DIPLOMARBEIT

Wireless Real-Time Communication for Smart Transducer Networks

ausgeführt am

Institut für Technische Informatik 182/1

der

Technischen Universität Wien

unter der Leitung von

O. Univ. - Prof. Dr. Hermann Kopetz

und

Univ. Ass. Dipl. - Ing. Dr. Wilfried Elmenreich

als verantwortlich mitwirkendem Assistenten

durch

Bernhard Huber
Matr. - Nr. 9926084
Edla 9, 3261 Steinakirchen/Forst

Wien, am 25. August 2004 . . . . . .

Wireless Real-Time Communication for Smart Transducer Networks

Due to the increasing demand for mobility and flexibility in the area of distributed control systems, wireless communication onwardly gain importance in automation engineering. However, especially in the field of real-time systems, wireless communication is not well established yet. A main reason therefore is the variable network structure caused by mobile communication nodes, which impairs predictability and dependability. Furthermore, the retransmission of erroneous data – a common approach in wireless networks – is not well-suited for real-time systems.

This thesis describes the approach and the implementation of a wireless real-time communication protocol for the establishment of mobile smart transducer networks. The conceptual basis has been mainly inspired by TTP/A, a real-time communication protocol for non-safety critical sensor and actuator networks, which has been developed at the Vienna University of Technology.

The wireless protocol is based on the time-triggered paradigm, which means that all communication activities are derived from the progression of physical time. A single dedicated master node is responsible for establishing a synchronized time base among all communication participants. This thesis presents a synchronization algorithm that is suitable for wireless communication networks. An important feature of the proposed algorithm, which highly enhances dependability, is the ability to tolerate certain failures of the master, the communication channel, and the communication links.

Monitoring and configuration support of individual networked nodes is a crucial demand for distributed control applications. A fixed amount of bandwidth, assigned to the monitoring and configuration service, guarantees that the real-time service is not disturbed in any case.

The design of the protocol aims at the use of commercial off-the-shelf components for implementation. The practicability of the proposed protocol is shown in a case study: An autonomous mobile robot, consisting of several smart transducer nodes, is used for the exploration of an indoor environment. The gathered information is transmitted to a host PC, where a representation of the environment is generated.

Drahtlose Echtzeitkommunikation in Smart Transducer Netzwerken

Flexibilität und Mobilität sind Anforderungen die in verteilten Steuerungs- und Kontrollsystemen immer mehr an Bedeutung gewinnen. Damit verbunden steigt in diesem Anwendungsbereich die Notwendigkeit von drahtlosen Kommunikationssystemen. Jedoch gerade für Echtzeitsysteme ist die Akzeptanz und Verbreitung drahtloser Kommunikation noch gering. Eine Ursache dafür ist eine mögliche Beeinträchtigung der Zuverlässigkeit solcher Systeme aufgrund der variablen Netzwerkstruktur, die sich aus der Mobilität des Systems ergibt. Zusätzlich sind in drahtlosen Netzwerken gängige Datenkorrekturmechanismen wie das wiederholte Übermitteln von fehlerhaften Daten für Echtzeitsysteme nicht sehr geeignet.

Diese Arbeit beschreibt den Ansatz und die Implementierung eines drahtlosen Echtzeitkommunikationsprotokolls für den Aufbau mobiler *Smart Transducer* Netzwerke. Die Basis vieler Designentscheidungen bildet hierbei TTP/A, ein Echtzeitkommunikationsprotokoll für nicht sicherheitskritische Anwendungen, das an der Technischen Universität Wien entwickelt wurde.

Das in dieser Arbeit beschriebene Protokoll basiert auf dem zeitgesteuerten Paradigma: Alle Kommunikationsabläufe werden ausschließlich vom Fortschreiten der physikalischen Zeit gesteuert. Ein einzelner dedizierter Masterknoten ist für den Aufbau einer synchronisierten Zeitbasis verantwortlich. Ein wichtiger Teil dieser Arbeit ist die Entwicklung eines Synchronisationsalgorithmus, der die gesteigerten Anforderungen in drahtlosen Netzwerken berücksichtigt. Eine wichtige Eigenschaft dieses Algorithmus ist die Fähigkeit, bestimmte Ausfälle des Masterknotens sowie des Kommunikationskanals zu tolerieren, um damit die Verlässlichkeit des Systems zu steigern.

Die Möglichkeit zur Überwachung und Konfiguration einzelner Knoten des Systems während des Betriebs ist eine entscheidende Anforderung an Kommunikationsprotokolle für Smart Transducer Netzwerke. Um ein störungsfreies Arbeiten des Echtzeitservices zu garantieren, wurde ein fixer Anteil der Bandbreite des Kommunikationskanals für das Überwachungs- und Konfigurationsservice reserviert.

Anhand einer Fallstudie wird die Praxistauglichkeit des Protokolls gezeigt: Ein aus mehreren Smart Transducern bestehender mobiler autonomer Roboter wird verwendet, um die Form und etwaige Hindernisse eines Raumes zu erkunden. Die gesammelten Informationen werden zentral zu einer grafischen Darstellung der Umgebung verarbeitet.

Danksagung

Ich möchte mich bei Prof. Dr. Hermann Kopetz bedanken, der es mir ermöglichte, diese Arbeit am Institut für Technische Informatik zu verfassen. Besonderer Dank gilt vorallem Dipl. Ing. Dr. Wilfried Elmenreich, für seine fachlich aber auch menschlich sehr wertvolle Betreuung während meiner Diplomarbeit.

Ein großes Dankeschön gilt auch meinen Eltern, ohne deren Unterstützung mein Studium in dieser Form nicht möglich gewesen wäre.

Zudem danke ich Wolfgang Haidinger und Christian Trödhandl für wertvolle Hinweise während der Implementierung der Arbeit, sowie Verena Katinger, Wolfgang Huber und Markus Bauer für das gewissenhafte Korrekturlesen.

Meiner Freundin Verena möchte ich schließlich für ihre Unterstützung und ihr Verständnis während der Entstehung dieser Arbeit danken.

Contents

List of Figures

List of Tables

Begin – to begin is half the work,
let half still remain;
again begin this,
and thou wilt have finished.

DECIMUS MAGNUS AUSONIUS

Chapter 1

Introduction

The gradually miniaturization of electronic components, combined with the increasing power of embedded processors, and the technical advance of micro-electro-mechanical systems (MEMS) enable the development of smaller, more powerful, and more cost-effective sensor/actuator elements. This trend assists the increasing popularity of embedded computer systems in many fields of application. Domains where these advances can be applied vary from industrial control systems like plant control, robot control, or supervision systems over building automation systems for elevator control, air conditioning, or lightning control to applications in the automotive sector used for engine management, diagnostics, or X-by-wire.

Due to the increased use of sensor/actuator elements, the communication effort among this networked elements gains; consequently, the complexity of the hole system increases. To contain these negative side-effects, an appropriate network structure has to be used to interconnect the individual components as effective as possible. This includes high data efficiency as well as low wiring costs and flexible network topology.

Although the fields of application of such distributed embedded computer systems can be quite different, there exist some common aspects that have to be considered:

In many of the application areas listed above, safety critical aspects have to be taken into account. Since the correct functionality of a distributed embedded application surely depends on the correct service of the underlying communication system, *dependability* [Laprie, 1992] is a crucial property. The use of fault-tolerant or replicated communication media is one possibility to increase dependability. The usage of data protection mechanisms like parity bits or checksums is another often applied technique to enhance the dependability of communication systems.

Real-time requirements are the second important aspect that should be considered. Many applications have to provide a service which requires the correctness of the data in the value as well as the time domain [Kopetz, 1997]. This applies to all applications where

the quality of the provided service is affected by an uneven communication delay between the sensing and actuating instant. Considering distributed control applications like Steer-by-Wire for cars, sensors and actuators may be connected to different nodes in the distributed system. Delays of few milliseconds, introduced by the communication system, may cause the control application to produce a jerky steering behavior or even lead to an accident.

One possibility to fulfill the real-time requirements is the time-triggered approach. A time-triggered system consists of a set of synchronized nodes, sharing a common global time base. All communication actions and tasks are executed with respect to this distributed global time.

As the complexity of a system increases, the importance of proper maintenance and configuration support gains. Particularly, *on-line configuration* and *monitoring* methods gain in importance since they can be used to determine the state of the system without the need to stop basic parts or even the whole service. The deterministic behavior of time-triggered systems is well-suited for the realization of on-line configuration and monitoring support. If a certain amount of bandwidth is granted to the configuration and monitoring service, it can be guaranteed that the intended service of the application is not disturbed.

Increasing the number of distributed sensor/actuator nodes implicates a main drawback: The needed interconnection effort increases regardless of the network topology. *Wireless communication media* can be used to circumvent this drawback. Furthermore, applications that require mobility, e. g., mobile industrial robots, or that operate in inhospitable environments where less or suboptimal communication infrastructure is provided, also suggest the use of wireless communication.

1.1 Related Work

This thesis mainly touches two scientific fields, which are extensively discussed in the scientific literature: distributed real-time systems and wireless communication. Nevertheless, the combination of both is still a quite recent field of research these days. Wireless real-time sensor networks and wireless fieldbus networks are representatives of this combined field of research, which are closely related to this thesis.

The popularity of wireless communication for automation systems is increasing. Decotignie [2002] gives a detailed survey about requirements and presently investigated solutions for wireless fieldbusses.

The goal of the OLCHFA fieldbus [Roberts, 1993] has been the development of a fieldbus system that transparently adds wireless stations to wired segments. The factory instrumentation protocol (FIP) had been the basis of the OLCHFA fieldbus. The deployed

microwave communication systems use spread spectrum modulation techniques to gain resistance against multi-path fading as well as interference rejection.
A research group at EPFL Lausanne have followed another approach to enhance the FIP fieldbus with wireless communication capabilities. They have used a dedicated gateway as link between the wired fieldbus segment and one or more wireless stations [Morel and Croisier, 1995]. The goal of this research project has been to make the data from the wireless stations appear to be local in the gateway. The developed wireless protocol uses time division multiple access (TDMA) for bus arbitration.

Kutlu et al. [1996] have introduced a concept for wireless controller area network (CAN). They have investigated two different approaches to satisfy the message delivery time for real-time applications as well as message prioritization: The RFMAC protocol is suited for centralized networks that means that there is one dedicated master node and all slave nodes have to stay in the range of the master. RFMAC uses an adopted idle signal multiple access (ISMA) protocol for bus arbitration. The WMAC protocol on the other hand, covers distributed wireless CAN networks. It uses CSMA/CA for bus arbitration. Priority management is realized by different message delay times.

The R-Fieldbus project is an ongoing project that aims at the development of a radio-based physical layer for modern production systems, based on the existing and available technologies in the LAN and WAN world [Hähniche and Rauchhaupt, 2000]. This project is based on a hybrid wired/wireless PROFIBUS solution where so-called base stations are used to link wireless nodes to wired segments. In order to avoid timing problems caused by the coexistence of heterogeneous transmission media in the same network, an extra inactivity (idle time) is inserted by the master stations [Alves et al., 2002].

1.2 Motivation and Objectives

The main objective of this thesis is to present a communication protocol that establishes a wireless time-triggered real-time network among several possibly mobile fieldbus clusters or intelligent sensor/actuator elements, so-called *smart transducers* [Frank, 2000]. A smart transducer is a device that integrates an analog or digital sensor/actuator element, a processing unit, and a communication interface.

Additionally, this protocol should provide an interface for runtime monitoring and configuration that guarantees that the real-time behavior of the communication is not disturbed in any case.

The design of the communication protocol predominantly aims at the interconnection of multiple mobile TTP/A fieldbus clusters [Kopetz et al, 2002]. However, the design is not subject to any restrictions forced by TTP/A; thus, it could be used in any non safety

critical application where a time-triggered wireless communication is necessary.

As case study and for demonstrating the protocol's capabilities, a mobile robot has been chosen whose task is to perceive the environment and to avoid collisions with obstacles by executing a path planning and navigation algorithm. The robot consists of several sensor and actuator nodes (ultra sonic and infrared distance sensors, steering servo, etc.) forming a TTP/A network.

The environmental representations generated by the robot are transfered via the wireless real-time interface to a host PC, where a map making application builds a global view of the robot's environment. Furthermore, the wireless monitoring and configuration interface is used to calibrate and update sensor/actuator settings during operation.

The here introduced communication protocol is not suited for so-called *ad-hoc sensor networks* because these networks are mainly intended for the interconnection of a vast number of widely distributed sensor elements with sporadic and randomized communication requests. The architecture of ad-hoc sensor networks has to be optimized for high flexibility, ad-hoc routing, plug-and-play, and power save mechanisms [Mhatre and Rosenberg, 2004], which is not the main focus of this thesis.

1.3 Structure of this Thesis

This thesis is organized as follows:

Chapter 2 gives an introduction into basic terms and concepts that are used throughout this thesis. Chapter 2.1 states general requirements of real-time systems, followed by a short classification of real-time systems. Furthermore, TTP/A as a representative for non safety critical time-triggered systems is introduced. The following section gives a short introduction to wireless communication technologies. Finally, fundamental map making techniques are explained.

In chapter 3 the core of this thesis, the wireless communication protocol, is presented. Section 3.1 and 3.2 are devoted to the design principles and a detailed description of the functional properties of the protocol. In Section 3.3 the fault-tolerant clock synchronization algorithm and its properties are presented. The remainder of this chapter treats the monitoring and configuration support.

Chapter 4 describes the implemented case study that demonstrates the practicability of the introduced communication protocol. The First, the intended goals and the system architecture of the case study are shortly denoted. The remainder is devoted to the description of hardware and software components that have been partially adopted from previous projects or have been newly developed during this thesis, respectively.

The results of the case study are discussed in chapter 5. Section 5.1 presents the results

of memory, performance, and timing analysis. Section 5.2 and Section 5.3 describe the results of an on-line calibration experiment and the generation of a global map out of the wireless transmitted sensor readings, respectively.

Chapter 6 concludes the thesis and gives an outlook to possible future research in this area.

I love deadlines.
I like the whooshing sound they make as they fly by.

Hitchhiker's Guide to the Galaxy, DOUGLAS N. ADAMS

Chapter 2

Basic Terms and Concepts

The concepts that form the basis of this thesis are outlined in this chapter. First, a general description and classification of real-time systems and their requirements is given. The TTP/A protocol, a representative of a time-triggered distributed real-time system, which influences many design decisions in this work, is introduced. Different wireless communication techniques are discussed and compared afterwards. Finally, some basics regarding mobile robot positioning and map making are presented.

2.1 Real-Time Systems

A *real-time system* consists of three parts [Kopetz, 1997]: a *real-time computer system*, a controlled object, and a human operator, whereby the latter two are often referred as the *environment* of the real-time system. A real-time computer system is *"a computer system in which the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical instant at which these results are produced"* [Kopetz, 1997, page 2]. The boundary between the human operator and the real-time computer system is called the *man-machine interface* and consists of input and output devices. The interface between the real-time computer system and the controlled object is called the *instrumentation interface*.

A real-time computer system must react to a stimuli from its environment – the controlled object or the operator – within a specific time interval, the so-called *deadline*. A deadline is called *soft deadline* if the result has still utility after the deadline has passed; otherwise, it is called a *firm deadline*. If missing a firm deadline could lead to a catastrophic result, the deadline is classified as *hard*.

As the controlled object changes its state as a function of time, a snapshot of the state variables at the same time instant have to be recorded to represent the current state of

the controlled object. Normally, only a subset of the state variables is essential for the real-time system. Such a significant state variable is called a *real-time entity*.

2.1.1 Real-Time System Requirements

The requirements for real-time systems can be divided into functional, temporal, and dependability requirements [Kopetz, 1997]:

Functional Requirements

The functions a real-time system has to perform can be summarized as:

Data Collection: A real-time system must observe the real-time entities in a controlled object and collect these observations. An observation is an atomic structure that consists of the name of the real-time entity, the instant of time when the observation was made, and the observed value of the real-time entity.

$$Observation = < Name, t_{obs}, Value >$$

This observations are represented by *real-time images*. The temporal accuracy of a real-time image depends on the dynamics of the controlled object: the higher the dynamic, the shorter the accuracy interval.

Direct Digital Control: Many real-time computer systems have to access the controlled object directly, i. e., without any underlying control system. Therefore, the real-time computer system itself must be capable of calculating the set points for the actuators.

Man-Machine Interaction: Due to the information flood real-time systems have to deal with, they have to inform the operator about the relevant current state of the controlled object; furthermore, they must assist the operator in his controlling responsibilities and therefore extract and prepare the relevant information.

Temporal Requirements

Considering the different points of contact of a real-time computer system with its environment, the most stringent temporal demands for real-time systems have their origin at the instrumentation interface, e. g., with fast control loops, and not at the man machine interface because the human reception delays are orders of magnitude higher than the latency requirements of fast control loops. Temporal requirements are:

Minimal Latency Jitter: In most control applications the time when a new observation is made is in the sphere of control of the real-time computer system and therefore, known in advance. Control algorithms are designed to handle a constant delay between subsequent sample points. A delay jitter introduces an additional uncertainty into the control loop that has an adverse effect on the quality of control. Therefore, the delay jitter should always be a small fraction of the overall delay [Kopetz, 1997].

Minimal Error-Detection Latency: Considering safety-critical systems, any error has to be detected as soon as possible and with very high probability. In order to be able to perform corrective actions or to bring the system in a safe state before an error can cause a severe system failure, the error-detection latency must be in the same order of magnitude as the period of the fastest critical control loop.

Dependability Requirements

Dependability of a computer system is *the ability to deliver a service that can justifiably be trusted* [Avižienis et al., 2001]. According to Avižienis et al., six different attributes of dependability can be distinguished:

Availability is a measure of the probability that a system is ready for providing its service at a specific point t in time. The availability depends on the *mean-time-to-failure* (MTTF, the inverse of the failure rate) and the *mean-time-to-repair* (MTTR, the inverse of the repair rate) of a system. Thus, a system designer can achieve high availability either by a long MTTF or a short MTTR.

Reliability is a measure of the probability that a system will provide its specified service during a given interval of time $[0,t)$, i. e., reliability is concerned with the continuity of service.

Safety is reliability regarding *critical failure modes* [Kopetz, 1997]. Therefore, the safety of a system is the probability that no critical failure occurs during a time interval $[0,t)$.

Confidentiality is concerned with preventing information from unauthorized access.

Integrity is concerned with preventing the system state from improper alteration. Both, confidentiality and integrity are often subsumed with the term *security*.

Maintainability is a measure of the time it takes to repair a system after the occurrence of a benign failure. According to the quantitative measurement of reliability, maintainability is quantified by the probability $M(d)$ that the system is restored within the time interval d.

Due to the lack of standard measures, it is usually difficult to quantify confidentiality and integrity.

[Laprie, 1992] distinguished three types of impairment of dependability:

Fault: A fault is the cause of an error. Faults can be classified by *fault nature* (chance and intentional faults), *fault perception* (physical and design faults), *fault boundaries* (internal and external faults), *fault origin* (originated in development or during operation), and *fault persistence* (transient and permanent faults).

Error: An error is an unintended internal state of a computer system caused by a fault. [Kopetz, 1997] distinguishes between *transient errors* and *permanent errors*. A transient error exists only a short time and disappears without an explicit repair action. Permanent errors, on the other hand, remain in the system until the system state is explicitly repaired.

Failure: A system is said to have failed whenever a user of the system notice a deviation of the actual service from the intended service [Kopetz, 1997]. Failures can be classified by *failure nature* (failures in value or timing domain), *failure perception* (consistent and inconsistent failures), *failure effect* (benign and malign failures), and *failure oftenness* (permanent and transient failures).

2.1.2 Classification of Real-Time Systems

[Kopetz, 1997] presents a classification of real-time systems from different perspectives. The first two classifications are based on characteristics *outside* the computer system. The latter are based on characteristics *inside* the computer system.

Hard Real-Time System versus Soft Real-Time System

A hard real-time system is distinguished from a soft real-time system by the consequences a missed deadline entails.

Hard Real-Time System: In a hard real-time system a missed deadline can lead to a catastrophic result. Hard real-time systems have at least one hard deadline.

Soft Real-Time System: In contrast to hard real-time systems, the violation of timing requirements does not cause severe consequences, but it may degrade the quality of the intended service.

Fail-Safe versus Fail-Operational

Depending on the system behavior in case of a critical failure, real-time systems can be classified in:

Fail-Safe Real-Time Systems: In order to enable fail-safe behavior, one or more safe system states have to be quickly identified after the occurrence of a critical failure. It solely depends on the characteristic of the application if such a safe state can be identified. Fail-safe systems must achieve a high error-detection coverage, i. e., the probability of the detection of a previously occurred error must be high.

Fail-Operational Real-Time Systems: If a safe system state can not be identified, the real-time computer system must provide a minimal level of service to avoid a catastrophic event even in the case of a critical failure. This class consists of, e. g., all safety critical applications in the area of aviation.

It is important to notice that the required behavior is determined by the characteristics of the application and not by the real-time system itself.

Guaranteed-Response versus Best-Effort

Different from both classifications above, this one is in the scope of the real-time system design and not of the application. If the specified timing is validated even in the case of peak load and fault scenarios, we can speak of a system with *guaranteed response*. To achieve such a behavior, careful planning and extensive analysis are required during the design phase.
Best-effort systems are real-time systems that cannot give such an analytic response guarantee. As a consequence, it is difficult to substantiate that these systems operate correctly in rare-event scenarios.

Event-Triggered versus Time-Triggered

Considering the flow of physical time as a directed time line that extends from the past to the future, an *event* is every occurrence that happens at a cut of this timeline [Kopetz, 1993]. A *trigger* is an event that causes the start of some action like the execution of a task or the transmission of a message [Kopetz, 1997]. Depending on the used trigger mechanism, two different real-time system design approaches can be distinguished:

Event-Triggered Systems: In event-triggered systems, communication and processing activities are also initiated by events other than a clock tick. This events can originate either from inside or outside the computer system.

Time-Triggered Systems: In time-triggered systems, all starting points for communication and processing activities are derived from the progression of physical time. These points are predefined and known a priori.

2.2 The TTP/A Fieldbus Protocol

The TTP/A protocol is a *time-triggered protocol* for the communication among smart transducer nodes within a cluster [Kopetz et al., 2001]. A smart transducer is the integration of an analog or digital sensor/actuator element, a microcontroller, and a communication interface forming a mechatronic component [Schlatterbeck and Elmenreich, 2001].

A single active master node controls the communication and is responsible for establishing a common time base within the cluster. If the active master fails, a secondary master can take over. A TTP/A cluster can consist of up to 255 slave nodes. Each slave is assumed to have a unique identification number with a scope limited to the cluster.

2.2.1 Communication Principle

The communication in TTP/A is organized in rounds. A round consists of several frames, which are sequentially transmitted bytes of a single node. These frames are separated by an *interframe gap*. Communication rounds are fully independent from each other. Two successive rounds are separated by an *interround gap*. The length of these gaps are implementation specific.

A round is identified by its name. The structure and the properties of all rounds are known a priori, and they are common knowledge to all communication participants. The name of a round is transmitted within the so-called fireworks byte. The fireworks byte is sent by the master, indicates the start of a new round and is the global synchronization event for all nodes. There are eight valid fireworks bytes; thus, eight different rounds can be distinguished. To enhance dependability, the fireworks bytes have a Hamming distance of at least 4 [Kopetz et al, 2002].

Three different types of rounds are distinguished in the TTP/A protocol:

Master-Slave (MS) Round: MS rounds are used by the master to read data from the interface filesystem (IFS), to write data to the IFS, or to execute a specified IFS

record of a remote slave. Each MS round consists of two parts: the master-slave address (MSA) round and the master-slave data (MSD) round. The MS rounds implement the configuration planning as well as the diagnostic and management interface of a smart transducer [OMG, 2003]. Using the *baptize algorithm* [Elmenreich et al., 2002c], new nodes can be integrated into TTP/A networks by means of MS rounds.

Multi-Partner (MP) Round: Multi-partner rounds are periodic communication rounds that implement the real-time service of TTP/A. Multi-partner rounds are defined by so-called *round definition lists* (RODLs), which are specified a priori and are common knowledge to all nodes in the cluster. A RODL entry primarily consists of the operation type (send, receive, or execute), the location of the data in the IFS, the temporal position within this communication round, and the frame length. In TTP/A up to eight different RODLs can be defined. The active RODL is determined by the fireworks byte, which is sent by the master.

Broadcast Round: The broadcast round is used for addressing all baptized nodes in the cluster at once, e. g., to send a command that puts all nodes of a cluster into sleep mode. The layout of a broadcast round is the same as of MSA rounds, except that the address field contains the logical name 0.

It is recommended to generate an interleaved schedule for master-slave and multi-partner rounds. The periodic MP rounds are used for updating the real-time images while MS rounds can be used to read specific configuration data from a node's IFS or even dynamically reconfigure a RODL entry. The round sequence is stored in the *round sequence file* (ROSE).

2.2.2 Interface File System

The IFS is the core of the conceptual model of TTP/A [Kopetz et al, 2002]. Every TTP/A node has its own local IFS which acts as source and sink of all communication activities. The IFS of the whole cluster is formed by the local interface file systems of all nodes in it.

The local IFS of a single node can contain up to 64 files. Every file is able to hold at most 256 four-byte records. This results to a maximum storage of 64 kB. The layout of the IFS is statically defined but can vary from node to node.

File Structure

The distributed interface file system of TTP/A clusters is hierarchically structured. A record, consisting of four bytes, is the smallest addressable unit. The address is composed of the following fields:

$$< \textit{cluster name}, \textit{node name}, \textit{file name}, \textit{record name} >$$

Inside a cluster, the *cluster name* can be omitted; considering only a single node, a record can be definitely identified by two fields, the *file name* and the *record name*.

File Operations

One can distinguish three different operations on IFS files: *send*, *receive*, and *execute*. The send operation is used to transmit data from the local IFS to the common communication bus. On the contrary, the receive operation stores data from the bus into the local IFS. The execute operation is used to start time-triggered tasks whereby the type of the task and additional parameters can be specified.

