no. 3, 1389–1406.

[36] Victoria Pimentel and Bradford G. Nickerson, *Wireless industrial control networks*, Tech. Report TR13-223, Faculty of Computer Science, University of New Brunswick, Fredericton, January 2013, `http://www.cs.unb.ca/tech-reports/reportpage20102015.shtml`.

[37] ———, *A safety function response time model for wireless industrial control*, 40th Annual Conference of IEEE Industrial Electronics Society, October 29-November 1 2014, Dallas, USA, pp. 3878–3884.

[38] R Core Team, *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013, ISBN 3-900051-07-0.

[39] N. Salman, I. Rasool, and A. H. Kemp, *Overview of the IEEE 802.15.4 standards family for low rate wireless personal area networks*, 7th International Symposium on Wireless Communication Systems (ISWCS), September 2010, York, pp. 701–705.

[40] Ioakeim K. Samaras and George Hassapis, *A flexible analytical markov model for the IEEE 802.15.4 unslotted mechanism in single-hop hierarchical wireless networks with hidden nodes*, Wireless Personal Communications **72** (2013), no. 4, 2389–2424.

[41] Thilo Sauter, *Fieldbus systems: History and evolution*, ch. 13 of Integration Technologies for Industrial Automated Systems, CRC Press, 2006, edited by Richard Zurawski.

[42] Zach Shelby and Carsten Bormann, *6LoWPAN: The wireless embedded internet*, Wiley, 2009.

[43] Thomas Watteyne, Xavier Vilajosana, Branko Kerkez, Fabien Chraim, Kevin Weekly, Qin Wang, Steven D. Glaser, and Kris Pister, *OpenWSN: A standards-based low-power wireless development environment*, Transactions on Emerging Telecommunications Technologies **23** (2012), no. 5, 480–493.

[44] Lianming Zhang, Jianping Yu, and Xiaoheng Deng, *Modelling the guaranteed QoS for wireless sensor networks: A network calculus approach*, EURASIP Journal on Wireless Communications and Networking **82** (2011), 1–14.

[45] Weiqi Zhang and Bradford G. Nickerson, *Wireless sensor network communication protocols*, Tech. Report TR11-208, Faculty of Computer Science, University of New Brunswick, Fredericton, May 2011, `http://www.cs.unb.ca/tech-reports/reportpage20102015.shtml`.

[46] Tao Zheng, Mikael Gidlund, and Johan Akerberg, *Deterministic medium access mechanism for time-critical wireless sensor network applications*, Personal Indoor and Mobile Radio Communications (PIMRC), 2013 IEEE 24th International Symposium on, Sept 2013, pp. 1598–1602.

Appendix A

Pololu Corporation Line Following Control Implementation

The control code shown here is for the on board version of the 3pi controller. For wireless and wired implementations, the `control_calculate_action` method was created using the lines between and including those marked with an *.

```
while(1)
{
  // Get the position of the line.
  unsigned int position = read_line(sensors,IR_EMITTERS_ON);

 *// The "proportional" term should be 0 when the 3pi is on the line.
  int proportional = ((int)position) - 2000;

  // Compute the derivative (change) and integral (sum) of the position.
  int derivative = proportional - last_proportional;
  integral += proportional;

  // Remember the last position.
  last_proportional = proportional;

  // Compute the difference between the two motor power settings,
  // m1 - m2.  If this is a positive number the robot will turn
  // to the right.  If it is a negative number, the robot will
  // turn to the left, and the magnitude of the number determines
  // the sharpness of the turn.
```

```
  int power_difference = proportional/20 + integral/10000 + derivative*3/2;

  // Compute the actual motor settings.  We never set either motor
  // to a negative value.
  const int max = 60;


  if(power_difference > max)
    power_difference = max;
  if(power_difference < -max)
 *  power_difference = -max;

  if(power_difference < 0)
    set_motors(max+power_difference, max);
  else
    set_motors(max, max-power_difference);
}
```

Appendix B

Wireless Experimental Set-up

Figures B.1 and B.2 show pictures of the wireless line following experimental set-up. The OpenWSN network is conformed of the TelosB DAG root and mobile motes. The DAG root mote communicates serially with the Ubuntu workstation, and the mobile mote communicates serially with the Pololu 3pi.

