Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Distributed People Counting Using a Wireless Sensor Network

Semester Thesis

Patrick Senti

patricksenti at student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Philipp Sommer
Prof. Dr. Roger Wattenhofer

February 2011 - August 2011

Acknowledgements

This thesis would not have been possible without the support by Philipp Sommer and Prof. Dr. Roger Wattenhofer, who provided me the opportunity to engage in this work. I thank Philipp Sommer for supporting my initial idea that lead to this thesis, and for his valuable feedback to intermediary results and to the draft of this report. He also provided helpful insights to implement a functional hardware/software prototype of the people-counting sensor node. I am thankful to Prof. Dr. Roger Wattenhofer for the opportunity to complement my studies programme, CAS in Computer Science (Distributed Systems), by this work and in the focus area of his research group.

Abstract

People observation and counting is of interest in many commercial and non-commercial scenarios. The number of people entering and leaving shops, the occupancy of office buildings or the passenger count of commuter trains provide useful information to shop merchants and marketers, security officials or train operators. To this end, this thesis develops a distributed people counting system, covering algorithms and their implementation on wireless sensor nodes.

The algorithms presented address the people counting problem by interpreting the infrared sensor signal as system state. In particular, a dynamically configurable FSM is proposed based on regular expressions. In combination with several parameters related to sensor signal patterns the algorithms are adjustable to a range of deployment scenarios. The algorithms are provided as both a simulation model and the corresponding sensor node implementation.

The prototype implementation provides a setup where each sensor node is equipped with one passive, dual-element infrared sensor (PIR). For evaluation, a prototyping processor board and low-cost PIR sensors were used to build two wireless sensor nodes from commercially available components. A feature of this implementation is that each sensor node is able to count, on-system and in real-time, the number of passers-by going in each direction, while requiring relatively little compute power.

A wireless sensor network is described to monitor the count values of distributed nodes and collect these data at a base station. The base station transmits the count data to a web-based data collection server, where it can be made available for further use to applications and client devices, such as smartphones. To enable the configuration of the sensor nodes to specific installation scenarios, and for the purpose of performance analysis, a validation framework is presented.

A stochastic state estimation model is considered to resolve the problem of relatively noisy people count measurements by sensors. In particular, the Kalman Filter is explored as a first approximation, and the corresponding model is evaluated using MATLAB.

Keywords: people counting, pir sensor, dual-element pir sensor, pattern matching, regular expression, kalman filter, wireless sensor network

Contents

CHAPTER 1

Motivation and Related Work

1.1 Motivation

The purpose of this thesis is to evaluate the use of passive infrared (PIR) sensors for people counting, with the sensors deployed in a wireless network. The people counting problem is an interesting problem as it serves as the basis for many commercial and security applications.

The specific application that motivated this thesis is the allocation of free space in commuter trains: When boarding commuter trains in a densely populated sub-urban area, commuters often find themselves boarding the train into the same passenger car as many others. Consequently it may be difficult to find a free seating place. While the train can appear overcrowded from a passenger's local perspective, there might in fact be a lot of free seats elsewhere. The situation can also fluctuate with different trains and days such that it becomes unpredictable for the individual commuter to find an "ideal" place. Using sensor nodes to count passengers automatically could help to improve the situation: observing the train occupancy would enable a system to indicate available seats to passengers, even prior to boarding. Directing commuters towards free seats would allow more convenience for commuters and make better use of the resources allocated by the train company. More generally, such a system could be integrated in a broader scenario linking multiple transport types (trains, buses, private cars) such that passengers could find the best route at any given time.

While preparing for the thesis, it became clear that a cost-efficient, distributed solution to the people counting problem is a precondition to the realization of the above application scenario. A first review of the research literature indicated that many technologies used are either too costly (e.g. video cameras), or apply wireless sensor networks (WSN) primarily to *in-network data analysis and distributed counting algorithms*, rather than the *communication* of actual people counts. In consequence, such systems hold an inherent level of complexity that increases the cost of deployment and operation for commercial scenarios. This background motivated us to focus the thesis on finding methods and algorithms, such that all processing and signal analysis, in respect to the counting process, *shall be integrated on each sensor node*, *and the sensor network shall be applied to communicating* final count data.

1.2 People Counting Technology

People counting is a widely studied and commercially exploited subject. This section briefly reviews the typical technologies used for people counting:

Video Cameras In [1] the authors describe an approach to people counting (and localization) using multiple video cameras. The focus lies on extracting the size and moving patterns of individuals passing. By means of motion histograms based on frame-differenced images, the histograms classify detected movements. Probabilistic correlation is applied to determine a people count. The results of multiple cameras are joined in order to form a movement vector for each individual recognized. In contrast, [2] proposes a solution based on a single ceiling-mounted camera, which identifies people by background extraction of the camera image. A non-background "blob" is recognized, and its size is estimated and compared to previously established bounds of people's pixel dimensions. A people count is derived from the results of this analysis. The system reaches a claimed accuracy of 98.5%. The major disadvantage of a camera-based system is that it requires an ambient light source and relatively powerful compute resources to perform image processing.

Ultrasonic Sensors The authors of [3] introduce a system employing ultrasonic sensors. Per each observed area a three-node sensor cluster is established, whereby each sensor node mounts an ultrasonic sensor. Multiple clusters are joined to cover a wider area. Nodes in each cluster communicate sensor readings by an RF link to the cluster's coordinator node. The latter contributes its own sensor measurements. By means of a distributed algorithm, nodes decide on whether to count a detected person. The sensor nodes require clock synchronization at the millisecond level in order to correlate the data exchanged. Despite the availability of clock synchronization protocols this imposes a disadvantage to this approach. The system achieves an overall counting accuracy of 90% using a probabilistic estimate of the total count, despite individual clusters achieving only around 50-70% accuracy.

Infrared Sensor Arrays A system using infrared (IR) arrays and pattern recognition algorithms is described in [4]. IR arrays combine a matrix of IR sensors to form array detectors. As the name suggests the sensor signals are provided as a matrix, where each element of the matrix corresponds to one IR sensor. Pattern recognition algorithms are able to detect people moving across the sensor's view at a claimed accuracy of 95%. This holds true even if two pedestrian's paths cross, or people walk in parallel. IR arrays provide a cost-effective solution and also operate without any ambient light source. IR arrays are widely used in commercial systems[1].

[1] e.g. http://www.pyreos.com/products/people-and-door-counting.html

Infrared Motion Sensors A people counting system based on PIR motion detectors is presented in [5]. For each passage monitored, three PIR sensors are installed at a distance of 0.8m. The sensors are connected to a coordinator by a wireless RF link. Sensors detect motion events and send these data to the coordinator. The coordinator infers a people count from correlating the number, phase and time-difference of peaks found in the signal. The system achieves a rate of 100% to detect the direction of movement, and accurately detects 89% of the number of people passing. PIR sensors provide an alternative to IR sensor arrays, however the cost and effort of employing multiple sensor nodes for each entry/exit point is a cost-side disadvantage. The goal of this thesis is to develop a system based on just one PIR sensor and one sensor node per each observed entry/exit point.

Sensor Fusion Results of a building occupancy estimation system applying different types of sensors is found in [6]. The system consists of camera, CO_2 and PIR sensors. It uses a Hidden Markovian Model (HMM) based on an Extended Kalman Filter (EKF) in order to derive building occupancy. The approach integrates historical data and current sensor readings to estimate the true state of the system, adjusting for sensor noise (false observations) and stochastic processes, e.g. uncertain people movement patterns.

1.3 Related Work

This section reviews different approaches specifically applying PIR sensors.

In [7] the authors evaluate a system of multiple PIR sensors[2] mounted in two parallel rows to detect the pedestrian's moving direction. The sensors are connected to Tmote Sky nodes and transmit their data to a base station computer for analysis by an echo-state neural network. The neural network is trained to recognize moving patterns and derives people counts accordingly. In this thesis we will present an algorithm that recognizes patterns of movement by analysing the output of only one PIR sensor, and where the analysis is carried out by the sensor node itself. As a result, there is no subsequent analysis step to be executed at the base station. This approach is motivated by the less complex setup in terms of network protocol and hardware requirements.

Several authors point out that PIR sensor signals contain noise, and thus motivate the need for signal filtering [8, 9]. A moving average filter is proposed in [9] combined with a movement detection algorithm that adapts to different noise levels in an outside environment. The authors observe that signal noise is generally of lower frequency than the signal of moving targets, and that the noise level depends on weather conditions. To counter this effect, their algorithm adapts an estimated noise level by probabilistic methods. This contrasts to our experience using dual-element PIR sensors: such sensors automatically compensate for changes in the environment, and the noise level in general is of higher frequency than an actual signal.

[2] Commercially available from Hygrosens, the same vendor that provided the PIR sensors for this thesis.

[8] develops signal processing algorithms for two dual-element PIR sensors mounted such that their field-of-view (FOV) is shifted to face opposite directions. The algorithms extract a sensor signal for each passing person by finding and separating higher-frequency windows. The direction of movement is derived as a function of the phase shift of the two sensors' signals. Accordingly, the people count per each direction is the number of identified windows for each direction. The authors observe that in case of multiple people walking in a queue the peaks are no longer separable by this approach. Our algorithm presented in Section 4.7 is able to detect multiple people walking in a queue as it does not rely on separating signal frequencies (other than for noise reduction) or signal phase shift. Instead, it identifies the direction of movement based on the pattern of the first detected movement, and counts the number of subsequent peaks.

Distributed people counting systems using wirelessly connected sensor nodes are discussed in [3, 5, 7, 9]. The systems employ existing networking algorithms and protocols, available as part of the hardware platforms and operating system used (Tmote Sky, Mica2, TinyOS). The ZigBee protocol and corresponding commercial RF modules are proposed in [5] because the protocol features a low-power operation mode, uses low data rates and enables to cover a wide area by means of multi-hop data collection. The same reasons motivate the use of a ZigBee RF module for our data monitoring and collection approach presented in Section 5.3.

Estimating the real state of a system based on uncertain measurements, such as people counts by PIR sensors, is addressed in [10]. In the context of a building evacuation scenario a non-linear stochastic process and sensor model is developed. An Extended Kalman Filter (EKF) is used to estimate the true state of room occupancy based on previously observed occupancy patterns and current measured sensor data. Likewise, a linear Kalman Filter is employed in [11] for a person tracking and localization application. Motivated by these results, Section 5.5 proposes our solution approach to estimate the true room occupancy.

1.4 Scope and Outline

The main focus of the thesis is on the development of a self-contained People Counting Sensor Node (PCSN), based on low-cost hardware and using a single dual-element PIR sensor. The distributed nature of people counting scenarios is taken into account by the description and realization of a web-based data collection server, as well as by providing a solution sketch for the problem of stochastic state estimation.

The remainder of the document is organized as follows. Chapter 2 introduces and defines the problems addressed. In Chapter 3, theoretical background to our solution approach is presented. The development of algorithms and the technical realization of the distributed people counting system are the subject of Chapters 4 and 5. An evaluation of the proposed algorithm and its sensor node implementation is presented in Chapter 6. The thesis concludes in Chapter 7 with a brief discussion of findings.

CHAPTER 2

Problem Definition

2.1 People Counting using Dual-Element PIR Sensors

We first define the term *People Counting*:

Definition 1 (People Counting)**.** The process of counting the number of people passing in and out of an observed area during a period of discrete time $k_0, \ldots, k_n$. At each instance of time k the process results in two counts $c_{k,in}$ and $c_{k,out}$ subject to the condition $c_{k,out} \leq \sum_{j=0}^{k-1}(c_{j,in} - c_{j,out})$.

Next we define the term *Dual-Element PIR Sensors*. In particular we emphasize the difference of one-element and two-element PIR Sensors. Note that for the remainder of this document the term *PIR Sensor* refers to dual-element PIR Sensors.

Definition 2 (PIR Sensor)**.** Passive Infrared Sensors (PIR) are commonly known as *movement detectors*. Such sensors measure the amount of infrared light radiating from objects passing in their view; a change in the measurement exceeding some defined threshold is considered a movement. *Dual-element PIR sensors* connect two pyroelectric detector elements. The sensor signal is equal to the difference of the elements' voltages. Combined with Fresnel lenses, focusing infrared light coming from different angels, such sensors allow for the extraction of directional information of moving objects [12].

Using Definition 1 and Definition 2 let us define the first problem as follows:

Problem 3 (People Counting using PIR Sensors)**.** Given a system compromised of a microcontroller connected to a dual-element PIR sensor, determine the count values $c_{k,in}$ and $c_{k,out}$ at discrete-time instances k.

2.2 Monitoring, Data Collection of Distributed People Counting Sensor Nodes

Let us define the notion of a *People Counting Sensor Node (PCSN)* and then give the problem of monitoring a distributed set of such nodes:

Definition 4 (People Counting Sensor Nodes, PCSN)**.** A device solving Problem 3. Distributed PCSNs are defined as a set of a predefined number of such devices connected to a base station by radio. Multiple base stations can be combined to form a network of distributed PCSNs.

Next we define the terms *Observed Space* and *Space Occupancy*:

Definition 5 (Observed Space, Space Occupancy)**.** A defined spatial area S_A with predefined entry and exit points j, each of which is observed by a PCSN c_j. All PCSNs observing an area S_A are connected to the same base station. The occupancy of an observed space is defined as the total number of people present in this space $occ_k = \sum_{i=0}^{k} \sum_{j=1}^{N} (c_{j,k,in} - c_{j,k,out})$ where N is the number of the PCSNs observing space S_A.

Based on Definition 4 and Definition 5 we give the following problems:

Problem 6 (Monitoring)**.** Given a set of distributed PCSNs $C = \{c_j : 1 \leq j \leq N\}$, at the base station monitor and display the count states $c_{j,k,in}$ and $c_{j,k,out}$ of each PCSN $c_j \in C$ at discrete-time instances k. k shall be chosen such that there is no interference by multiple PCSNs, in order to combine a collective state of all monitored devices.

Next we extend the concept of monitoring to a network of distributed PCSNs:

Problem 7 (Data Collection)**.** Given a network of distributed PCSNs, collect the count states of all PCSNs assigned to an observed space S_A and derive a combined occupancy state for this area.

2.3 Estimation of Space Occupancy

Definition 8 (State Estimation)**.** The process of estimating the true (internal, process) state of a system at discrete-time instances k in light of noisy measurements, where the process and measurement noises are within the bounds of some defined probability distribution function.

Deriving from Definition 5 and Definition 8 we define the following problem:

Problem 9 (Estimation of Space Occupancy)**.** Given the data collection of monitoring results of observed spaces (measurement state), for each space estimate the true occupancy (process state).

CHAPTER 3

Theoretical Background

This chapter provides a brief review of the theoretical background to the algorithms described in Chapter 4. The following sections are based on [13, 14, 15, 16] as referenced in the respective context.

3.1 Digital Filters

This section reviews the purpose and mathematical background of digital filters. The purpose of a filter is to separate certain frequencies in a signal in order to either remove undesired noise or extract information. Digital filters are relevant to the system presented in this thesis to reduce noise present in the sensor signal, and to extract relevant information from this signal.

There are several basic types of filters:

- low-pass filters remove all frequencies below a specified frequency.
- high-pass filters remove all frequencies above a specified *cut-off* frequency.
- band-pass filters remove all frequencies above and below specified lower and higher *pass* frequencies, and thus let a certain *band* of frequencies pass the filter.
- band-reject filters work the same way, but *reject* the band of frequencies within the specified lower and higher bounds.

While filters separate frequencies represented in the *frequency domain*, all filters used in this thesis operate on the signal in the *time domain*. Such filters can be implemented by two distinct methods which are described in the following sections.