Special Files

Besides configuration data and sensor readings, the local IFS of a node contains some special files:

Round Descriptor Lists (RODLs): As mentioned above, the RODL file contains the information about the structure of a particular round. In order to make a communication among the nodes in a cluster possible, different RODL definitions on different nodes must not conflict with each other (e. g., two send operations at the same time).

Configuration File: The configuration file holds the current *logical name* of the node. This file is used for the *baptize algorithm* [Elmenreich et al., 2002c]; therefore, it is mandatory for all nodes supporting plug and play.

Membership File: It is only useful for master nodes to implement the membership file. It consists of two independent membership vectors. While the first vector contains all slaves, which have sent a live-sign during the last MP round, the second membership vector contains all slaves, which have responded correctly to the most recent MS operation.

Round Sequence (ROSE) File: The ROSE file determines the sequence in which the rounds are scheduled by the master. Again, only the master uses this file and ordinary slaves do not need an implementation of the ROSE file.

Documentation File: This file is mandatory for all nodes in the cluster. It is used to identify the node. Therefore, the first and the second record contain the unique physical name of the node, a 8 byte (64 bit) integer.

For detailed information on the layout and the meaning of different IFS files as well as for a detailed description of RODL file generation see [Elmenreich et al., 2002a].

2.3 Wireless Communication

Wireless communication systems are applied in more and more application areas. While they are heavily used for telephone systems and computer networks like office networks or multimedia networks, the use of wireless communication systems for real-time systems is still in its beginnings.

This sections gives a short introduction to three commonly used classes of wireless links, *radio links*, *optical links*, and *acoustic links*, that might be capable for establishing wireless real-time networks [Losert and Obermaisser, 2001].

2.3.1 Radio Links

Radio waves are the most commonly used technology for wireless data transmission. Broadly spoken, the information is modulated onto a carrier signal whereby the frequency of the carrier determines important communication properties.

Due to the fact that the available frequency spectrum is limited and has to be shared among different services, radio devices have to be certified by a national or international authority, e. g., the Federal Communications Commission (FCC), to ensure that the use of this radio device does not disturb any other service.

Nowadays, there are a lot of different sophisticated wireless communication standards. Some of the most important are:

WLAN (IEEE 802.11): The IEEE 802.11 standard was released in 1997 and primarily defines the physical and the data-link layer of the OSI layer model. According to the standard defined in [IEEE, 1999], the frequency band at 2400.0–2483.5 MHz is used for data transmission. Data rates from 1 Mbit/sec (IEEE 802.11) up to 11 Mbit/sec (IEEE 802.11b) or 54 Mbit/sec (IEEE 802.11g) can be realized. WLAN uses differential binary phase shift keying (DBPSK) [see Rappaport, 1996, page 242] at data transmission speeds of 1 Mbit/sec or differential quadrature phase shift keying (DQPSK) [see Rappaport, 1996, page 249] at higher data rates for the modulation of the data.

The primarily addressed area of application is the wireless interconnection of workstation computers in the home office area or multimedia applications; thus, the use of WLAN IEEE 802.11 network cards in embedded computers is seldom feasible.

Bluetooth: In contrast to WLAN, Bluetooth aims at the appliance in the embedded sector. Therefore, a great advantage compared to IEEE 802.11 devices is the small and highly integrated shape of commercially available Bluetooth devices. As defined in [Bluetooth SIG, 2001], the Industrial, Scientific, Medicine (ISM) frequency band from 2400.0 – 2483.5 MHz is used. This frequency band is subdivided into 79 channels and the data is modulated via Gaussian frequency shift keying (GFSK).
Using a transmission power of 1 mW, Bluetooth is specified for data rates up to 1 Mbit/sec over a distance of about 10 meters.
Bluetooth supports the creation of so-called piconets. Each piconet consists of one master and up to seven slave nodes whereby a member of piconet A can also belong to other piconets, e. g., piconet B and/or piconet C. In this case, the three piconets are forming a so-called scatternet.

DECT: The abbreviation DECT stands for "Digital Enhanced Cordless Telecommunications" and was developed by the European Telecommunication Standards Institute (ETSI).
A DECT system consists of base station and several mobile members. DECT operates on a frequency band of 1880 – 1900 MHz with Gaussian Minimum Shift Keying (GMSK) modulation, and it is subdivided into 10 different carrier frequencies with a channel distance of 1.728 MHz. Medium access is done via time division multiple access (TDMA). Each carrier frequency is split up into 24 time slots (12 time slots for up-link and 12 time slots for down-link). Thus, DECT can handle up to 120 full duplex channels with only one radio receiver.
The maximum transmitter power of DECT systems is about 250 mW. This enables a in-house communication distance of 50 meters.
Due to the fact that DECT is primarily used for mobile phones, the dimensions of DECT devices are quite small. Therefore, it is suitable for the use in embedded systems.

Low Power Radio Devices: This item summarizes all devices, which are not covered by any standard but can be used to implement any communication protocol. Normally, this devices consist of an analog input, an analog output and a couple of control inputs for selecting the operation mode of the device.
Depending on the device, commonly frequency bands are 433 MHz, 868 MHz, or the 2.4 GHz ISM-band mentioned above. The reachable data rates vary from a few kbit/sec up to 1 Mbit/sec or more. It is important to note that not all modules are certified all over the world and thus national terms have to be considered when

using such devices. Due to the lack of any pre-implemented protocol stack, this devices offer the possibility of direct and instantaneous access to the communication medium, which enables the implementation of communication systems with low jitter and highly predictable behavior. On the other hand, the complexity of the needed software increases because the lower layers of the protocol have to be implemented too.

2.3.2 Optical Links

Wireless communication via optical links is used as strategy to overcome the worst drawback of radio based communication systems: the limited RF spectrum especially for high-bitrate data transmissions and the licensing needs for these frequency bands.

In general, one can distinguish two different fields of application when dealing with wireless optical links: free-space and indoor systems.

Free-Space Communication Systems

Lasers are a commonly used technology in this category of wireless optical communication systems. In order to establish a reliable communication, the adjustment of transmitter and receiver has to be highly accurate and both have to be in direct line-of-sight (LOS) which means that no obstacle stays there between.
Lasers enable the establishment of long range and high speed data links, e. g., up to 1 Gbit/sec over several kilometers.
Besides the limitation of point-to-point communication, laser communication links suffer from a critical limitation in practice: In contrast to wired fiber optic communication, where high data rates over long distances are achieved by a high transmission power, lasers pose a potential safety hazard in wireless applications, if they are operated incorrectly [Heatley and Neild, 1999].

Indoor Communication Systems

For indoor environments two different approaches can be used: line-of-sight and diffuse communication links.

Line-of-sight Links: As mentioned above, transmitter and receiver must always have mutual line of sight to achieve a reliable communication. This maximizes bandwidth but roaming on, e. g., different working places, is difficult. Two different approaches are outlined in [Heatley and Neild, 1999]: optical telepoint and IrDA systems.

Optical telepoint systems create communication cells, which may be shared by multiple users. A user who wishes to access a high capacity link enters the cell and is free to move within but not beyond. Several experimental developed telepoint systems reach cell sizes between 0.5 m to 10 m and data transmission speeds from 1 Gbit/sec to 10 Mbit/sec, respectively.
The Infrared Data Association (IrDA) standard on the other hand establishes a point-to-point communication without roaming support. This standard was developed in 1993 and is part of almost every portable computer and cellular phone nowadays.

Diffuse Links: The main design objective of this category of wireless optical communication systems is to facilitate roaming to a certain degree instead of maximizing bandwidth. Two properties are required to enable a diffuse communication link: First, the beams radiate over a very wide angle and are reflected off surfaces and objects in the vicinity. Second, optical receivers are constructed in a way that they can nearly capture all beams above the plane of the photo diode. Consequently, multiple signal paths reach the receiver and cause interferences that reduce the achievable bandwidth compared to LOS systems.

In order to avoid safety risks when dealing with optical transmitters in indoor environments, a safety standard has been established in which optical sources are classified in accordance with their total emitted power [Heatley and Neild, 1999].

2.3.3 Acoustic Links

Communication via acoustic links is comparable to communication via narrow band radio links, except that the information is modulated on an acoustic carrier instead of a radio carrier.
Compared to the technologies mentioned above, the propagation velocity of acoustic signals is very low. Additionally, the velocity is heavily influenced by environmental parameters like the basic communication medium (air, liquids, or solid material), the homogeneity of the communication medium, and the environmental temperature. Furthermore, acoustic links are very interference-prone to multipath fading and environmental noise. Therefore, the dependability of the communication medium is low.
Out of this considerations, acoustic links may not be the first choice for building wireless real-time communication systems.

2.4 Mobile Robot Positioning

[Leonard and Durrant-Whyte, 1991] summarize the challenging task of mobile robot navigation by three general questions: “Where am I?”, “Where am I going?”, and “How should I get there?”. Mobile robot positioning deals with the answering of the first question. [Rencken, 1993] defines the process of localization as: “Given a set of measurements and a set of features, what is the robot’s position?”.

2.4.1 Methods for Mobile Robot Positioning

[Borenstein et al., 1996] outlines four different methods that can be applied to determine the position of a mobile robot:

Odometry-based positioning: Odometry is the simplest, least expensive, and most widely used navigation method for mobile robots. The measurement of wheel rotation and steering orientation enables the generation of an estimation of the robot’s position. Due to the fact that odometry depends on the integration of incremental motion information, the estimation error accumulates over time. Nevertheless, odometry is used in almost every mobile robot because combined with other positioning methods, it can be used to enhance the system performance or the quality of the estimation results. In map-based positioning methods, e. g., the search space and the processing time can be reduced [Cox, 1991].

Active beacon navigation: Applications that require high reliability and highly accurate positioning results often use the active beacon navigation method. This method requires accurate mounting of active beacons in the environment; thus, it is only feasible in applications where the environment can be partially structured. The need of additional hardware and the accurateness of the mounting position can lead to high costs in installation and maintenance. Generally, there are two different methods for active beacon navigation: trilateration and triangulation. In trilateration the position is determined by distance measurements to three or more known beacon sources in the environment. The global positioning system (GPS) is an example of trilateration. For triangulation, three or more active transmitters are needed, which are mounted at known locations in the environment. A rotating sensor on the robot determines the angles to the beacon sources relative to the longitudinal axis of the vehicle. These values are used to estimate the position and the orientation of the robot.

Landmark navigation: Similar to active beacons, either natural or artificial landmarks are used to determine the absolute position of the robot according to a set of sighted

landmarks. [Borenstein et al., 1996] defines the terms *natural landmarks* and *artificial landmarks* as follows: *"Natural landmarks are those objects or features that are already in the environment and have a function other than robot navigation; artificial landmarks are specially designed objects or markers that need to be placed in the environment with the sole purpose of enabling robot navigation."* The main difficulty of this navigation method is the correct recognition of appropriate landmarks in the environment.
The typical position estimation process looks like as follows: First, sensory information is acquired. Thereafter, the data has to be preprocessed, landmarks have to be detected and segmented. The identified landmarks have to be matched against the a priori stored map. Out of this information, the position can be calculated, e. g., with triangulation methods.
A special and simplified derivation of landmark-based navigation is line-based navigation which is widely applied in industry. A line is used as continuous landmark, which the robot has to follow. In contrast to simplicity and high reliability, the main drawback of this approach is the limited and fixed operation range of the robot.

Map-based positioning: The most challenging but on the other hand most universal localization approach is map-based positioning or *"map matching"*. Similar to landmark navigation, the a priori stored or during the exploration process evolved map is matched against features detected by the robot's sensory units. Thus, the map matching process can be described as follows: Sensory information of the environment is acquired. By means of filtering and sensor fusion, data is reduced and relevant information is extracted. Using this data, a local map of the robot's environment is generated. This map is matched against an already known global representation of the environment. In case of successful matching, the robot's position can be estimated.
Map-based navigation methods have the main advantage that the environment has not to be modified, the new generated local maps can be used to update the already stored global map, and the accuracy of the estimation process can be enhanced. One drawback is the increasing need of sensory and processing power compared to the previous mentioned navigation methods and that the environment must provide enough distinguishable features, which are feasible for map generation.

2.4.2 Map Making and Sensor Fusion

It is almost impossible to derive a reliable and robust model of the environment out of single sensor information. A better and more accurate model of the world can be achieved by combining the information delivered from multiple sensors.

The term sensor fusion can be defined as:

> The combining of sensory data or data derived from sensory data such that the resulting information is in some sense better than would be possible when these sources where used individually.
> [Elmenreich, 2002, page 8]

[Bosse et al., 1996] and [Grossmann, 1998] outline the benefits that can be gained from the fusion of sensor data:

Robustness and reliability: Even in case of partial sensor system failures, useful information can be provided.

Extended spatial and temporal coverage: Increasing the number of applied sensors reduces the probability of missing relevant information in the spatial as well as the time domain.

Increased confidence: The correctness of individual sensors as well as of their measurement values can be monitored by multiple sensors covering the same domain.

Reduced ambiguity: More information about the same observed object reduces the probability of misinterpreting the measured values.

Robustness against interference: Different types of sensors can be used to tolerate different types of interferences and thereby enhancing the system robustness.

Improved resolution: Using different, independent measurements, a fused value can be generated with higher resolution than any of the used sensors can produce.

2.4.3 Geometric and Topological Map Matching

In general, there are two common representations for features in map-based positioning: geometric and topological maps. Geometric maps are similar to the human view of a map: Objects are stored according to their absolute geometric relationship. In topological maps on the contrary, relationships between features of the environment are stored rather than the absolute coordinates of the detected object. Due to the fact that these relationships are relative, the robot can obtain them from his current position without the knowledge of his absolute position. Therefore, unlike than in geometric maps, position estimation errors do not influence the quality of the generated topological map [Taylor, 1991].

Geometric Maps

The simplest way for modeling the robot's environment are occupancy grids. Occupancy grids are usually two-dimensional grids. Each grid cell represents the binary information of an absolute positioned area in the robot's working space: either the cell is occupied by an object or not.

[Moravec, 1988] introduces the *Certainty Grid* approach. Similar to occupancy grid-based representations, the robot's environment is subdivided into square cells. Each cell represents the probability of the existence of an object at this absolute position. To take the uncertainty of the sensor values into account and to handle faulty sensors that permanently submit incorrect values, [Elmenreich et al., 2002d] developed the *Robust Certainty Grid* algorithm. This approach uses the fusion of overlapping sensory information from multiple sensors to estimate the confidence of the individual sensors.

[Borenstein and Koren, 1991a] have introduced the *Histogram Grid* approach for representing obstacles in the environment. This approach is derived from the *Certainty Grid* described above and differs only in the way it is generated and updated. The computational complexity of the grid generation is reduced by updating only one histogram cell per sensor reading and not the whole probably effected area. A continuously and rapidly sampling while the robot is moving, results in a histogramic probability distribution, where higher certainty values are assigned to cells close to the location of the obstacle [Borenstein and Koren, 1991a]. Due to the lower computational complexity, this approach is well suited for real-time obstacle avoidance applications [Borenstein and Koren, 1990, 1991b].

Topological Maps

Instead of mapping observed obstacles into a global reference coordinate frame, map making procedures for topological maps extract essential relationships between observed features.

An important representative for localization methods based on topological maps is the class of roadmap-based localization approaches. Roadmaps are analogous to highway systems and have the properties *accessibility*, *connectivity*, and *departability* [Choset et al., 1997]. Accessibility guaranties that the robot is able to reach the "highway" from anywhere in the environment. Due to the property of connectivity, a collision free path to the vicinity of the goal can be found by traversing the roadmap. Finally, due to departability, a collision-free path from the roadmap to the goal can be constructed. Canny and Lin presented a path planning algorithm using this roadmap scheme [Canny and Lin, 1993].

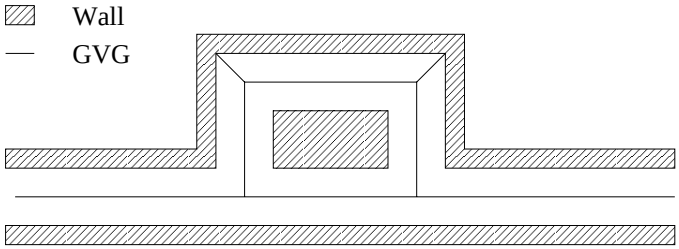


Figure 2.1: Example of a generalized Voronoi graph

Choset et al. use the *generalized Voronoi graph* (GVG) for the implementation of roadmaps. The GVG is an extension of the *generalized Voronoi diagram* (GVD), which is defined as the locus of points equidistant to two or more obstacles. The GVG is the one-dimensional set of points in *m* dimensions, equidistant to *m* obstacles. In the planar case GVG and GVD coincide [Choset et al., 1997]. Figure 2.1 shows an example for a GVG.

The wirless telegraph is not difficult to understand.
The ordinary telegraph is like a very long cat.
You pull the tail in New York, and it meows in Los Angeles.
The wireless is the same, only without the cat.

ALBERT EINSTEIN

Chapter 3

Wireless Communication Protocol

This chapter introduces the core component of this thesis, the wireless communication protocol. The design principles, functional properties, the clock synchronization approach, and the monitoring and configuration support of the protocol are explained in the following.

3.1 Design Principles

The protocol design was guided by several requirements inherently arising out of the intended area of application. This includes the establishment of a real-time communication and closely associated therewith, a reliable mechanism for clock synchronization that also copes with the aggravating circumstances caused by the use of wireless communication.
For fieldbus systems, respectively for the transmission of sensory data in general, message lengths are mostly very short; therefore, data efficiency for short telegrams is a very important design guideline. Also the detection or, if feasible, the correction of transmission errors is a crucial demand.
To enable the on-line configuration of important system parameters or to retrieve information about the internals of the system, monitoring and configuration have to be supported by the protocol.
A detailed description of the design principles is given in the following:

Time-Triggered Communication: Most areas of application covered by this communication protocol have stringent timing requirements, e. g., when several smart transducers or whole fieldbus clusters are interconnected to a distributed control application.

In this context the time-triggered approach offers the advantage that message jitter and communication delays are minimized. These are important properties when dealing with control applications. Additionally, communication activities and task execution are highly predictable hence they are only derived from the progression of physical time. This reduces the complexity of the application software that is based on the communication protocol.

Fault-Tolerant Clock Synchronization: The quality of service of time-triggered systems, especially message jitter, and associated therewith, the maximum reachable communication speed, is tightly coupled with the quality of the synchronized time base.

A wireless communication system complicates the construction of a high quality and reliable synchronized timebase, because the applied algorithms have to take into account that the full connection of the communication cluster cannot be guaranteed all the time. Therefore, a clock synchronization algorithm is needed that enables a sufficient synchronization of all communication participants even in the case of a transient loss of the connection to a bounded set of communication partners.

Data Efficiency: When dealing with smart transducer communication, the message length of the transmitted data packets is usually very short. The transmitted data is normally a pre-processed digital representation of possibly analog sensory information, e. g., transformation of a temperature measurement to a standardized 8 bit representation.

Thus, it is important to keep the communication overhead as small as possible to achieve a efficient communication. The use of the time-triggered paradigm supports the protocol efficiency because the transmission of message parameters like sender identification, message length, message priority, etc. is implicit due to the predefined communication schedule.

Error Detection: Depending on the used technologies, wireless communication channels are exposed to certain interferences by the environment. When applying optical communication techniques, every kind of obstacle may cause a reduction of the communication quality or even a complete data loss. If radio links are used, data transmission is disturbed particularly by too long transmission ranges or impenetrable walls. Therefore, methods for error detection or error correction are crucial for wireless communication systems.

Monitoring and Maintenance Support: In order to configure system parameters online, e. g., in systems that cause high costs in case of a shut-down or where a temporary shut-down is not possible at all, as well as to perform on-line diagnosis to detect erroneous system states, the support of maintenance and monitoring at

protocol level is necessary. Due to the limited interfaces a distributed embedded computer system provides to the outside, monitoring and debugging is much more complex than in systems consisting e. g., solely of PC applications.
Thus, the communication protocol itself has to provide means for monitoring the internal state of the system but without generating an indeterministic system behavior. Particularly, the real-time behavior must not be violated.

3.2 Functional Description

The communication protocol introduced in this thesis is based on the time-triggered paradigm. The main design objective was predominantly to establish a wireless real-time communication among several, possibly mobile, TTP/A clusters and to provide the possibility for monitoring the distributed system from outside (see Figure 3.1). A short introduction to TTP/A can be found in section 2.2.

The communication is controlled by a single dedicated master node. This master is responsible for establishing a synchronized time base among all nodes within the cluster. The notion of a global time on which all communication participants agree, is absolute essential for distributed systems, in which the start of task execution and all communication activities is only derived by the progression of real time.

To overcome the main drawback of a single master solution – the problem of single point of failure – the design of this protocol tolerates certain failures of the master node. This is very important in respect of the wireless communication medium because transient link failures are not exceptionally there. The duration of the tolerated master failures mostly depends on the required precision of the synchronized time base (see section 3.3).

An eight-byte long time format is used to represent the global time. This time format, introduced in [Kopetz et al, 2002], is based on GPS time and is split up into two parts. A 40-bit value defines a horizon of 2^{40} seconds of this representation; that is more than 10 000 years. The other part consists of a 24-bit value that supports a possible granularity of 2^{-24} seconds; that is about 60 nanoseconds.

In order to detect a possible corruption of the transmitted data, the protocol includes some error detection mechanisms. On byte level the transmitted data contains a parity bit for error detection. Several sequentially transmitted bytes form a frame. Every frame is concluded by a check byte, e. g., an eight-bit checksum.

In addition to the real-time service, this protocol also supports monitoring of all nodes within the communication cluster without any disturbance of the real-time service or the cause of a so-called “Probe Effect” [McDowell and Helmbold, 1989].

The master provides the diagnostic and monitoring interface as well as the configura-

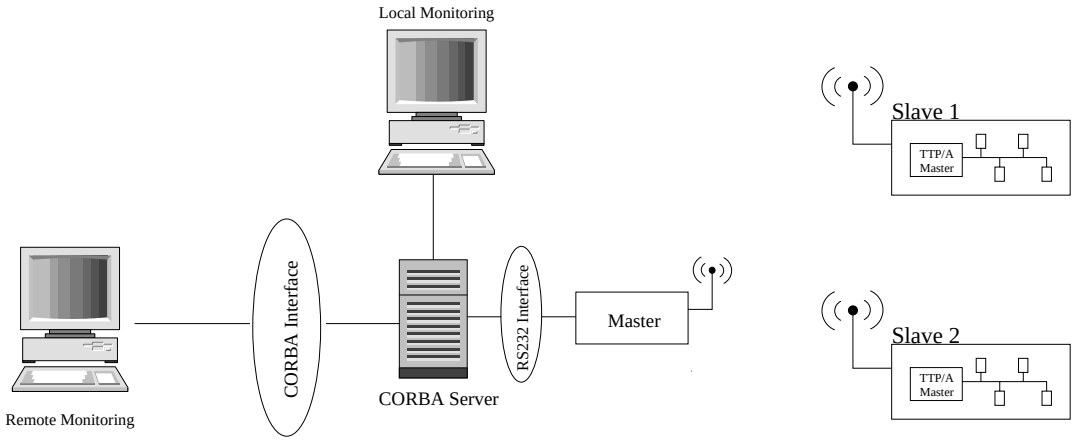


Figure 3.1: Wireless interconnected TTP/A clusters

tion planning interface [Kopetz et al, 2002] to the outside. These interfaces can be used, e. g., with the *TTP/A Gateway* [Krywult and Steiner, 2004] to monitor and alter the internal state of the wireless real-time system. Additionally, these interfaces enable the configuration of the TTP/A clusters controlled by the wireless slaves.

Furthermore, it is assumed that each slave has assigned a unique identifier, a 16-bit address. Thus, theoretically $(2^{16}) - 1 = 65535$ slaves (the address 0x0000 is reserved for the master) could be addressed. In practice, especially when fieldbus networks are interconnected via the wireless protocol, it is more useful to regard the first eight-bit as unique slave address (or fieldbus cluster address in that case) and the second eight-bit as a in depth reference to a sub-node or sub-component of the addressed slave. Thus, $(2^8) - 1 = 255$ different slaves and $(2^8) - 1 = 255$ sub-components at each slave can be addressed.

It is important to note that the addressing of slaves is only needed for the maintenance service. All communication activities of the real-time service are specified a priori and therefore no explicit addressing is needed.

3.2.1 Communication Schedule

The protocol uses a round-based communication schedule. Each round consists of one or more frames. A frame is a sequence of bytes transmitted from a single node. Any two frames are separated by a so-called interframe gap, a duration without communication. Subsequent rounds are separated by an interround gap (see figure 3.2). The length of these gaps are parameters that mostly depend on the reached synchronization quality.

The structure and duration of every round is static and defined a priori. Thus, TDMA lends itself for bus arbitration. The complete set of predefined communication rounds has

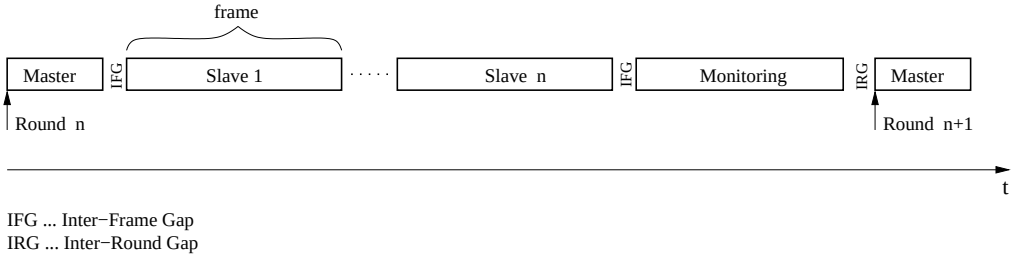


Figure 3.2: Typical communication round layout

to be common knowledge for all slaves in the system. The master selects the active round by transmitting a *round identifier* at the start of each new round. The round identifier has a similar role as the fireworks byte of the TTP/A protocol introduced in [Kopetz et al, 2002].