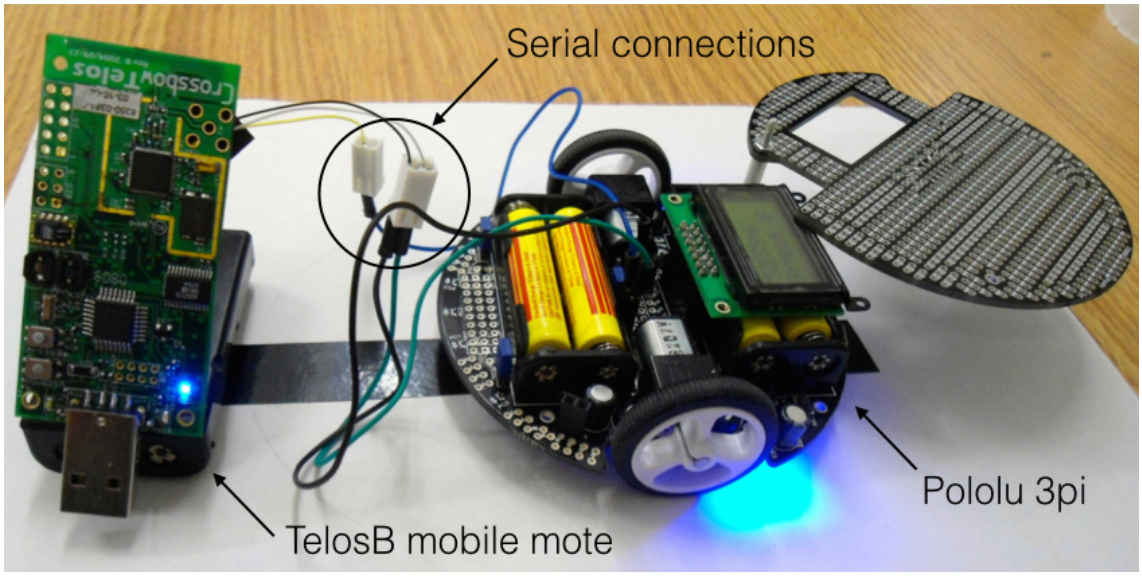


Figure B.1: TelosB mobile mote connected to the Pololu 3pi.

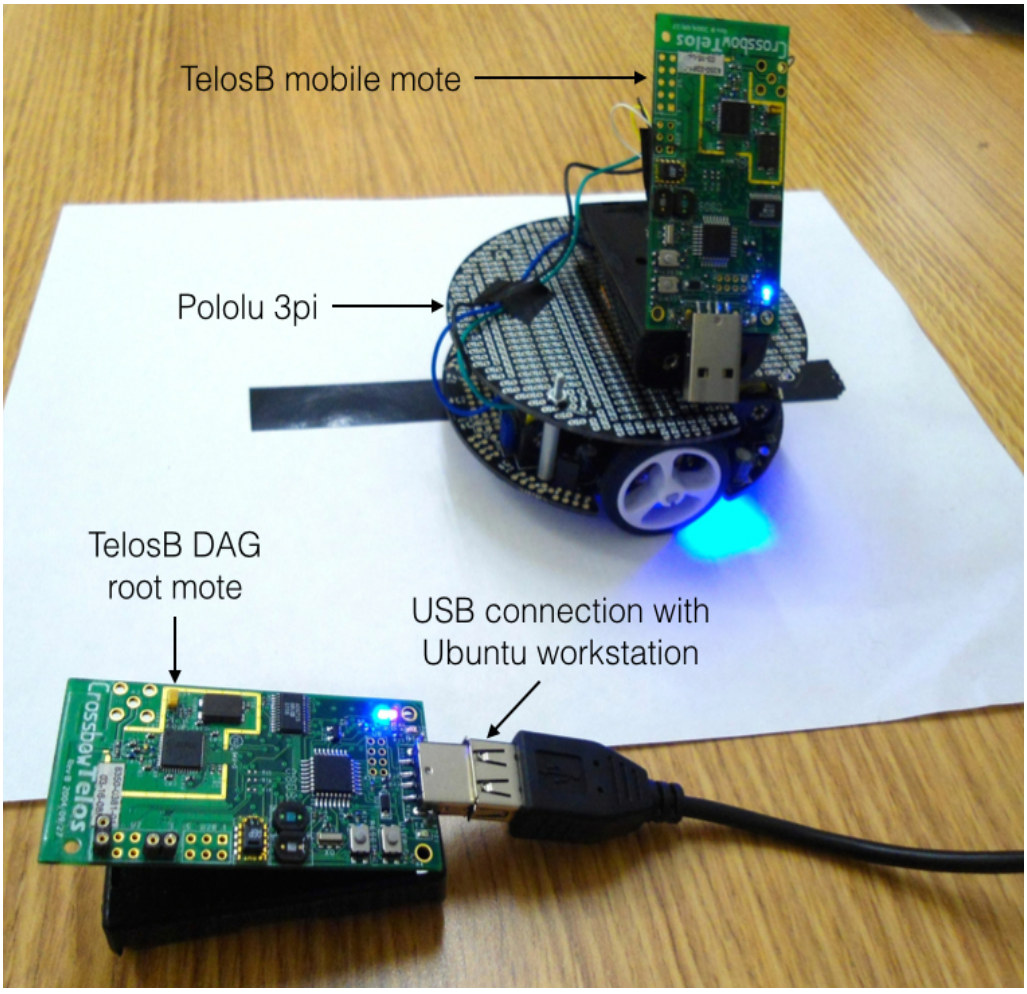


Figure B.2: The TelosB DAG root mote communicates wirelessly with the TelosB mobile mote.

Appendix C

Wired Experimental Set-up

Figures C.1 and C.2 show pictures of the wired line following experimental set-up. Serial communication is used between the Pololu 3pi, the Pololu USB AVR programmer, and the Ubuntu workstation.