3.1.1 Finite Impulse Response Filters

FIR (finite impulse response) [13] digital filters are based on mathematical convolution of the filter's kernel and the input signal, i.e.

$$y[k] = \sum_{j=0}^{M-1} h[k] \cdot [k-j] \tag{3.1.1}$$

where $x[k]$ is the sensor signal of discrete time k as converted by the ADC[1], h is the filter kernel, M is the size in samples of h, and $y[k]$ holds the convoluted sum, that is the filtered sample of discrete time k. The filter kernel defines the *impulse response* of the filter, which is its output if the input is a delta function δ. A delta function is an input signal whose samples u_i are zero except for sample u_0. Convolution of a signal and a filter kernel result in the filtered output signal y.

3.1.2 Infinite Impulse Response Filters

IIR filters [13] apply recursion rather than convolution to calculate the filter's output, defined as

$$y[i] = a_0x[i] + a_1x[i-1] + ... + a_nx[i-n] + b_1y[i-1] + ... + b_ny[i-n] \tag{3.1.2}$$

where a, b are the vectors of recursion coefficients. Using recursion coefficients a, b allow a IIR filter to "simulate" a mathematical convolution with less computational effort. IIR filters calculate the current output sample $y[i]$ by taking into account both the signal samples $x[\cdot]$ and previously calculated output samples $y[i-n]$, $1 \leq n \leq 12$, instead of only unfiltered samples as a FIR filter does. Each coefficient is said to be a *pole* of the filter, and the number of coefficients is the number of poles (or *order*) of the filter. A single-pole low-pass filter is calculated by using coefficients a, b such that $a_0 = 1-x$, $b_1 = x$ where $0 < x < 1$. According to [13], single-pole filters are analogous to a lower-pass filter implemented as a RC-network in hardware: "x is the amount of decay between adjacent samples (...), the higher the value of x, the slower the decay."

Note that the IIR filter described in Section 4.4 is a four-pole Butterworth filter whose filter coefficients are based on applying the Laplace-transform and z-transform to an impulse response [13].

3.2 Automata and Regular Languages

This section reviews the concepts of automata and regular languages. Both concepts are relevant to the people counting system and in particular in relation to a solution of Problem 3.

[1] Analog Digital Converter

3.2.1 Deterministic Finite Acceptors

A *Deterministic Finite Acceptor* (DFA) [14] is an automaton that accepts a series of discrete-time events $e_1, ..., e_n$ iff there is a transition function δ defined for the transition from the current state to the next state as a result of event e_k. Formally, the DFA accepts the input symbol y_k generated by event e_k iff δ is defined for

$$\delta(q_i, y_k) = q_{i+1}, i \geq 0 \tag{3.2.1}$$

where q_i defines the current state of the automaton. The automaton is formally specified by

$$M = (\mathbb{Q}, \Sigma, \delta, q_0, F) \tag{3.2.2}$$

where $\mathbb{Q}$ is the set of all internal states, Σ is the set of valid input symbols (generated by some event e_k), $\delta : \mathbb{Q} \times \Sigma \to \mathbb{Q}$ is the transition function, $q_0 \in \mathbb{Q}$ is the initial state, and $F \subseteq \mathbb{Q}$ is the set of final states.

Alternatively, Σ can be given as Σ^* to denote a set of strings of symbols in Σ. Then the transition function becomes $\delta^* : \mathbb{Q} \times \Sigma^* \to \mathbb{Q}$, and is said to be the *extended transition function* where the second argument is a string rather than just one symbol.

DFAs are often referred to as *Finite State Machines* (FSM), and we will use the two terms interchangeably.

3.2.2 Regular Languages and Regular Expressions

An automaton M that accepts all strings defined by symbols in Σ is said to generate the language $L(M) = \{w \in \Sigma^* : \delta^*(q_0, w) \in F)$. A language L is said to be regular iff there exists some DFA such that $L = L(M)$. A regular language is therefore an exact representation of a DFA, and the two concepts can be used interchangeably.

A regular expression r defines a language L by combining the set of symbols $w \in \Sigma$ using the following notation. If $\Sigma = \{a, b, c\}$ then let the regular expression r define $L(r)$ as follows:

r	$L(r)$	Meaning of special character
$a\|b$	$L(r) = \{a, b\}$	$\|$ denotes alternative of a or b
$a?b$	$L(r) = \{ab, b\}$	? denotes optionality of a before b
$a+b$	$L(r) = \{ab, aab, aaab, ...\}$	$+$ denotes the occurrence of a for n times, $n \geq 1$
$a*b$	$L(r) = \{b, ab, aab, aaab, ...\}$	$*$ denotes the occurrence of a for n times, $n \geq 0$
$(ab)+c$	$L(r) = \{abc, ababc, abababc, ...\}$	$()$ denotes grouping of symbols

Table 3.2: Definition of regular languages by regular expressions

Note that the special characters $|?*+()$ of this notation can be chained by all symbols defined in Σ such that a language of any length can be generated.

In conclusion, regular expressions define DFAs: a regular expression defines a regular language, and a regular language is the exact representation of a DFA. For example, let $\Sigma = \{a, b, c)$ and $r = (ab) + c*$, which generates the language $L(r) = \{abc, ababc, abababc, ..., abc, abcc, abccc, ...\}$. Then $L(r)$ is the equivalent of the DFA $M = (\{q0, q1, q2\}, \{a, b, c\}, \delta^*, q0, \{q2\})$ where δ^* is defined for

$$\begin{aligned} \delta^*(q0, ab) &= q1 \\ \delta^*(q1, ab) &= q1 \\ \delta^*(q1, c) &= q2 \\ \delta^*(q2, c) &= q2 \end{aligned}$$

3.2.3 Graphical Representation

For the graphical representation of a DFA we use vertices to represent states and edges to represent the transitions between states, as defined by the transition function. For example, let $M = (\{q0, q1, q2\}, \{a, b, c\}, \delta, q0, \{q2\})$ where δ is defined for

$$\begin{aligned} f(q0, a) &= q0 \\ f(q0, b) &= q1 \\ f(q1, c) &= q2 \end{aligned}$$

Then the following transition graph represents this DFA (Figure 3.1):

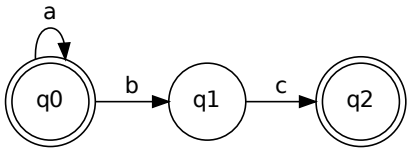


Figure 3.1: A graphical representation of a DFA as a transition graph.

3.3 Stochastic State Estimation

This section reviews the definition and mathematical concepts of discrete-time stochastic systems. In particular, we review the Kalman Filter as a means to minimize the estimated error in a prediction of the future system state. Stochastic state estimation is relevant to the people counting system in order to improve the counting accuracy of multiple sensors, as stated in Problem 9. Section 5.5 makes use of the material presented here.

3.3.1 Discrete-time Stochastic System

The people counting system presented in this thesis can be seen as a discrete-time stochastic system [15], defined in its general form as

$$x_{k+1} = f(x_k, u_k, k) + w_k \tag{3.3.1}$$

$$y_k = g(x_k, u_k, k) + v_k \tag{3.3.2}$$

where matrix x_k denotes the state of the system at discrete-time instance k, which in our case consists of the *true* "in" and "out" or "occupancy" scalar values. Matrix y_k denotes the measured state at discrete time k, consisting of the *measured* "in" and "out" counts (and possibly other information that is of relevance and can be measured). Function $f(\cdot)$ denotes the state transition function describing the process transition of state x_k to state x_{k+1}, using input (control) values u_k. Function $g(\cdot)$ relates the state x_k to the measurement y_k. w_k, v_k represent the process and measurement noises, respectively.

In words, given the linear or non-linear functions $f(\cdot)$ and $g(\cdot)$, stochastic state estimation attempts to predict the future state of a system. For the linear case the above state equations are transformed into the following equations:

$$x_{k+1} = A_k x_k + B_k u_k + w_k \tag{3.3.3}$$

$$y_k = H_k x_k + v_t \tag{3.3.4}$$

where A is the state transition matrix, B is the input transition matrix, and H is the measurement matrix which relates the current state x_k to the measurement y_k. Note that all variables in these equations are matrices.

3.3.2 The Kalman Filter

The Kalman Filter is a stochastic state estimator algorithm which is widely used in industrial applications to control stochastic systems. The algorithm implements a system of linear stochastic difference equations, which minimize the error covariance of the state estimation. The discrete Kalman Filter algorithm consists of five steps at each time instance k as defined in [16]:

1. State projection:

$$x'_k = Ax_{k-1} + Bu_k \tag{3.3.5}$$

2. Estimation error covariance projection:

$$P'_k = AP_{k-1}A^T + Q \tag{3.3.6}$$

3. Computation of the Kalman gain (gain of the filter):

$$K_k = P'_k H^T (HP'_k H^T + R)^{-1} \tag{3.3.7}$$

4. Correction of the estimate with measurement y_k :

$$x_k = x'_k + K_k(y_k - Hx_k) \tag{3.3.8}$$

5. Correction of the estimation error covariance:

$$P_k = (I - K_k H)P'_k \tag{3.3.9}$$

In the above equations $^{-1}$ denotes the inverse of a matrix, and ^{T} means the transpose of a matrix. All parameters A, B, P, H, K, I, Q and R are matrices. Note that unlike Equations 3.3.3, 3.3.4, we assume A, B, Q, R to be constant in our scenario. P denotes the current error covariance of the estimation, I is the identity matrix and K denotes the Kalman Filter gain. The Kalman Filter assumes a linear system with zero-mean ("white") Gaussian process noise covariance Q and measurement noise covariance R. Q and R are defined as follows:

$$Q = E[w_k w_k^T], R = E[v_k v_k^T] \tag{3.3.10}$$

where $E[\cdot]$ means the expected value of the process and measurement noise. The Kalman Filter requires random variables w_k, v_k to be independent of each other at each time instance k, and zero on average, i.e.

$$w_k \sim N(0, \sigma^2(w)) \tag{3.3.11}$$

$$v_k \sim N(0, \sigma^2(v)) \tag{3.3.12}$$

Remark. The Extended Kalman Filter (EKF) is the non-linear equivalent of the discrete Kalman Filter presented above. As this thesis does not make use of the EKF we refer the reader to [16].

CHAPTER 4

People Counting Algorithm for PIR Sensors

The focus of this chapter is to analyse the signals by PIR sensors, and to develop an algorithm for people counting. The chapter is organized as follows. To solve Problem 3, we start by analysing the signal patterns of PIR sensors, in particular focusing on those aspects relevant to the people counting problem. Algorithms are developed step-by-step, taking into account the insights gained in the process. All algorithms are formally specified and described in pseudo-code.

4.1 PIR Sensor Signal Basics

Typical commercially sold PIR sensors emit a binary signal, where a HIGH (binary 1) signal indicates *no movement*, and a LOW (binary 0) signal indicates the detection of *movement* by the sensor. For our application this information is of little value, as we need to detect the direction of the movement. Therefore we use PIR sensors provided by Hygrosens, which in addition to the digital signal also provide an analogue output (details see Section 8.2).

PIR sensors are built from two infrared segments, arranged along a horizontal split. The sensor's analogue signal equals the difference of the infrared radiation received on each side of the horizontal split, which allows to detect on where in the sensor's field of view a movement originated. Figure 4.1 illustrates the unfiltered analogue signal emitted from the sensor in the event of a single person passing from left to right and back again. The signal as shown was sampled at 50Hz. The pattern reflects the IR radiation, as seen by the sensor, for a single person moving from left to right (range $3000 - 4000$ms), and right to left (range $6000 - 7500$ms). The continuous signal is interpolated for readability.

As is easily visible from the figure the signal pattern has several different components:

1. Signal amplitude. The signal amplitude is a function of the distance of the sensed

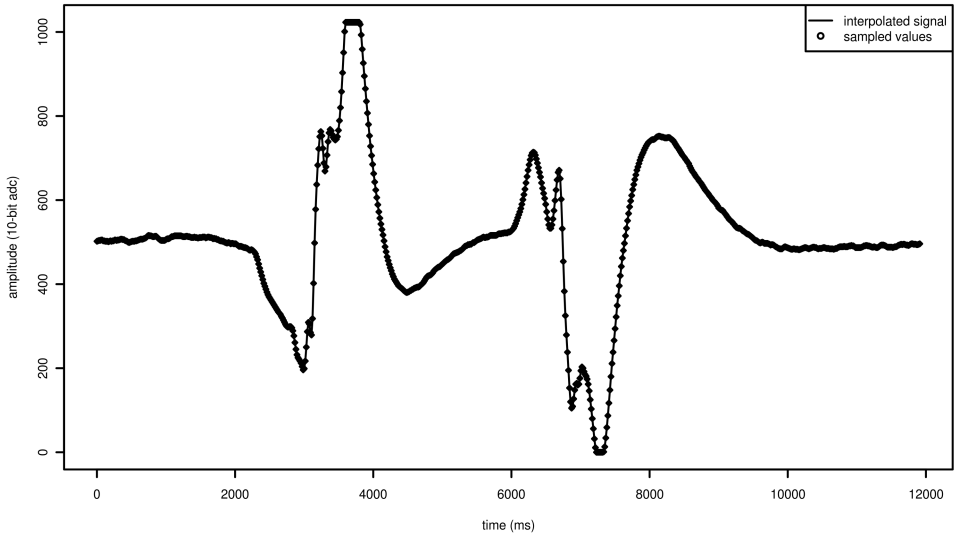


Figure 4.1: Unfiltered PIR sensor analogue output for a single person traversing from left to right and back again. The midpoint of the signal indicates no movement, whereas signal peaks above and below the midpoint refer to maximum incident IR radiation to the respective side of the PIR sensor.

object passing in front of the sensor. In general, the closer the object to the sensor, the higher the amplitude [17], although the speed of the passing object may also influence the amplitude [8].

2. Signal frequency. The signal frequency is an indication of the speed of the passing object. According to the vendor documentation, the signal frequency for our PIR sensors has a bandwidth from 0.2Hz to 10Hz.

3. The phase of the signal (as visible in the time-domain). The phase is an indication of the side of the sensor where a movement was detected.

The direction of movement can be detected by analysing the phase of the signal. We simply approximate this information by using the fact that the PIR sensors output a signal at $\approx VCC/2$ while no movement is detected, and the side S of the movement is given by

$$S = \begin{cases} left & \text{if } s \geq VCC/2 + v \\ right & \text{if } s \leq VCC/2 - v \\ silent & \text{otherwise} \end{cases} \tag{4.1.1}$$

where s indicates the signal's amplitude and v is a noise threshold above and below $\frac{VCC}{2}$ [5]. Based on our experiments a threshold of $v \sim 260$mV works best. Note that sides

right, left are relative to the mounting direction and can in practice be exchanged for each other.

The PIR sensors used in the prototype cover a detection range of $r = 12$m at a horizontal angle of $\pm 50°$. Consider Figure 4.2: assuming the typical installation scenario will use sensors ceiling- or side-mounted in entrance doorways of maximum height/width of 2.5m, we can estimate the distance in relation to the sensor's range as follows. Let the maximal distance $d_{max} = \frac{(2.5-h)}{\sin(180-90-50)}, 0.5 \leq h < 2.5 \rightarrow 0.02 \leq d_{max} < 3.2$m, where h is the person height. Then the maximum fraction r' of the sensor's range covered by our application will be $r' = \frac{d_{max}}{r} \leq \frac{1}{3}r$. Even though the sensor's amplitude is not strictly linear to the distance [18], we consider the sampled signal for any movement will be in the range $v_{adc} = 80 < \lceil (1-r') \cdot \frac{2^{adc_w}}{2} \rceil \leq \frac{2^{adc_w}}{2} = 512$, where $adc_w = 10$ is the precision of the ADC in bits and $v_{adc} = \lceil adc_w \cdot \frac{v}{VCC} \rceil$ is the signal noise converted to its digital representation. Therefore we choose to ignore the variation of the sample signal (due to person height) for the purpose of this discussion. Note we use the signal noise v as a lower bound because the sensors report the *difference* of the incident IR radiation between the two IR elements rather than an absolute value for each. Thus, in practice, the signal can be well below the theoretical minimum of $\lceil \frac{2}{3} \cdot \frac{2^{adc_w}}{2} \rceil = 342$ at $r' = \frac{1}{3}$, however, we consider all signals above the noise level as by Equation 4.1.1.