If the master activates another communication schedule that is equally structured as the preceding one, i.e. the amount and ownership of all frames is equal and the round differs only in the meaning of the transmitted data, any ambiguities and misinterpretations can be avoided. Therefore, it is important to inform also slaves that are not connected to the master. To accomplish this, the current round identifier should be included in every slave frame.
If the structure of the new round differs from the preceding one, a consensus protocol has to be established to guarantee that no communication errors occur.

We distinguish between three different frame types:

Master Frame: This frame is transmitted by the master node. The first byte indicates the start of a new round and acts as the global synchronization point for all slaves. The master frame contains the *round identifier*, the next *monitoring request* and two *status/command* bytes.

Slave Frame: The slave frames are used to implement the real-time service. Depending on the individual needs, every slave can own one or more time slots for sending data frames. A configuration with some slaves acting as a kind of “passive members” without sending anything is also possible.

Monitoring Frame: Different from the master and slave frame, the monitoring frame is not statically assigned to a particular node. Depending on the type of the request, this frame is either transmitted by the master or by one of the slaves.

3.2.2 Startup Procedure

Due to the different functionalities, the protocol startup has to be considered separately for the master and for slave nodes.

Startup Procedure of the Master Node

Figure 3.3 shows the startup procedure of the master node: After a power-on or a reset of the node, the protocol starts in *Idle* state where it resides inactively until a certain amount of time, the *idle_timeout*, is elapsed. The idle_timeout is used to ensure that all previously activated slaves have stopped their communication, to provide a safe startup without disturbing any ongoing communication. To ensure this behavior, the idle_timeout is calculated as shown in equation (3.1). C^i_{init} is the initial confidence value (see 3.3.3) of node *i*, the number of slaves in the system is denoted by n_s, and variable K is the number of defined communication schedules.

$$t_{time\_out} \geq \max_{0 \leq i < n_s} (C^i_{init}) \cdot \max_{0 \leq k < K} ({t_{round}}^k) \tag{3.1}$$

Unlike to the normal operation, the master transmits a special frame, the so-called *startup_frame* (see Figure 3.4(a)), at the beginning of each round of the *Startup* phase. The startup_frame has the same length as the *request_frame* and differs only in the meaning of the transmitted bytes. The eight-byte time value that is transmitted via the startup phase can be used to synchronize every slave to an external absolute time reference.

During the startup phase, responses from the slaves consist of initial configuration data

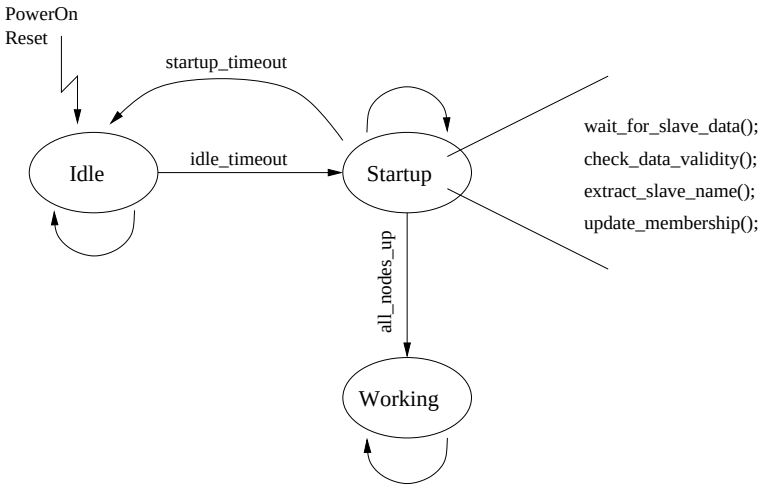
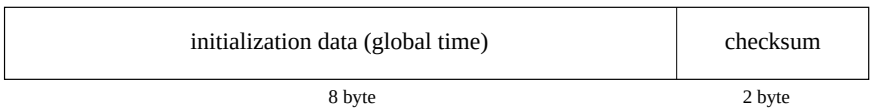
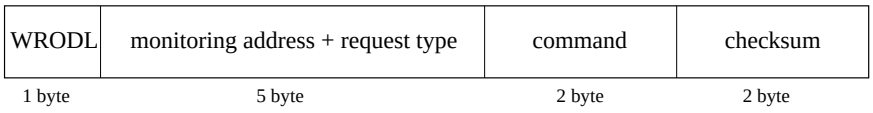


Figure 3.3: Protocol startup at master node



(a) Layout of the startup_frame



(b) Layout of the request_frame

Figure 3.4: Different types of master frames

like the membership vector if the slave controls a TTP/A fieldbus cluster. This data is checked for validity, i. e., the temporal properties of the received data and the checksum are proofed. The master uses the reception time and the static *wireless round descriptor list* (WRODL), to extract the sender and to update its membership table. If all expected nodes have correctly responded inside a predefined time interval, the startup_timeout, the master switches to the *Working* state. Otherwise it returns to *Idle* state.

In the *Working* state, so-called request_frames (see Figure 3.4(b)) are transmitted. The first byte selects the active communication schedule out of a set of a priori defined rounds. The following 5 bytes contain the address and type of the monitoring request in this round. Byte 7 and 8 are reserved to send certain commands to all attached slaves. Such commands might be *sleep*, *resynchronize*, *abort communication*, etc. Similar to the startup_frame the last two bytes are used to identify the frame type and to build a checksum over the data bytes.

Startup Procedure of Slave Nodes

The startup procedure of a slave node is shown in Figure 3.5. The slave starts in *Idle* state after power-on or reset; there, it inactively waits until some data from the master is received. If a correct master frame has been received, the slave enters the *Ignore Data* state where all incoming data is ignored until the time of a complete round has elapsed. Thereafter, a receive window is opened and a new master frame is expected. This should prevent the slave from misleadingly interpreting data from other nodes as a master frame, which would lead to a completely wrong synchronization.

In *Startup* state, the slave only considers master frames, extracts the frame type and

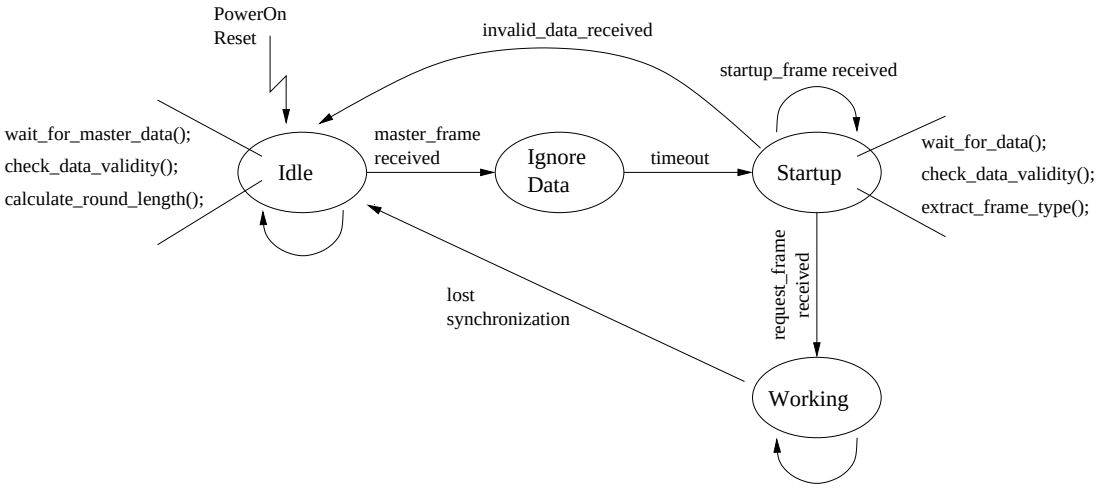


Figure 3.5: Protocol startup at a slave node

uses its own slave frame for the transmission of initial configuration data. As long as startup_frames are received, the slave resides in this state. A missing or invalid master frame immediately returns the slave back to *Idle* state.

If the master starts sending request_frames, all synchronized slave nodes proceed in *Working* state, providing the real-time service as well as the configuration and maintenance service. In case of a definite loss of synchronization, the slave deactivates itself and returns to *Idle* state (see section 3.3.3).

3.3 Fault-Tolerant Clock Synchronization

In time-triggered systems, task execution and all communication activities are derived from the progression of real time and the therefrom arising events. Thus, in a distributed time-triggered system, a synchronized timebase is a crucial requirement to enable a reliable system behavior.

In wireless networks, especially when dealing with mobile nodes, e. g., mobile robots, the toleration of transient faults of the synchronization unit, in our case the single master node, is a determining factor. It can not be assured that a permanent connection between all slaves and the master persists. Consider for instance a scenario where a mobile robot, acting as a wireless slave node, exceeds the maximum transmission distance. There should be the possibility that this robot can react on this situation and return into the operational transmission range of the wireless module without the need to abort any ongoing communication.

3.3.1 Synchronization Approach

Any frame in a round, except the monitoring frame, can be used for synchronization. The reception of the first byte of the master frame acts as global synchronization event for all slaves in the system. Nevertheless, every correct synchronized node can overtake the role of a “second-rate time master” for a subset of other slaves to keep them synchronized. Figure 3.6 depicts the internal protocol execution of a slave node after a successful protocol startup.

The correct function of the synchronization algorithm depends on the suggestion that the master node possesses a reliable clock with a sufficient high quality. The used synchronization algorithm consists of a two-staged approach, whereby stage two is omitted if stage one works successful:

Stage 1: Synchronization by the master node.

Stage 2: Synchronization by another slave node with a better clock.

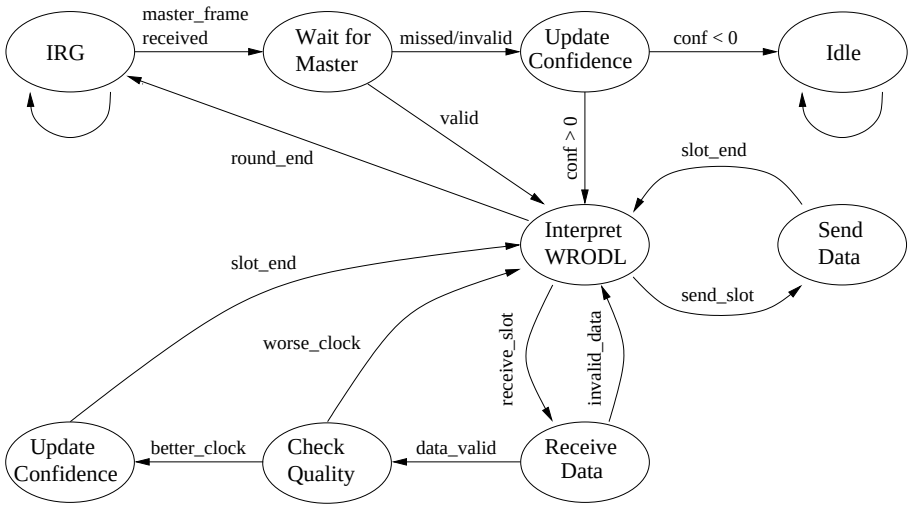


Figure 3.6: Description of the internal processes of the Working State

Stage 1 - Synchronization by the Master

In the normal case all slave nodes are synchronized by the master. Therefore, the point in time, the first byte of the master frame has been received, serves as the synchronization event for each slave node. Due to the statically defined communication schedule, every node knows in advance when the next master frame should start. Thus,

the difference Δt between the expected and the actual arrival time of the first byte of the master frame can be used to adjust the slaves clock (see Figure 3.7).

Because of the very limited range of most wireless modules, especially in the sector of low power radio devices and the high propagation speed of radio waves, the communication delay caused by the wireless transmission can be neglected. Therefore, the overall communication delay resulting from different parameters like code execution time, wireless module activation, etc. is accounted as constant. In this case, the clock synchronization reduces to a simple adjustment of the individual node's time value (3.2).

$$t_{synced} = t_{old} + \Delta t \tag{3.2}$$

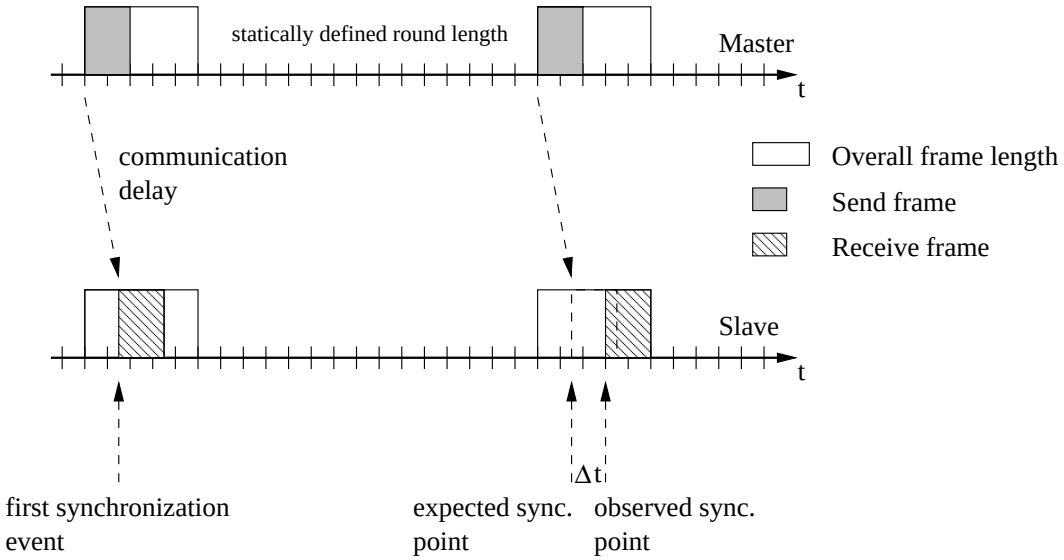


Figure 3.7: Normal clock synchronization by the master node

Stage 2 - Synchronization by other Slave Node

The protocol is designed to tolerate several subsequent omissions of master frames without forcing a slave to stop its communication service. As long as the temporal partitioning of the communication medium is not endangered, temporary unsynchronized slaves are allowed to proceed. The self-deactivation of improper synchronized nodes is described in section 3.3.3.

In order to avoid the onward synchronization error and as a result to avoid the self-deactivation of a slave, the clock synchronization algorithm enables the slave to synchronize its clock without assistance of the master. Figure 3.8 shows an accordant situation where slave 2 loses communication to the master and hence is synchronized by slave 1.

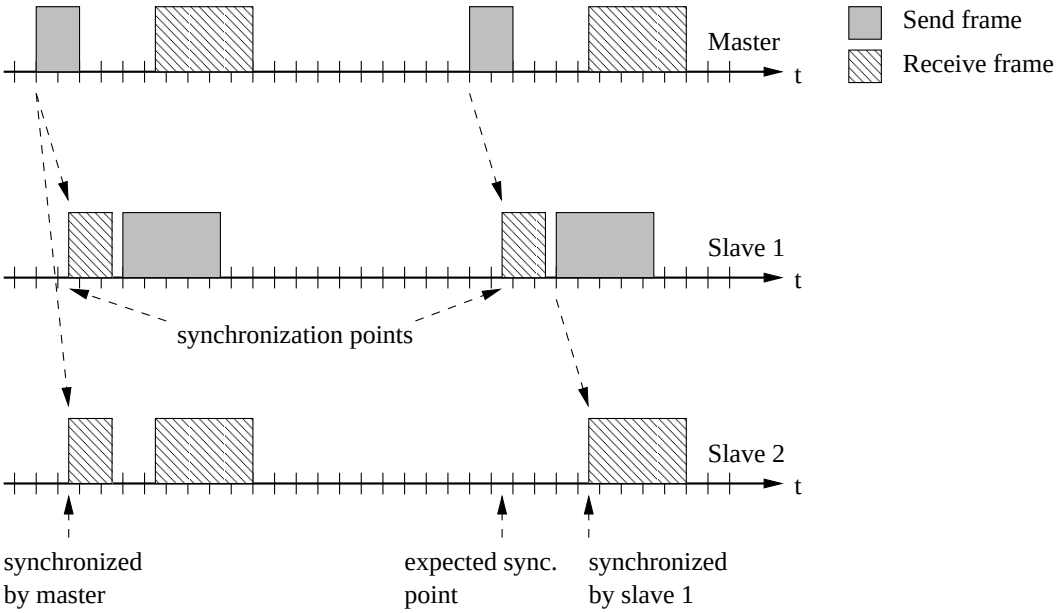


Figure 3.8: Clock synchronization in case of communication loss to master node

Again, due to the statical defined communication rounds, every node knows a priori when data from other slaves should be received. This information is used to calculate the deviation of the expected and the actual arrival time of the frame.

To decide, if a received slave frame should be used for synchronization, two confidence values are assigned to each slave. The *initial confidence* value denotes the quality of the node's clock. The *actual confidence* value on the other hand represents the quality of synchronization. The initial confidence value is a constant parameter, whereby the actual confidence value changes according to the number of communication rounds without synchronization.

After the reception of a slave frame, each node k updates its actual confidence value C^k. The updated confidence $C^{k\prime}$ of node k, which possesses an initial confidence C^k_{init}, is calculated according to the received frame from node s with the parameters C^s and C^s_{init}:

$$C^{k\prime} = \left\lfloor \frac{C^s}{C^s_{init}} \cdot C^k_{init} \right\rfloor \tag{3.3}$$

If $C^{k\prime} > C^k$, node k adjusts its time according to the time deviation and alters its actual confidence value as defined in equation (3.3). Equation (3.3) ensures that the actual confidence of a node remains less or equal than the node's initial confidence and that the adjustment takes the quality differences of both clocks into account.

3.3.2 Quality of Synchronization

The quality of the global time, the distributed time base shared by all nodes of the system, is an essential property of a time-triggered system. Standard measures for the quality of an ensemble of clocks are *accuracy* and *precision* [Kopetz, 1997].

The accuracy of clock k at micro-tick i is defined as:

$$accuracy_i^k = \max\{offset_i^k\} \tag{3.4}$$

The offset denotes the time difference between two respective micro-ticks of clock k in respect to the reference clock. Given a *period of interest,* this maximum offset over all micro-ticks in this period is called $accuracy^k$ of clock k.

$$\Pi_i = \max_{\forall 1 \leq j,k \leq n} \{offset_i^{jk}\} \tag{3.5}$$

Π_i is called the precision of an ensemble of clocks at micro-tick i. The maximum of Π_i over a *period of interest* is called precision Π of the ensemble. An externally synchronized ensemble of clocks with an accuracy A is also internally synchronized with a precision of at most 2A [Kopetz, 1997].

The offset of a clock k to a reference clock depends on the drift rate ρ^k of the clock. As mentioned above, the external synchronization event for all clocks in the ensemble is the first byte of the master frame. Due to the fact that the period between two master frames depends on the round length, the synchronization interval is equal to the round length of the active communication round. Thus, the maximum round length t_{round_max} and ρ^k determine the worst case accuracy of clock k.

$$A_{worst\_case}^k = \rho^k \cdot t_{round\_max} \tag{3.6}$$

Considering the ensemble of slave nodes being externally synchronized by the master, the worst case precision of the ensemble is defined as:

$$\Pi_{worst\_case} = 2 \cdot \rho^k \cdot t_{round\_max} \tag{3.7}$$

To guarantee that no slave in the cluster misses the start of any frame, the length of the interframe gap t_{IFG} must be at least the worst case precision.

$$\frac{\Pi_{worst\_case}}{t_{IFG}} = k \qquad k \geq 1 \tag{3.8}$$

Out of (3.7) and (3.8) the lower bound of the inter-frame gap t_{IFG} is given as:

$$t_{IFG_{min}} = \frac{2 \cdot \rho \cdot t_{round}}{k - 2 \cdot \rho \cdot (n+2)} \qquad k \geq 1 \tag{3.9}$$

t_{round} is the overall time needed for the transmission of the master frame, the monitoring frame, and n slave frames.

3.3.3 Self-Deactivation of Inexactly Synchronized Nodes

In systems with a TDMA bus arbitration scheme, bus access is solely controlled by the progression of time. Thus, timing violations of the communication scheme have to be avoided in any case to prevent collisions on the shared communication medium.

To adhere to this condition, we have established the notion of a *time confidence value* on each node. The time confidence value denotes an estimation of the precision in the worst case, i.e., the node has drifted with the maximum expected drift rate. This estimation determines the maximum permitted time between two subsequent synchronization events that will not cause a violation of the communication scheme.

The maximum synchronization interval t_{sync} differs from node to node. It depends on the globally defined tolerance gap between two subsequent frames t_{IFG} and the tolerance gap between two subsequent rounds t_{IRG} on the one hand, and on the drift rate ρ of the individual node's clock on the other hand. Thus, considering $t_{IFG} = t_{IRG}$ it is defined as:

$$t_{sync} = \frac{t_{IFG}}{2 \cdot \rho} \tag{3.10}$$

The initial time confidence value is an integer expression of this estimation and results from:

$$C_{init} = \left\lfloor \frac{t_{sync}}{t_{round\_max}} \right\rfloor \tag{3.11}$$

The self-deactivation process is shown in Figure 3.6 where the internals of the *Working* state of the slave nodes are described. A slave node decrements its actual confidence value in every round it fails to synchronize with the master. To ensure that it does not violate the communication scheme, it is only allowed to transmit data over the shared communication medium if the confidence value is greater than zero. Otherwise, the node returns to *Idle* state and hence communication is aborted. Thus, the initial time confidence value is a quality measure for the node's clock and determines the number of tolerated rounds without synchronization.

After being correctly synchronized by the master, the actual confidence value is reseted to the initial value. In case of synchronization by another node, the actual confidence value is calculated as shown in equation (3.3). It also guarantees that the initial confidence value is not excelled.

It is important to note that the delayed self-deactivation mechanism applies only to the real-time service of the communication protocol. Every node has to abort monitoring or configuration requests in case of a communication failure with the master immediately because the validity of the actual request is not guaranteed any longer.

3.3.4 Fault Tolerance and Error Recovery

This section states some assumptions about the behavior of the fault-tolerant clock synchronization approach in the case of certain types of faults. The therefore used formal description, the fault hypothesis, is defined by Kopetz [1997, page 73] as:
"The fault hypothesis is [...] a statement about the assumptions that relate to the type and the frequency of faults that the computer system is supposed to handle".

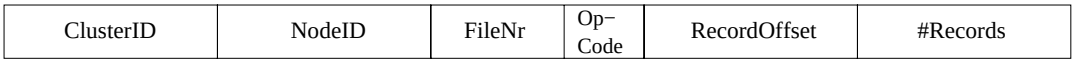
For the here introduced clock synchronization algorithm, following assumptions can be made [Huber and Elmenreich, 2004]:

1. Permanent fail-silent behavior of the slave nodes is tolerated.
2. Transient fail-silent behavior of the master is tolerated, if the down time of the master node is less than the maximum synchronization interval (see section 3.3.3).
3. Transient failures of all links in the communication channel are allowed, if the time the links are broken is less than the maximum synchronization interval.
4. Permanent failure of $\lfloor \frac{n_s}{2} \rfloor$ (n_s is the number of slave nodes in the system) links in the communication channel is tolerated. If we assume the probability of a link failure with 1%, the probability of the occurrence of $\lfloor \frac{n_s}{2} \rfloor$ *independent* link failures is $0.01^{\lfloor \frac{n_s}{2} \rfloor}$.
5. With an absolute reference time at the master, the reintegration of the master after a transient failure is always possible.
6. Without an absolute reference time all slaves have noticed a permanent master failure and on this account safely stop their service not later than $\max_i (C_{init}) \cdot t_{round\_max}$ $(0 \leq i \leq n_s)$, which is known in advance. A safe restart of the system can then be performed.

3.4 Runtime Configuration and Monitoring

Debugging and analyzing the incorrect behavior of a distributed system is a very challenging task, where *monitoring* is often used to locate the cause of the incorrect behavior. Monitoring allows to gather runtime information that cannot be obtained by a static analysis of the source code [Joyce et al., 1987].

A supplementary integration of monitoring functionality into a distributed real-time system may change its behavior. This indeterministic effect is called *"Probe Effect"* [Gait, 1986]. Applied to software, often the term *"Heisenberg Uncertainty"* [LeDoux and



ClusterID	system wide unique identifier	(8 bit)
NodeID	cluster wide unique identifier	(8 bit)
FileNr	adressed file number	(6 bit)
Op−Code	operation mode	(2 bit)
RecordOffset	adressed record number (1 record = 4 bytes)	(8 bit)
#Records	number of adressed records	(8 bit)

Figure 3.9: Address format used in master frame

Parker, Jr., 1985] is used. Consider, e. g., a software monitor that is usually implemented as an additional software task. The resources needed by the software monitor (processor time, memory, etc.) may change the whole system behavior, so that a rare scenario may never occur. Thus, it is of utmost importance that monitoring capabilities are part of the system design.

To cope with this requirements, a static amount of bandwidth, the so called monitoring frame described in section 3.2.1, is used to provide a dynamic configuration service as well as a monitoring service.

The configuration and monitoring service is controlled by the master. The master frame, transmitted at the beginning of each round, contains among other information a 5-byte address and the mode of operation (two bits included in the slave address). The address format (see figure 3.9) aims at the use of the *interface file system* (IFS) of the TTP/A protocol as the underlying source and sink of all communication activities.

Depending on the operation type, the monitoring frame is used either by the slave (monitoring mode) or by the master (configuration mode). Due to the fact that the monitoring frame is scheduled a priori, the non-disturbance of the real-time communication service can be guaranteed. If a configuration or monitoring request exceeds the static size of the monitoring frame, the request is proceeded in the monitoring frame of the succeeding rounds. It is important to note that the master as well as the slave have the possibility to release and abort an ongoing configuration or monitoring request in certain circumstances. This is especially important if the connection between the master and the currently monitored slave has been lost.