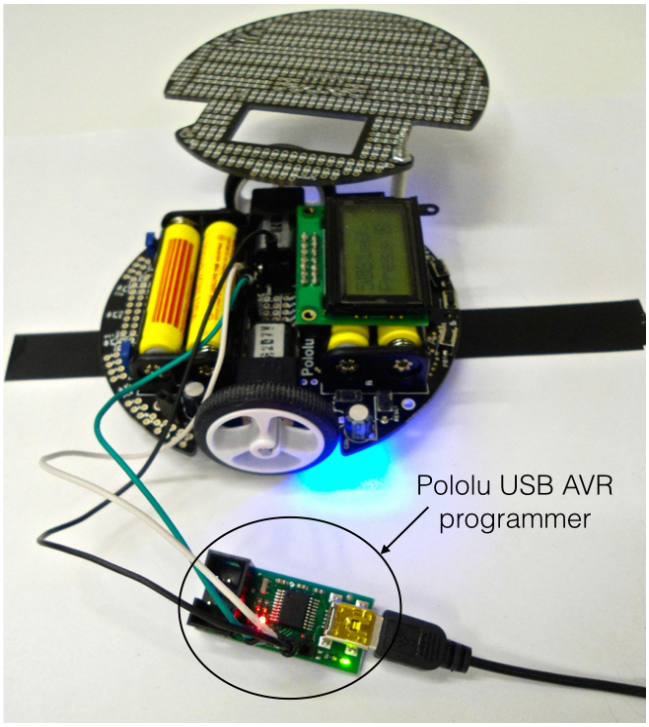


Figure C.1: Pololu USB AVR programmer connected to the UART RX and TX lines on the Pololu 3pi.

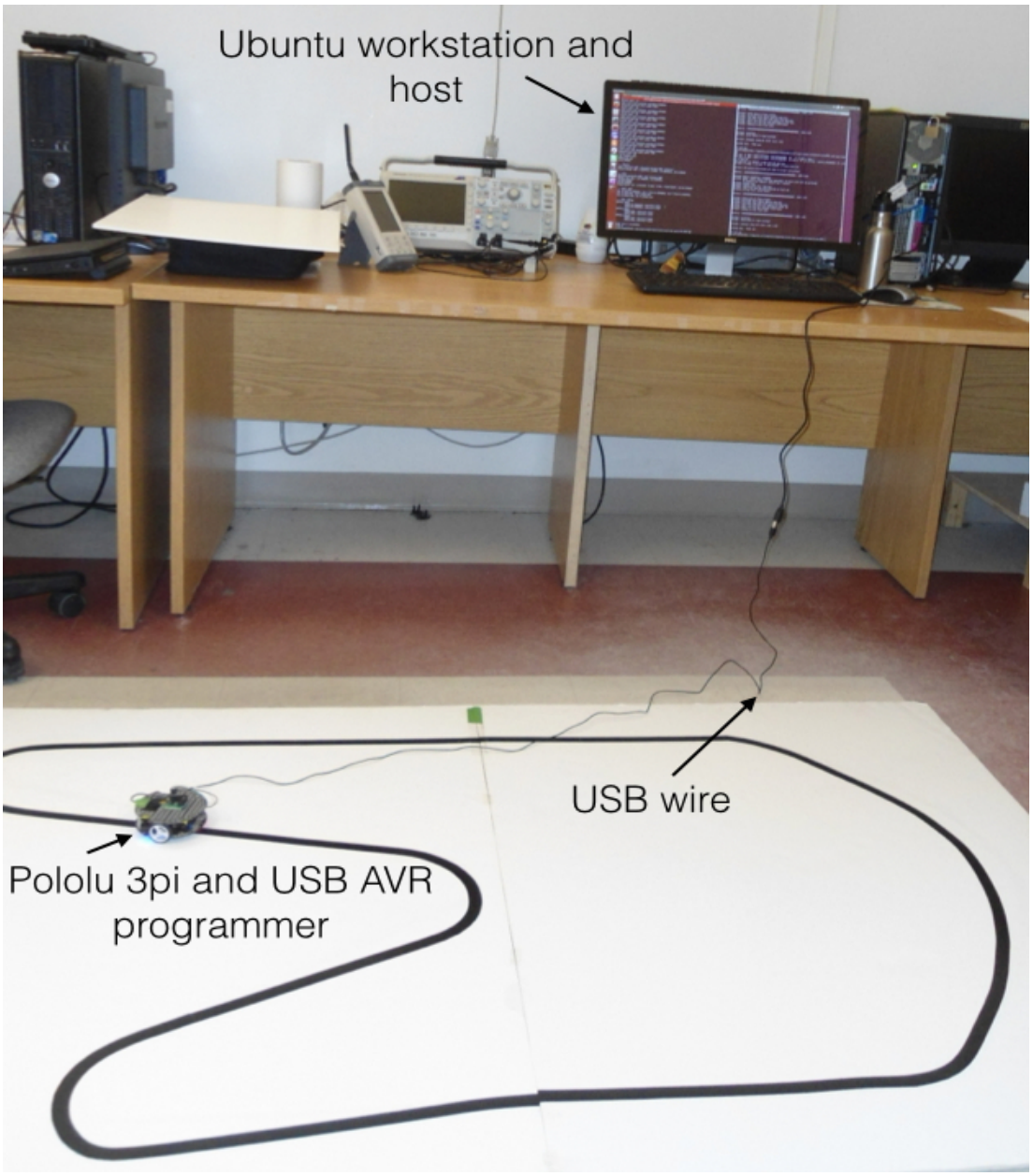


Figure C.2: Host connected over a USB wire to the Pololu USB AVR programmer on the line following course.