We will later show that the signal frequency is of marginal importance in our application since we are only interested in the fact that a passage has occurred, but not in how fast this happened. However, the frequency bandwidth is of relevance to the sampling frequency of the sensor as it provides a natural bound in terms of the Nyquist frequency. Hence, the sample frequency shall be at least $f_{max} \cdot 2 \geq 20$Hz where $0.2 \leq f \leq 10$ is the PIR sensor's frequency spectrum. Based on our experiments $f = 50$Hz yields a sufficient resolution and sample spacing in the time domain.

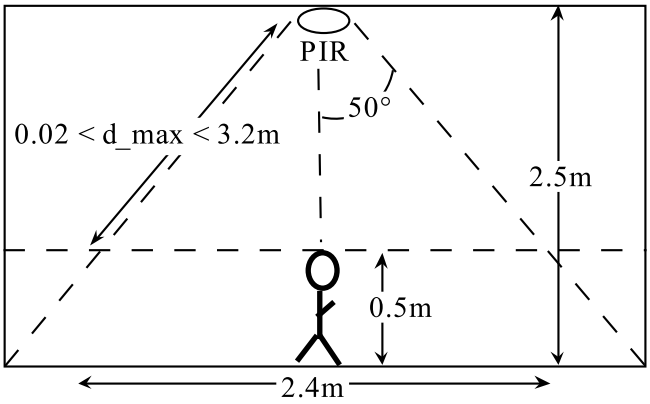


Figure 4.2: Assumed maximal distances and minimal person height. The distances allow to approximate the sensor signal strength to fall within a limited range despite people height. That is, we can ignore variation in the signal due to person height. Note that this may no longer be true if the door height exceeds the given limits.

4.2 Deriving a People Count

Consider again Figure 4.1 depicting the PIR sensor output for one person, passing once in each direction. We can see two distinct peaks at the lowest and highest points of the amplitude range. The peaks signify the moment of strongest incident IR radiation on the *left* or *right* side of the sensor's field-of-view, respectively. The passages from left to right and vice versa are clearly visible: for each passage, the signal peaks on the *from* side, next on the *to* (opposite) side. From this analysis, the people count seems to equal the number of transitions $t(from, to)$. Unfortunately, this rule is not always applicable in the case of multiple people passing in a queue and with little distance, as will be discussed next.

Consider Figure 4.3. It depicts the signal output for four people passing by the sensor from left to right. Unlike the single-person example, no specific left/right passages are visible, except for the first. Further, and unlike the single-person passage, the signal does not provide clues to separate one person from the other. Clearly, counting the transitions as in the first example will produce false positives.

Looking further, the signal contains several *high frequency* ranges. The signal frequency corresponds to the angular speed of each person passing by. Since we are interested in the number of people we might simply count the number of distinct high-frequency areas. However, this turns out to be impractical as we can see in Figure 4.4: applying a high-pass filter correctly yields the high-frequency areas, but these areas are too close to each other to count the number of passers-by. Therefore we conclude that isolating the high-frequency areas of a sampled signal is not practical to solve our problem.

Let us consider again Figure 4.3, but this time focus on *low-frequency peaks*. Four peaks are clearly visible in the amplitude range of $0 \leq s \leq 200$ and two peaks in the range $900 \leq s \leq 1023$: each of these peaks is a result of one person passing the sensor at a distance close enough to yield an amplitude above the *silent* threshold level. The four peaks exactly reflect the number of people *arriving* at the passage's opposite side. Recall that samples are spaced $\frac{1}{50}\text{s} = 20\text{ms}$ apart, and consider the distance between these peaks: each person takes between 0.5s and 1s to pass. This is reasonable considering the sensor's range we are interested in, the installation height, a walking speed of $\sim$ 5km/h, and an average person height of 1.7m.

In conclusion, to count the number of people passing in a queue at short distance, the sensor signal must be interpreted from the *first* detected passage, coming from a silent state. The first change in amplitude indicates the *from* direction, and the immediate subsequent amplitude peak on the opposite side indicates the *to* direction. Any subsequent *low-frequency* peak indicates one more person passing in line. The counting sequence completes with a *silent* phase.

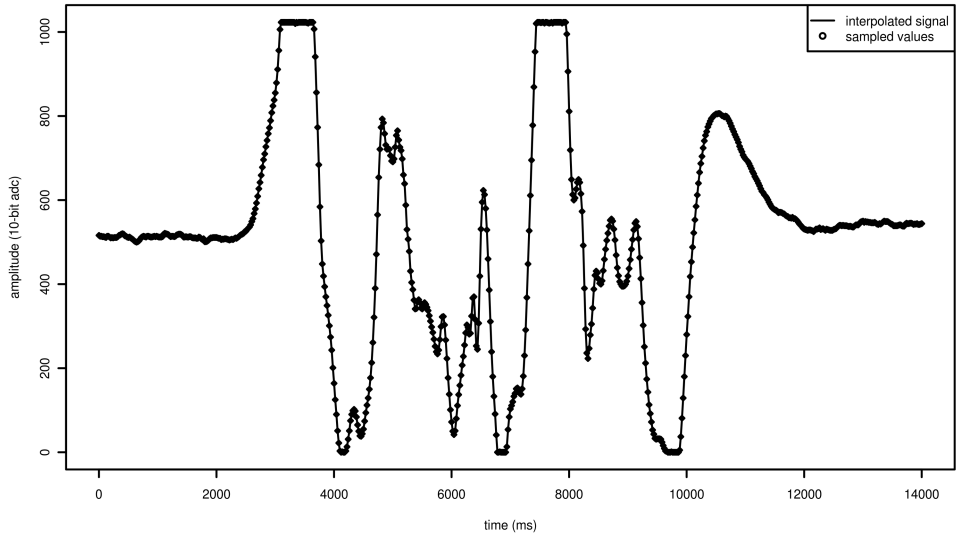


Figure 4.3: Signal pattern for four people in a row traversing from right to left.

4.3 Basic Algorithm

As shown in the last section the raw sensor signal allows for a visual analysis by counting the number of low-frequency peaks on one side of the sensor, after detecting the first passage. The COUNT algorithm (Algorithm 1) applies this insight. Assume the signal has been analysed for low-frequency peaks outside the silent amplitude range. The algorithm works in two stages:

1. Identify a first passage *left/right* or *right/left*. This becomes the *direction of movement.*
2. Count the number of transitions $c = (direction)$ by counting the peaks on the *to* side of the sensor.

While this basic algorithm works correctly for the previously identified patterns of PIR signals, it is not sufficient for practical applications due to several reasons:

1. Distortions and noise in the signal are not accounted for and will lead to false peak identification and thus invalid count values. An example is given in Figure 4.5, where a single person has passed the sensor yet yielding two peaks on the *to* side of the amplitude range. Such a pattern may be caused by a relatively slow moving person, where the sensor indicates IR radiation at different angles. This kind of distortion can be eliminated by filtering the signal to include only the relatively low-frequency components indicating amplitude shifts.

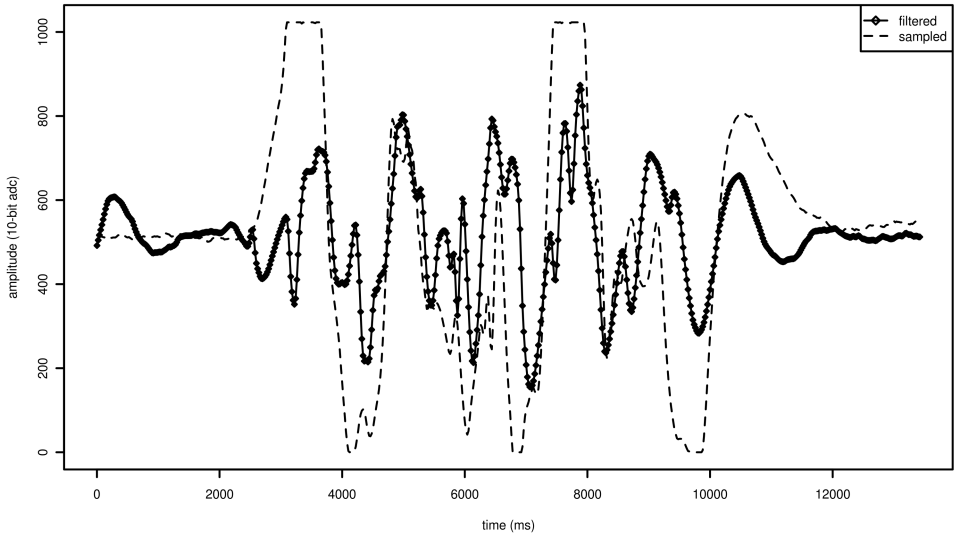


Figure 4.4: Signal pattern for four people in a row traversing from right to left, with high-pass filter applied (filtered signal adjusted for visual overlay).

2. The first transition from an identified *from* peak to an adjoint *to* peak must happen within a limited amount of time, that is below time distance $t \leq td_{max}$. If too much time passes between these two peaks the algorithm should ignore the previous peak and assume it has started from a silent position. Applying this rule simplifies the handling of fast changes in the direction of movement, such as when a group of people passes by the sensor in one direction first, followed by another group that walks in the opposite direction.

3. Multiple peaks in short distance such as in Figure 4.5 may cause false counts. To limit the number of counts within a sequence, a minimal time distance $t \geq td_{min}$ between peaks must be observed.

4. To distinguish people standing/waiting in the sensor's field of view, vs. someone actually passing it, an absolute distance of peaks in terms of the signal's amplitude $|s(t) - s(t-1)| \geq a_{min}$ should be observed (Figure 4.6).

The following sections discuss improvements to the COUNT algorithm to alleviate these problems. In particular, Section 4.4 introduces signal filtering to reduce distortions and noise; Section 4.5 provides a peak detection algorithm; Section 4.6 describes the implementation of the algorithm as a finite state machine, and Section 4.7 extends this idea by using regular expressions to enable run-time adjustments of the state machine.

Algorithm 1 Basic COUNT algorithm

```
1  function COUNT(s):
2    // initialize
3    c(LEFT):=0
4    c(RIGHT):=0
5    from:=NONE
6
7    // analyse signal
8    for t := 1 to length(s)
9      if s(t) is "silent" then
10       direction:=SILENT
11
12     // state 1: identify first peak
13     if direction==SILENT and s(t) peaks LEFT and s(t+1) peaks RIGHT
14       direction:=LEFT
15     else
16      if direction==SILENT and s(t) peaks RIGHT and s(t+1) peaks LEFT
17        direction:=RIGHT
18     else
19       direction:=SILENT
20
21     // Stage 2: direction indicated, count the number of people
22     if direction is LEFT|RIGHT and s(t) peaks LEFT|RIGHT
23       c(direction) += number of peaks on "to" side
24   end
25 return c
```

4.4 Signal Filter

The signal filter's purpose is to smooth the raw sensor signal such that the COUNT algorithm is able to identify amplitude peaks. As discussed in the previous sections the raw signal contains both noise from the sensor's electrical circuit, and high-frequency components which originate in the speed of movement of a person or object passing. A FIR 50-point low-pass (moving-average) filter to remove both noise and high-frequency components yields signal u as shown in Figure 4.7. Empirical testing indicated the ideal cut-off frequency of the filter at $\frac{1}{35} \sim 0.7$Hz to exclude all high-frequency components, including signal noise. The FILTER algorithm implements this filter of either FIR or IIR type. Using the filtered signal the COUNT algorithm is able to identify *from* and *to* sides of the signal at any given sample.

4.5 Peak Detection

Peak detection according to [19] is the process of identifying a peak in a given signal $u = \{u_0, ..., u_t\}$ where each sample u_t is considered to satisfy a peak condition. For any

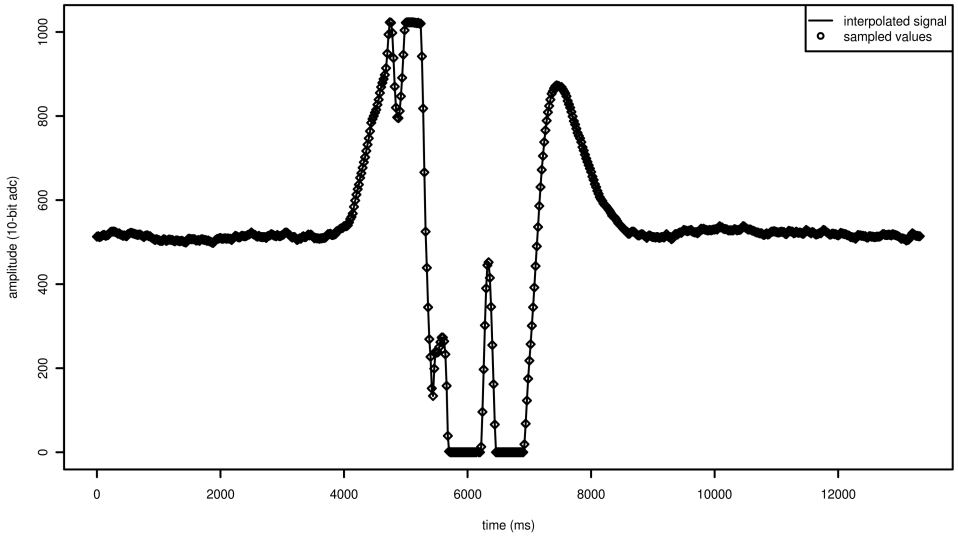


Figure 4.5: PIR signal with multiple peaks for a single person traversing in one direction. The pattern yields two peaks, and thus causes the basic COUNT algorithm (Algorithm 1) to count two people. This can be alleviated by applying a filter to the sensor signal, such that there is only one peak.

given sample u_t we derive the standard deviation

$$\sigma_t = \sqrt{\frac{1}{2sw} \sum_{i=t-sw}^{t+sw} (u'_i - m)} \tag{4.5.1}$$

of all samples $u' = \{u_s : t - sw < s < t + sw\}$, where sw is the number of samples before and after u_t (i.e. window width), m is the mean of all samples in u', that is

$$m = \frac{1}{2sw+1} \sum_{i=1}^{2sw+1} u'_i \tag{4.5.2}$$

A peak is identified for all samples u_t whose variance $v_t = |u_t - m| \geq \sigma$. Statistically, the idea is to identify all samples which exceed the threshold of one standard deviation of their $\pm sw$ neighbours. All peaks within the signal are identified accordingly as $P = \{p_t : u_t \in u, v_t \geq \sigma\}$. The PEAK algorithm (Algorithm 2) is implemented accordingly. Note that

$$p[t] = \begin{cases} u[t] & \text{if peak identified} \\ \perp & \text{otherwise} \end{cases}, \perp \text{denotes no peak.} \tag{4.5.3}$$

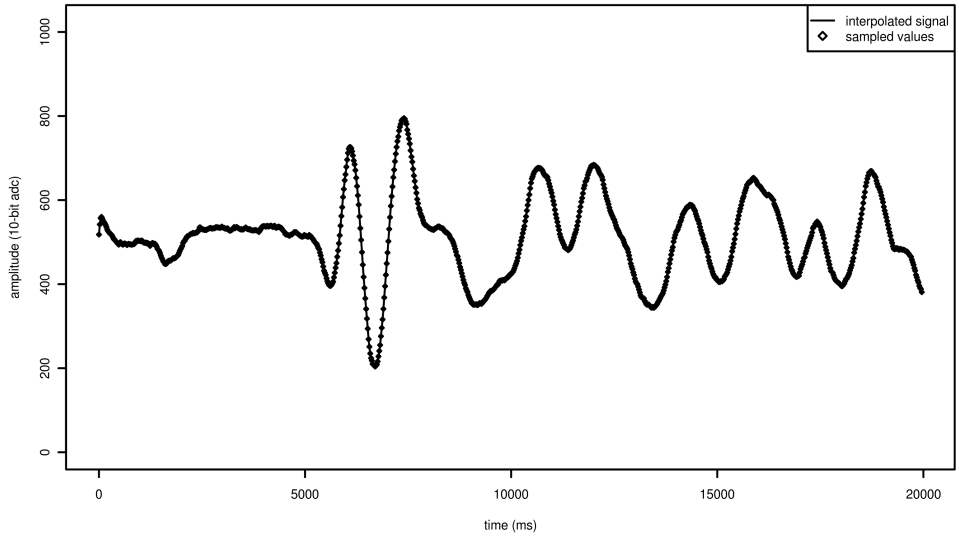


Figure 4.6: PIR signal for people waiting in front of sensor (no passage). The signal peaks are relatively close and decreasing in absolute distance as compared to an actual passage. This insight can be applied to reduce the number of false counts.