If all nodes, the master and all slaves, keep their protocol and node setup in a structured writable memory space like the IFS of TTP/A, the configuration mode can be used to adopt node properties (e. g., sensor/actuator settings, self-describing information) as well as protocol properties (e. g., communication scheme) [Peti et al., 2002].

Der Unterschied zwischen Theorie und Praxis
ist in der Praxis weit höher als in der Theorie

ERNST FERSTL

Chapter 4

Case Study

This chapter describes the design and implementation of a wireless communication cluster, which is used to show the practical relevance of the introduced communication protocol. The core of the case study is an autonomous mobile robot, the so-called *Smart Car*. This demonstrator was developed during the last years through the effort of several people. It was predominantly developed as a demonstrator for the *Dependable Systems of Systems* project (DSoS, European Research Project IST-1999-11585) [Elmenreich et al., 2002b].

4.1 Problem Description

Based on preceding work on the Smart Car (see Figure 4.1) done by [Elmenreich, 2002, Peti, 2001, Schneider, 2001], the Smart Car is able to navigate autonomously through a course with static obstacles. The environment of the robot is perceived by multiple infrared and ultrasonic sensors, which are fused to obtain a more reliable representation of the environment. This information is used by a path planning algorithm to select the least dangerous direction to achieve a safe path through the obstacles.

The main focus of this case study is to implement the presented wireless communication protocol on commercial-off-the-shelf microcontroller. The usage of a mobile robot shows that the communication protocol can deal with difficulties arising out of the use of a wireless communication medium. The Smart Car has been chosen to demonstrate the usability of the wireless real-time communication service as well as the wireless monitoring and configuration service in a practical application.

Via the real-time service, the actual sensory information of the Smart Car is transmitted. Using this information, a temporary valid *local map* of the Smart Car's environment is generated, e. g., to reconstruct and verify the navigation decision of the autonomous

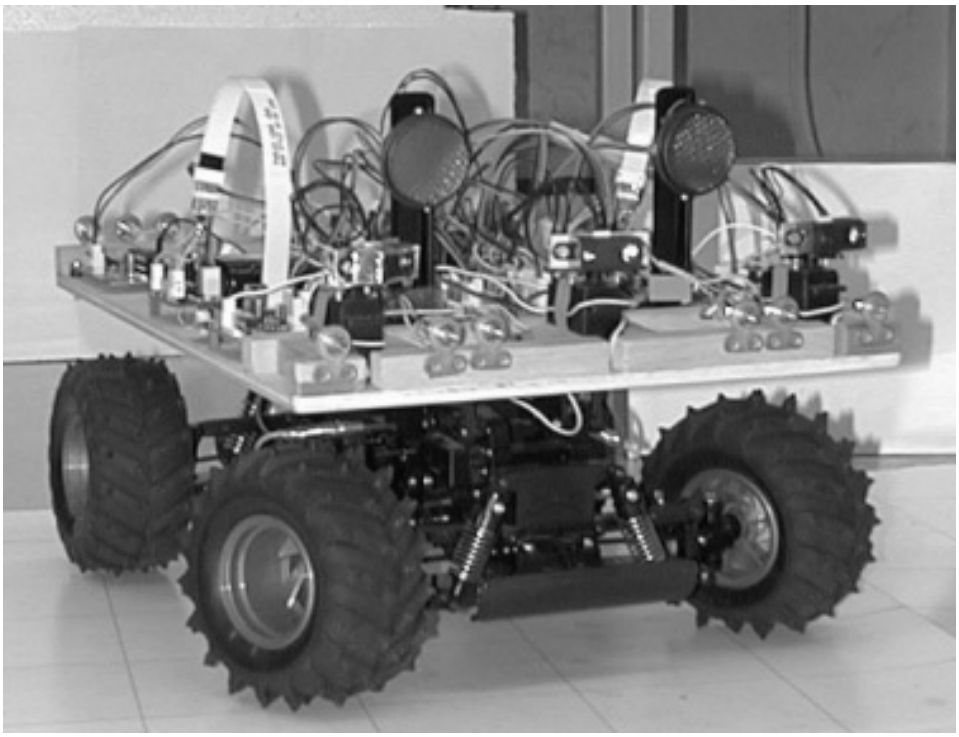


Figure 4.1: Smart Car – the core of the case study

robot or to assemble the pieces to a global map of the environment. The latter task is realized by a software described in Section 4.6.

Via the monitoring tool *TTP/A Gateway* [Krywult and Steiner, 2004], the monitoring and configuration interface provided by the wireless protocol is utilized to monitor or update current parameters of the Smart Car remotely. These parameters are, e. g., the current distance to an obstacle, the steering or speed parameters of the Smart Car, or the risk of obstacle collision while moving in the chosen direction.

4.2 Case Study Setup

The setup of the case study consists of the components shown in Figure 4.2.
The wireless network is built up on three nodes: the wireless master and two slave nodes. The wireless master concurrently acts as a kind of gateway between the wireless network and applications outside (e. g., map making software or monitoring and configuration software). The interconnection of the master with the outside, in our case a PC, is realized via a RS232 interface.

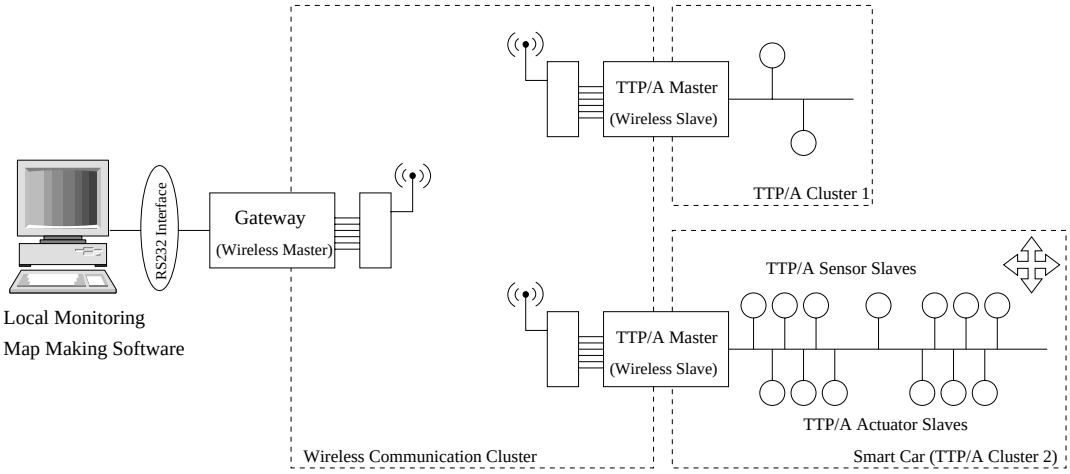


Figure 4.2: Case study setup

The wireless slave nodes have to concurrently provide two different functionalities: On the one hand, they are wireless slave nodes and have to provide the specified functionality offered by the wireless communication protocol. On the other hand, each of them acts as TTP/A master and is therefore responsible for the correct function of a self-contained TTP/A fieldbus cluster.

All nodes, the master as well as the slaves, are connected via a serial interface to a so-called *wireless module*. The wireless module is the hardware component that is responsible for encoding and transmitting the data over the wireless communication medium. The design of this hardware component is presented in Section 4.3.1.

The TTP/A cluster 1 shown in Figure 4.2 is a dummy cluster that only consists of two slaves nodes. This cluster was used for testing and developing purposes (e. g., for the remote monitoring of particular IFS files) as well as verifying the correct functionality of the proposed clock synchronization algorithm.

TTP/A cluster 2, representing the Smart Car, is a quite complex fieldbus cluster that connects several different types of slaves. A more detailed explanation of the hardware components of the Smart Car is given in Section 4.3.2.

4.3 Hardware Components

This section describes the hardware components that were employed and particularly refined during the implementation of this case study.

Basically, it can be distinguished between two different sets of hardware components: The hardware components used for the wireless communication and those for building

the Smart Car demonstrator. This section places emphasis on the hardware developed for the wireless communication. The demonstrator's nodes and their electro-mechanical sensor or actuator elements are only shortly described. A detailed description of the building blocks of the Smart Car can be found in [Elmenreich, 2002, Schneider, 2001].

4.3.1 Hardware for Wireless Communication

The wireless network comprises of two different types of nodes: the master node and the slave node. The hardware modules have not been newly developed, but they have been taken from existing applications [Elmenreich, 2002, Peti, 2001, Schneider, 2001]. Figure 4.3 shows a slave node (the hardware module on the left) already connected to a wireless module. The master node is constructed identically and differs only in the used microcontroller.

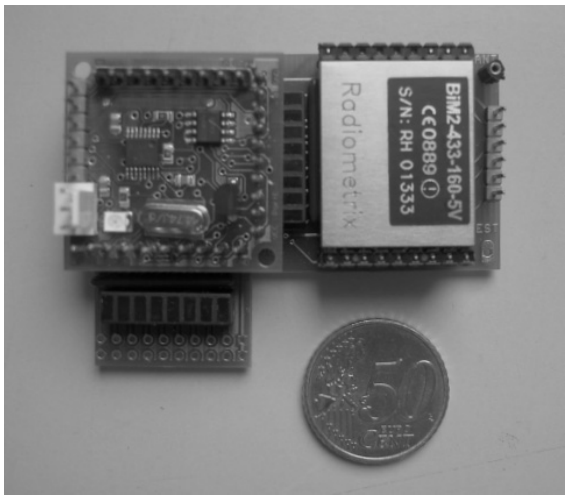


Figure 4.3: Wireless module connected to slave node

Master Node: The core component is an Atmel ATmega128 microcontroller. This is a representative of the 8-bit low power microcontroller family based on the Atmel AVR RISC architecture. The controller is clocked by a 14.7456 MHz quartz oscillator, which enables the use of standard baud rates for UART communication at high quality.

The Atmel ATmega128 features 128 kByte of in-system programmable flash, 4 kByte of EEPROM, and 4 kByte SRAM. Table 4.1 summarizes some features of those microcontrollers.

An important feature of this microcontroller is the availability of two hardware UARTs, which enable serial communication without producing much CPU load.

This is needed because of the master's gateway functionality between the wireless network and the outside.
Since the node is used in other TTP/A applications, it is also equipped with an ISO-K bus driver, which enables the access of, e. g., a TTP/A bus. This feature, however, is not used in this application.

Slave Node: As mentioned above, the hardware of the slave node differs from the master only in the used microcontroller: The slave node is built up on an Atmel ATmega128 microcontroller, which is driven in ATmega103 compatibility mode. This constraint is forced by the currently used TTP/A implementation, which is subject to some restrictions in terms of register usage. Therefore, it requires a certain version of the AVR-GCC assembler which cannot handle microcontrollers of type ATmega128. A description of the development environment is given in Section 4.4.1.

		Master	Slave	Wireless Module
Micro Controller		ATmega128	ATmega 103*	AT90S2313
Available Memory	Flash	128K Bytes	128K Bytes	2K Bytes
	EEPROM	4K Bytes	4K Bytes	128 Bytes
	SRAM	4K Bytes	4000 Bytes	128 Bytes
Clock Frequency	max.	16 MHz	16 MHz	10 MHz
	used	14,7456 MHz	14,7456 MHz	9,8304 MHz
Peripheral Features	8-bit Timer	2 (0 used)	2 (1 used)	1 (1 used)
	16-bit Timer	2 (1 used)	1 (1 used)	1 (1 used)
	Serial U(S)ART	2 (2 used)	1 (1 used)	1 (1 used)

*ATmega128 micro controller in ATmega103 compatibility mode

Table 4.1: Microcontrollers used in the case study

Wireless Module

The actual wireless communication is done by the so-called *wireless module* (shown in Figure 4.3, the hardware module on the right). This hardware module has been newly developed, the board schematic and layout can be found in Appendix A.

In general, the wireless module consists of a *BIM2-433-64 radio transceiver module* from Radiometrix Ltd.[1] and an Atmel AT90S2313 microcontroller. The transceiver module provides half-duplex data transmission at ranges up to 200 meters external and 50

[1]Data sheet can be found at `http://www.radiometrix.co.uk/products/bim2.htm`

meters within buildings. It operates on 433.92 MHz, an European license exempt frequency band and enables data rates up to 64 kBit/s. Additionally, the BIM2-433-160 transceiver module, a plug-in replacement for the older version that enables data rates up to 160 kBit/s, has also been used. The main advantage of these transceiver modules in connection with real-time systems is high timing predictability due to the ability of low level access to the wireless channel.

The transceiver module can deal with serial digital data as input and output, respectively, and thus can easily communicate with a standard *UART* interface available on most current microcontrollers. We implemented *Manchester coding* and *modified frequency modulation* (MFM) as encoding scheme for the wireless transmitted data. In order to decouple and disburden the master and slave nodes from the data encoding, they transmit the raw data to the AT90S2313 microcontroller on the wireless module. Additionally, this unit controls the transceiver module. Although these are rather simple tasks, they consume a lot of processing power and hence the wireless module is the bottleneck of our current implementation. Analysis results are presented in Section 5.1.2. A more detailed description of the software of this controller is given in Section 4.5.1.

4.3.2 Demonstrator Hardware

The Smart Car consists of a series of different electro-mechanical and TTP/A hardware components forming a distributed system. Figure 4.4 shows the network and communication structure of the demonstrator.

The actual electro-mechanical components used for the Smart Car are summarized in the following. A detailed description of functional properties, timing behavior, and data acquisition issues is presented in [Schneider, 2001].

Infrared Sensor: The Smart Car is equipped with three Sharp GP2Y0A02YK infrared sensors. These are low cost sensors, which provide a detection range from 20 to 150 cm. The employed distance measuring technique is triangulation. The Sharp GP2Y0A02YK infrared sensors provide an analog output that can be directly connected to a microcontroller.

Ultrasonic Sensor: The detection of obstacles farther than 150 cm is done by two Polaroid 6500 ultrasonic sensors. These components transmit a sonic ping with a frequency of about 50 kHz. The reception of the echo, generated at the surface of an obstacle, produces a high signal state that can be used to measure the time the sonic ping needs for traveling to the obstacle and back again. Using the known speed of the sonic ping, about 340 $\frac{m}{s}$, the distance can be calculated.

Servo Motor: The servos handling the steering of the car are standard servo motors as they are employed in radio controlled cars. They provide a three-wire interface:

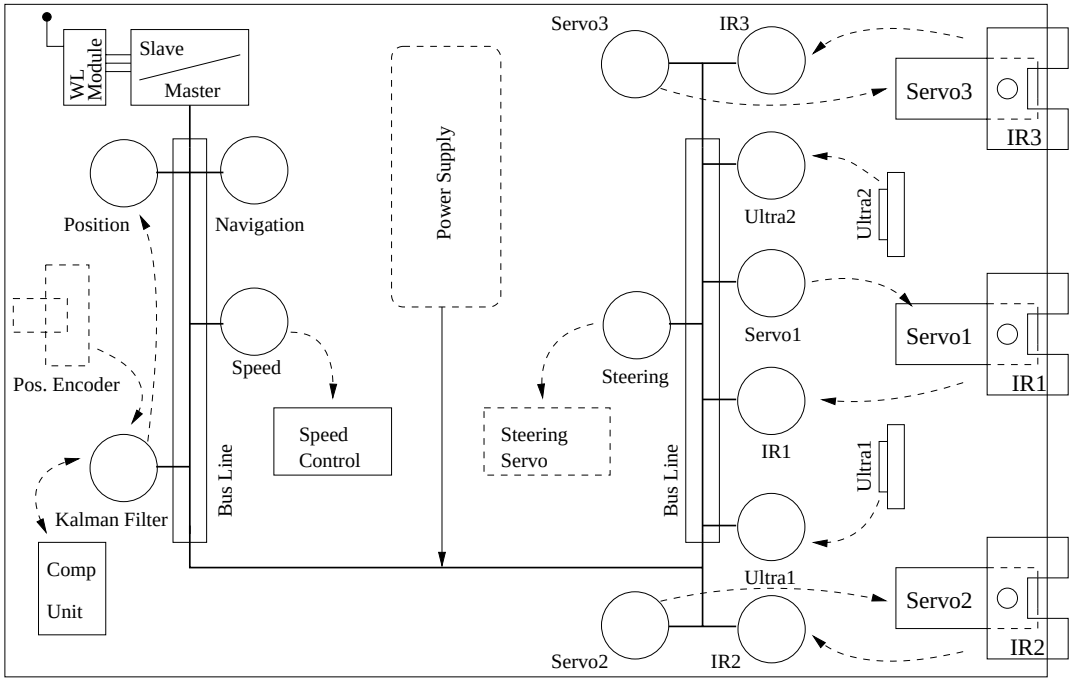


Figure 4.4: TTP/A network structure of the Smart Car demonstrator

power, ground, and control. The servo rotation is determined by a pulse-coded control signal.

All TTP/A nodes are based on Atmel AVR 8-bit RISC microcontrollers. As a result of the capability of TTP/A to build an inhomogeneous network based on different controller types, they have been chosen according to the performance requirements of the individual nodes. Table 4.2 summarizes the different nodes and their actual hardware configuration, respectively.

4.4 Demonstrator Software

This section describes the different tasks executed by the Smart Car. Due to the exact and complete specification of the system's timing behavior in TTP/A, the use of TTP/A as link between the different functional units of the demonstrator allows an independent consideration of each task. The Smart Car has to perform low level tasks like infrared and ultrasonic data acquisition, servo control, and motion planning as well as sophisticated ones like position estimation, grid generation, and path planning.

Node	Microcontroller	Function	Additional Comment
Navigation	ATmega103	path planning, grid generation	additional ext. RAM
Position	AT90S4433	provides actual position	
Kalman Filter	ATmega103*	position data preprocessing	
Speed	AT90S4433	speed control	
Steering	AT90S4433	steering control of front wheels	
Ultra[1-2]	AT90S4433	long distance obstacle detection	
IR[1-3]	AT90S4433	short distance obstacle detection	
Servo1	ATmega103*	control of IR1 turning angle	on-line configurable
Servo[2-3]	AT90S4433	control of IR[2-3] turning angle	

* ATmega128 micro controller in ATmega103 compatibility mode

Table 4.2: Slave nodes forming the TTP/A network

4.4.1 Embedded Development Environment

There are two main criteria for the choice of the programming language during the development of software for embedded systems: The need for full control over the generated machine code to maximize performance or for sophisticated mechanisms for debugging, type checking, etc.

Writing embedded software in assembler surely applies to the first argument and is often used for implementing core services of real-time systems as, e. g., the implementation of the TTP/A protocol [Trödhandl, 2002] used throughout this case study. For the application software of the demonstrator's TTP/A nodes, the programming language C was chosen. C offers well-suitable constructions for system level programming. Additionally, there exists C compiler support for many different platforms, which enhances the portability of the developed software.

The used target systems are equipped with Atmel AVR 8-bit RISC microcontrollers, as there exists a freely available open source cross-compiler: AVR-GCC. This compiler is a port of the commonly used GNU GCC, which is available on a series of operating systems, including MS Windows and Linux. The TTP/A application code was developed using the AVR-GCC compiler in version v3.0. This is a stable version of the compiler, which is well-tested for usage with the TTP/A protocol implementation. A drawback of the AVR-GCC compiler v3.0 is that it does not support the Atmel ATmega128 microcontroller. Therefore, this embedded controllers have to be used in ATmega103 compat-

ibility mode.

The implementation of the wireless master is based on an ATmega128 controller and therefore, the next stable version, AVR-GCC v3.2, has been chosen, since no support for the TTP/A protocol implementation is needed.

4.4.2 State-of-the-Art Software Components

This section shortly describes the software components that have already been available prior to this case study and that could have been reused from previous work [Elmenreich, 2002, Schneider, 2001].

Infrared sensor data acquisition: The infrared sensor element delivers an analog voltage value according to the obstacle distance. An analog-digital converter digitizes this measurement value. The sensor characteristics are shown in Figure 4.5. In order to minimize the measurement jitter, the average value over four measurements is calculated.

Ultrasonic data fusion: Due to the wide diffusion of ultrasonic sensors, they have not been used for the generation of the certainty grid. Therefore, the measurements returned from these sensors are used for a coarse estimation of the free space in front of the Smart Car. According to the returned distance value, a confidence value is assigned to each sensor. The distance and the confidence of both ultrasonic sensors are taken to calculate a fused distance. See [Elmenreich, 2002, page 110] for a detailed description of the fusion algorithms.

Servo control: The three servos, mounted in the front area of the Smart Car, adjust the infrared sensors, so that they obtain a complete view of the environment in front of the car. The software controlling the servo receives an 8-bit angle as its input. This value has a resolution of two degrees; thus, all possible turning positions can be encoded in one single byte. A pulse code modulated signal is generated out of the input value, which adjusts the servo towards the appropriate direction.

Position estimation: The Smart Car is equipped with a simple wheeled position encoder to determine the covered distance by calculating the ticks delivered by this encoder. To cope with discontinuities of the surface, which can heavily influence the position estimation result, Trojer [2004] has enhanced this simple estimation approach with a Kalman filter that basically uses the history information of the encoder to filter such discontinuities. As a side-effect, the Kalman filter implementation provides the current acceleration and velocity of the Smart Car additionally to the improved estimation of the covered distance.

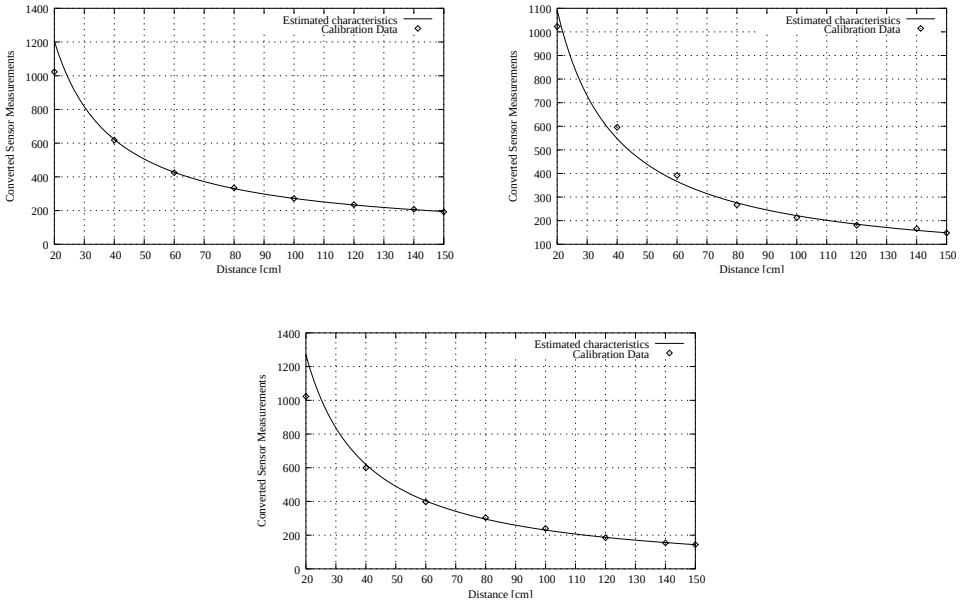


Figure 4.5: Characteristics of different instances of Sharp GP2Y0A02YK infrared sensors

Motion planning: The results of the ultrasonic data fusion determine the velocity of the Smart Car. If no obstacle is detected within a certain range (about 200 cm), the Smart Car resides in the so-called *rabbit* mode. This means that the robot drives straight forward with full speed until an obstacle enters the detection range of the ultrasonic sensors. In *turtle* mode, which is activated if an obstacle is detected, the driving speed is reduced and a complete sensor sweep is performed after every movement of 20 cm. Using this information, the navigation and path planning algorithm, determines the further course of the Smart Car.

4.4.3 Grid Generation

The integration of the measurements of the three infrared sensors – the generation of the certainty grid – is done by the navigation node. The original grid generation implementation has been optimized for low processor load and low memory consumption. Therefore, this algorithm was based on several assumptions which restrict the flexibility of the Smart Car: First, the car's perceivable environment is limited to a grid of 17 x 11 cells, each corresponding to an area of 10 cm x 10 cm. This yields a maximum visible environment of 170 cm x 110 cm.
Additionally, the number of viewing angles of the sensors has been fixed and they have

all been statically defined (shown in Figure 4.6). The dark box at the bottom of the grid represents the Smart Car. The three sensors are placed at the left, middle, and right front of the car.

Thereby, the end point of the line of sight of each sensor-angle-combination is known in advance. Thus, a straight line between the sensor's origin and the known endpoint is calculated. All cells along this line between the origin and the measured distance value are marked as free, the cell corresponding to the measurement value itself as occupied, whereas all others remain as they are.

In this original approach, the calculation and update of the active cells in the grid can be done very fast. It has one major drawback, however: Changes in the grid size, e. g., caused by new sensors with a better operation range, or in the viewing angles, e. g., to improve the spatial resolution of the grid, need a complete recalculation of all statically defined parameters.

Therefore, we implemented a new approach where the line of sight of the individual infrared sensors and the covered grid cells at a specific viewing angle are calculated dynamically. This enables, e. g., an operator to adjust the viewing angles during operation without the need to stop and recompile the Smart Car's software. The equations (4.1) and (4.2) are used to calculate the relative distance of the observed obstacle from the sensor origin. In this equations *distance* refers to the measured distance value and $Scale_X$ as well as $Scale_Y$ to the grid resolution in horizontal and vertical direction, respectively.

$$x_{diff} = \frac{distance \cdot \cos(\alpha)}{Scale_X} \tag{4.1}$$

$$y_{diff} = \frac{distance \cdot \sin(\alpha)}{Scale_Y} \tag{4.2}$$

Additionally, the model of the infrared sensors has been changed. In the original model (shown in Figure 4.7(a)), the viewing angle α and the distance measurement value are taken to calculate a single line of sight from the sensor's origin to the detected obstacle. The certainty value of all grid cell along this line of sight is set to 0x00 meaning that there is guaranteed no obstacle. The value of the last cell is set to 0xFF. The line of sight can be calculated very fast using, e. g., the Bresenham line drawing algorithm [Hearn and Baker, 2003].