Appendix D

Udpprint Application

The udpprint application runs on the mobile TelosB mote as part of the OpenWSN firmware. The udpprint application forwards UDP frames received wirelessly to the TelosB UART1 serial pins. The udpprint application is implemented as follows:

```
#include "openwsn.h"
#include "udpprint.h"
#include "openqueue.h"
#include "openserial.h"

//=========================== variables =======================================

//=========================== prototypes ======================================

//=========================== public ==========================================

void udpprint_init() {
   //Udpprint is for communication from host to mobile only
}

//This method will be triggered when the mobile receives a packet from
//host. This methods forwards the data received on UDP packet over serial
void udpprint_receive(OpenQueueEntry_t* msg) {
   //No handshake required
   if(!idmanager_getIsDAGroot()){
      openserial_printData((uint8_t*)(msg->payload),msg->length);
   }
   openqueue_freePacketBuffer(msg);
}
```

```
void udpprint_sendDone(OpenQueueEntry_t* msg, owerror_t error) {
   openserial_printError(
      COMPONENT_UDPPRINT,
      ERR_UNEXPECTED_SENDDONE,
      (errorparameter_t)0,
      (errorparameter_t)0
   );
   openqueue_freePacketBuffer(msg);
}

bool udpprint_debugPrint() {
   return FALSE;
}

//========================== private ==================================
```

Appendix E

Udpinject Application

The udpinject application runs on the mobile TelosB mote as part of the OpenWSN firmware. The udpinject application generates a UDP packet from a received serial frame. The udpinject application is implemented as follows:

```
#include "openwsn.h"
#include "udpinject.h"
#include "openudp.h"
#include "openqueue.h"
#include "openserial.h"
#include "packetfunctions.h"

//========================== variables =================================
uint8_t host_addr_128b[] =
  {0xbb, 0xbb, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01};


//========================== prototypes ================================

//========================== public ====================================

void udpinject_init() {
  //Udpinject is for communication from mobile to host only
}

//This method will be triggered when the mobile mote receives a U serial
//frame
void udpinject_trigger() {
```

```
  OpenQueueEntry_t* pkt;
  uint8_t number_bytes_from_input_buffer;
  uint8_t input_buffer[255];
  int i;

  number_bytes_from_input_buffer = openserial_getInputBuffer(
    &(input_buffer[0]),
    sizeof(input_buffer));

  //Prepare packet
  pkt = openqueue_getFreePacketBuffer(COMPONENT_UDPINJECT);
  if (pkt==NULL) {
    openserial_printError(COMPONENT_UDPINJECT,ERR_NO_FREE_PACKET_BUFFER,
      (errorparameter_t)0,
      (errorparameter_t)0);
    return;
  }
  pkt->creator                     = COMPONENT_UDPINJECT;
  pkt->owner                       = COMPONENT_UDPINJECT;
  pkt->l4_protocol                 = IANA_UDP;
  pkt->l4_sourcePortORicmpv6Type   = WKP_UDP_INJECT;
  pkt->l4_destination_port         = WKP_UDP_INJECT;
  pkt->l3_destinationAdd.type      = ADDR_128B;

  //Address to host
  memcpy(&(pkt->l3_destinationAdd.addr_128b[0]),&host_addr_128b,16);
  packetfunctions_reserveHeaderSize(pkt,number_bytes_from_input_buffer);
  for(i=0; i<number_bytes_from_input_buffer; i++)
    ((uint8_t*)pkt->payload)[i]  = input_buffer[i];

  //Send packet
  if ((openudp_send(pkt))==E_FAIL)
    openqueue_freePacketBuffer(pkt);
}

void udpinject_sendDone(OpenQueueEntry_t* msg, owerror_t error) {
  msg->owner = COMPONENT_UDPINJECT;
  if (msg->creator!=COMPONENT_UDPINJECT) {
    openserial_printError(COMPONENT_UDPINJECT,ERR_UNEXPECTED_SENDDONE,
      (errorparameter_t)0,
      (errorparameter_t)0);
  }
  openqueue_freePacketBuffer(msg);
}

//This method will be triggered when the mobile receives a handshake
//packet from host
void udpinject_receive(OpenQueueEntry_t* msg) {
```

```
  OpenQueueEntry_t* pkt;
  pkt = openqueue_getFreePacketBuffer(COMPONENT_UDPINJECT);
  if (pkt==NULL) {
    openserial_printError(COMPONENT_UDPINJECT,ERR_NO_FREE_PACKET_BUFFER,
      (errorparameter_t)0,
      (errorparameter_t)0);
    return;
  }
  pkt->creator                    = COMPONENT_UDPINJECT;
  pkt->owner                      = COMPONENT_UDPINJECT;
  pkt->l4_protocol                = IANA_UDP;
  pkt->l4_sourcePortORicmpv6Type  = WKP_UDP_INJECT;
  pkt->l4_destination_port        = WKP_UDP_INJECT;
  pkt->l3_destinationAdd.type     = ADDR_128B;

  //Address to host
  memcpy(&(pkt->l3_destinationAdd.addr_128b[0]),&host_addr_128b,16);
  packetfunctions_reserveHeaderSize(pkt,2);
  ((uint8_t*)pkt->payload)[0]     = 'O';
  ((uint8_t*)pkt->payload)[1]     = 'K';

  //Send packet
  if ((openudp_send(pkt))==E_FAIL)
    openqueue_freePacketBuffer(pkt);
  openqueue_freePacketBuffer(msg);
}

bool udpinject_debugPrint() {
  return FALSE;
}
//========================== private ==================================
```