4.6 Counting Algorithm as a DFA

The counting algorithm can be considered a deterministic finite acceptor (DFA)

$$M = (\{SILENT, FROMLEFT, FROMRIGHT, CR, CL\}, \{M, L, R, I\}, \delta, SILENT, \{SILENT\}) \quad (4.6.1)$$

where δ is defined for

$$\begin{aligned} \delta(SILENT, L) &= FROMLEFT \\ \delta(SILENT, R) &= FROMRIGHT \\ \delta(FROMLEFT, R) &= CR \\ \delta(CR, L) &= FROMLEFT \\ \delta(CR, R) &= CR \\ \delta(CR, I) &= SILENT \end{aligned} \quad (4.6.2)$$

(continued)

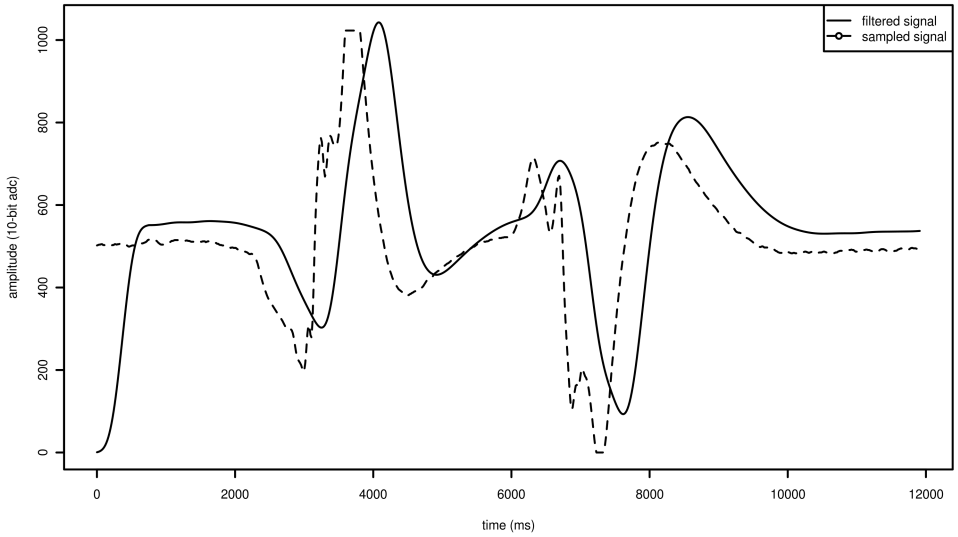


Figure 4.7: Filtered signal using a FIR 50-point moving average filter. Note both the input signal and the filter output have been interpolated to form continuous lines for visual emphasis of the differences.

$$\begin{aligned}
\delta(FROMRIGHT, L) &= CL \\
\delta(CL, R) &= FROMRIGHT \\
\delta(CL, L) &= CL \\
\delta(CL, I) &= SILENT \\
\delta(FROMLEFT, I) &= SILENT \\
\delta(FROMRIGHT, I) &= SILENT
\end{aligned}$$

Using the output of the PEAK algorithm (Algorithm 2), COUNT assigns each peak $p_t \in P$ a character $c_t \in \Sigma$ according to the rules of Equation 4.1.1, where t is the discrete time. The symbol I ("*ignore*") is used to denote a peak p_t which has too large a time distance to the previous peak, that is for its peak time pt_i it holds that $pt_i - pt_{i-1} > td_{max}$. Two associated counters c_{right}, c_{left} keep track of the number of transitions to states *CR* and *CL*, respectively.

In words, *SILENT* is the initial state and a state transition to *SILENT* occurs upon processing a sample whose time distance to the previous peak is larger than the threshold, td_{max}. A state transition to *FROMLEFT* occurs upon detecting a peak on the left of the signal amplitude range, and a state transition to *CR* occurs subsequently as many times as required, until a transition to *SILENT* is raised. The same applies respectively to the states *FROMRIGHT* and *CL*.

Algorithm 2 PEAK algorithm to detect peaks in the signal

```
1  function PEAK(u, sw):
2    Initialize P={}
3    for u[t] in u begin
4      u' := {u[t-sw],...,u[t], u[t+sw]}
5      sd := STDDEV(u')
6      m  := MEAN(u')
7      if |u[t]-m| > sd then
8        insert u[t] into P
9    end
10 return P
```

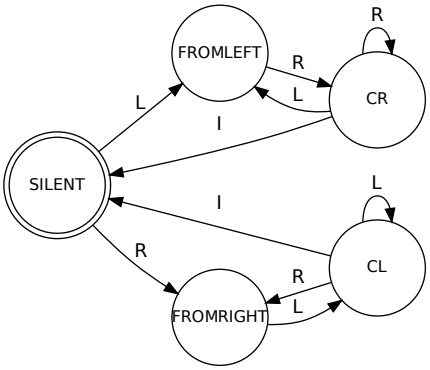


Figure 4.8: The DFA to implement the COUNT algorithm.

Implementation Concerns

Our first implementation of the COUNT algorithm used this DFA as a hard-coded construct. This soon proved to bring with it several disadvantages, in particular:

- Experiments indicated that some installation scenarios might require alternative state transitions, e.g. due to different spatial characteristics of the environment where the sensor node is installed. For example, if the sensor node is installed such that one side of it is exposed to direct sunlight, whereas the other side is in the shadow. As a result, the signal amplitude for peaks changes. In this case, the sensor may no longer be able to accurately detect a transition from *SILENT* to *FROMLEFT*, as it would rather indicate a transition from *SILENT* to a peak around the midpoint of the sensor's signal range. Such a scenario would require the change of the state machine's implementation.

- Any changes to the state machine are cumbersome and error-prone to implement, because the code has to be rewritten to match the new scenario, and each state transition has to be coded explicitly.

As an alternative the next section considers an implementation where the FSM is implemented by means of a regular expression.

4.7 Counting Algorithm as a Regular Expression

The DFA introduced in Section 4.6 can be described as a regular language. Let

$$\Sigma = \{\varepsilon, R, L, I\}. \tag{4.7.1}$$

define a set of symbols, where ε is the empty input. As in the case of the DFA, algorithms PEAK, COUNT are used to assign each peak $p_t \in P$ a symbol $c_t \in \Sigma$ according to the rules given by Equation 4.1.1. Again, counters c_{right} and c_{left} keep track of the respective number of transitions. A state string $s = \bigcup c_t$ is built and evaluated against the following regular expressions r_{from}:

- Transitions are counted in c_{left} given transitions $SILENT \rightarrow FROMLEFT \rightarrow CR$ and any subsequent state *CR*, which is the equivalent to the regular expression $r_{left} = I + LR(L?R)*$
- Transitions c_{right} are counted given transitions $SILENT \rightarrow FROMRIGHT \rightarrow CL$ and any subsequent state *CL*, which is the equivalent to the regular expression $r_{right} = I + RL(R?L)*$
- State *SILENT* is equivalent to the regular expression $R_{silent} = I+$

Implementation Concerns

Note the split of the DFA into three partial regular expressions for each state *SILENT, FROMLEFT, FROMRIGHT*. This is a result of the requirement to increase the counters on particular state transitions: using different regular expressions simplifies the test for each transition to the CR and CL states, respectively[1].

Implementing the COUNT algorithm based on regular expressions instead of a hard-coded automaton has the advantage that modifications become a matter of introducing respective regular expressions. For new states the alphabet can be easily extended and aligned with new rules to assign symbols to observed signal patterns. The algorithm as such is left unchanged.

In the context of our scenario this introduces the possibility to implement a dynamic configuration mode, whereby a sensor node can be configured at run-time to match a particular installation's needs. Thus it is possible to use the same node for multiple scenarios without the need for reprogramming. Another advantage is that regular expressions can be implemented efficiently in hardware, hence provide for a cost-optimal and high-throughput implementation on an embedded system.

[1] Note some implementations of regular expression matching require to specify start and end-of-string matchers, that is r_{left} and r_{right} start with a caret symbol (^) and r_{silent} ends with a $ special character.

4.8 Finalized COUNT Algorithm

Taking into account the previous sections, we can now specify the finalized COUNT algorithm (Algorithm 3) as follows. The algorithm uses four stages:

1. Apply a low-pass filter to the input signal to reduce signal noise and extract the information indicating an object passing by the sensor.

2. Identify the signal peaks p_t as those samples that identify either the beginning of a traversal, or a subsequent peak on the opposite side.

3. Assign each peak a character of the regular language alphabet Σ of the counting state machine, and build a state string $s = \bigcup c_t$.

4. Use the regular expressions $R_{right}, R_{left}, R_{silent}$ to identify patterns in the state string. On a match, count the respective traversal (or ignore the peak in the case of a *SILENT* state).

Algorithm 3 The finalized COUNT algorithm (part 1)

```
1  FUNCTION COUNT(s):
2    # initialize
3    c(left, right) = 0
4    c(right, left) = 0
5    pt_max    = 100          # max# samples between two peaks
6    s  = ""                  # state string
7    si = 0                   # state string index
8    MIDPOINT = ADC(VCC/2)    # define the ADC midpoint
9    MIDTHRES = 80            # +/- 80 (2x80 <= 169)
10   ABSTHRES = 150           # minimal amplitude distance
11   MAXDIST  = 100           # maximal time distance
12   MATCH_LEFT    = /I+LR(L?R)*/
13   MATCH_RIGHT   = /I+RL(R?L)*/
14   MATCH_SILENT = /I+$/     # only match at end of string
15   # Stages 1,2
16   y = FILTER(s)
17   p = PEAK(y)
18   # Stage 3,4 analyse signal
19   for(each peak t in p)
20     if pt[t] - pt[t-1] <= MAXDIST and (pt-pt[t-1] > ABSTHRES) then
21       # Stage 3
22       si = si + 1
23       if p[t] - MIDPOINT > 0 then s[si] = "L"
24       if p[t] - MIDPOINT < 0 then s[si] = "R"
25       if abs(p[t] - MIDPOINT) <= MIDTHRES then s[si] = "M"
26 (continued)
```

Algorithm 4 The finalized COUNT algorithm (part 2)

```
27 (continued from part 1)
28       # Stage 4 a) left/right states
29       if matches(/MATCH_LEFT/, s) then
30         c(left, right) += 1
31       else
32        if matches(/MATCH_RIGHT/, s) then
33         c(left, right) += 1
34     else
35       # ignore the sample
36       if pt[t] - pt[t-1] > pt_max then
37         s[si] = "I"
38       # Stage 4 b) silent state
39       if matches(/MATCH_SILENT/, s) then
40         #reset the state string (but start in state I)
41         si = 1
42  end for
43 return c
```

4.9 Summary of Parameters

Several parameters were introduced for algorithms FILTER, PEAK, COUNT. Table 4.2 provides an overview of all parameters.

Parameter	Mnemonic	Description
$\frac{2^{ADC_w}}{2}$	MIDPOINT	Midpoint value of ADC range, sensor silent state
v	MIDTHRES	Noise threshold in sensor silent state (in ADC range)
a_{min}	ABSTHRES	Minimal absolute distance between peaks (in ADC range)
td_{max}	MAXDIST	Maximal time distance between peaks (number of samples)
td_{min}	MINDIST	Minimal time distance between peaks (number of samples)
sw	PEAKWDTH	$\frac{1}{2}$ width of window considered for peak detection (samples)
r_{left}	MATCH_LEFT	Regular expression for states $FROMLEFT, CL$
r_{right}	MATCH_RIGHT	Regular expression for states $FROMRIGHT, CR$
r_{silent}	MATCH_SILENT	Regular expression for state $SILENT$

Table 4.2: Summary of parameters used by algorithms FILTER, PEAK, COUNT.

CHAPTER 5

Implementation

In this chapter we discuss the implementation of the algorithms in the C-language in order to program a sensor node, an embedded system with limited computing and memory resources. An overview of the validation framework is presented, which enables the comparison and validation of the performance of algorithms as implemented in the C and R programming languages.

In solving Problem 6 and Problem 7 the technical aspects of a monitoring application and a data collection server are summarised. To conclude, we present a solution sketch to the space occupancy estimation Problem 9. The Appendix lists reference information and technical details.

5.1 System Overview

Figure 5.1 depicts the architecture of the system. The system consists of the following elements.

People Counting Sensor Nodes A People Counting Sensor Node (PCSN) uses a double-element passive infrared sensor (PIR) connected to a microcontroller. The microcontroller continuously analyses the PIR's output signal to detect and count the number of people passing it. A successful pass is a traversal from one side of the PIR's field of view (FOV) to the other. Each pass increases the count in either the *left* or the *right* direction as seen from the PCSN's installation point. The PCSN periodically transmits the determined number of passes to a base station in the Sensor Network.

Sensor Network The Wireless Sensor Network (WSN) is a hierarchical or mesh-style topology of sensor nodes, in particular PCSNs. Each PCSN is connected to a base station via the ZigBee network protocol[1]. The protocol provides the infrastructure for node connection, routing and data collection. The WSN's base station has a

[1] Specified by the ZigBee Alliance (http://www.zigbee.org) and based on the IEEE 802.15.4 standard for low-rate wireless personal area networks.

double role: First, it acts as the sink node in the sensor network. Second, it is the gateway to the web-based space observer. It collects the count data from all PCSNs and forwards these to the web-based Space Observer.

Space Model and Monitor The Space Model defines the physical distribution of PCSNs to cover a specific area for which to count people and derive the occupancy level. The Monitor is an application installed on the WSN's base station to collect all count data of associated PCSNs, and to forward these observations to the web-based space observer.

Web-based Space Observer The Space Observer allows for the definition of multiple (spatial) spaces. For each space the server records the "in" and "out" counts and calculates the current occupancy of each defined space by use of the Process Estimator. The occupancy is measured as an absolute number and related as a percentage to the known capacity of an area.

Occupancy Estimator The space model serves as the basis for the Occupancy Estimator to estimate the occupancy of each space. Because the PCSNs' count data are diluted by measurement noise (false counts), the occupancy of a space cannot simply be calculated. Instead, a stochastic process is assumed, and the occupation is estimated by the use of a stochastic process estimator, such as the Kalman Filter (Section 5.5). The space model defines a space as some spatial area that is observed by sensors for the number of people entering ("in") and leaving ("out") the area. The stochastic process assumes noisy measurements of in and out counts (the process measurement), and derives an estimation of the actual occupation (process state) for each space. This data can then be used to display an occupation of spaces to the user e.g. via smartphone user interface.

Smartphone User Interface The Smartphone User Interface is supposed to signify some end-user device capable of querying the web-based space observer. It is listed here for completeness but is not realized as part of this thesis.

Validation Framework The validation framework provides the means to analyse the PCSN's performance off-line. That is, the validation framework provides algorithms implemented in the R-language and allows to analyse their respective C-language implementation output based on previously recorded data.

5.2 People Counting Sensor Node

For programming the prototype sensor nodes, the algorithms were transformed from their *R* language implementation into the *C* language. This chapter first introduces the hardware setup, and next outlines the changes applied due to the specific constraints of the sensor node as an embedded system.