It is important to note that this is a idealized model of the infrared sensors that completely neglects the cone-shaped view field of the sensors. This limitation yields a new model that allows the specification of γ, which depicts the angle of beam spread of the sensor (see Figure 4.7(b)). Additionally, certainty values inside the view field but not directly on the line of sight are diminished to represent an undecided state.

If the increment of the different viewing angles equals $2 \cdot \gamma$, the grid cells along the line from the sensor's origin to the right corner of the view field would be overlapped by

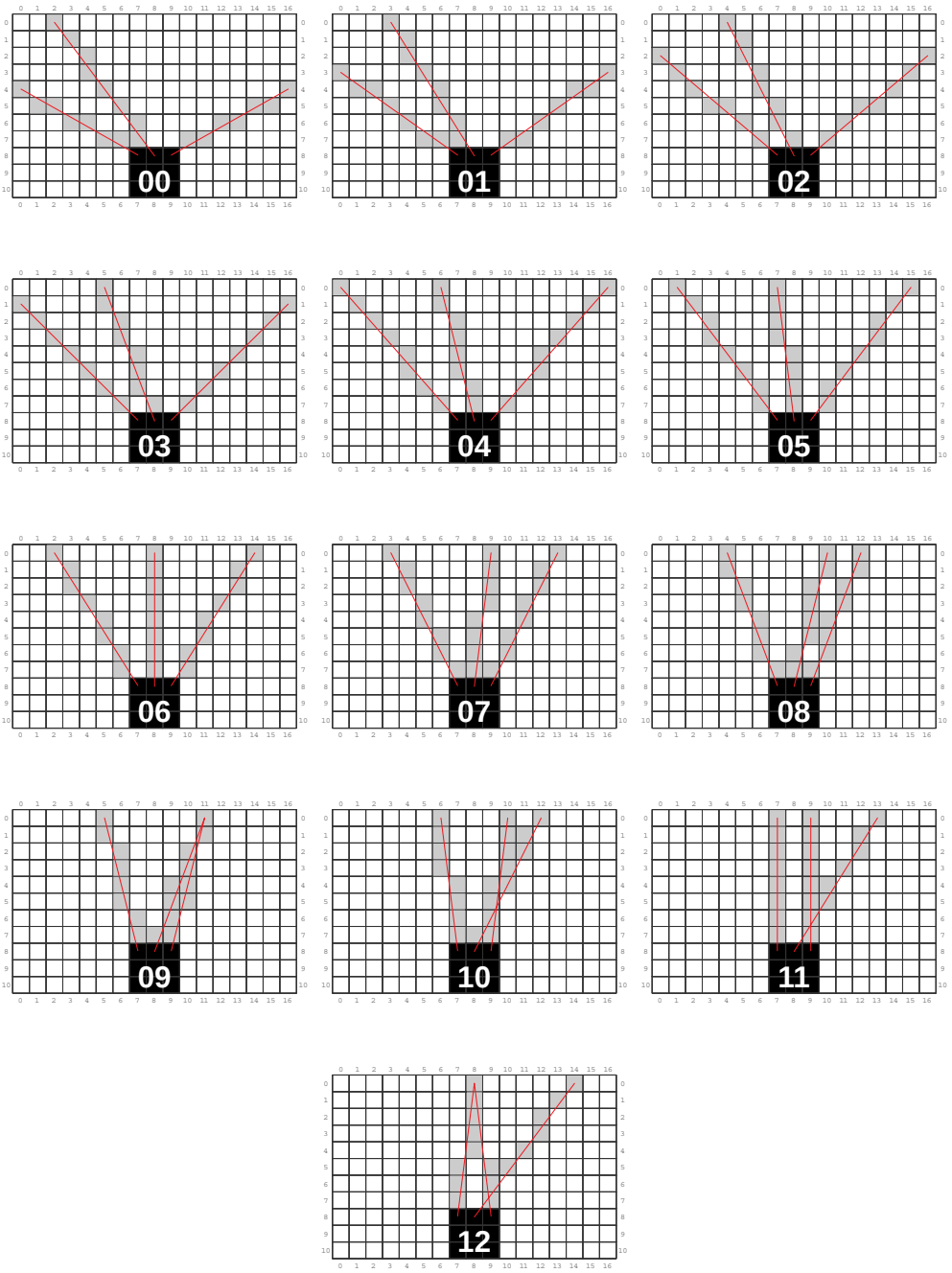
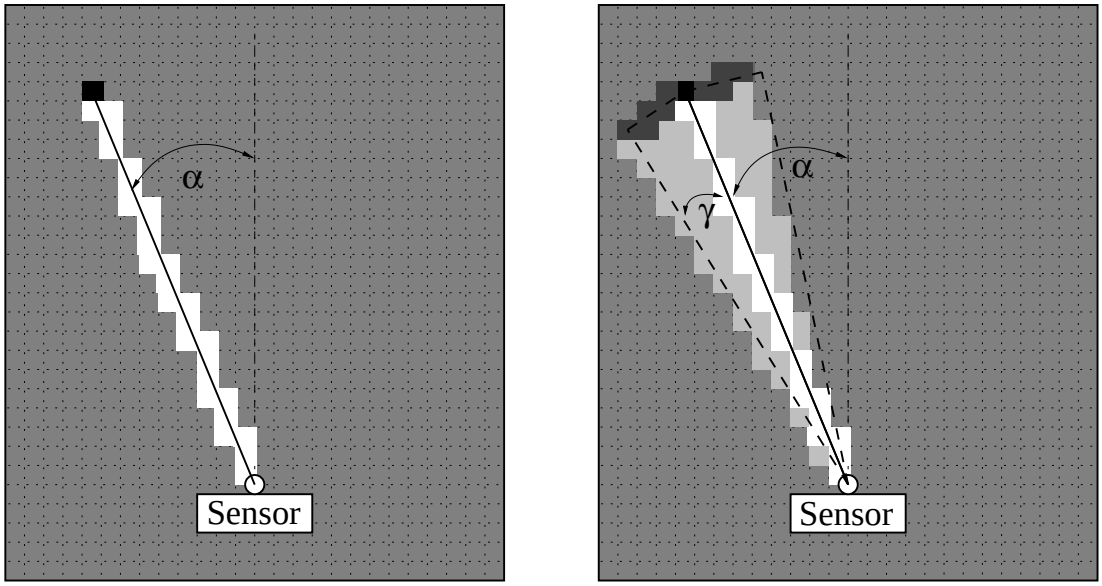


Figure 4.6: Statically defined viewing angles (from [Dias and Irran, 2002])

the view field of the next measurement. To avoid this problem, the overlapping line is removed from the view field. Figure 4.7(b) shows that the leftmost line of the cone-



(a) Original IR sensor model

(b) Enhanced IR sensor model

Figure 4.7: Two different infrared sensor models used with the Smart Car

shaped sensor model is included in the view field, whereas the rightmost line is excluded.

Each sensor generates its own local certainty grid that contains the measurement values of a complete sweep (usually 13 different viewing angles). Using these three independent local representations of the robot's environment, a fused certainty grid is generated. The fusion is either done by a simple confidence weighted averaging algorithm, Bayesian fusion, or the robust certainty grid algorithm [Elmenreich, 2002].

4.4.4 Navigation and Path Planning

As mentioned above, the Smart Car provides two different motion modes: rabbit mode and turtle mode. In the original Smart Car implementation, the turtle mode is only activated when an obstacle has been detected. One of the objectives of this thesis lies on the generation of a global map of the robot's environment. Therefore, the Smart Car resides in turtle mode, even if there is no obstacle in front because the infrared sensors are only activated in this mode.

The navigation and path planning algorithm presented by Elmenreich [2002] performs three different steps:

Path planning: This task deals with the evaluation of the possible paths the Smart Car

is able to take, given a fixed set of steering constants. Out of the possible steering constants, a configuration of 13 constants has been chosen that provide 13 well-distinguishable paths. Figure 4.8 depicts the chosen paths in relation to a certainty grid of size of 17 x 11 cells.

Path assessment: To assist the decision making process, each available path has to be evaluated according to its risk of causing a collision with an obstacle. Consequently, the grid cells belonging to a specific path have to be determined. On this "active" cells, a simple risk distribution scheme is applied. The algorithm uses concentric rings of 20 cm width with distance-dependent risk values. The overall risk of a path results out of equation (4.3) [Elmenreich, 2002]:

$$risk_{path} = \sum_{cell \in path} cell.risk \cdot cell.occ \tag{4.3}$$

In this equation *cell.risk* represents the assigned risk value of a specific cell on the path and *cell.occ* the occupancy value of this cell.

Decision making: The evaluated risk values of all 13 paths are used to decide which path the Smart Car should take. First, all paths having a risk value higher than a particular threshold are discarded. Out of the set of remaining paths the one with highest preference is chosen. The assignment of a preference value can be used, e. g., to force the Smart Car to drive always straight ahead if feasible. If more paths with the same preference exist, the one with the lowest risk value is chosen.

The original implementation of the Smart Car calculates the available paths, together with the assignment of the effected cells per path, and the risk distribution scheme statically and holds them fixed during operation. In this thesis the flexibility of the path planning phase has been improved in the way that the paths are calculated by the Smart Car itself during the startup phase. This enables the modification of important system parameters like steering constants and grid size which have an impact on the resulting paths, without the need of an extensive offline re-calculation and re-programming of the navigation node. Additionally, the risk distribution scheme that heavily depends on the used grid size is calculated during the startup phase as well.

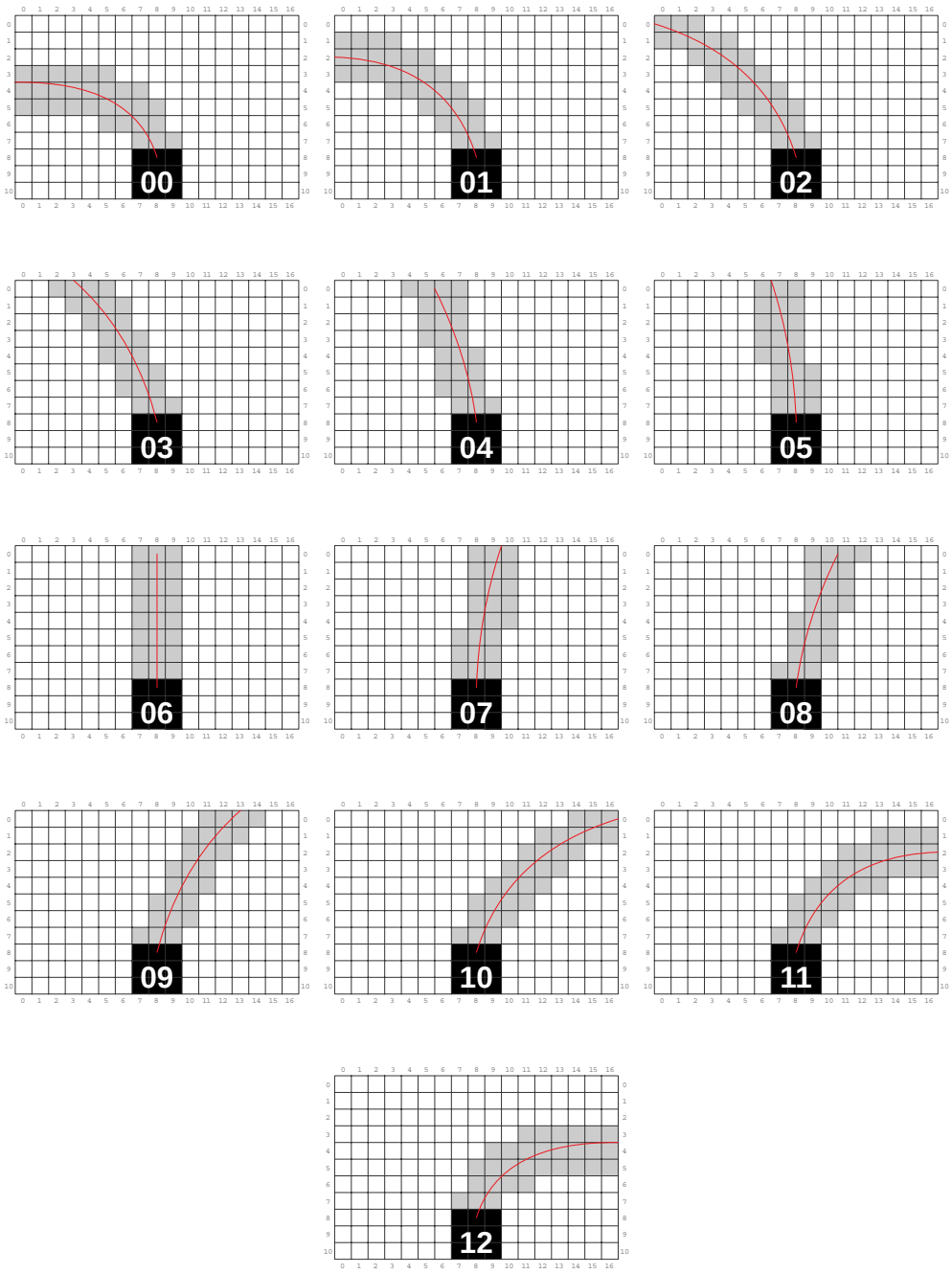


Figure 4.8: Path planning options (from [Dias and Irran, 2002])

4.5 Wireless Protocol Implementation

This section describes the software modules developed for the implementation of the wireless communication protocol.

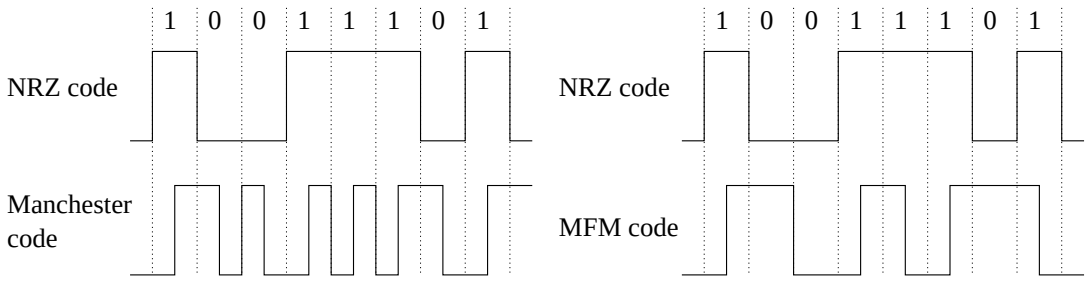
4.5.1 Data Encoding and Transmission

The *wireless module*, a hardware extension that is connected to the master as well as to the slave nodes, basically consists of a wireless transceiver module and a microcontroller. It has to fulfill two different functions:

Data Encoding:

The simplest form of data encoding is *non return to zero* (NRZ) coding. It is used, e. g., by standard UART interfaces: The logical states “1” and “0” are represented by significant signal conditions, e. g., a high signal level for logical “1” and a low level for “0” or vice versa. The drawback of the NRZ coding is its disadvantageous bit patterns, e. g., a constant transmission of “1” or “0”, where the receiver can lose the synchronization to the sender that immediately leads to data loss. Thus, the wireless module has to transform NRZ coded data either to *Manchester* or *modified frequency modulation* (MFM) encoded data.

Figure 4.9(a) depicts the translation of a NRZ coded data stream to Manchester coded data. The Manchester code uses signal transitions instead of signal levels for information representation. A logical “1” is represented as a transition from low to high signal level, a logical “0” otherwise. Therefore, it is guaranteed that a signal change occurs in every bit cell. A drawback of this coding scheme is the bandwidth usage. Manchester coded



(a) Manchester encoded data (b) MFM encoded data

Figure 4.9: Comparison of Manchester and MFM coded data

data streams need twice as much bandwidth than, e. g., NRZ coded data.

A data coding scheme that utilizes both advantages, permanent signal transitions and low bandwidth usage, is the modified frequency modulation (MFM). The MFM code was primarily developed for efficient data storage on magnetic disks, but it can also be applied to data transmission. In Figure 4.9(b) MFM coded data is depicted. The MFM code works as follows: Every time the signal level of the raw data is "1", a signal transition in the middle of the bit cell of the resulting data stream occurs. Additionally, there are also signal transitions at the border of a bit cell between two succeeding "0". Figure 4.9 shows that MFM coded data needs less bandwidth than Manchester coded data.

Transceiver Module Control

The used wireless transceiver modules have three different functional modes: power down, receive, and transmit. The state machine depicted in Figure 4.10 shows the logical structure of the software implemented on the encoding unit.

After an initial reset, the encoding unit resides in *power save* mode where the appropriate control signals are generated to activate the transceiver's power save mode.
A low level of the *activate_receiver* signal, which is provided by the host microcontroller, switches the encoding unit to the *receive mode* where it first waits for the detection of the wireless carrier signal. If the carrier has been detected, the receiver synchronizes itself with the sender by the reception of several regular high-low transitions. This synchronization pattern is transmitted at least 3 ms and is followed by the start pattern.
After the correct reception of the *start pattern*, two certain sequences of eight bit, the data reception is started. In case of Manchester encoded data, the data is grouped into frames of 16 bits, decoded, and the resulting eight data bits are transmitted to the host (either master or slave node). In case of synchronization or data errors, the encoding unit immediately returns to *power save* state. Additionally, a high level of the *activate_receiver* signal any time during the data reception causes an abort and the return to the initial state.

The wireless transmitter is activated by the reception of data via the UART interface from the host microcontroller. If the whole message is received, the transmission process is started. The transmitter has to start with a preamble of about 3 to 5 ms of a synchronization pattern (0x55). Thereafter, two *start characters* are transmitted and then the actual data transmission is started.

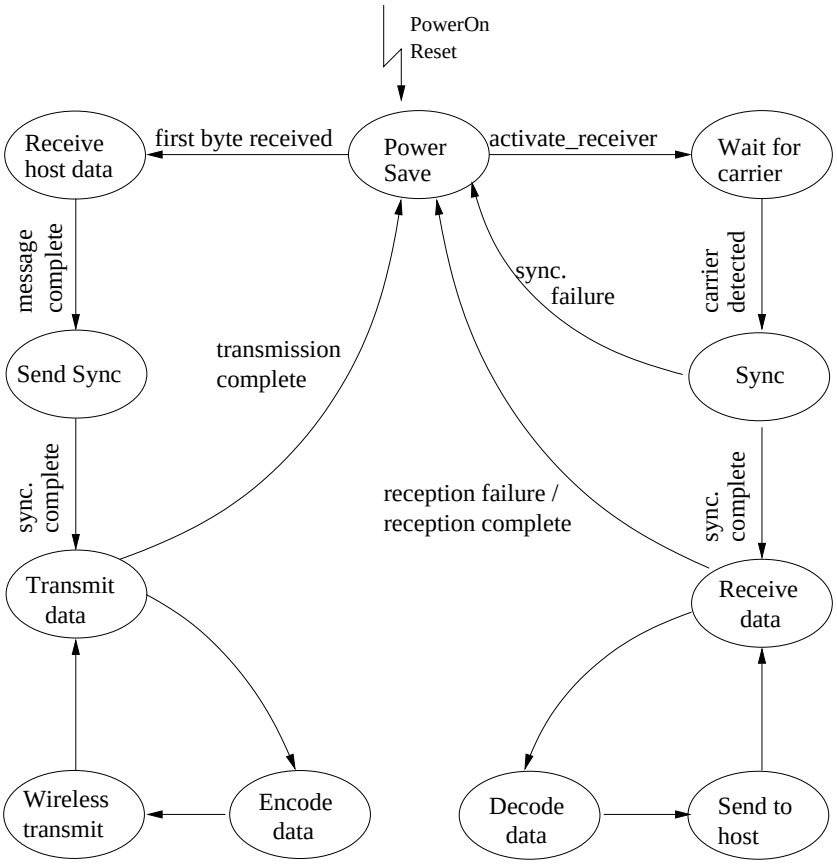


Figure 4.10: State machine of the encoding unit

4.5.2 Wireless Master Implementation

Basically, the implementation of the wireless master follows straight away from the state machine shown in section 3.2.2. Therefore, this section only covers some specific topics related to the implementation of the wireless master.

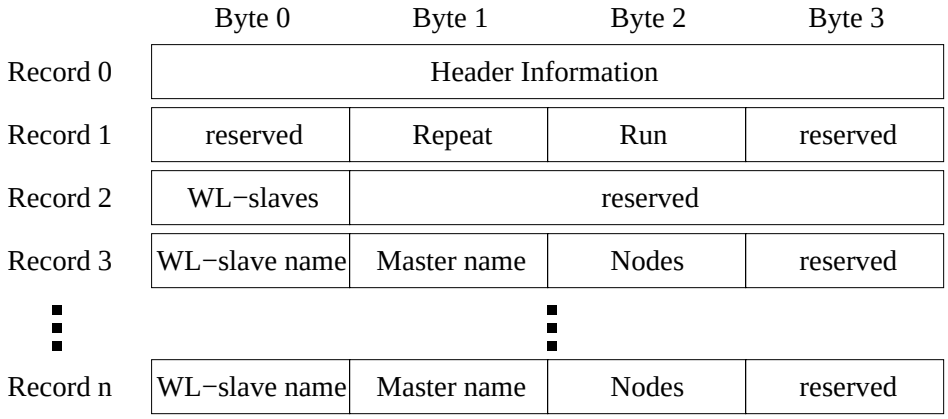
Generation of the time base

This communication protocol is based on the single time master principle. Consequently, the quality of the master's clock is a crucial feature of the implementation. As mentioned in Section 4.3.1 the core of the master node is an Atmel ATmega128 microcontroller clocked by an off-the-shelf external quartz oscillator with a frequency of 14.7456 MHz. In order to achieve the maximum possible timing quality, the 16-bit timer provided by the Atmel microcontroller has been configured to generate interrupt service requests exactly at the time instants defined by the communication schedule.

Important IFS files

For the implementation of the wireless master the concept of the interface file system (IFS) has been used to store all protocol relevant data.

Config file: This file hosts information about the state of the monitoring interface on the one hand and the setup of the wireless cluster on the other hand (see Figure 4.11). For wireless slaves which are also TTP/A masters, additional information about the TTP/A cluster is stored.



Repeat	constantly repeat monitoring request list
Run	activate monitoring
WL−slaves	number of attached wireless slaves
WL−slave name	logical name of wireless slave k
Master name	logical name of TTP/A master node
Nodes	number of attached TTP/A slaves

Figure 4.11: Contents of the configuration file

Wireless RODL: The communication schedule of a round is defined by the so-called wireless round definition list (WRODL). The layout of the WRODL was overtaken from the TTP/A specification. A detailed description of the layout can be found in [Elmenreich et al., 2002a]. A minor change was made to this specification: An additional OP-code has been added which indicates a monitoring frame.

Request List: The *Request List* implements the monitoring and configuration service. The structure of the request list is shown in Figure 4.12. Basically, the information stored in a single request entry is mapped into the address fields of the master frame. For configuration requests, the request type is *Send*, the *File Pointer* and the *R/W Offset* determine which file from the master's IFS is transmitted during request. For monitoring requests, the request type is *Receive*, these two fields

	Byte 0	Byte 1	Byte 2	Byte 3
Record n	Slave Name	Node Nr.	File / OP−Code	Start Rec.
Record n+1	Request Length	File Pointer	R/W Offset	End / Error

Slave Name	name of the addressed wireless slave node
Node	name of the addressed TTP/A node (master/slave)
File	addressed file number (0 .. 63), 6 bit
OP−Code	request type (send or receive), 2 bit
Start Rec.	record offset within addressed file
Request Length	overall request length (in records)
File Pointer	pointer to storage place in masters IFS
R/W Offset	record offset within file (denoted by File Pointer)
Error	actual error code (bit 0 to 6)
End	indicator for the end of the request list (bit 7)

Figure 4.12: Request List entry

are only needed if the combination of *Slave Name* and *Node* address a TTP/A slave node. Furthermore, *File Pointer* and *R/W Offset* refer to a storage cell of the according TTP/A master. The reason therefore is explained in the paragraph below.

Virtual IFS mapping

The monitoring tool *TTP/A Gateway* uses a standardized interface for communication with a CORBA gateway that enables the interconnection of several TTP/A clusters. In this thesis, the role of the CORBA gateway is overtaken by the wireless master node. Thus, the wireless master has to host the information of all slave nodes (TTP/A clusters) requested by the monitoring tool.

Figure 4.13 depicts the two-staged architecture that is used to fulfill this task. The wireless master contains an *IFS table* consisting of several *IFS pages*. Each page forms a separated name space and is assigned to a specific slave. Thus, the name of the slave can be used as index into the IFS table. By convention, index 0 addresses the IFS page of the wireless master.

The wireless master periodically processes the entries of the request list and updates the contents of the according IFS page (right part of Figure 4.13). If the request addresses a TTP/A slave connected to a wireless slave and not the wireless slave directly, the File Pointer and R/W Offset introduced above are needed. The combination of both fields determines the location in the IFS page, assigned to the addressed TTP/A cluster, where the data from the TTP/A slave should be stored. Consider following example:

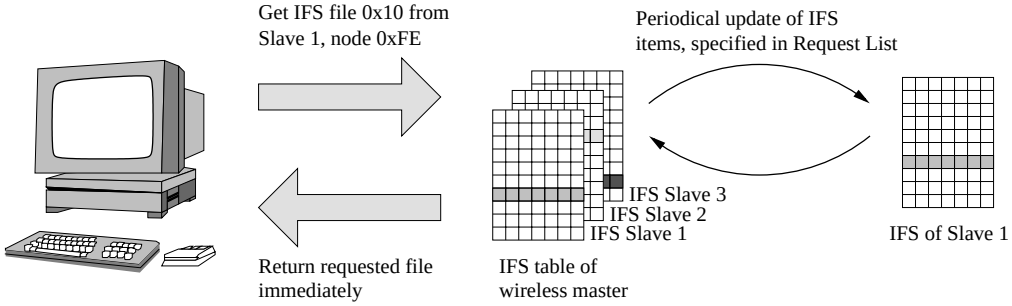


Figure 4.13: IFS table at wireless master node

```
Slave Name: TTP/A cluster 1, Node: Infrared Sensor 1, OP-Code: Receive
IFS Address: File: 0x10
             Start Rec.: 0x01
             Request Length: 0x04
File Pointer: 0x30, R/W Offset: 0x10
```

This entry is a request for four records (16 bytes) starting at record 0x01 from file 0x10 of "Infrared Sensor 1", which resides in TTP/A cluster 1. The data is stored at the wireless master in file 0x30 of the IFS page of "TTP/A cluster 1", starting at record 0x10.