Appendix F

Handling Request Frames

Request frames are sent from the mobile TelosB mote to the Pololu 3pi once during every slotframe, i.e. once every 120 ms. When a request frame is received at the 3pi, the following method is executed:

```
//3pi got a REQUEST frame, handle it and generate the response with
//the latest line position reading
void hdlcRes_request(long serial_duration)
{
   int i, j, action_delay;
   uint8_t one_byte;
   unsigned int tmp_sensors[5];
   long time, w_input, p_input;
   long  p_start, p_end, refresh;

   //Start counting for processing time
   p_start = get_ms();

   //Get time since last R frame and store maximum as
   //waiting time from input entity
   if(w_input_0 == 0)
      w_input_0 = get_ms();
   else {
      w_input_1 = get_ms();
      w_input = w_input_1 - w_input_0;
      if(w_input > max_w_input)
         max_w_input = w_input;
      w_input_0 = w_input_1;
   }
```

```
//If we have not reached the end of test
if(!check_end()){
   //On every SERIALRX we will send new line position
   //Won't check CRC - no payload
   outputHdlcOpen();                          //1b
   outputHdlcWrite('U');                      //1b

   //Indicate this is position message
   outputHdlcWrite('P');                      //1b

   //Read line position
   line_position = read_line(tmp_sensors, IR_EMITTERS_ON);

   //Store current time
   time = get_ms();
   if(array_index == ARRAYSIZE)
      array_index = 0;
   sequence_numbers[array_index] = Rframe;
   reading_times[array_index] = time;
   array_index++;

   //Add line position
   for(i=0; i<NUMBYTES; i++){
      one_byte = (line_position >> 8*i) & 0xff;
      outputHdlcWrite(one_byte);              //1B
   }

   //Add sequence number
   for(i=0; i<NUMBYTES; i++){
      one_byte = (Rframe >> 8*i) & 0xff;
      outputHdlcWrite(one_byte);              //1B
   }

   //If there are any action delays to report, add them
   if(action_delays_index > 0){
      for(i=0; i<action_delays_index; i++){
         action_delay = action_delays[i];
         for(j=0; j<NUMBYTES; j++){
            one_byte = (action_delay >> 8*j) & 0xff;
            outputHdlcWrite(one_byte);        //1B
         }
      }
      action_delays_index = 0;
   }
   outputHdlcClose();                         //3b

   //Send U frame
```

```
      serial_send_blocking((char *)openserial_vars.outputBuf,
         openserial_vars.outputBufIdxW);
      openserial_vars.outputBufIdxW = 0;
      Rframe++;
   }

   //Refresh all input WDTimes
   refresh = get_ms();
   for(i=0; i<NWDTIMES; i++)
      input_refresh[i] = refresh;

   // Processing ends
   p_end = get_ms();
   p_input = p_end - p_start;
   p_input += serial_duration;
   if(p_input > max_p_input)
      max_p_input = p_input;
}
```

Appendix G

Generation of a Corrective Action

Corrective actions are generated at the host running at the Ubuntu workstation. Every time a new line position is received at the host, the corresponding corrective action is calculated and transmitted back to the 3pi. The controller shown in Appendix A is implemented by the `control_calculate_action` method, which is called by the following method:

```
//A new line position has been received at the host with the sequence
//number seq_num, calculate its corresponding corrective action
//and send it over the socket
double act(int seq_num){

   struct timeval process_start, process_end;
   double p_fhost, wdtime;
   uint8_t byte;
   uint8_t action_msg[MSGBYTES-1];
   int i;

   //Calculate corrective action
   action = control_calculate_action(line_position);

   //Start measuring process
   gettimeofday(&process_start, NULL);

   //Add corrective action
   int_to_uint8(action_msg, action, 0);
```

```
   //Add sequence number
   int_to_uint8(action_msg, seq_num, 4);

   //Send message
   fprintf(log_file, "Corrective action %d number %d\n", action, seq_num);
   printf("Corrective action %d number %d\n", action, seq_num);

   //We are not sending corrective action for first line position received
   //since it is always the maximum corrective action
   if(positions > 1){
      if(sendto(socket_udpprint, action_msg, sizeof(action_msg),
            MSG_DONTWAIT, (struct sockaddr *) &mobile_udpprint,
            sizeof(mobile_udpprint)) < 0) {
         perror("Error sendto failed");
         fprintf(log_file, "Error sendto failed while sending corrective
            action");
      } else {
         actions++;
         fprintf(log_file, "Corrective action sent\n");
         gettimeofday(&current_action_sent, NULL);

         //WDTime calculation
         wdtime = (double) (current_action_sent.tv_usec -
            prev_action_sent.tv_usec) / 1000000 +
            (double) (current_action_sent.tv_sec -
            prev_action_sent.tv_sec);

         //Report if any WDTime expired
         for(i=0; i<NWDTIMES; i++){
            if(wdtime > wdtimes_fhost[i])
               wdexpirations_fhost[i]++;
         }
         prev_action_sent = current_action_sent;
      }
   } else
      gettimeofday(&prev_action_sent, NULL);

   //Process done
   gettimeofday(&process_end, NULL);
   p_fhost = (double) (process_end.tv_usec - process_start.tv_usec)
      / 1000000 + (double) (process_end.tv_sec - process_start.tv_sec);

   return p_fhost;
}
```