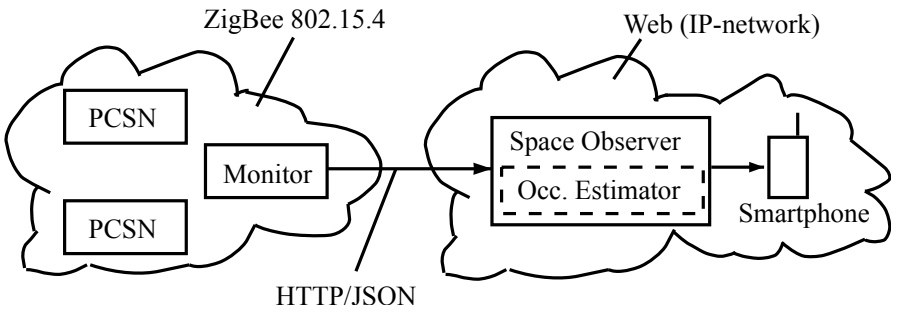


Figure 5.1: The architecture of the distributed people counting system. The People Counting Sensor Nodes (PCSNs) are positioned at entry/exit points of an area to be observed to count the number of people entering and leaving. The Monitor acts as the network's base station, and transmits the count values to the web-based Space Observer. The Occupancy Estimator determines the true occupancy of the system to minimize the error in the final result. The result is made available to client applications and devices, e.g. running on smartphones.

5.2.1 Hardware Configuration

For custom-development of the sensor nodes, the Arduino[2] platform was used as it provides a configurable and programmable processor board, which can be easily combined with commercially available production-quality sensors and other electronic components. The sensor nodes evaluated for the PCSN carry 1-2KB of main memory and a 16MHz-operated ATMEGA168 or ATMEGA328 processor. Each sensor node consists of the processor board, a PIR sensor and a radio module for communication with the base station. The CPUs operate on a 5V basis, while the radio module operates on 3.3V. Built-in voltage regulators ensure a stable power source, provided by a 9V PP3 battery. Both the processor board and the radio module support low-power modes, which we make no use of in this prototype scenario. Details of all components and the circuit diagram are shown in Section 8.2 and Section 8.3.

5.2.2 Limiting the Sensor's Field of View

The PIR sensor uses Fresnel lenses to focus the incident IR radiation, such that the sensor is enabled to cover a wider area and to detect movement in both the horizontal and vertical axes. Since we are only interested in signals within a specific horizontal range, let us restrict the sensor's coverage by applying a blocking material to the respective areas of the Fresnel lenses. This effectively limits the covered area by the sensor and thus reduces the risk of false counts. [12] discusses technical alternatives to blocking

[2] http://arduino.cc/

IR radiation such as constructing a blind cover. For the purpose of our experiments we have found white adhesive office-supply gum to be sufficient.

5.2.3 Need for Iterative Algorithms

The algorithms considered so far operate on the full length of a given signal. On the sensor node this is unrealistic since the available memory and CPU processing capacity are limited. It is essential that the algorithms FILTER, COUNT, PEAK are modified, such that they process one sample at a time and only use a small amount of memory in doing so.

Processing Scheme The algorithms for the sensor node follow an iterative processing scheme, such that the signal $u[k]$ is transformed into output $c[k]$ as

$$c[k] = COUNT(PEAK(FILTER(u[k]))) \tag{5.2.1}$$

where $u[k]$ denotes the sample at time k as given by the ATMEGA328's internal analogue-digital converter (ADC), and FILTER, PEAK, COUNT are the modified low-pass filter, peak detection and counter algorithms, respectively. $c[k]$ is a vector to contain the transition counts, $c[k] = [c_{left}, c_{right}]$. The ADC converts the analogue input signal provided by the sensor at a rate of 50Hz using a timer interrupt. Upon completion of the conversion, the sampled signal is transferred and processed as defined above.

Ring Buffer for Iterative Algorithms To process the input signal iteratively, FILTER, PEAK, COUNT need to access both the current input sample $u[k]$ and a certain number of previously calculated (filtered) output samples $y[k]$. Due to the memory constraints on the sensor node, it is infeasible to hold all calculated output samples in memory. Ring buffers are implemented for the input u, filter output y and peak p vectors. Each ring buffer is a vector of dimension b which denotes the buffer size as the maximum number of items stored in the buffer. The ring buffer operates such that the accessed sample index $i = k$ is calculated as the ring buffer's vector index i' given by

$$i' = \begin{cases} i \bmod b & \text{if } i \geq 0 \\ b + i & \text{otherwise} \end{cases}, |i| \leq b \tag{5.2.2}$$

5.2.4 Iterative FILTER algorithm

A first C-language implementation of the FILTER algorithm using the FIR kernel demonstrated that this implementation is infeasible given the limits of the sensor nodes. The algorithm in principle would work sufficiently fast, and does not as such exhaust the memory. For the filter to work, the data structure requires an array of $50 + 100 - 1 = 149$ entries in order to store both, the samples and the filter kernel. Each sample takes 4 bytes

of memory (of type *float*). However, the sensor node also stores the count vector, debug strings and requires other temporary storage. Thus, the total heap memory requirement in excess of 1KB memory would break the limits of the ATMEGA168-based sensor node, which served as a test board.

Based on experiments using the R Signal[3] package [20], a recursive implementation of an infinite impulse response (IIR) was chosen. Several different filters were considered and compared; Figure 5.2 depicts a comparison of several filters applied to the same input signal.

The figure depicts in (a) the FIR low-pass filter described in Section 4.4 and the resulting output signal. (b) shows the result of a four-pole Butterworth low-pass filter, (c) represents the result of a Chebychev four-pole filter. The peaks shown are those found by the PEAK algorithm. All filters use the same cut-off frequency ($\frac{1}{35}$ of the Nyquist frequency $f/2$, equals 0.71Hz at sampling frequency $f = 50$Hz). The Chebychev filter has a band-pass ripple of 1%, which is clearly visible in (c) around samples 0-500, 600-650, and 1250-1500. This ripple is a potential source for the false detection of peaks in the filtered signal. The Butterworth filter performs best in this respect among all filters (0% ripple), and also matches the result of FIR filter (a) closely.

5.2.4.1 Iterative PEAK and COUNT algorithms

The PEAK and COUNT algorithms are modified such that they accept a single input sample $u[k]$, process and store the result $y[k]$ into the respective ring buffer. PEAK accesses FILTER's output from ring buffer y, and then stores the identified peaks $p[k]$ in ring buffer p. COUNT accesses ring buffer p and updates the count values in vector c. Note that vector c is not a ring-buffer as it holds the scalar sum of the transition counts.

5.2.4.2 Iterative Regular Expression

Building the state string s on the PCSN may lead to buffer overflow or overwriting of previous states in the ring buffer. The following approach solves this problem by using a simplified regular expression evaluator, REGEXP: it accepts single characters iteratively. Thus, there is no need to buffer s. REGEXP is implemented as a C++ class and works as follows[4]:

- REGEXP:: *make*() accepts a regular expression r as an input string. It parses the regular expression and builds an internal representation of the DFA. The internal representation equals the language $L(r)$, i.e. the set $S = \{s_0, ..., s_n\}$ of all strings

[3] A MATLAB-equivalent implementation of digital signal processing functions

[4] REGEXP was implemented as a proof of concept and is not optimized for space nor time complexity. For example, *matchc*() has a worst-case time complexity $O(n^2)$ where n is the number of states in the DFA. An improved version using a more efficient internal encoding could perform significantly better, e.g. using a binary tree to store and access states, reducing *matchc*()'s worst-time complexity to $O(n \log n)$.

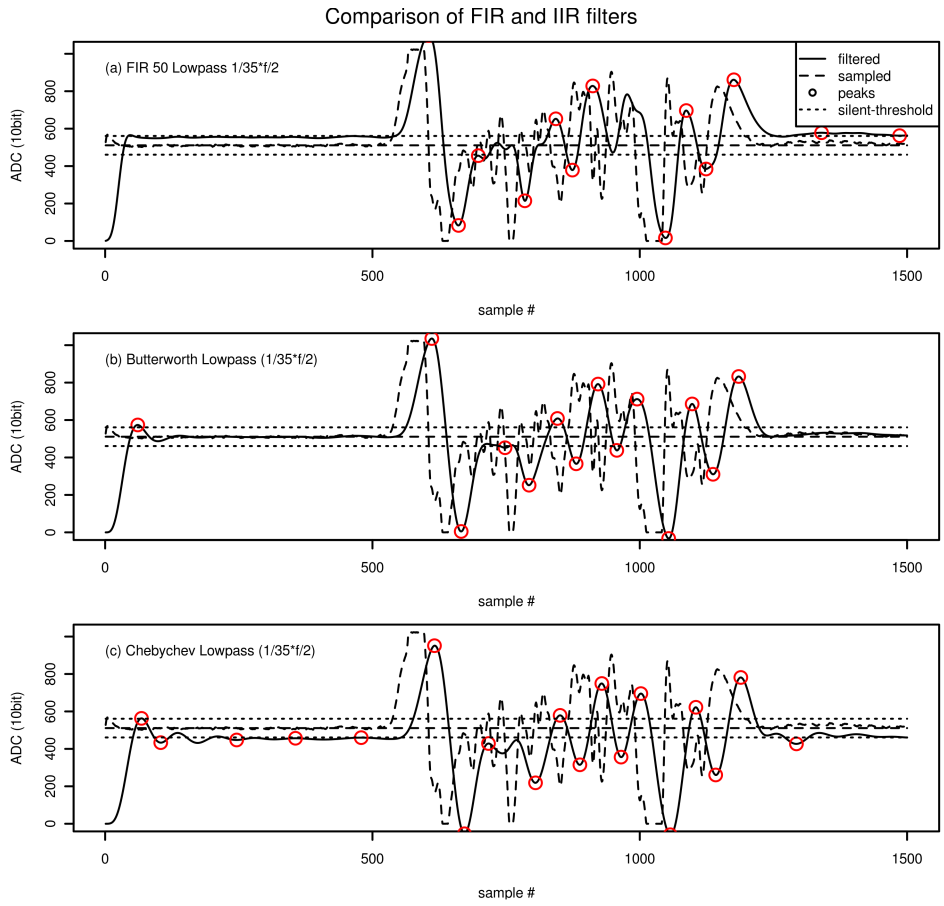


Figure 5.2: comparison of FIR and IIR filters. Different filter kernels result in more or less number of peaks detected. In particular, the (b) Butterworth filter performs best in terms of detecting relevant peaks for identification of traversals.

generated by the regular expression. In the general case this is the same as the input string, i.e. $s_0 = r$. Special characters $|?$ generate new sub-states thus result in duplication of all previous strings. The special characters for repetition and grouping $+*()$ are not supported.

- REGEXP:: $matchc()$ accepts a single character c as input. Each string s_i is checked for equality to c at position v_i. If there is a match, v_i is increased. If there is no match, it is set to the missing value $\perp$. If v_i reaches the end of string s_i the full string was matched and the function returns true, else false.

- REGEXP:: $reset()$ resets all $v_i = 0$.

Using this class, the PCSN is configurable to new installation scenarios by defining regular expressions r_{left} and r_{right}. Because the REGEXP does not currently support

repetition or grouping, several changes to the C-language implementation of the COUNT algorithms were necessary:

- The algorithm tracks the states $FROMLEFT$ and $FROMRIGHT$ explicitly.
- The regular expressions are redefined to $r_{left} = LRL?$, $r_{right} = RLR?$, a match is only attempted if the time distance $pt_i - pt_{i-1} \leq td_{max}$ (to compensate for the missing character *I* in the regular expressions).
- The regular expression R_{silent} is replaced by a direct test for the maximal time distance td_{max}.
- Each time a match occurs on either r_{left} or r_{right} the respective instance of REGEXP is reset. This is the equivalent of matching the start of the string the next time around.

5.2.5 Operation Modes of the PCSN

The PCSN implements two modes of operation: a *LOGGER* mode, which is used to sample unfiltered values, and a $COUNTER$ mode which is used to operate the sensor node as an independent people counter. The *LOGGER* mode is designed to be used for experimental data collection: all sensor values are transmitted to and stored by a PC for subsequent analysis. In this mode, the PCSN does not filter or process the data in any way. In $COUNTER$ mode the PCSN only transmits (a) the scalar count values c_{left}, c_{right}, and (b) string s since the last peak was processed. Note that for debugging purpose, s is amended by $+$ and $-$ signs to mark transitions to *CL* and *CR* states, respectively.

The PCSN accepts a configuration value *SERIAL* and *RF*. The *SERIAL* configuration implies use of the sensor node's USB port to transmit the data. The *RF* configuration implies use of the *ZigBee* radio transmission module.

5.3 Space Model, Monitor and Observer

5.3.1 Monitoring Model

The PCSN is designed to be installed such that the PIR sensor's *left/right* sides correspond to an observed space's *entry/exit* directions, respectively. If multiple PCSNs are combined, a larger spatial area can be observed, and the occupancy of the area can be derived from the total of the count.

Thus, the monitoring model is defined as follows:

Definition 10 (Monitoring Model)**.** A spatial area S of size S_A subdivided by N sub-areas $S_i, 0 < i \leq N$ of size $S_{A(i)}$ such that $S_A = \sum S_{A(i)}$. Each sub-area S_i has a predetermined maximal occupancy (number of passengers), and is monitored by a set of

PCSNs $C_i : \{c_0, ..., c_k\}$. Each node is positioned to monitor one entry/exit point of the respective sub-area.

By Definition 10, all nodes in a set N_i form a segment of the sensor network with a local root node acting as the base station for this segment. A segment is defined as follows:

Definition 11 (Monitoring Segment)**.** A set of PCSNs interconnected as a spanning tree, where the root node is said to be the base station. At an appropriate frequency, nodes send their acquired data to the root node. In general, a recursive hierarchy of nodes may be used to represent segments of segments.

5.3.2 Sensor Monitor (Base Station)

Each base station implements a Sensor Monitor which connects to one or multiple PCSNs operated in *COUNTER* mode. The Sensor Monitor receives count data via a radio link. A prototype implementation of two PCSNs connected to a base station has been implemented as a Java application (SENSMON). For this purpose, both the base station and the PCSNs are extended by RF modules using the ZigBee 802.15.4 networking protocol.

For each PCSN connected, SENSMON displays a visual representation of the count values, symbolising *in*/*out* counts for an observed area, respectively. Upon reception of a new count value, SENSMON updates the display accordingly (see screenshot in Subsection 8.5.1).

An extension to SENSMON was added to enable the interfacing with the Data Collection Server described in Subsection 5.3.4.

5.3.3 Sensor Network Data Transmission

The ZigBee 802.15.4 protocol supports star, meshed and hierarchical network topologies. Note that the RF modules used for the prototype implementation only support the star topology, which is sufficient for the purpose of the prototype. Each RF module is assigned a pre-defined node ID. A hardware configuration setting defines the role of each RF module in the network (client or controller).

A simple application protocol layer was developed to enable the binary transmission of arbitrary data structures. It is based on the open-source library *xbee-arduino*[5], which implements an application programming interface (API) to the RF modules. The application protocol layer provides the following functions and a message data structure:

- *RFMESSAGE* is the message data structure for both, requests and responses. The structure contains a 1-byte field for application use, e.g. to indicate the type of

[5] http://code.google.com/p/xbee-arduino/

message exchanged, and a payload field of maximum size *RFMESSAGESIZE* (defaults to 10 bytes). The payload field can be accessed as an array of bytes, as a string of characters, as a 16-bit integer or an unsigned byte.

- *getRequest*() to receive a packet from the MAC layer; returns a pointer to the *RFMESSAGE* structure.
- *sendResponse*() to send a reply package; uses a *RFMESSAGE* structure to marshal payload data in little endian coding.