Every time a specific data item is requested by the monitoring tool, the wireless master returns its local copy of this item (left part of Figure 4.13). It is important to note that the IFS table of the wireless master contains only items, which are requested by any entry of the Request List. Thus, if this list is not configured properly, the returned data might be outdated. If a requested file does not exist in the IFS page assigned to the addressed slave, standardized error codes are returned to the monitoring tool.
Since the Request List itself resides in the IFS of the wireless master, updates and changes can be done during operation of the system.

4.5.3 Wireless Slave Implementation

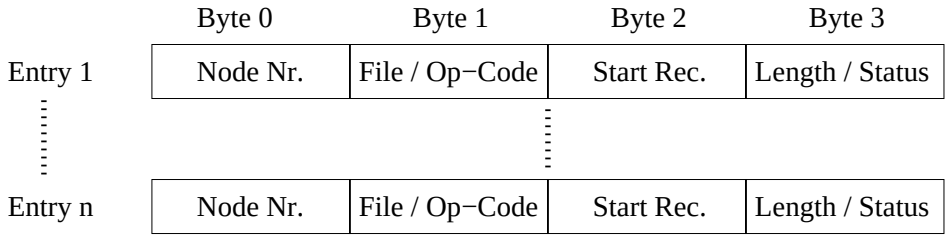
Some important facts regarding the implementation of the wireless slave nodes are presented in this section. The wireless slaves concurrently act as TTP/A master; this section, however, is only concerned with issues that deal with the wireless network.

Important IFS files

Besides the information needed for the implementation of the actual TTP/A master, the IFS contains two files necessary for the implementation of the wireless protocol.

Wireless RODL As described in the section above, the WRODL determines the layout of the time-triggered communication. The entries differ from slave to slave, but they must not interfere each other.

Request Buffer The *Request Buffer* file contains requests transmitted by the master that regard to a particular TTP/A slave connected to this node. The structure of a buffer entry is shown in Figure 4.14. The concrete scheduling of stored requests is explained below. A second IFS file is used for the temporary storage of the monitoring or configuration data that has to be exchanged between the wireless master and the addressed TTP/A slave.



Node Nr.	logical name of addressed slave
File	addressed file (0 .. 63), 6 bit
Op−Code	type of request (send or receive)
Start Rec.	record offset within addressed file
Length	number of requested records (max. 8)
Status	processing state of the entry

Figure 4.14: Request Buffer entry

Scheduling Monitoring Requests for TTP/A Slaves

In TTP/A networks, so-called Master/Slave rounds enable the master to obtain arbitrary data from particular slave nodes. Thus, the requests stored in the Request Buffer of the wireless slave have to be scheduled to corresponding Master/Slave rounds.

In general, the real-time data exchange in TTP/A – the multi-partner rounds – is alternatively interleaved by Master/Slave address rounds (MSA) and Master/Slave data (MSD) rounds, respectively. This alternating schedule is used by the wireless slave to obtain the requested data from the TTP/A slaves. Based on the addressing information stored in the Request Buffer, a valid MSA round is generated. Since in one MSD round only a single IFS record can be returned, the completion of a request can last longer than one Master/Slave round.

The last byte of a request entry represents, beside the number of requested records, the processing state of this entry. A buffer entry can reach following different states:

Unused: This entry is not filled with a request, or a former configuration request was successfully completed.

Waiting: The waiting state means that this entry was recently occupied by a configuration request, but the according data from the wireless master is still missing. This situation occurs, because the request itself is transmitted in the master frame, whereas the data follows in the succeeding monitoring frame at the end of a round.

Passive: A entry is set to *Passive*, if the complete request was received, but was not processed yet.

Active: *Active* means that this entry is currently processed.

Done: An entry referring to a monitoring request is set to *Done* immediately after the successful reception of the last requested byte.

Redo: *Redo* has the same meaning as *Active* with the difference that this request was successfully processed at least once.

Error: This indicates the occurrence of an error, e. g., the requested file or record does not exist. Only monitoring requests produce error codes because TTP/A does not support any error messages or problem reports when writing data to the IFS. The success of a write operation can only be verified by a succeeding read operation on the same data item.

The Request Buffer is periodically traversed. Entries with status *Passive* or *Done* are processed. To avoid the transmission of out-dated configuration data to a slave, the status field of a completed configuration requests is set to *Unused* instead of *Done*. This guarantees that configuration requests are only executed once.

The storage of new requests works as follows: The complete buffer is traversed in order to find an entry with a status field set to *Unused*. If one is found, the request is stored at this location. Otherwise, the last processed entry is evaluated. If the state of this entry is *Done* or *Error* it can be overwritten. Otherwise, the wireless slave returns an error message indicating that this request cannot be fulfilled.

4.6 Map Making Software

The objective of this software is to evaluate and graphically represent the sensor data received via the wireless communication protocol. It consists of two small applications:

The first one is responsible for the reception and storage of the sensory data. The second application is used to transform this numerical information to graphical representations and to assembly the individual snapshots to a global representation of the robot's environment.

Sensory data acquisition

Due to the missing possibility to store data received via the monitoring tool in a file, a simple C application was developed that waits for incoming data at the serial interface of the PC and sequentially stores it into a file. Since two different applications are waiting for data at the same interface, the monitoring tool and the data acquisition application can not be used concurrently.

Add-to-grid algorithm

The "Add-to-grid" algorithm – the core element of the map making software – generates the certainty grid representing the robot's environment. Altogether, the environmental representation consists of four certainty grids: An independent grid for any of the three infrared sensors as well as a resulting grid generated by means of sensor fusion. The same algorithms developed for the map making application are also used for the navigation and path planning application on the Smart Car.

The generation of the certainty grid for a complete sensor sweep requires the following steps:

1. Initialization
2. Sensor Value Evaluation
3. Affected Cell Calculation
4. Cell Update
5. Iteration of steps 2–4 with the distance values from *Sensor 2* and *Sensor3* obtained at *Viewing Angle 1*.
6. Sensor Fusion
7. Iteration of steps 2–6 for all 12 remaining viewing angles
8. Output Generation

Initialization: Each grid cell consists of an *occupancy* value, denoting the probability of being occupied by an obstacle, which is initialized with 0x80, meaning no assertion can be made if there is an obstacle or not. A value of 0x00 means that this cell is not occupied, 0xFF states the opposite. Additionally, an *owner* value is assigned to all cells of the fused sensor grid that holds the number of the sensor that has been least recently updating this cell. This value is initialized with *NOSENSOR*.

Sensor Value Evaluation: The distance value obtained by a sensor at a certain viewing angle is evaluated and validated. Therefore, out of norm assertions like negative distances or values larger than the maximum sensor range are used.

Affected Cell Calculation: Using the known position of the sensor in the grid, the viewing angle, and the distance value, the grid cells affected by this sensor reading are calculated. Therefore, two different modes can be used: *line* or *cone* mode. In line mode, only those grid cells are considered that lie on a straight line from the sensor's origin to the target cell. In cone mode on the other hand, a cone shaped area of grid cells, as shown in Figure 4.7(b), is taken into account.

Cell Update: This means that all cells that are ascertained to be between the sensor's origin and the target cell are set to *not occupied* (value 0x00); the target cell is set to *occupied* (value 0xFF). In cone mode, cells farther away from the middle of the cone are given a more uncertain value, e. g., 0x20 for probably free and 0xD0 for probably occupied.

Sensor Fusion: The three individual sensor grids are fused to a resulting certainty grid, representing the robot's environment. The fusion of the individual grids is either done by a simple confidence weighted averaging algorithm, Bayesian fusion, or the robust certainty grid algorithm [Elmenreich, 2002].

Output Generation: Out of the resulting certainty grid, a TIFF image file is generated. The occupancy values of the individual grid cells are mapped to gray-scales: *White* corresponds to an occupancy value of 0x00 while *black* depicts a value of 0xFF. Some sample images are shown in Section 5.3.

Global grid generation

As described above, the Smart Car performs a complete sensor sweep after every covered distance of 20 cm. Thus, during the exploration of its environment, the Smart Car generates a series of local environment maps which all represent a specific part of the entire surroundings.

The values obtained by the position encoder and the steering commands are used to correlate these individual maps. Due to the lack of any absolute reference points in

the environment, errors caused by wheel slippage and steering imprecision influence the resulting global map.

The global map generation process works as follows: At the beginning, the Smart Car is located at a known absolute position within the environment waiting for exploration. Out of the values from the position encoder and the steering commands from the navigation node, the absolute position of the robot within its environment is updated and the corresponding grid cells of the global map are matched with the current local map. Due to the relatively small change of the robot's position compared to the size of the local grid, the same cells of the global grid are updated in more than one robot position. Therefore, fusion algorithms can be used to enhance the quality of the resulting representation.

A parameter is stored that denotes the deviation of the Smart Car's longitudinal axis to the y-axis of the global map. If the robot follows a curved path, the angle between the longitudinal axis of the robot and the y-axis of the global map is calculated. The local map is rotated according to this angle and matched with the global map.

When rotating a raster image by an arbitrary angle, the digitalization error has to be considered. The obvious method is to loop over the pixels in the source image, transform each coordinate, and copy the pixel to the destination. The drawback with this approach is that it leaves holes in the destination image caused by digitalization errors. To avoid this effect, the "reverse" transformation is applied on the coordinates of the destination image by using the negative rotation angle. Afterwards, the corresponding pixels from the source image are copied. There are, however, also much faster methods e. g., a method proposed by [Paeth, 1986] that uses three shear operations for accomplishing a rotation, but this alternatives were not considered in this implementation.

There are two possible outcomes:
If the result confirms the hypothesis,
then you've made a discovery.
If the result is contrary to the hypothesis,
then you've made a discovery.

ENRICO FERMI

Chapter 5

Evaluation

The following chapter presents performance and quality measures of the introduced wireless communication protocol as well as experimental results obtained with the Smart Car demonstrator and the map making application on the PC.
The first part describes the minimal requirements needed for implementing the protocol on an embedded controller platform. Furthermore, some quality measurements are presented. The second section shows the ability of the implemented wireless communication system to calibrate the demonstrator's sensors/actuators on-line. The last experiment shows the generation of a global map of the robot's environment derived from the sensor measurements of the Smart Car.

5.1 Protocol Analysis

5.1.1 Memory Consumption

As shown in Table 4.1, the used controllers consist of three different types of on-chip memory: flash ROM, EEPROM, and internal SRAM. The *avr-gcc* compiler suite (see Section 4.4.1) that has been used for the implementation of the case study provides the tool *avr-objdump* to dump the contents of a compiled object file. Figure 5.1 shows the usage of avr-objdump to determine the static memory consumption of a given object file. The different sections listed in the output (.text, .data, .bss, and .eeprom) refer to the different available types of on-chip memory.

Flash ROM: The size of the needed amount of flash ROM is expressed by the section *.text*. Generally, this refers to program code or special attributed constants.

Internal RAM: The amount of statically allocated internal SRAM is shown by the sec-

```
master.eep_elf:     file format elf32-avr

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000ae2  00000000  00000000  00000094  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000002  00800060  00000ae2  00000b76  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          0000000e  00800062  00800062  00000b78  2**0
                  ALLOC
  3 .eeprom       000000a6  00810000  00810000  00000b78  2**0
                  CONTENTS, READONLY
  4 .stab         00000b64  00000000  00000000  00000c20  2**2
                  CONTENTS, READONLY, DEBUGGING
  5 .stabstr      00000a8c  00000000  00000000  00001784  2**0
                  CONTENTS, READONLY, DEBUGGING
```

Figure 5.1: Output of avr-objdump to determine the static memory consumption

tions *.bss* and *.data*. Whereby data that is kept in section .data is initialized with predefined values during startup, all data kept in section .bss is initialized with zeros.

EEPROM: As the name implies, the size of the used EEPROM can be found in section *.eeprom*.

The static memory consumption of the three different nodes is shown in Table 5.1. As the task of the wireless slave is twofold – slave in the wireless protocol and master for the TTP/A cluster – the resources consumed by TTP/A and by the wireless protocol have to be considered separately. To demonstrate the memory requirements for TTP/A, a minimal TTP/A master was compiled. The difference between the rows “TTP/A Master” and “Wireless Slave” denote the memory requirements for the slave implementation of the wireless protocol.

In order to complete the memory consumption analysis, the amount of dynamically allocated memory, only SRAM in this case, has to be considered. Due to the lack of sophisticated support for dynamic memory allocation, subroutine calls and interrupts as well as locally defined variables determine the amount of dynamically allocated memory.

Table 5.2 shows the overall SRAM usage including the dynamic memory analysis. The static SRAM usage of all nodes slightly deviates from the values presented in Table 5.1. This is caused by the additional code needed for analyzing the dynamic memory consumption. Therefore, the dynamic memory analysis could not have been carried out on the encoding unit: Any addition to the existing source code would have exceeded the

limits of the employed microcontroller.
The high dynamic memory usage of the pure "TTP/A Master" and the "Wireless Slave", compared to the "Wireless Master", are caused by the size of their local IFS. After starting up the TTP/A protocol, the IFS structure is mapped into SRAM and thus "dynamically" allocates a not negligible amount of SRAM.

	Controller Type	Flash ROM	EEPROM	int. RAM
Encoding Unit	AT90S2313	2040 Bytes (99.6%)	0 Bytes (0.0%)	77 Bytes (60.2%)
TTP/A Master[1]	ATmega103[2]	2786 Bytes (2.1%)	166 Bytes (4.1%)	16 Bytes (0.1%)
Wireless Slave	ATmega103[2]	16010 Bytes (12.2%)	390 Bytes (9.5%)	805 Bytes (20.1%)
Wireless Master	ATmega128	8954 Bytes (6.8%)	0 Bytes (0.0%)	2128 Bytes (52.0%)

[1] consists only of TTP/A protocol code and RODL definition
[2] AVR ATmega128 micro controller in ATmega103 compatibility mode

Table 5.1: Static memory consumption of case study implementation

	Total SRAM	Static	Dynamic	Overall Consumption
Encoding Unit	128 Bytes	77 Bytes	—	$\geq$ 77 Bytes ($\geq$ 60.2%)
TTP/A Master	4000 Bytes	56 Bytes	234 Bytes	290 Bytes (7.3%)
Wireless Slave	4000 Bytes	807 Bytes	523 Bytes	1330 Bytes (33.3%)
Wireless Master	4096 Bytes	2129 Bytes	126 Bytes	2255 Bytes (55.1%)

Table 5.2: Overall SRAM consumption

5.1.2 Performance Analysis

Payload and protocol overhead considerations

As mentioned in Section 3.1, the efficiency of the information transport should be high when dealing with sensory information. Using time-triggered communication has the advantage that the communication schedule is deterministically defined, so there is no need, e. g., to negotiate about the bus access, to transmit an explicit sender identification, etc. Thus, the efficiency of the protocol solely depends on the payload to communication overhead ratio. Table 5.3 depicts the length of the individual data frames, the amount of payload and protocol specific data in bytes, and thereto related the resulting protocol overhead of our current implementation.

The overall length of a communication round is given by:

$$round\_length = gateway\_frame + n \cdot slave\_frame + monitoring\_frame \quad (5.1)$$

With n being the number of interconnected slaves (TTP/A clusters). In our case study setup, two TTP/A cluster have been connected, the therefore overall round length is $round\_length = 10 + 2 \cdot 45 + 34 = 134$ Bytes. Accordingly, the protocol overhead is 22 Bytes or 16,4 % of the overall data.
Taking a look at Table 5.3, it is obvious that an increasing number of slave frames per round reduces the overall communication overhead but is limited downward by the overhead of the slave frame itself, which is about 11 %.

	Length	Payload	Protocol data	Overhead
Master frame	10 Bytes	0 Byte	10 Bytes	100.0 %
Slave frame	45 Bytes	40 Bytes	5 Bytes	11.1 %
Monitoring frame	34 Bytes	32 Bytes	2 Bytes	5,9 %

Table 5.3: Wireless protocol overhead in the case study implementation

Data transmission rate

Two different transceiver modules have been used, which are equivalent in their functional and operational specification, except the maximum data transmission rate. The *BIM2-433-64* transceiver allows data rates up to 64 kBit/s, while its successor, the *BIM2-433-160*, has a specified maximum data rate of 160 kBit/s.

Unfortunately, the maximum ratings of these devices could not have been used to full extend because the encoding unit forms the bottleneck of our implementation. At the encoding unit, the data has to be received from the host microcontroller via hardware UART, afterwards encoded, and finally transmitted via software UART to the radio transceiver module. Additionally, in order to reduce the harmonic parts of the transmission signal, the signal transitions are unsharpened by a digital raised cosine filter. Both, the need for a software UART implementation and the raised cosine filter cause the encoding unit to be the bottleneck of the case study implementation with respect to the communication speed.

Thus, the current implementation reaches a maximum wireless transmission rate of 10240 Bit/sec. An FPGA implementation of the encoding unit that overcomes this limitation has been designed, but has not been applied to the case study yet. By using the FPGA implementation, much higher data rates can be achieved.

The following experiment was carried out to analyze the amount of transmission errors occurring at different transmission speeds (shown in Table 5.4 and 5.5) and to demonstrate the practicability of the fault-tolerant clock synchronization approach. The experiment setup consists of the master node as well as two slave nodes (C1 and C2). Both

slaves have assigned one communication frame per round. The actual used communication schedule is shown in Figure 5.2:

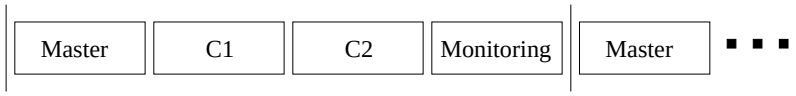


Figure 5.2: Communication schedule for evaluation of maximum data rate

The monitoring frame is alternatively assigned to one of the slaves and is used to transfer the local error statistics to the master. In both experiments, all communication partners are equipped with the same transceiver module, either the BIM2-433-64 (Table 5.4) or the BIM2-433-160 (Table 5.5).

Data errors recorded during the experiments are originated from different reasons: The error recognition and recording takes place at the application level of the slaves. Each recognized omission error, data error or synchronization loss is written to the slaves' local IFS. The monitoring support of the protocol is used to transfer this data to the master, where it is available to the monitoring tool *TTP/A Gateway*. Consequently, the error values subsume problems arising out of the data en-/decoding at the wireless module, interferences on the communication channel, and synchronization and timing difficulties at the wireless slave nodes and the master node, respectively.

The first rows of Table 5.4 and Table 5.5 depict the data rate at which the experiment was accomplished. *"Omission Errors"* summarizes all master frames which, have not been started during a given tolerance interval determined by the *interframe gap*. The duration of the interframe gap used during this experiment has been set to 5 ms. All other errors that might occur while receiving the master frame (too few bytes, parity error, etc.) are subsumed in *"Data Errors"*. The overall number that a slave has to restart the protocol because of a complete synchronization loss is depicted in the row *"Sync. Losses"*.

The vast number of data errors shown in Table 5.5 compared to the experiment results shown in Table 5.4 seems not to be reliable: The same software and communication schedule have been used and the different types of the applied transceiver modules should be plug-in replacements. Further analysis has shown that, out of the three identical applied BIM2-433-160 transceiver modules, only one module is either defective or operates slightly out of specification, causing this huge amount of transmission errors (see Table 5.6). Furthermore, Table 5.6 shows that only the BIM2-433-160 modules seem to be vulnerable to this faulty transceiver.

Although a defective transceiver module might be the reason for these observed results, it shows the relevance of the fault-tolerant clock synchronization approach developed during this thesis. Even though the number of transmission errors is very high, the slave nodes do not lose synchronization with the master.

	Data Rate [bit/s]		4800	6400	9600	10240
Master Frame	Frames Sent		1024	1024	1024	1024
	Omission Errors	C1	0	0	0	0
		C2	1	0	0	0
	Data Errors	C1	6	0	0	2
		C2	0	2	0	0
	Sync. Losses	C1	0	0	0	0
		C2	0	0	0	0

Table 5.4: Comparison of transmission error quantity at different transmission speeds (applied wireless modules: Radiometrix BIM2-433-64)

	Data Rate [bit/s]		4800	6400	9600	10240
Master Frame	Frames Sent		1024	1024	1024	1024
	Omission Errors	C1	62	64	87	99
		C2	21	49	98	88
	Data Errors	C1	95	51	6	28
		C2	5	3	0	14
	Sync. Losses	C1	0	0	0	0
		C2	0	0	0	0

Table 5.5: Comparison of transmission error quantity at different transmission speeds (applied wireless modules: Radiometrix BIM2-433-160)

Master	Slave C1	Slave C2		Omission	Data	Sync.
BIM2-433-64	BIM2-433-160	BIM2-433-64	C1	0	0	0
			C2	0	0	0
BIM2-433-64	BIM2-433-160*	BIM2-433-64	C1	0	0	0
			C2	0	0	0
BIM2-433-160	BIM2-433-160*	BIM2-433-64	C1	101	0	0
			C2	0	0	0
BIM2-433-160	BIM2-433-64	BIM2-433-160	C1	0	0	0
			C2	7	0	0
BIM2-433-160	BIM2-433-64	BIM2-433-160*	C1	0	0	0
			C2	83	0	0
BIM2-433-160*	BIM2-433-64	BIM2-433-160	C1	0	0	0
			C2	96	0	0

* the faulty BIM2-433-160 transceiver module

Table 5.6: Identification of faulty transceiver module (9600 Bit/s, 1024 communication rounds)

Evaluation of the maximum transmission range

The same setup as described above has been used for this experiment. Slave C2 has been equipped with a mobile power supply and thus attains mobility while slave C1 has been kept stationary. The different locations where the measurements have been performed are shown in Figure 5.3. The distance between the "Base station" (BS), consisting of the master node and the stationary slave C1, and the mobile slave C2 vary from 50 cm to 26 m. Likewise, the radio signal has to overcome different obstacles. While the locations A, B, and C shown in Figure 5.3, are in direct line-of-sight (LOS) to the BS and position D is only separated by a cupboard, the radio signal has to penetrate several walls, when being received at all other measurement points.

The experiment size is 1024 communication rounds at each location. The results (see Table 5.7) show that there occur few communication errors, even if the mobile slave node is not in direct LOS to the BS (experiments A to E). As expected, the amount of communication errors increases proportional to the transmission distance and the number of walls that have to be penetrated (experiments F to I). The error frequency, however, remains in the domain that can still be handled by the communication protocol, since no synchronization losses occur. The last measurement (J) highlights the limits of the used transceiver module in combination with the hardware and software components in use.