Appendix H

Handling Data Frames

When a corrective action is received at the mobile TelosB mote, the mote sends a data frame to the Pololu 3pi. When a data frame is received at the 3pi, the following method is executed:

```
//3pi got a DATA frame, check it, extract the corrective action
//(power_difference) and apply it to the motors
void hdlcRecv_data(long serial_duration)
{
   int i, in_array, action_delay;
   int j = 0;
   int power_difference = 0;
   int seq_num = 0;
   uint8_t one_byte = 0;
   long  w_output, p_output, p_start, p_end, refresh;

   //Start counting for processing time
   p_start = get_ms();

   //Get time since last D frame and store maximum as waiting time
   //from output entity
   if(w_output_0 == 0)
      w_output_0 = get_ms();
   else {
      w_output_1 = get_ms();
      w_output = w_output_1 - w_output_0;
      if(w_output > max_w_output)
         max_w_output = w_output;
      w_output_0 = w_output_1;
```

```
}

inputHdlcOpen();
for(i=0; i<pointer; i++){
   inputHdlcWrite((uint8_t) message[i]);
}
inputHdlcClose();

//Check CRC
if(openserial_vars.inputBufFill != 0){             //PASS
   Dframe++;
   //Get corrective action
   for(i=8; i<8+NUMBYTES; i++){
      one_byte = openserial_vars.inputBuf[i];
      power_difference |= one_byte << 8*j;
      j++;
   }

   //Apply corrective action
   if(power_difference < 0)
      set_motors(max+power_difference, max);
   else
      set_motors(max, max-power_difference);

   //Store current time
   refresh = get_ms();

   //Refresh all WDTimes
   for(i=0; i<NWDTIMES; i++)
      output_refresh[i] = refresh;

   //Get sequence number
   j = 0;
   for(i=8+NUMBYTES; i<8+2*NUMBYTES; i++){
      one_byte = openserial_vars.inputBuf[i];
      seq_num |= one_byte << 8*j;
      j++;
   }

   //Calculate action delay
   in_array = search_in_array(seq_num);
   if(in_array != -1){
      if(action_delays_index == ARRAYSIZE)
         action_delays_index = 0;
      action_delay = refresh - reading_times[in_array];
      action_delays[action_delays_index] = action_delay;
      action_delays_index++;
   }
```

```
   } else                                                    //FAIL
      Eframe++;

   //Processing ends
   p_end = get_ms();
   p_output = p_end - p_start;
   p_output += serial_duration;
   if(p_output > max_p_output)
      max_p_output = p_output;
}
```

Appendix I

Box Plots

The box plot [30] [26] is a standardized way of displaying a data set, an example is shown in Figure I.1. The box plot incorporates five important elements: the lower and upper extremes (labelled in Figure I.1 with the letters F and B, respectively), the median (labelled with the letter D in Figure I.1), and the first and third quartiles (labelled in Figure I.1 with the letters E and C, respectively). The median splits the data set in half, which means that 50% of the data is contained above the horizontal line labelled D in Figure I.1, and the other 50% of the data is contained below.

The first quartile or lower quartile splits the lowest 25% of the data, and the second quartile splits the lowest 75% of the data from the highest 25%. The inter-quartile range (the rectangle in the box plot in Figure I.1) is defined as the range from the first quartile to the third quartile, and it includes 50% of the samples.

The box plot is a powerful representation of a data set, since it shows the range of the data set (from the lower extreme, labelled F in Figure I.1, to the upper extreme, labelled B in Figure I.1), the likely range of the data set (the inter-quartile range, from labels E to C in Figure I.1), and the typical value (the median, labelled D in Figure I.1).

Label A in Figure I.1 indicates the outliers present in the data set. Outliers

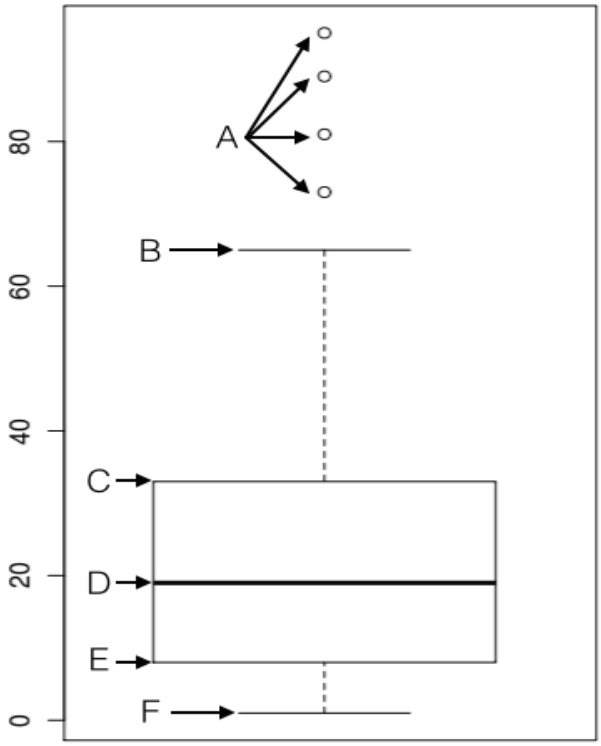


Figure I.1: Example of a box plot.

are defined as samples with statistically unexpected high or low values. There are different methods for calculating outliers, for example values that are above three times the inter-quartile range from the third quartile, and values that are below three times the inter-quartile range from the first quartile. A high concentration of outliers shows up in a box plot as a vertical thick line resulting from many overlapping circles.