If configured for RF transmission, the PCSN invokes the *sendResponse*() function periodically to transmit the content of the transition count vector c. The periodicity is configurable. If a period (default: 1000ms) has elapsed, the transmission is scheduled, immediately following completion of the COUNT algorithm. The *sendResponse*() function transmits the payload data via serial interface to the RF module, which subsequently and asynchronously transmits the data.

5.3.4 Data Collection Server

In order to utilize the count data collected by a base station, the data needs to be served from some central source. For this purpose a prototype implementation of a web-based data collection server (DCS) has been implemented as a Python web application.

DCS implements the model as specified in Subsection 5.3.1, that is it knows about spatial areas, called *Spaces*. For each *Space* it manages an occupancy count, which is the delta of all of the in/out counts received for a particular *Space*. The occupancy count is adjusted based on observations as reported by base stations. An observation is either a pre-calculated occupancy value, an in or out count, or a delta count ($delta = in - out$). For each observation DCS records a *SpaceEvent*, which is a log of all observations received.

DCS provides the following set of JSON-enabled web services. JSON was chosen as the protocol since there are implementations for many programming languages and platforms, and thus enables arbitrary clients, e.g. smartphones or tablet PCs, to interact with the server.

- Observe ($/observer/observe$) to report an observation for a *Space*
- Query ($/observer/query$) to query the occupancy count for a *Space*

In addition to these web services, the DCS also provides a set of XML-based low-level web services to support the querying, inserting and updating of specific *Space* and *SpaceEvent* objects (see screenshot in Subsection 8.5.2). These web services also provide a query interface to search for and list objects qualifying for certain criteria. The provision of JSON and XML web services are enabled by two open-source libraries (details see Section 8.2).

5.4 Validation Framework

Validation of the PCSN requires the capability to analyse raw (unfiltered) sensor readings off-line, to configure and test the FILTER, PEAK AND COUNT algorithms. For this purpose, a validation framework was developed:

PC Logger Script This *bash* script is used as the PC-counterpart to the *LOGGER* mode of the PCSN. It captures the node's output into an ASCII file.

Analysis Routines Using the R statistical package, a collection of routines was developed to analyse the sensor data captured. These routines include implementations of the FILTER, PEAK, COUNT algorithms in the R-language. Further routines provide for analysis and visualization of runs of these algorithms, e.g. to compare the performance of different types of filters. For example, Figure 5.2 was created by the routine $plotbyfilter()$: it accepts a set of different filters and the full input signal of an experiment to produce a visualization. Similarly, routine $ex.run()$ executes these algorithms for multiple experiments and filters, and thus it allows to compare the performance of the algorithms for multiple scenarios in a single step. Several helper functions made it possible to work efficiently when translating algorithms from R- to C-language - e.g. routine $filter2c()$ translates an IIR filter object into the corresponding C structure.

Algorithm Test Driver Recall the sensor node implementation of the algorithms. In order to evaluate the performance of these routines, a PC-based test driver was developed. It reads the experiment data output by the *LOGGER* mode of the PCSN. Then it calls the C implementations of the algorithms iteratively, as if they were executed on a sensor node. The outputs of each algorithm are subsequently stored in a log file to be analysed by the R routine $testccode$. This yields a direct, visual comparison of the performance of the R and C implementations. Figure 5.3 depicts an example.

5.5 Space Occupancy Estimator

Remark. This section provides a solution outline to the problem of state estimation in relation to the people counting scenario. As a first approximation, the idea is to linearize the people counting problem by making several assumptions about the people counting process and measurement system. In a more realistic setting, these assumptions are likely to be inaccurate. In particular, the assumptions about the probability distribution functions of the process and measurement noises are likely unrealistic in a real scenario [6].

The counting algorithm presented in Chapter 4 has a fundamental limitation: its counting is only accurate in regards to some probability function:

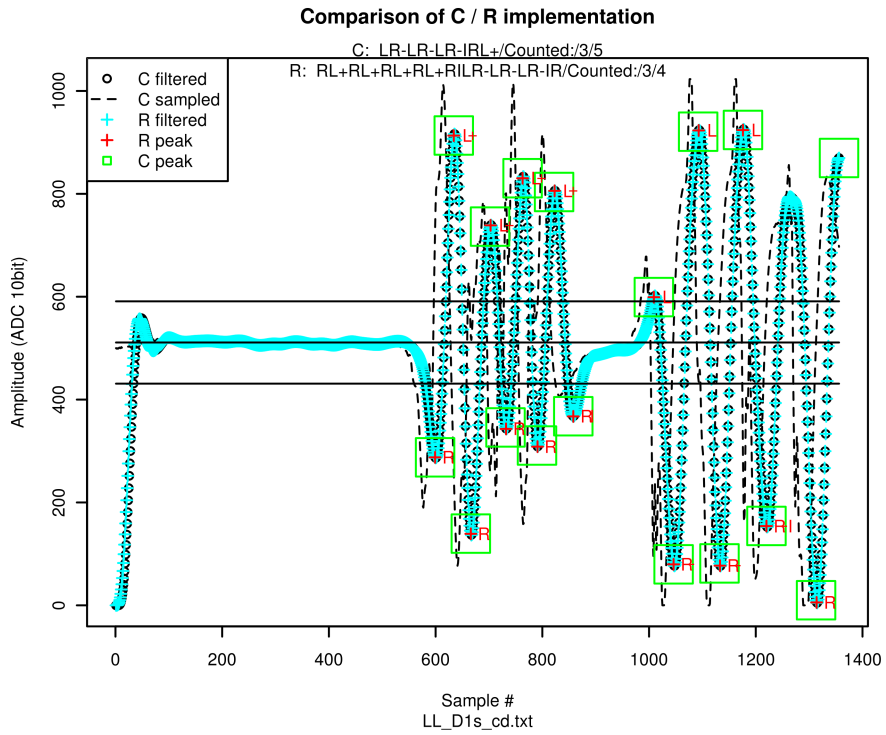


Figure 5.3: Comparison of respective results of the R- and C-language implementations of the COUNT algorithm. The evaluation framework runs both implementations using the same input data. The two lines below the title show the state strings and the count values respectively with the *CR* and *CL* state activations annotated by plus and minus signs. Note that the C-language state string is reduced to the size of the iteration buffer.

- The people count used is polluted by false counts, or *measurement noise*. While filtering the signal reduces or even removes the noise present in the sensor signal, the filter cannot reduce false counts. False counts may occur for many reasons, including: as a result of overlaps in the signal, e.g. due to multiple people walking by the PIR sensor at the same time, or in opposite directions; too many directional changes below the sensor's resolution; people waiting in front of the sensor instead of passing, yet causing a detection of a traversal.

- The measured process - people passing through an entrance door - is itself a stochastic process. Some sources of the *process noise* include: the rate of people passing through the gate may change intermittently and unexpectedly; some people increasing or decreasing their speed while passing, or turning half-way past the passage; sudden build-up of waiting queues in the observed spatial area due to some blockage.

The above sources of measurement and process noise cause the individual as well as the

aggregate people count of any people counting system to be limited within the bounds of process noise, accuracy of the sensor and performance of the algorithms used.

For the purpose of this discussion we view the people counting process (Definition 1) as a linear system, where the future state solely depends on the present *occupancy* (state) and the *in* and *out* counts (measurements) at discrete-time instance k.

Consider the model of spatial areas of Definition 10. Assume that for each area S_A and discrete-time instance k, the system collects N measurements of occupation defined as

$$occ_{k,i} = occ_{k,i-1} + delta_k, delta_{k,i} = (in_{k,i} - out_{k,i}), 0 \leq i < N, occ_0 = 0 \tag{5.5.1}$$

Then let us model the state space of each spatial area as a column vector

$$x_{k,i} = \begin{bmatrix} occ_{k,i} \\ delta_{k,i} \end{bmatrix} \tag{5.5.2}$$

(w.l.g. we will subsequently drop the index i).

Further assume we have prior knowledge of the rate of change α_k that $delta_k$ is subject to, at timesteps k. a_k is assumed to be derived from previously established measurements or defined by the probability distribution function. Then we define the process model as

$$occ_{k+1} = occ_k + \alpha_k \cdot delta_k + w \tag{5.5.3}$$

where k denotes the discrete-time instance and w is the process noise. Similarly, we define the measurement model as

$$delta_{k+1} = \alpha_k \cdot delta_k + w \tag{5.5.4}$$

Since we know α, it is reasonable to assume that we also have a notion of previous occupation levels at time instance k. We can use this knowledge to (re-)initialize the state by "injecting" a control input u_k in case of drastic error of the state estimate (such as an occupation less than zero). Transforming these equations into a linear difference equation, we have process equation

$$x_{k+1} = \begin{bmatrix} 0 & \alpha_k \\ 0 & \alpha_k \end{bmatrix} x_k + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_k + w \tag{5.5.5}$$

and measurement equation

$$y_k = \begin{bmatrix} 0 \\ 1 \end{bmatrix} x_k + v \tag{5.5.6}$$

The process and noise covariance matrices w and v are assumed to be normally distributed. These equations form the basis for the Kalman Filter to predict the *true* space occupation.

A simulation was implemented using MATLAB based on the above assumptions and equations as follows:

1. A set of D "known true" deltas $ktruedelta$ and space occupation $kocc$ are randomly defined as the baseline for the measurements. D defines the number of discrete time steps for the simulation. From these sets the α and α'change rates are calculated.

2. To simulate an actual series of observations, a set of D *true* values of deltas ($truedelta$) and occupations (occ), are calculated, assuming a normally distributed random process noise w.

3. For the duration of D time steps, random measurements $y_k = truedelta(k) + v$, $0 < k < D$ are generated, assuming a normally distributed random measurement noise v. The initial state x_0 is initialized to control input $u_0 = ktruedelta(1)$.

4. Each simulated measurement is processed by the Kalman Filter.

5. The state and measurement matrices x and y are stored for later analysis.

Figure 5.4 represents the result of a simulation run for $D = 60 \cdot 4$, assuming one measurement per minute during four hours. The plot depicts the occupation as estimated by the Kalman Filter, the true occupation as given by occ and a naive calculation of the occupation by simply adding the simulated measurement of delta y_k.

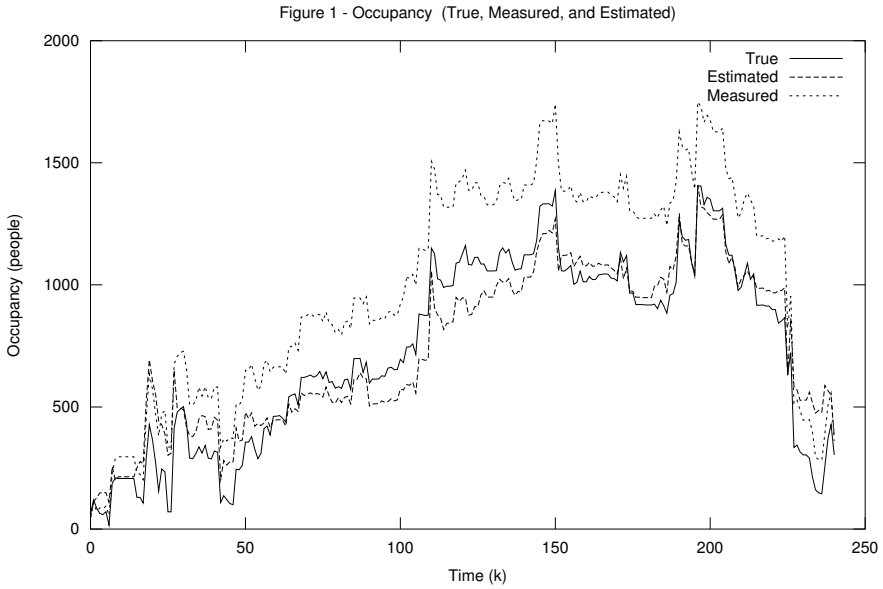


Figure 5.4: A successful simulation run reflecting the space occupancy as estimated by the Kalman Filter vs. the true situation. The simulation covers a time span of 240 minutes, assuming one measurement per minute.

Unfortunately, the simulation is not stable: as shown in Figure 5.5 the randomization can cause drastic estimation errors which the Kalman Filter is unable to correct.

In particular, an improved definition would need to take into account the constraints imposed on the occupation *occ* and delta *delta* values, that is, at all times it holds that $occ \geq 0$, $delta \geq -occ$.

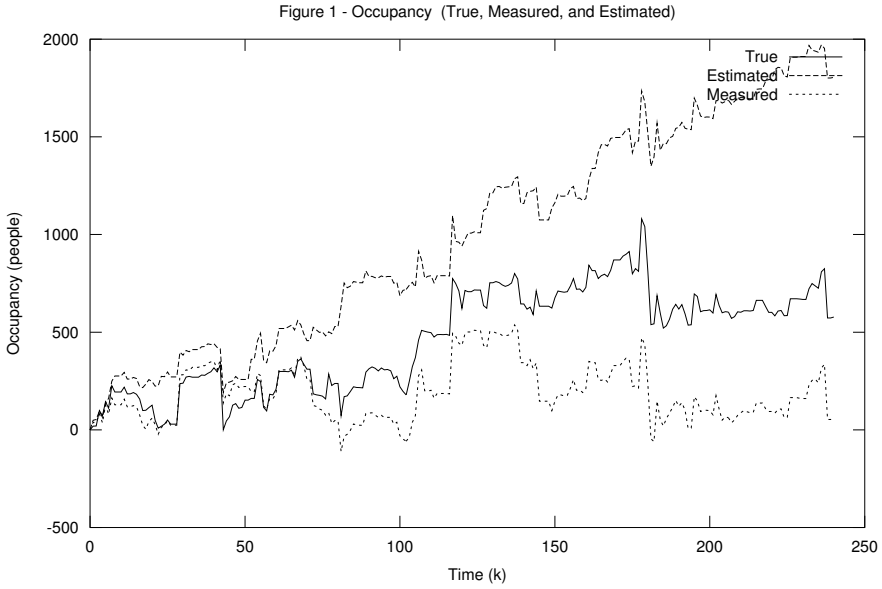


Figure 5.5: Unstable simulation run of the Kalman Filter. Drastic errors in the estimation occur, which are not corrected by the Kalman Filter. One approach to alleviate this problem is to constrain the process and measurement models to positive values.

CHAPTER 6

Evaluation

To evaluate the algorithm and the PCSN's performance, two distinct series of experiments were conducted. The first series was used to capture sensor data resulting from several different movement patterns, including single-person passages and multiple people passing in line. For evaluation, the sensor output was subsequently processed by the C-implementations of the COUNT, PEAK and FILTER algorithms, using the test driver. Recall from Section 4.6 that the C implementation uses a restricted regular expression as a realization of the FSM. To understand the impact of this limitation, one evaluation applied the R implementation based on a fully specified DFA. All experiments were timed in order to gauge the effect of higher rates of passages. The same experiments were conducted using different timing parameters in order to simulate people walking up-close or at larger distances from one person to the next. The evaluation was conducted in an office room, where the PCSN observed the room's door to an adjacent hallway.

The second series of experiments consisted of two installation scenarios a, b of two installed PCSNs. In scenario a, the PCSNs were installed at two opposite doors of a meeting room, to count people walking in and out of this room. The nodes were configured to periodically transmit (interval 1000ms), via the ZigBee RF link, their respective count values to the Monitor application set-up at the base station. In scenario b, the PCNSs were mounted to count people entries and exits at the main entrance of building ETZ at ETH. A concurrent manual count provided the baseline for later analysis.

6.1 COUNT Algorithm Performance

The following experiments were conducted for each of the time distances d indicated in Table 6.1. The time distances give an approximate indication of how close to each other the participants walked in line. The speed at which to walk was not regulated, however, the lower the distance in between passing events, the quicker each preceding individual would make space for the next. A timer-based, toggling-LED installed in view of the participants indicated the time to start walking. As a result of slightly

differing reaction latencies to this signal among participants the time distances should be considered approximate values.