Location	Distance	Slave	Omission	Data	Sync.	Comment
A	0.5 m	C1	1	3	0	direct LOS
		C2	0	0	0	
B	4.0 m	C1	0	6	0	direct LOS
		C2	0	0	0	
C	10.0 m	C1	0	11	0	direct LOS
		C2	1	3	0	
D	4.5 m	C1	1	10	0	LOS obstructed
		C2	0	3	0	by cupboard
E	6.3 m	C1	0	11	0	LOS obstructed
		C2	2	10	0	by a wall
F	7.0 m	C1	3	9	0	LOS obstructed
		C2	0	0	0	by several walls
G	11.9 m	C1	2	13	0	LOS obstructed
		C2	0	13	0	by several walls
H	21.0 m	C1	0	17	0	LOS obstructed
		C2	0	38	0	by several walls
I	26.0 m	C1	0	15	0	LOS obstructed
		C2	0	26	0	by several walls
J	19.3 m	C1	0	1	0	aborted after
		C2	65	112	16	300 rounds

Table 5.7: Evaluation of data errors at different transmission distances

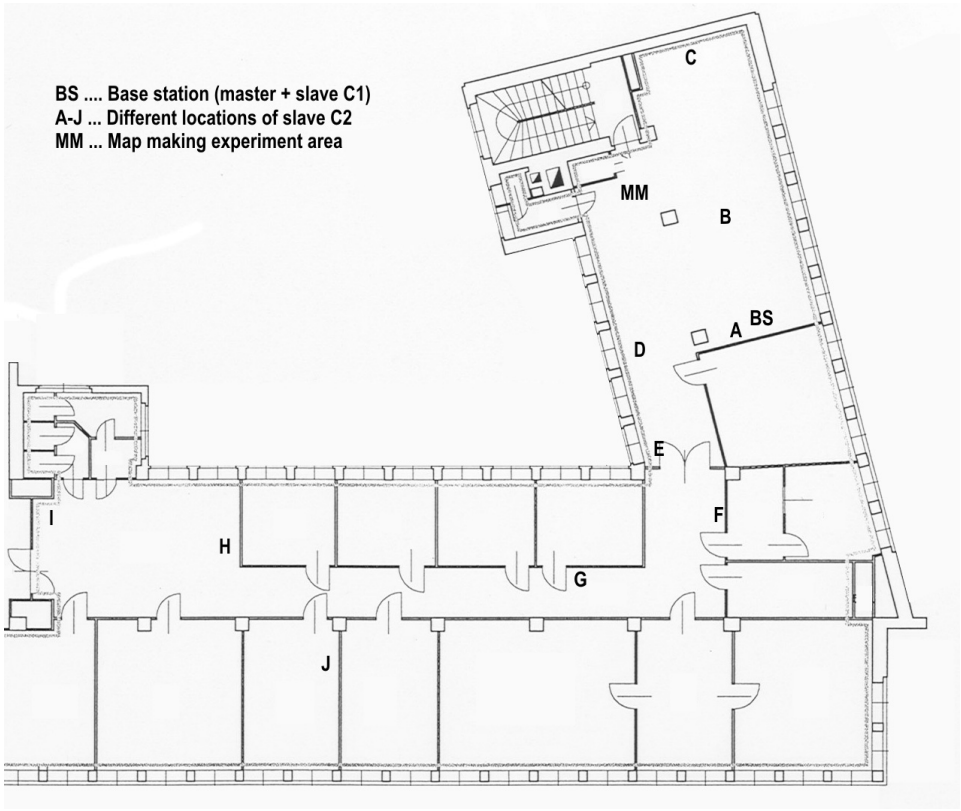


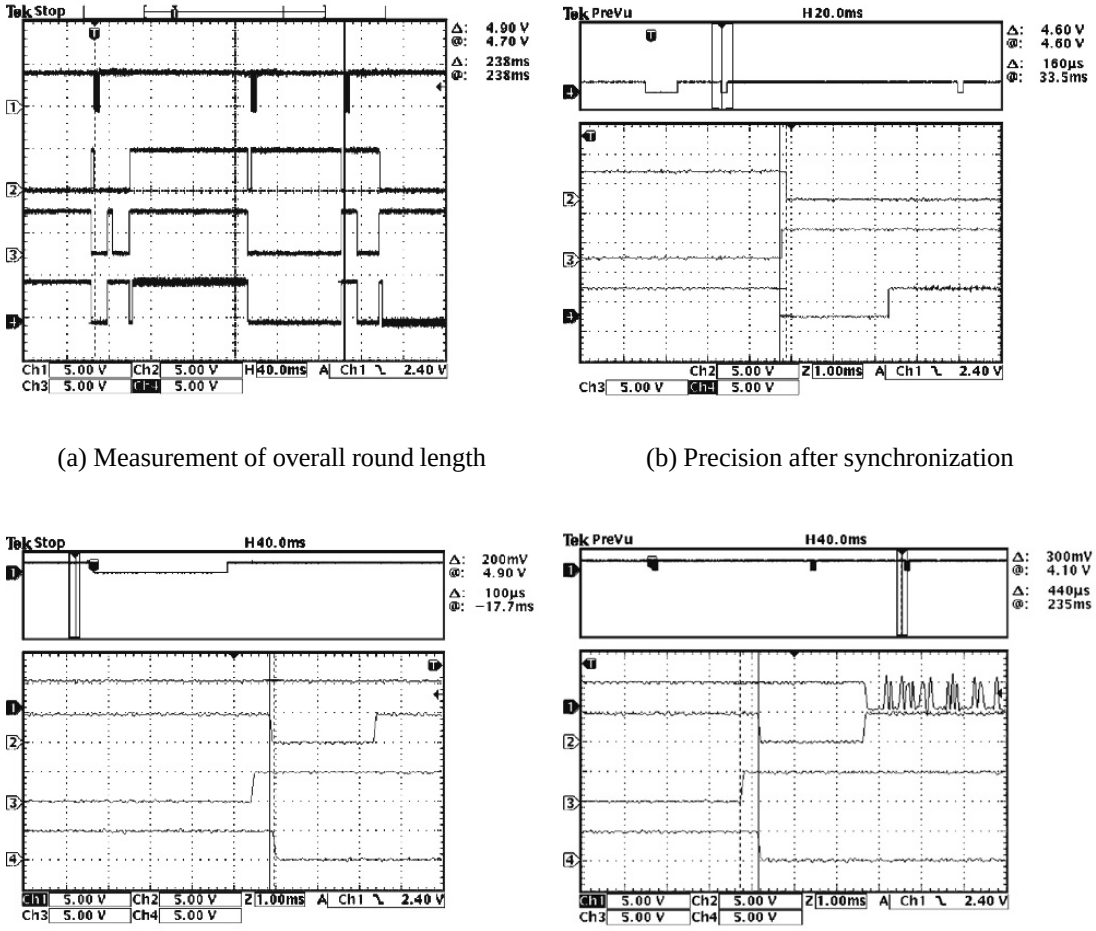
Figure 5.3: Experiment locations for maximum transmission range evaluation at the Institute for Computer Engineering

5.1.3 Fault-tolerant Clock Synchronization

The experiment described in this section proofs the practicality of the fault-tolerant clock synchronization approach. As mentioned above, the test setup consists of a master and two slave nodes (C1 and C2) forming a wireless network. However, the clock of slave C2, more precisely the drift rate of the clock, has been degraded per software in order to test the clock synchronization capabilities. The difference is depicted in Figure 5.4.

Figure 5.4(a) shows the overall length of one communication round. The communication schedule used in this experiment consists of the master frame, one slave frame assigned to slave C1, and a monitoring frame. The round length determines the minimum time between two succeeding synchronization points. This is an important property because with a static drift rate and an initial accuracy after synchronization (Figure 5.4(b) shows the typical precision of the clocks after synchronization), the round length determines the worst case accuracy of a clock (slave C1 or C2) to its reference (master).

It has been simulated by software that slave C2 loses the connection to the master after



(a) Measurement of overall round length

(b) Precision after synchronization

(c) Accuracy of “normal” clock

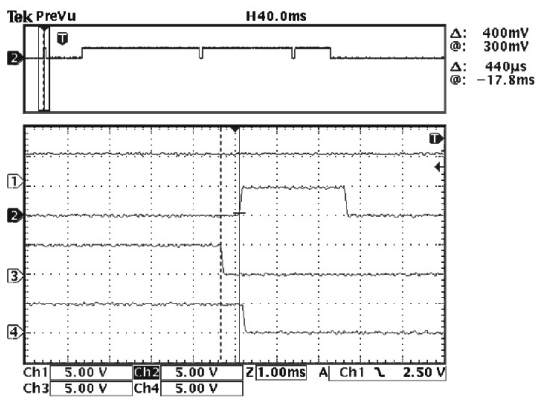
(d) Accuracy of “degraded” clock

Figure 5.4: Synchronization quality properties: Channel 1: HW-UART send signal of master node; Channel 2: Timing diagram master node; Channel 3: Timing diagram of slave C2; Channel 4: Timing diagram of slave C1

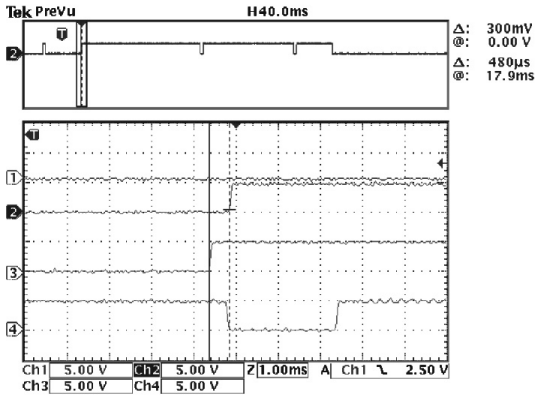
a certain amount of time. The protocol code of slave C2 has been modified, so that the slave ignores all incoming data packets by the master after a certain amount of successful communication rounds. This modification has been used in two different measurement series:

1. Simulation of link loss without fault-tolerant clock synchronization

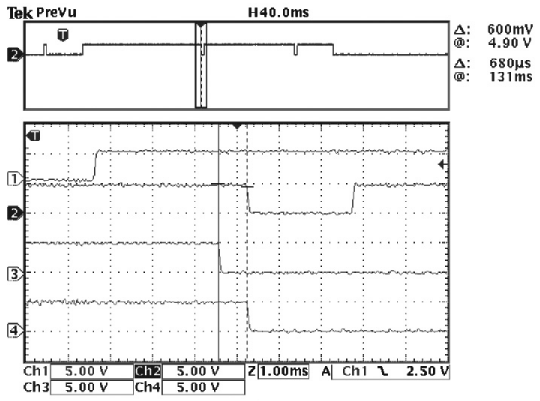
2. Simulation of link loss with activated fault-tolerant clock synchronization



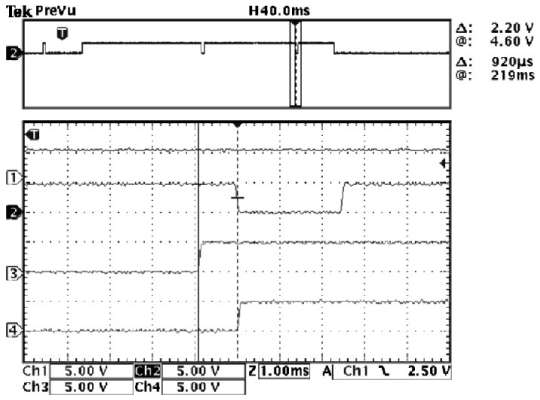
(a) Accuracy at the beginning of the round



(b) Accuracy after master frame



(c) Accuracy after slave frame



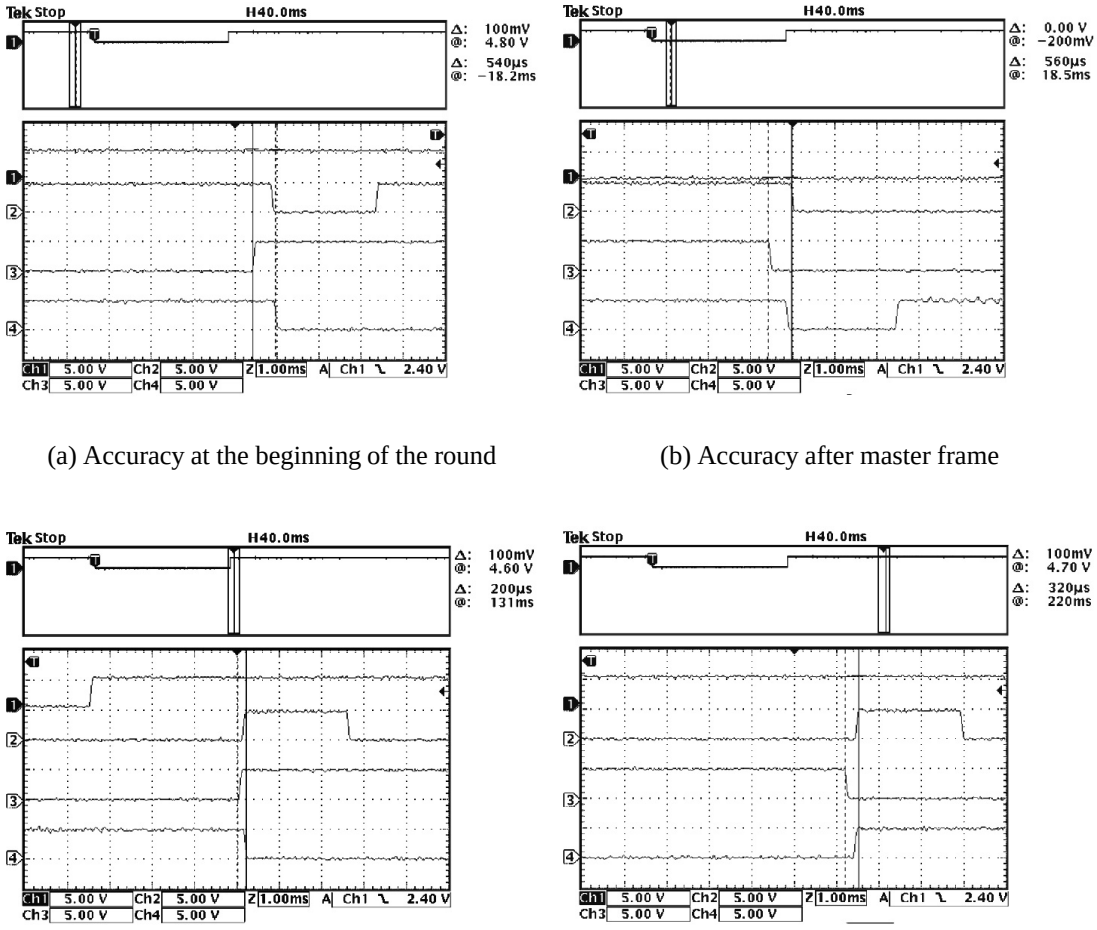
(d) Accuracy at the end of the round

Figure 5.5: Experiment with disabled fault-tolerant clock synchronization

The accuracy of the clock of slave C2 to the master clock was recorded via an oscilloscope at different points in time: before the simulated link loss, after the (ignored) master frame, after the slave frame sent by slave C1, and at the end of the round.

Figure 5.5 shows the results of the first measurement series. As illustrated in Figure 5.5(a), the accuracy of the degraded clock to the master's clock is 440 μs at the beginning of the round. This value results from an initial synchronization error and the clock drift. Due to the lack of synchronization by the master (Figure 5.5(b)), the accuracy degrades by time, leading to an inaccuracy of 920 μs after the whole communication round (see Figure 5.5(d)).

These measurements have been repeated, whereby the second time the fault-tolerant



(a) Accuracy at the beginning of the round

(b) Accuracy after master frame

(c) Accuracy after slave frame

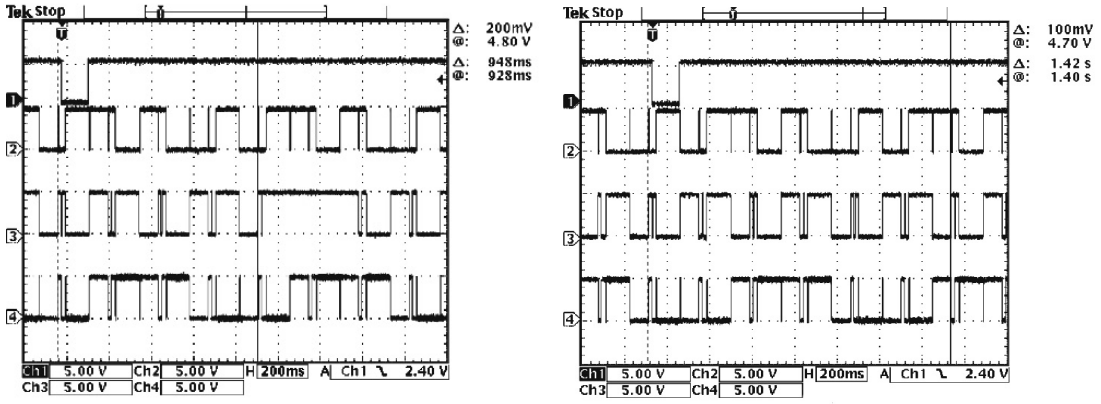
(d) Accuracy at the end of the round

Figure 5.6: Experiment with enabled fault-tolerant clock synchronization

clock synchronization has been enabled. The results are shown in Figure 5.6. Similar to the measurements above, the initial inaccuracy is 540 μs. Again, the slave is not synchronized by the master frame (Figure 5.6(b)). In contrast to the first experiment, the frame sent by slave C1 is used to adjust the clock of slave C2. A synchronization quality of 200 μs is achieved. Instead of continuously drifting away, the use of the fault-tolerant clock synchronization bounds the inaccuracy of slave C2's clock to 320 μs at the end of a complete communication round. Further measurements have shown that even several rounds without synchronization later, the accuracy has not degraded.

The quintessence of these measurements is depicted in Figure 5.7: In the left picture (fault-tolerant synchronization is disabled) slave C2 stops its communication to avoid the

violation of the timing schedule, and has to resynchronize again with the next successful received master frame. The right picture shows the behavior of slave C2, if the fault-tolerant synchronization method is activated. Due to the bounded inaccuracy of its clock, the slave can sustain communication without violating the timing schedule.



(a) Loss of synchronization after 5 rounds without link to master

(b) Ongoing communication after 5 rounds without link to master

Figure 5.7: Comparison of results without (left image) or with (right image) fault-tolerant clock synchronization

5.2 On-line Servo Calibration

The servos employed on the Smart Car are car servos, often used with radio-controlled model cars. They use a pulse-code modulated control signal to attain and keep the intended servo position. By convention, a 1.5 ms pulse holds the servo in a neutral position, a 1.0 ms or a 2.0 ms pulse turns the servo counter-clockwise or clockwise to it's maximum angle (see Figure 5.8), respectively.

This experiment shows the on-line maintenance and configuration support of the Smart Car demonstrator using the wireless protocol. The goal is to adjust the parameters which determine the timing of the control signal during operation, without the need to stop and re-program the Smart Car. Thus, the sweeping angles of the Smart Car's infrared sensors can be fine-tuned or even completely re-configured during the normal operation of the demonstrator.

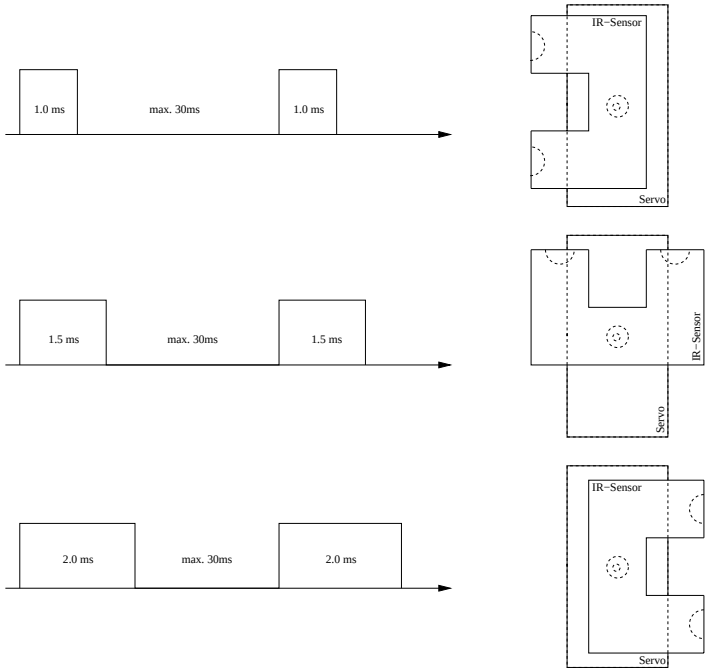


Figure 5.8: Timing diagram for the servo unit

5.2.1 Experiment Setup

Figure 5.9 shows the experiment setup. The servo control node is connected to the TTP/A bus of the Smart Car. The Smart Car master node, TTP/A master and wireless slave, schedules the TTP/A communication rounds of the Smart Car. Via the monitoring application on the host PC, calibration data is sent to the wireless master where the data is transformed into the actual messages of the wireless protocol and is finally scheduled for transmission.

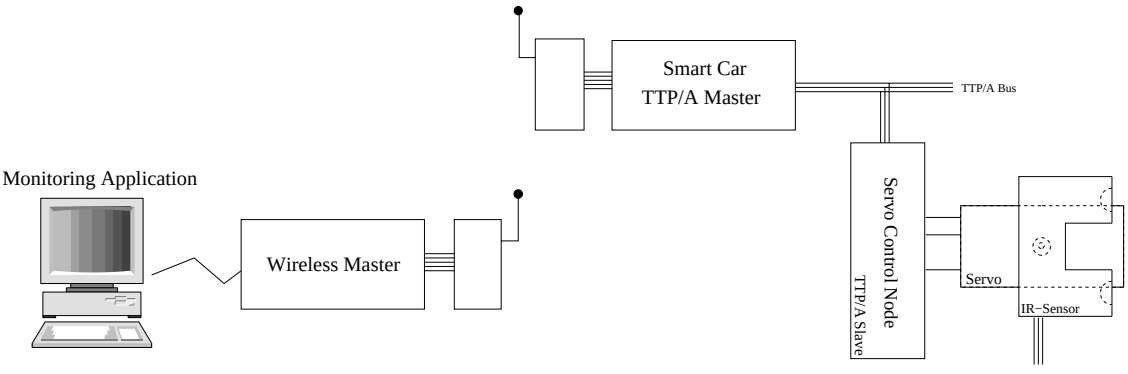


Figure 5.9: Experiment setup for on-line servo calibration

As mentioned in Chapter 2, all relevant data for the correct operation of a TTP/A node should be stored into its IFS. The intended angle, transmitted by the navigation node, and the timing parameters are stored in file 0x10 of the servo's IFS (see Figure 5.10). Thus, for the calibration of the servo node, the calibration parameters stored in the IFS have to be changed with appropriate write commands.

5.2.2 Calibration Background

In record 1 of file 0x10 only byte 0 is used for storing the set value of the servo angle. This record has to be separated from the calibration process because maintenance access to the IFS is done record-wise and a modification of the set value with an outdated value would lead to an unintended behavior of the servo node.

```
define FILE10
.global file10
file10:
    .byte   0x00, 0x00, 0x00, 0x00 // actual angle
    .byte   0x46, 0x9C, 0x3C, 0x00 // testangle, min_angle, max_angle
    .byte   0x48, 53,   45,   0x00 // calib_offset, grad_num, grad_denom,
                                   // raw_calib
.global file10_end
file10_end:
```

Figure 5.10: IFS contents for servo calibration

Record 2 consists of a testing angle as well as the minimum and maximum rotation angle. Due to physical limitations (sensor casing, limited space, etc.), the whole operation area of the servo cannot be used and therefore has to be limited by the software to avoid damages on the Smart Car hardware. The testing angle can be used, e. g., to check the current calibration with rotation angles that are easy to verify, e. g., zero degree that causes the servo looking straight ahead, or 90 degree in clockwise or counter-clockwise direction, respectively.

The third record holds the intrinsic calibration data. Equation 5.2 describes the transformation from an intended rotation angle to a timing value:

$$timeout = offset - (angle \cdot gradation) \tag{5.2}$$

Variable *offset* denotes the time that the control signal has to last at least to keep the servo in the zero degree position. The *gradation* denotes the amount of time that have to be added to or subtracted from the control signal pulse to change the servo rotation about

one degree. To permit the usage of integer arithmetics and to enable the storage of the gradation in the IFS, the gradation factor is split up in numerator and denominator (5.3).

$$timeout = offset - \frac{\text{angle} \cdot \text{grad\_num}}{\text{grad\_denom}} \tag{5.3}$$

Thus, the bytes 0 ... 2 in record 3 hold the values for offset, gradation numerator, and gradation denominator. The fourth byte of this record is used to enable the raw calibration mode. Writing a value different from zero causes the servo control node to use this byte as raw timeout value. This is very useful to determine, e. g., the offset value.

5.2.3 Calibration Process

The request list of the wireless master consists of two items, which are periodically scheduled (see Figure 5.11):
The first request demands the records one to three of file 0x10 from node 0x31 which is the logical name of the first servo control node. At the master the contents of this file are mapped into file 0x21 of the IFS page of the addressed slave, slave 0xA1 in that case.
The second request writes two records of the *IFSWriteBuffer* of the master node to the file specified above. That means, that the contents of the servo's IFS can be indirectly changed by overwriting the appropriate values in the master's IFS.

```
/* request 1 */
IFS_RequestList[8]  = 0xA1;            /* cluster name --> Smart Car   */
IFS_RequestList[9]  = 0x31;            /* node name    --> Servo1      */
IFS_RequestList[10] = (0x10<<2)|READ;  /* file 0x10, read request      */
IFS_RequestList[11] = 0x01;            /* start reading at record 1    */
IFS_RequestList[12] = 0x03;            /* read 3 records = 12 bytes    */
IFS_RequestList[13] = 0x21;            /* map data to master file 0x21 */
IFS_RequestList[14] = 0x01;            /* start record for mapping     */
IFS_RequestList[15] = 0x00;            /* last request flag = false    */

/* request 2 */
IFS_RequestList[16] = 0xA1;            /* cluster name --> Smart Car   */
IFS_RequestList[17] = 0x31;            /* node name    --> Servo1      */
IFS_RequestList[18] = (0x10<<2)|WRITE; /* file 0x10, write request     */
IFS_RequestList[19] = 0x02;            /* start writing at record 2    */
IFS_RequestList[20] = 0x02;            /* write 2 records = 8 bytes    */
IFS_RequestList[21] = 0x00;            /* not used in  write request   */
IFS_RequestList[22] = 0x04;            /* index into write buffer      */
IFS_RequestList[23] = 0x80;            /* last request flag = true     */
```

Figure 5.11: Request List of wireless master

These requests are only transmitted to the TTP/A master of the addressed slave, not directly to the addressed TTP/A slave. Therefore, these requests have to be scheduled to corresponding master-slave rounds of the TTP/A protocol. Figure 5.12 depicts the resulting communication demands, arising out of the two requests described above:

Receive file 0x10, record 1-3: Immediately after receiving the master frame, the master of the Smart Car TTP/A cluster adds the request to its local request buffer. If no other remote request is active, the next master-slave rounds are configured according to this request. If the entire requested data is ready before the start of the monitoring slot, the Smart Car master returns these values to the master. Otherwise, an *IFS_NOT_READY* [Elmenreich et al., 2002a] error message is returned. If the same request has been received earlier – this is quite likely due to the periodic scheduling of the stored requests at the master – the records received during the last master-slave rounds are returned.

Send file 0x10, record 1-2: This time, the configuration data is sent by the master. After the receiving of the data, the Smart Car TTP/A master acts as described above: The request is added to it's local request buffer and scheduled. This time, the request is carried out exactly once and is then removed from the request buffer.

The complete calibration process can be described as follows:

1. Use the *TTP/A Gateway* to display the values stored in the servo node's IFS (these values are mapped to file 0x21 of the Smart Car's TTP/A master).
2. Change the the *raw calibration value* by writing byte 3 of the wireless master's *IFSWriteBuffer* (file 0x24) to enter the calibration mode of the servo node.
3. Modify the *raw calibration value* to obtain the zero degree servo position and store it as calibration offset into the IFS (byte 0).
4. Use the *raw calibration value* to turn the servo to an easy to verify angle, e. g., 90 degree.
5. Calculate the *gradation* and store *numerator* and *denominator* in the according bytes of the IFS.
6. Explore the physical extreme values for the rotation angle by varying the *raw calibration value* and store the gathered information.
7. Use the *testing angle* to check the new calibration. Accordingly, set the *testing angle* to a valid value (value has to be between *minimum* and *maximum* rotation angle). Set the *raw calibration value* to zero to leave the calibration mode.