Box plots in this thesis were generated using the R software [38].

Appendix J

Jackdaw Sniffer Set-up

The Atmel RZUSBstick is a USB stick with a 2.4GHz transceiver and a USB connector. The Contiki Raven firmware was loaded to the RZUSBstick. This combination of hardware and firmware is referred to as the Jackdaw.

The download site for the pre-compiled binaries for the Contiki Raven firmware can be accesed at the following location:

`http://cydanil.net/wiki/index.php?title=Flashing_Contiki_Jackdaw_onto_Raven_USB_Stick`

Once the pre-compiled binaries for the Contiki Raven firmware are downloaded and unpacked at the `contiki-raven-2.5` directory, the following commands load the Contiki Raven firmware to the RZUSBstick:

```
>$ cd contiki-raven-2.5/
>$ avr-objcopy -R .eeprom -R .fuse -R .signature -O ihex ravenusbstick.elf
     ravenusbstick.flash
>$ avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load"
     --change-section-lma .eeprom=0 -O ihex ravenusbstick.elf
     ravenusbstick.eeprom
>$ sudo avrdude -c jtagmkII -P usb -p usb1287 -Ueeprom:w:ravenstick.eeprom
     -Uflash:w:ravenstick.flash
```

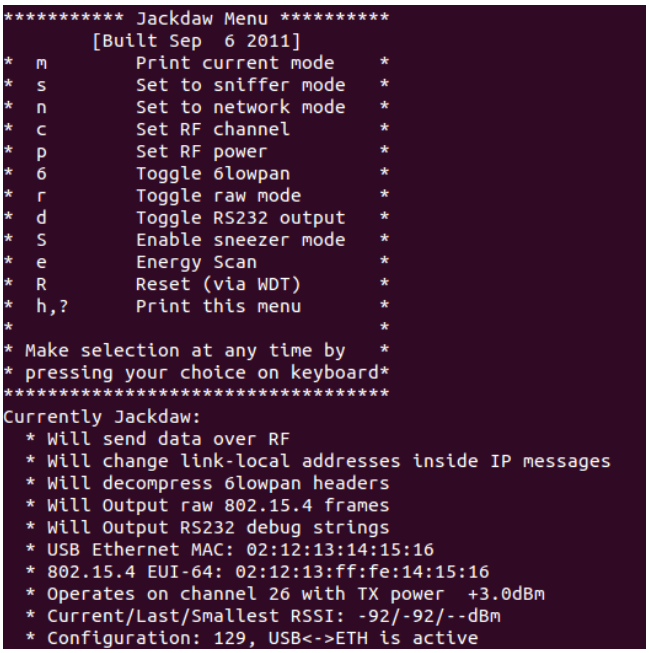


Figure J.1: Jackdaw menu and configuration.

After running these commands, the RZUSBstick should be unplugged and plugged again, so the firmware is rebooted. After plugging the RZUSBstick again, running the `dmesg` command should show an output containing similar messages as the following:

```
[13538.905094] usb 2-1: New USB device found, idVendor=03eb, idProduct=2021
[13538.905102] usb 2-1: New USB device strings: Mfr=1, Product=2,
   SerialNumber=3
[13538.905107] usb 2-1: Product: Jackdaw 6LoWPAN Adaptor
[13538.905112] usb 2-1: Manufacturer: Atmel
[13538.905116] usb 2-1: SerialNumber: 021213141516
```

Running the `ifconfig` command should show a `usb0` network interface to access 6LoWPAN devices. This interface can be accessed from Wireshark to display the messages captured by the Jackdaw. A terminal emulation progran, e.g. Minicom, can be attached to serial port USB0 to access and configure the Jackdaw. A screenshot of the Jackdaw menu and configuration is shown in Figure J.1.

Appendix K

OpenWSN Medium Access

Figure K.1 shows a screenshot of a Wireshark window, where packets transmitted on the OpenWSN network are captured. The address of the mobile mote ends in 6d11, and the address of the DAG root mote ends in 6c49. The 4 packets highlighted in red contain line positions sent from the mobile mote (source address ends in 6d11) to the host via the DAG root mote. The acknowledgement for the first packet highlighted in red is the following message captured, with the Info field starting with Ack. Figure K.1 shows that the DAG root mote answers with the corresponding acknowledgement to 3 of the 4 packets highlighted in red. For the third packet highlighted in red, Wireshark does not display the expected acknowledgement, but instead a beacon request is captured. This third packet is reported at the mobile mote as a failed transmission. Wireshark does not display the source of the beacon request, but the source is possibly the DAG root mote. The beacon request sender seems to be transmitting when the mobile mote is waiting for an acknowledgement from the DAG root mote, which indicates a conflict in the medium access. The beacon request is a broadcast message, as shown in Figure K.1. Broadcast messages should not be transmitted during guaranteed timeslots.