Experiment [code]	Time distance d between passing (seconds)
One Person walking back and forth [1cd]	$10, 5, 2.5, 1$
Queue of four people, all same direction [queue]	$10, 5, 2.5, 1, 0.5$
Queue of four people in each direction [cd]	$10, 5, 2.5, 1$
Opposite passage of two or more people [od]	$10, 5, 2.5, 1$

Table 6.1: Experiments

Consider Figure 6.1. The graph depicts the COUNT algorithm's results *cin*, *cout* for each experiment and in relation to expected count values *cexpin*, *cexpout*. The *queue* experiment indicates good performance for distances of $d = 10, 2.5, 1$s, whereas the performance for all other distances and experiments is rather poor. The algorithms' performance can be improved by changing the respective parameters. For reference, the algorithm parameters for all experiments are provided in the appendix (Table 8.1).

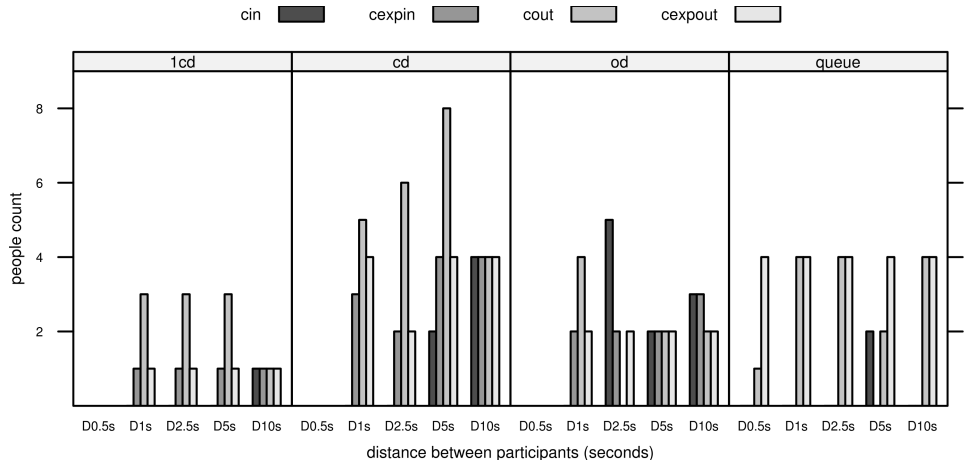


Figure 6.1: Experiment results using algorithm parameters (a) of Table 8.1. The results are acceptable for some scenarios, however, fail in other scenarios.

Careful consideration of different parameter settings and the cut-off frequency of the Butterworth low-pass filter resulted in an improved performance as shown in Figure 6.2. These results are mainly due to the parameters $MAXDIST$, $PEAKWDTH$, as well as the filter's cut-off frequency. The decrease in $MAXDIST$ causes the algorithm to consider an *ignore* state sooner, and widening $PEAKWDTH$ results in improved peak detection. Increasing the cut-off frequency of the low-pass filter helps in detecting more peaks, in particular for distances smaller than 5s.

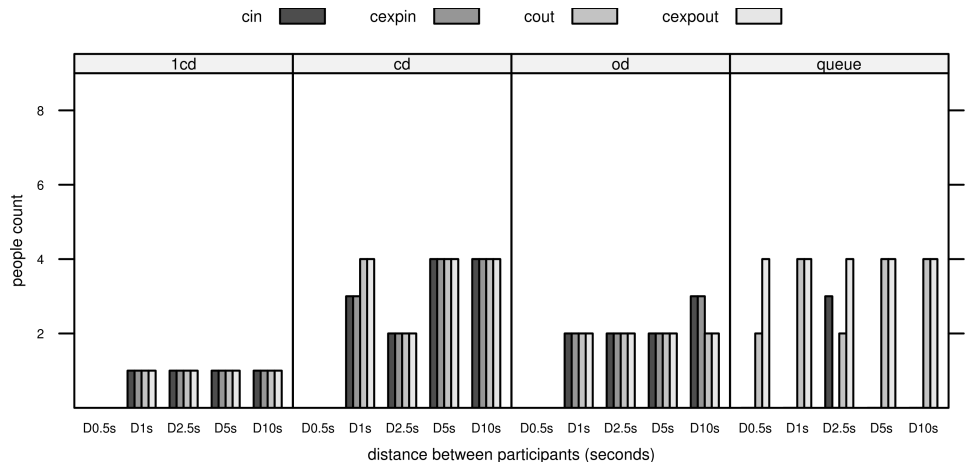


Figure 6.2: Experiment results using improved parameters (b) of Table 8.1. In particular, setting the cut-off frequency of the low-pass filter at 1Hz, and changing the COUNT algorithm's parameters to more appropriate values (e.g. $MAXDIST$, $PEAWIDTH$) produces improved results for most scenarios and time distances.

The people count (y-axis) indicates the total count in each direction. Despite the improved parameters, some problems persist. Let us consider distance $d = 2.5s$ of the *queue* experiment. In this case, the cut-off frequency $cf = 1$Hz causes the filter to "over-smooth" the signal, and in turn the algorithm misses to consider relevant peaks. As a result, the directional counts are invalid, and the total count overshoots by 25%. A similar pattern is responsible for the result at distance $d = 0.5s$. Both cases can be alleviated by setting a higher cut-off frequency, and by applying the regular expression as implemented by the R version of the COUNT algorithm. However, these changes result in a much diminished performance for all other experiments.

Consider Figure 6.3 for a normalized comparison of algorithm performance at various cut-off frequencies and respective parameters. Across experiments, the parameters (b) in Table 8.1 work best. For some installation scenarios parameters (c) might be applicable, e.g. in an area of high-frequency pedestrian or passenger volumes and where there is mostly one-directional traffic.

6.2 PCSN Performance

For scenario a, two PCSNs were configured using parameters (a) of Table 8.1. Corresponding to the respective evaluation of the algorithms (Figure 6.1), the combined counts were accurate for distances $t = 10,5s$, and increasingly erroneous for other distances. Note that people left the room at timed intervals, while the distances at the room entrance were chosen at random by the participants arriving in line. The doors are $2.1\text{m}x0.8\text{m}$ (height x width) in dimension, which is within the assumed specifications

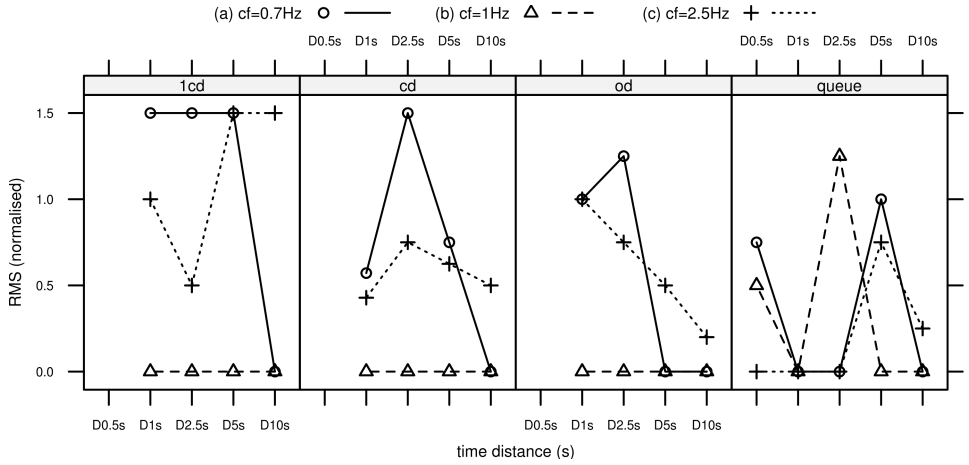


Figure 6.3: Normalized RMS of people counted and traversing at timed distances. The figure depicts the result as a function of the filter cut-off frequency (a), (b), (c) and other parameters as in Table 8.1. Overall, the best results are achieved by a cut-off frequency of 1Hz. However for distances of 0.5, 2.5Hz and the scenario of people traversing in line, a low-pass filter of 2.5Hz would improve the results. The COUNT algorithm could be improved to select the most appropriate filter automatically.

as defined in Section 4.1 For both cases, four people participated. Table 6.2 summarises these results.

	PCSN #1 (entry)	PCSN #2 (exit)	Total for room
Distance	In/Out	In/Out	In/Out
10s	4/0	0/4	4/4
5s	4/0	0/4	4/4
2.5s	1/2	3/1	4/3
1s	4/0	0/2	4/2
0.5s	4/0	0/1	4/1

Table 6.2: Results of PCSN installation for scenario *a*

In scenario *b*, two PCSNs were mounted at the upper frame of the entrance door. The door is 2.4mx1m in dimension (height x width), which is also within the assumed specifications. Before performing the evaluation, we tested and configured Sensor 1 for this particular environment by setting the parameters to configuration (b) of Table 8.1, while Sensor 2 used the default configuration (a). The evaluation was conducted for multiple periods and the count values recorded in intervals of five minutes. The sensors needed resetting during the experiment due to new insight on the effect of the

environment. Due to a technical problem and the battery running empty, Sensor 2's values were only recorded for one period. Table 6.3 summarises the results of scenario *b*.

Time	Manual Count	Sensor 1 Count	Sensor 2 Count	Notes
0:00	4 / 3	3 / 4	not recorded	
0:05	4 / 1	6 / 4	not recorded	left: group
0:10	1 / 2	7 / 4	not recorded	
0:15	2 / 3	12 / 4	not recorded	
0:20	1 / 3	13 / 4	not recorded	right: group
0:25	0 / 2	16 / 4	not recorded	
Total	*12 / 14*	*16 / 4*	not recorded	
Reset sensor regular expression R_{left}				
0:30	2 / 2	3 / 0	0 / 1	
0:35	2 / 3	4 / 0	0 / 2	right: group
0:40	2 / 1	7 / 0	1 / 2	
0:45	1 / 2	10 / 0	2 / 2	
Total	*6 / 6*	*7 / 0*	2 / 2	
Reset sensor viewing angle to 50° /20°				
0:45	3 / 1	3 / 1	out of battery	
0:55	2 / 0	5 / 1	out of battery	
1:00	4 / 4	9 / 2	out of battery	right: group
1:05	4 / 4	13 / 2	out of battery	right: group
1:10	4 / 1	16 / 4	out of battery	
Total	*14 / 9*	*16 / 4*	out of battery	

Table 6.3: Results of the evaluation at a building entrance door. The results indicate the need for further analysis of the impact of differences in temperature on the two sides of the sensor and the traversal pattern of groups of people.

The first evaluation period clearly showed that the directional counts did not work as expected. Instead, the sensors counted some traversals as if they happened in the same direction. Closer inspection of the entrance area provided the insight that one side of the entrance door was warmed by sunlight. As a result, the signal by the dual-element sensors were biased towards the cooler side, and thus were wrongly interpreted by the algorithm.

In order to compensate for this effect, we changed the covering of the PIR sensors such that the field of view included up to 40° on the cooler side, while we limited the other side to 20°. In a first attempt, the algorithm's regular expression was changed to compensate for the temperature difference by setting $R_{left} = LL?R?$. Using this expression, however, resulted in lower sensitivity on the side warmed by sun-light. The next step was to reset the sensor's viewing angle, which improved the directional count, but still did not yield the accuracy expected.

It is apparent from the data that the moving pattern for groups is different than in the previous section. We assumed that groups would pass the sensor in line. The experiments conducted in this scenario indicate, in contrast, that groups approach the sensor by two or more people at once, and only then start to pass the sensor one-by-one. We did not analyse the sensor signal in this particular scenario. The count values indicate that group passages are either counted as none, or as two people. Concluding from this experiment, there is a need to further analyse the impact of differences in temperature on the two sides of the sensor, to better understand the impact of the environment (e.g. door height, room height), and the traversal pattern of groups of people.

CHAPTER 7

Conclusion

The goal of this thesis was to develop a distributed people counting system by inexpensive hardware. Dual-element PIR sensors were chosen for their simplicity and effectiveness. This type of sensor has several advantages: it operates at low-power levels, no ambient light source is required and the effects of slow changes to the environment are cancelled automatically. In contrast to other approaches used for people counting, e.g. cameras or infrared arrays, interpreting a PIR sensor signal imposes relatively low requirements in regards to compute power.

We have considered the people counting problem from different perspectives. This included the development of algorithms to parse PIR sensor signals, the collection of results from a distributed network of sensor nodes, and the estimation of a total space occupancy based on such measurements. The algorithms developed apply a novel approach to the people counting problem. In particular, the application of a configurable FSM, based on regular expressions and combined with several other parameters enables a sensor node to adopt to a range of deployment scenarios. The prototype implementation of these algorithms on sensor nodes demonstrated their feasibility for deployment in a wireless sensor network. Using a number of base stations as the gateway to the web-based data collection server, a setup covering larger areas is possible.

7.1 Future Work

While the evaluation demonstrates the adaptability of the people counting algorithm to various movement patterns, it reveals several open problems. In a real-life setting, people cannot be expected to pass an observed passage at presumed or timed intervals. In consequence, the FILTER and COUNT algorithms should be able to detect the rate of passages, and adopt the algorithm parameters and filter strategy automatically.

Further, the COUNT algorithm should be changed such that it automatically compensates for temperature differences on either side of the sensor, e.g. by estimating a signal bias to either side, and by evaluating the relative maximum strength of the sensor signal amplitude. The algorithm could also be improved to derive and take into account a probabilistic estimate in relation to the ratio of directional changes it detects.

In some scenarios the calculated total occupancy for some observed space may be within acceptable limits of accuracy, despite each sensor node's count values. Stochastic state estimation combined with previously recorded statistics can be applied to solve this problem in principle. However, our solution approach using a linear Kalman Filter is not sufficiently stable. Future work should include evaluation of non-linear estimation models.

The prototype implementation of the PCSN does not make use of the dynamic characteristics of the counting algorithms. While the prototype code exposes parameters to configure the various operating modes and to change run-time parameters, effecting these changes requires re-programming of each sensor node by USB cable. For realistic and productive use, the PCSN needs to be extended such that the algorithms can be configured through ZigBee. That is, the PCSN requires a management mode, which can be activated from the base station in order to dynamically apply changes to filter coefficients and other parameters. This includes the extension of the iterative regular expression algorithm to support character repetition and grouping.

The ZigBee protocol was chosen for its low-rate, low-power capabilities. The prototype PCSNs do not realize a low-power mode, and hence are inefficient to operate on battery. In a productive scenario, a low-power mode is required to include both the CPU and the RF module. The PIR sensors ability to detect motion independent of a host processor could be exploited as a trigger to wake-up the node.

CHAPTER 8

Appendix

8.1 Evaluation Parameters

Parameter	Value
MIDTHRES	80
MAXDIST	200
MINDIST	20
ABSTHRES	80
PEAKWDTH	5
MATCH_LEFT	LRL?
MATCH_RIGHT	RLR?
FSM	C, simplified RegExp
Filter	Butterworth, $cf = 0.71$Hz
Sensor Field of View	$\pm 20°$

(a) Results see Figure 6.1

Parameter	Value
MIDTHRES	80
MAXDIST	60
MINDIST	5
ABSTHRES	100
PEAKWDTH	10
MATCH_LEFT	LRL?
MATCH_RIGHT	RLR?
FSM	C, simplified RegExp
Filter	Butterworth, $cf = 1$Hz
Sensor Field of View	$\pm 20°$

(b) Results see Figure 6.2

Parameter	Value
MIDTHRES	80
MAXDIST	60
MINDIST	5
ABSTHRES	80
PEAKWDTH	10
MATCH_LEFT	^(I)+LR((L)*R)*$
MATCH_RIGHT	^(I)+RL((R)*L)*$
FSM	R, PERL RegExp
Filter	Butterworth, $cf = 2.5$Hz
Sensor Field of View	$\pm 20°$

(c) Results see Figure 6.3

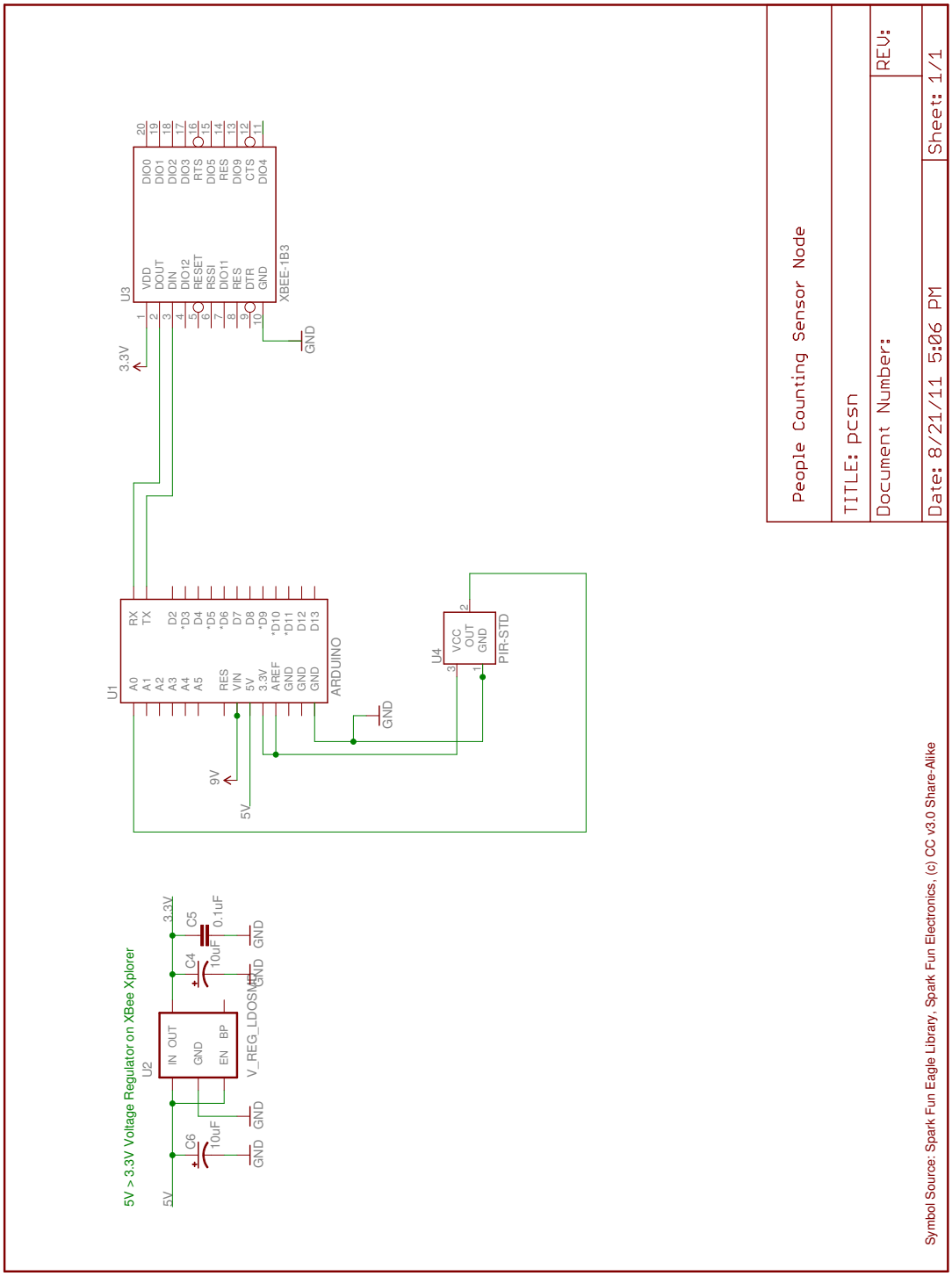
Table 8.1: Algorithm parameters used in evaluation

8.2 Technology Overview

The following table provides a list of all third-party technology used for the prototype implementation.

Area	Description	Details [software license]	Vendor/Source
Hardware	Sensor Node Prototype	Name: Arduino UNO Microcontroller: ATmega328 Operating Voltage: 5V Flash Memory: 32KB, SRAM: 2KB EEPROM: 1KB Clock Speed: 16MHz	SparkFun Electronics www.sparkfun.com Product ID 9950
	Sensor Node Testboard	Microcontroller: ATmega168 Operating Voltage: 5V Flash Memory: 16KB, SRAM: 1KB EEPROM: 512 bytes	Dshop Electronics www.dshop.ch Product ID ARD2K
	Sensor Node RF Module ZigBee, 3V break-out regulated	DigiKey XBee Series 1 Frequency: 2.4GHz Standard: IEEE 802.15.4 Operating Voltage: 3.3V, 50mA Data Rate: 250kbps Range: 100m	SparkFun Electronics www.sparkfun.com Product ID 8664 (XBee module), 9132 (XBee break-out regulated)
	PIR Sensor	Operating Voltage: 3-12V / 1.4mA Range: 4-12m Coverage angle: $v: \pm 30° / h: \pm 50°$ Signal bandwidth: 0.2Hz – 10Hz	Conrad Electronics www.conrad.ch Product ID 172500-62
Software	Sensor Node	Arduino Development Environment GCC 4.4.5 [GPL] avr-libc 1.6.8 [Copyleft] XBee API 0.2 [GPL]	www.arduino.com http://gcc.gnu.org/ http://www.nongnu.org/avr-libc http://code.google.com/p/xbee-arduino/
	Sensor Monitor	Processing 1.5 [GNU] OpenJDK Java 1.6 [GNU] Xbee API 0.9 [GNU]	http://processing.org/ http://openjdk.java.net/ http://code.google.com/p/xbee-api/
	Data Collection Server	Python 2.6.6 [PSF] appengine-rest-server [Apache] pyrest [MIT] mpmath [BSD]	http://python.org/ http://code.google.com/p/appengine-rest-server/ http://code.google.com/p/pyrest/ http://code.google.com/p/mpmath/
	State Estimator	kalmanf.m [MATLAB Central]	www.mathworks.com

8.3 Circuit Diagram



8.4 Implementation Reference

The following table provides a summary of all implemented program modules and functions by the validation framework and the prototype.

Area	Program/Function	Description	Programming Language
Validation Framework (Simulation)	countlib.r	R implementation of algorithms and helper functions	R
	ex.run()	Run the algorithms for experiment data using different filters, display count results in tabular format	R
	ex.plotbyfilter()	Run algorithms using different filters, plot results	R
	testccode	Run R algorithms and visually compare output to C implementation's result	R
	hots()	Identify peaks in signal using standard deviation around *sw* points	R
	count()	Count peaks according to FSM specification (several variants of count exist: count_fix, count_fsm, count_dynfsm) for execution of simple algorithm, hard coded FSM, regular expression based FSM)	R
	count_c()	R implementation of C version of count() (restricted Regular Expression)	R
	filter.CArma	R implementation of C version of filter()	R
	filter2c	Output C data structure of coefficients of a R filter defined by the signal package	R
	logger.sh	Logging routine to act as counterpart to the sensor node's LOGGER mode	Bash
Sensor Node	filter.h	Algorithm implementation for C-language	C
	hots()	Identify peaks in signal using standard deviation around *sw* points	C
	count()	Count peaks according to simplified regular expression	C
	filter()	IIR signal filter implementation	C

Area	Program/Function	Description	Programming Language
Sensor Node (c'td)	BI()	Macro for ring-buffer access of C array using negative indicies	C
	RegExp.cpp+.hpp RegExp::	Simplified regular expression for iterative pattern matching	C++
	RFMessage.c+.h	Application-level ZigBee messaging to simplify Digikey's XBee API	C
Monitor	MonSens.java	Monitor application using the Processing framework	Java
	RFMessage.java	Java implementation of application-level ZigBee messaging	Java
	Counter.java	Display of in/out counters	Java
Data	observer.py/main	REST and XML interface driver	Python
Collection Server	observer.py/ObserverSvc	REST and XML service component	Python
	observer.py/SpaceObserver	Python Kalman Filter implementation (deactivated)	Python
	space.py/Space	Space model implementation (incl. persistency)	Python
	space.py/SpaceEvent	Space model implementation for observation events (incl. persistency)	Python
State Estimation	occest.m	Occupation estimation based on Kalman Filter	MATLAB
	kfilter.m	Kalman Filter implementation	MATLAB

8.5 Screenshots

8.5.1 Sensor Monitor - SensMon

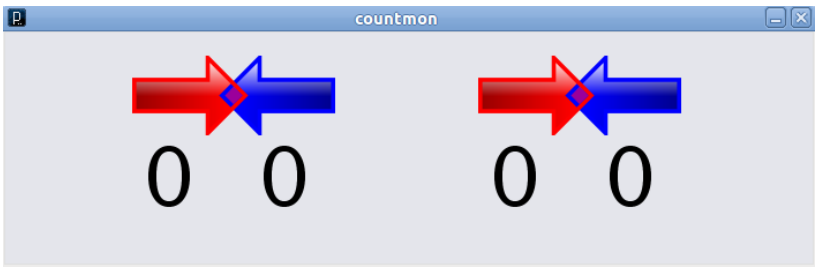


Figure 8.1: Monitor display at the base station

8.5.2 Data Collection Server - DCS

http://cleverp....com/api/event

This XML file does not appear to have any style information associated with it. The document tree is shown below.


```
−<list offset="50">
  −<event>
     <key>ag9zfmNsZXZlcnBlbmRlbG5yEAsSClNwYWNlRXZlbnQYBAw</key>
     <count>2</count>
     <time>2011-07-03T16:46:08.688960</time>
     <type>out</type>
     <space>ag9zfmNsZXZlcnBlbmRlbG5yCwsSBVNwYWNlGAEM</space>
   </event>
  −<event>
     <key>ag9zfmNsZXZlcnBlbmRlbG5yEAsSClNwYWNlRXZlbnQYBQw</key>
     <count>4</count>
     <time>2011-07-03T16:49:10.220284</time>
     <type>out</type>
     <space>ag9zfmNsZXZlcnBlbmRlbG5yCwsSBVNwYWNlGAEM</space>
   </event>
  −<event>
     <key>ag9zfmNsZXZlcnBlbmRlbG5yEQsSClNwYWNlRXZlbnQY6gcM</key>
     <count>1</count>
     <time>2011-07-03T16:46:06.845458</time>
     <type>out</type>
     <space>ag9zfmNsZXZlcnBlbmRlbG5yCwsSBVNwYWNlGAEM</space>
   </event>
  −<event>
     <key>ag9zfmNsZXZlcnBlbmRlbG5yEQsSClNwYWNlRXZlbnQY0Q8M</key>
     <count>5</count>
     <time>2011-07-03T16:42:33.062399</time>
     <type>in</type>
     <space>ag9zfmNsZXZlcnBlbmRlbG5yCwsSBVNwYWNlGAEM</space>
   </event>
```


Figure 8.2: Event log of the data collection server

Bibliography

[1] T. Teixeira and A. Savvides. Lightweight People Counting and Localizing in Indoor Spaces Using Camera Sensor Nodes. In *Proceedings of Conference on Distributed Smart Cameras, 2007, ICDSC*, pages 36 –43.

[2] S. Velipasalar, Ying-Li Tian, and A. Hampapur. Automatic Counting of Interacting People by using a Single Uncalibrated Camera. In *Proceedings of IEEE International Conference on Multimedia and Expo, 2006*, pages 1265 –1268.

[3] Quanbin Chen, Min Gao, Jian Ma, Dian Zhang, Lionel M. Ni, and Yunhao Liu. Moving Object Counting using Ultrasonic Sensor Networks. *International Journal of Sensor Networks, 2008*, pages 55–65, volume 3.

[4] K. Hashimoto, K. Morinaka, N. Yoshiike, C. Kawaguchi, and S. Matsueda. People Count System using Multi-Sensing Application. In *Proceedings of International Conference on Solid State Sensors and Actuators, 1997. TRANSDUCERS*, pages 1291 –1294, volume 2.

[5] P. Zappi, E. Farella, and L. Benini. Enhancing the Spatial Resolution of Presence Detection in a PIR based Wireless Surveillance Network. In *Proceedings of IEEE Conference on Advanced Video and Signal Based Surveillance, 2007, AVSS*, pages 295 –300.

[6] S. Meyn, A. Surana, Yiqing Lin, S.M. Oggianu, S. Narayanan, and T.A. Frewen. A Sensor-Utility-Network Method for Estimation of Occupancy in Buildings. In *Proceedings of IEEE Conference on Decision and Control, held jointly with the Chinese Control Conference, 2009, CDC/CCC*, pages 1494 –1500.

[7] Emi Mathews and Axel Poigné. Evaluation of a "Smart" Pedestrian Counting System based on Echo State Networks. *Journal on Embedded Systems, EURASIP*, pages 8:1–8:9, January.

[8] Muhammad Tahir, Peter Hung, Ronan Farrell, Sean McLoone, and Tim McCarthy. Lightweight Signal Processing Agorithms for Human Activity Monitoring using Dual PIR-Sensor Nodes. In *Proceedings of the China-Ireland information and communications technologies conference, CCIT, 2009*, pages 150–156.

[9] Lin Gu, Dong Jia, Pascal Vicaire, Ting Yan, Liqian Luo, Ajay Tirumala, Qing Cao, Tian He, John A. Stankovic, Tarek Abdelzaher, and Bruce H. Krogh. Lightweight Detection and Classification for Wireless Sensor Networks in Realistic Environments. In *Proceedings of Conference on Embedded networked sensor systems, 2005, SenSys*, pages 205–217.

[10] Robert Tomastik, Satish Narayanan, Andrzej Banaszuk, and Sean Meyn. Model-Based Real-Time Estimation of Building Occupancy During Emergency Egress. In *Pedestrian and Evacuation Dynamics 2008*, pages 215–224. Springer, 2010.

[11] V. Vaidehi, S. Vasuhi, K.S. Ganesh, C. Theanammai, N.T. Naresh Babu, N. Uthiravel, P. Balamuralidhar, and G. Chandra. Person Tracking using Kalman Filter in Wireless Sensor Network. In *Proceedings of Second International Conference on Advanced Computing (ICoAC), 2010*, pages 60 –65.

[12] Jian-Shuen Fang, Qi Hao, David J Brady, Mohan Shankar, Bob D Guenther, Nikos P Pitsianis, and Ken Y Hsu. Path-dependent Human Identification using a Pyroelectric Infrared Sensor and Fresnel-lens Arrays. *Optics Express*, pages 609–624, volume 14, 2006.

[13] Steven Smith. *Digital Signal Processing*, pages 107–111, 116–121, 319–322, 333–342. Newnes, 2003.

[14] Peter Linz. *An Introduction to Formal Languages and Automata, Fifth Edition*, pages 38–46. Jones & Bartlett Learning, 2011.

[15] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems*, pages 23–31,61–72. Springer, 2007.

[16] Greg Welch and Gary Bishop. *An Introduction to the Kalman Filter*. University of North Carolina at Chapel Hill, Department of Computer Science, 1995.

[17] Zhiqiang Zhang, Xuebin Gao, J. Biswas, and Jian Kang Wu. Moving Targets Detection and Localization in Passive Infrared Sensor Networks. In *Proceedings of International Conference on Information Fusion, 2007*, pages 1 –6.

[18] P. Zappi, E. Farella, and L. Benini. Pyroelectric InfraRed Sensors based Distance Estimation. In *Proceedings of IEEE Sensors, 2008*, pages 716 –719.

[19] G. Palshikar. Simple Algorithms for Peak Detection in Time-Series. In *Proceedings of 1st IIMA International Conference on Advanced Data Analysis*. Business Analytics and Intelligence, 2009.

[20] Tom Short. *R Signal Processing Package*, 2011. R package version 0.6-2.