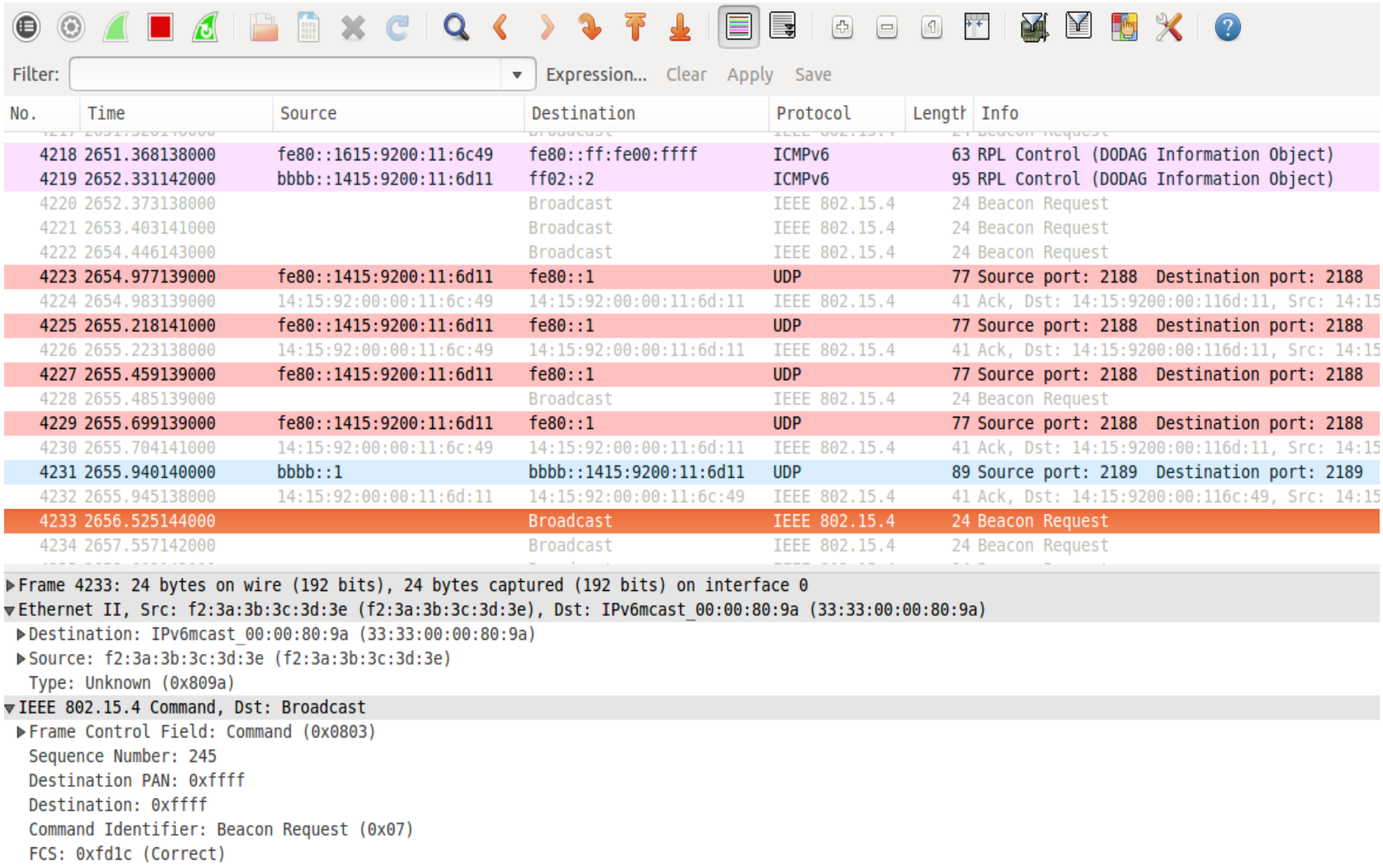


Figure K.1: Wireshark window showing OpenWSN frames.

Vita

Candidate's full name: Victoria Pimentel Guerra
Place and date of birth: Caracas, Venezuela, November 3, 1989

Universities attended:
Master in Computer Science (2012-2015)
University of New Brunswick
Fredericton, Canada

Computing Engineering (2007-2012)
Universidad Simón Bolívar
Caracas, Venezuela

Publications:
Victoria Pimentel and Bradford G. Nickerson. Communicating and Displaying Real-Time Data with WebSocket. IEEE Internet Computing Magazine, vol. 16, no. 4, pp. 45-53, July/Aug. 2012

Conference presentations:
Victoria Pimentel and Bradford G. Nickerson. A Safety Function Response Time Model for Wireless Industrial Control. The 40th Annual Conference of the IEEE Industrial Electronics Society, Oct. 29-Nov. 1 2014, Dallas, US, pp. 3878-3884
Best presentation recognition, special session 16

Victoria Pimentel and Bradford G. Nickerson. Estimating the Safety Function Response Time for Wireless Sensor Networks. University of New Brunswick Faculty of Computer Science Research Expo 2014, Fredericton, Canada

Other documents:
Victoria Pimentel and Bradford G. Nickerson, Wireless Industrial Control Networks, Technical Report TR13-223, Faculty of Computer Science, University of New Brunswick, Fredericton, January 2013.