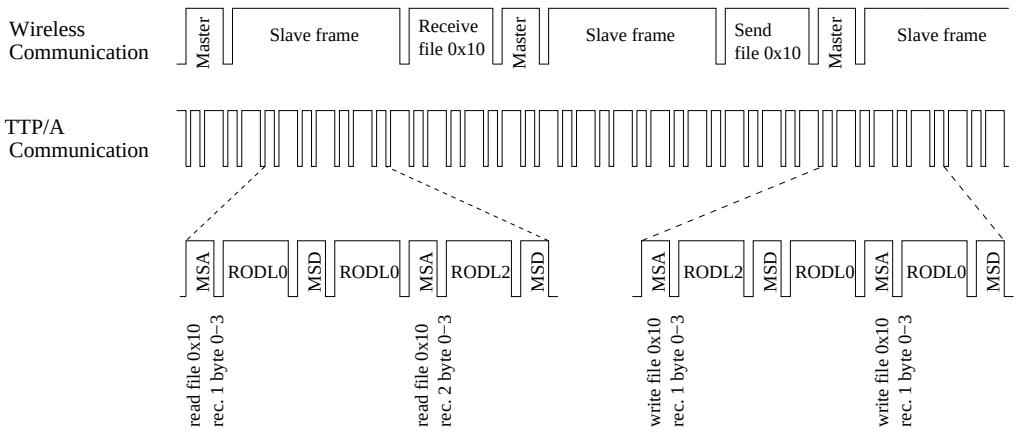


Figure 5.12: Scheduling of wireless protocol and TTP/A

8. Complete the testing phase by setting the *testing angle* outside the allowed range. Now, the servo node enters normal operation mode and uses the rotation angle specified by byte 0 of record 1.

5.2.4 Results

Due to the fact that the wireless protocol is not synchronized with the TTP/A protocol, the response or execution time of the receive or send requests depend on the state of the TTP/A protocol as well as on the execution state of the respective other request.

Response time of Monitoring Request

The request demands three records of the IFS of the servo node. Considering the Smart Car's *ROSE* file, the following schedule of TTP/A rounds is used:

$$\text{RODL } 0 \longrightarrow \text{MSA} \longrightarrow \text{RODL } 0 \longrightarrow \text{MSD} \longrightarrow \text{RODL } 0 \longrightarrow \text{MSA} \longrightarrow \text{RODL } 2 \longrightarrow \text{MSD}$$

Table 5.8 shows the duration of the TTP/A rounds listed above when using a TTP/A bus speed of 19200 Bit/sec. Thus, the collection of the requested data from the TTP/A slave node takes 82.0 ms in best case and 85.4 ms in worst case.
The length of a slave frame of the wireless protocol (see Table 5.9) at a wireless transmission speed of 9600 Bit/sec is given by 107.8 ms. Consequently, if this request is scheduled shortly after its reception, the data is available before the start of the monitoring frame and the Smart Car TTP/A master will respond correctly in the same round.
In worst case, the stored *configuration request* is started just before the end of the master

frame. It takes the TTP/A master 63 ms to store the two records received by the wireless master in the IFS of the servo node. In this case, the worst case overall time needed to process both requests is 148.4 ms which is longer than a single slave frame lasts. Thus, the wireless slave will either return an *IFS_NOTREADY* error message or the data, gathered during the previous execution of this monitoring request.

Round Type	Frames	Time
RODL 0	16 frames	10,8 ms
RODL 2	21 frames	14,2 ms
MSA	6 frames	4,1 ms
MSD	6 frames	4,1 ms

Table 5.8: Timing of different TTP/A rounds at 19200 Bit/sec bus speed

Frame Type	Bytes	Time
Master frame	10 bytes	31,2 ms
Slave frame	45 bytes	107,8 ms
Monitoring frame	34 bytes	83,7 ms
Gap time	—	1 ms

Table 5.9: Timing of different wireless frames

Execution time of Configuration Request

Due to the fact that writing to the remote IFS of the servo control node does not produce any response, the execution time of this request should be evaluated.

The same considerations as above, yield the following timing characteristics:
The best case occurs when no other request is active after the reception of the configuration data. In this case, the overall time between starting the request at the wireless master (start of master frame) and the effective change of all records in the servo's IFS results from:

$$t_{master} + t_{GAP} + t_{slave} + t_{GAP} + t_{monitoring} + t_{write} =$$
$$31.2\,ms + 1\,ms + 107.8\,ms + 1\,ms + 83.7\,ms + 63\,ms = 287.7\,ms$$

Variable t_{write} denotes the overall time to schedule two master-slave rounds including the interleaving multi-partner rounds.

In worst case, this time can be extended by a recently started *monitoring request*. Thus, the worst-case time needed to complete the record update is given by:

$$t_{master} + t_{GAP} + t_{slave} + t_{GAP} + t_{monitoring} + t_{read} + t_{write} =$$
$$32.5\,ms + 1\,ms + 108.7\,ms + 1\,ms + 82.5\,ms +$$
$$85.4\,ms + 63\,ms = 373.1\,ms$$

This calculation does not include the time needed to transmit the data from the *TTP/A Gateway* to the wireless master or vice versa. Using a relative high communication speed for this connection, in our case 115200 Bit/sec, the delay is held relative low: 1.1 ms for the transmission of 2 records (due to the communication overhead, each record results in 6 bytes) and 1.7 ms for the reception of 3 records, respectively.

5.3 Map Making Experiment

The final application of our case study is the generation of a global map of the robot's environment. Therefore, the temporary valid representations of the robot's visual field are joined using the covered distance and the steering angle between two succeeding sensor sweeps.

The map making software facilitates the distinction of multiple robots. It would be possible to concurrently use multiple mobile robots for the exploration of the environment.

5.3.1 Experiment Setup

The data gathered during a complete sensor sweep is used in two different ways (see Figure 5.13): On the one hand, the navigation node of the Smart Car applies a grid generation algorithm on it, executes a path planing algorithm on the result, and thereafter controls the actuators of the Smart Car according to the extracted information. On the other hand, the sensory information is transmitted via the TTP/A master (wireless slave) and the wireless master to a host PC. At this point, the same grid generation algorithm is applied to the data and the resulting grid is mapped into a global map of the robot's environment. Additionally, the generated local representation can be used to monitor and evaluate the decision making process of the navigation node.

The experiment has been carried out at the laboratory of the Institute for Computer Engineering (marked with *MM* in Figure 5.3).

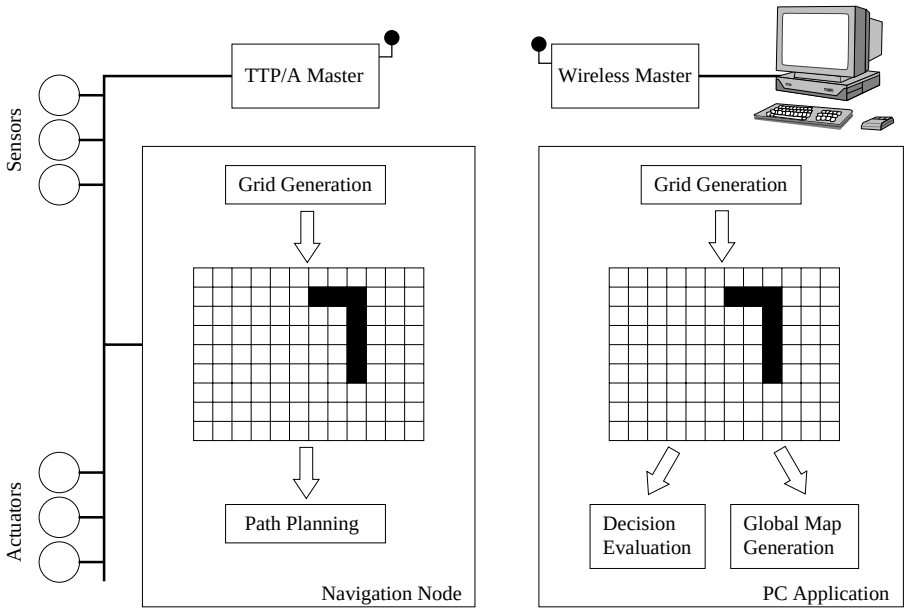


Figure 5.13: Data flow in map making experiment

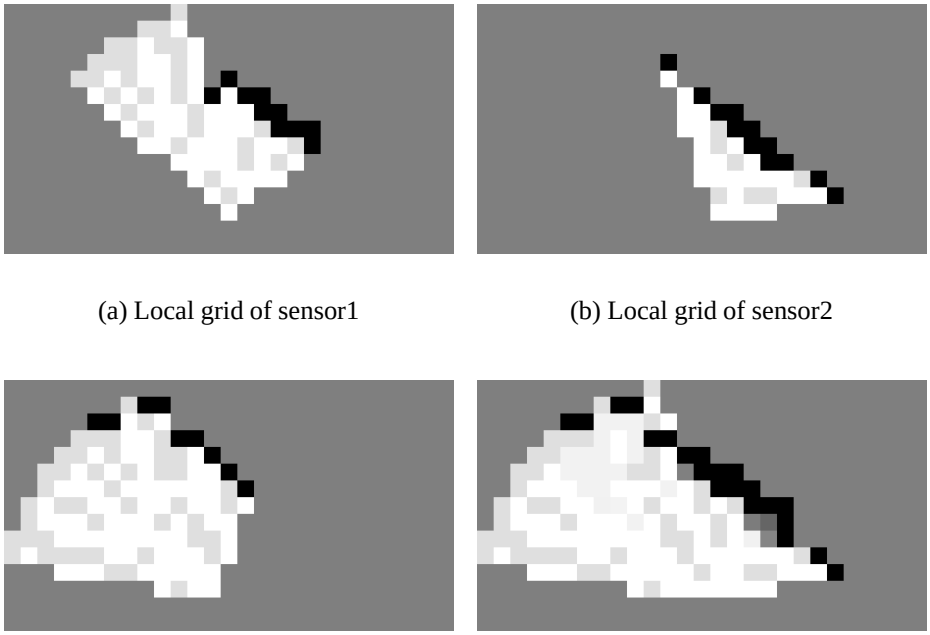
5.3.2 Local Map

For the generation of the local map, the data provided by the three infrared sensors is utilized. Each sensor covers a different part of the area in front of the robot. The viewing fields are selected in a way that a overlapping region between two neighbored sensors exists (see Figure 5.14). In these sections, a sensor fusion algorithm – the robust certainty grid algorithm in our case – can be applied to improve the overall result.

(a) Sensor 3 (b) Sensor 1 (c) Sensor 2

Figure 5.14: Viewing fields of the infrared sensors

Figure 5.15 shows an example of a local map: The figure depicts a wall, right in front of which the robot has been placed with an arbitrary angle. The resulting fused image shows that all sensors have created a very similar representation of the environment.



(a) Local grid of sensor1 (b) Local grid of sensor2

(c) Local grid of sensor3 (d) Result of sensor fusion

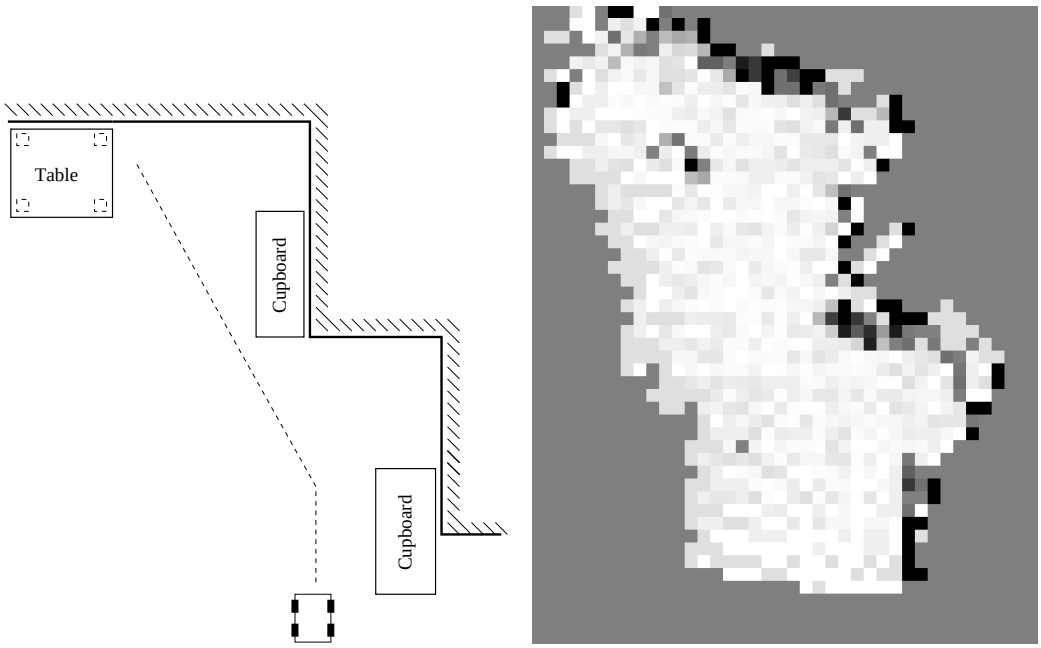
Figure 5.15: Representation of the robot's local environment

5.3.3 Global Map

As mentioned above, the experiment has been carried out indoors. Thus, the test conditions have been nearly ideal. The homogeneous soil condition and the planar floor assist the position encoder to deliver reliable measurements. High accuracy of the position encoder and a small deviation between intended and de facto achieved steering angle are crucial requirements for an accurate mapping of the individual local maps into the overall global representation of the environment.

In Figure 5.16(a) the area of the laboratory the robot had to explore is shown to scale. The starting point and the chosen path of the robot are marked in this image. In Figure 5.16(b) the result of the map making process is presented. The global map consists of 15 individual local maps which are joined together. Altogether, the information of 585 measurements has been used to generate this global map.

The map shown in Figure 5.16(b) clearly shows the drawback of the chosen map making approach: In the generated map, the wall besides to the robot in its end position is not orthogonal to the other wall. This map error results out of inaccuracies of the position encoding sensor and the steering servo which accumulate over time.
Another problem is the *event semantics* of the transmitted steering angle. If the steering



(a) To scale representation of the lab environment

(b) Generated global map of the lab environment

Figure 5.16: Result of the map making process

angle cannot be successfully transmitted between two succeeding sensor sweeps, the position of the robot cannot be recovered anymore. This problem could be easily solved by the application software that controls the autonomous robot, but this issue has not been considered during the implementation of the case study for the following reasons: The movement of the car and the sensor information gathering lasts longer than needed for the transmission of the sensory information to the wireless master. Therefore, there is plenty of time to transmit the information several times. Additionally, the protocol analysis has shown that data transmission errors are very seldom and multiple successive transmission errors are very unlikely.

There will come a time when you
believe everything is finished.
That will be the beginning.

LOUIS L'AMOUR

Chapter 6

Conclusion

The wireless communication protocol introduced in this thesis is well-suited for the interconnection of several, possibly mobile fieldbus networks or single smart transducer nodes. It is conceptual similar to TTP/A, a time-triggered fieldbus protocol for non safety-critical applications, which has been developed at the Vienna University of Technology. Thus, it is also based on periodic, statically defined communication rounds, which are controlled by a single dedicated master node. Similarly to TTP/A, the here described protocol reserves a certain amount of bandwidth for configuration and monitoring support to guarantee that the real-time service is not disturbed in any case.

The original TTP/A protocol is not well-suited for the use with a wireless communication medium. Many wireless receiver modules need a so-called *settle time* to synchronize themselves with the sender when being activated. During this time, no data can be transmitted. The short packet lengths of TTP/A would highly increase the communication overhead due to the permanent need of synchronization to different senders. Additionally, the synchronization mechanism of TTP/A requires a permanent connection of any slave to the master, which cannot be guaranteed in wireless mobile applications.

This leads to the main design decisions of the protocol: Several subsequent bytes from the same slave are subsumed to a frame in order to increase the size of the data chunks transmitted at once. Additionally, there are no special master/slave rounds as in TTP/A, but a certain amount of bandwidth is reserved for monitoring and configuration in every communication round.
Nevertheless, the main contribution of this thesis is the fault-tolerant synchronization mechanism. In general, the dedicated master node is responsible for synchronizing the slaves. In case of transient failures of the master or the communication channel or a permanent link failure to the master of a certain amount of slaves, the frames transmitted by the slaves are used for synchronization. Due to the static and a priori defined communication schedule, the timing of the master and the slave frames is known. Thus, the

deviation of the expected and the actual timing could be easily calculated and utilized for synchronization.

This rather simple and easy to implement synchronization mechanism is sufficient to keep the real-time service working, while the monitoring and configuration service on the other hand needs the addressing information of the master, and therefore, has to be stopped on slaves not connected to the master. This could be seen as a degraded service mode where the core service, the real-time service, is kept alive and the higher level service has to be stopped.
The practical relevance is shown by the carried out measurements. Even in the case of high transmission errors, e. g., caused by a transceiver module working slightly out of specification or a too high transmission range, the slave nodes have never lost the synchronization with the master.

The case study implementation shows the resource efficiency of the communication protocol. The complete case study has been built on 8-bit commercial-off-the-shelf microcontroller, which host the implementation of the wireless protocol and TTP/A as well as the application program. Due to the fact that no special hardware is needed, the implementation costs could be kept very low.

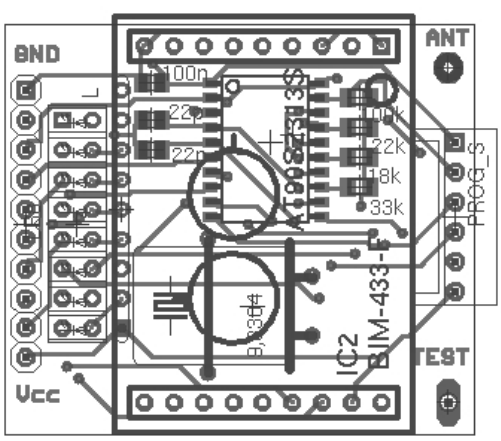
Out of the accomplished measurements, it is apparent that the presented implementation of the wireless communication protocol is well-suited for mobile free field or indoor applications with moderate communication demands like the map making application featuring an autonomous mobile robot implemented in this thesis.
Future improvements in the applied hardware, e. g., the use of an FPGA implementation for the encoding unit, could enhance the data transmission speed by a factor four to ten, depending on the version of the applied transceiver module. An enhancement of the maximum transmission distance or further speed gains could be achieved by the use of more powerful wireless transceivers. When using a different wireless module or even a different wireless technology, it should be considered that the module should provide low level access to the communication medium to avoid inconsistent bus access delays, which may disturb the real-time behavior.

The map making application generates a fairly accurate representation of the robot's environment. The emerging errors are mainly introduced by odometry inexactness and wrong sensor readings in regions where only one sensor is active and thus the fusion algorithms have no effect. The application is designed to support multiple robots, which then may concurrently explore the environment and share their information to cooperatively build a more detailed representation.

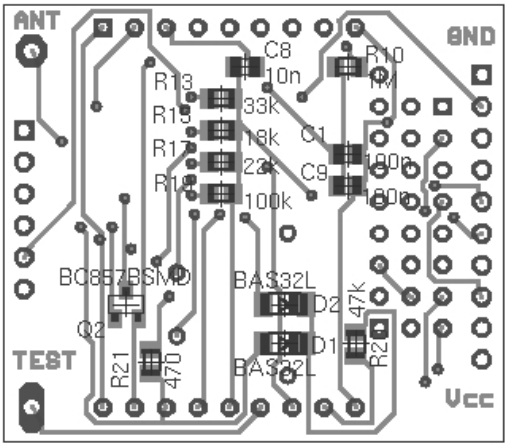
Appendix A

Layout of Wireless Module

A.1 Wireless Module Board Layout

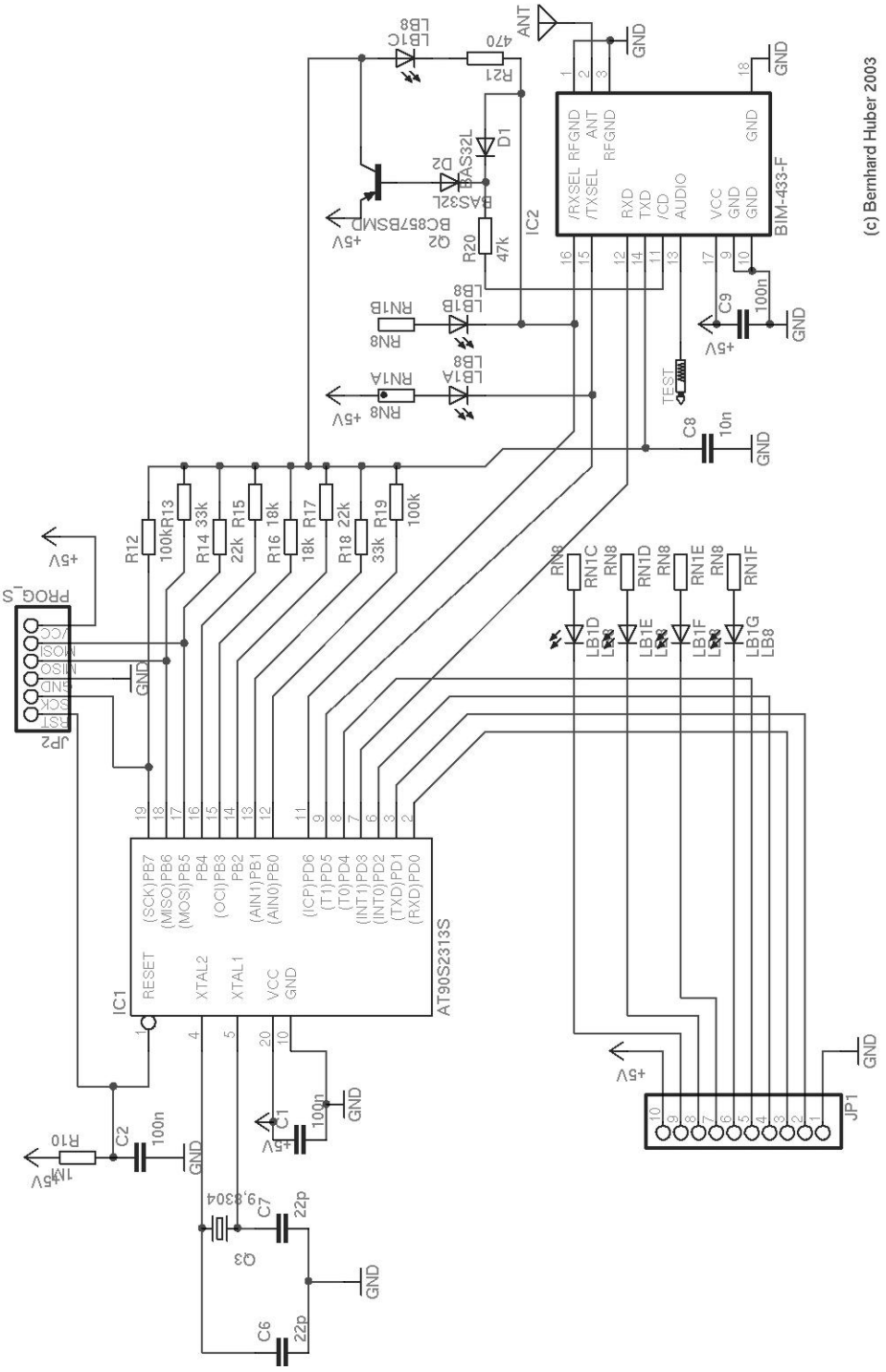


(a) Top View



(b) Bottom View

A.2 Wireless Module Schematic



Bibliography

M. Alves, E. Tovar, F. Vasques, G. Hammer, and K. Röther. Real-Time Communications over Hybrid Wired/Wireless PROFIBUS-based Networks. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems (ECRTS'02)*, June 2002.

A. Avižienis, J.-C. Laprie, and B. Randell. Fundamental Concepts of Dependability. Research Report 01-145, LAAS-CNRS, Toulouse, France, April 2001.

Bluetooth SIG. *Specification of the bluetooth system version 1.1*. Bluetooth SIG, core specification 1st edition, February 2001.

J. Borenstein, H. R. Everett, and L. Feng. Where am I? Sensors and Methods for Mobile Robot Positioning. Report, University of Michigan, April 1996.

J. Borenstein and Y. Koren. Real-Time Obstacle Avoidance for Fast Mobile Robots in Cluttered Environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 572–577, Cincinnati, USA, May 1990.

J. Borenstein and Y. Koren. Histogramic In-Motion Mapping for Mobile Robot Obstacle Avoidance. *IEEE Journal of Robotics and Automation*, 7(4):535–539, 1991a.

J. Borenstein and Y. Koren. The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288, June 1991b.

E. Bosse, J. Roy, and D. Grenier. Data Fusion Concepts Applied to a Suite of Dissimilar Sensors. In *Proceedings of 2nd Canadian Conference on Electrical and Computer Engineering*, volume 2, pages 692–695, May 1996.

J. F. Canny and M. C. Lin. An Opportunistic Global Path Planner. *Algorithmica*, 10: 102–120, 1993.

H. Choset, I. Konuksven, and A. Rizzi. Sensor Based Planning: A Control Law for Generating the Generalized Voronoi Graph. In *Proceedings of 8th International Conference on Advanced Robotics (ICAR '97)*, pages 333–338, Monterey, CA, July 1997.

I. C. Cox. Blance - An Experiment in Guidance and Navigation of an Autonomous Robot Vehicle. *IEEE Transactions on Robotics and Automation*, 7(2):193–204, April 1991.

J.-D. Decotignie. Wireless fieldbusses - A survey of issues and solutions. In *Proceedings of the 15th IFAC World Congress on Automatic Control*, Barcelona, Spain, 2002.

A. Dias and J. Irran. Documentation of the Smart Car Demonstrator. Research Report 45/2002, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2002.

W. Elmenreich. *Sensor Fusion in Time-Triggered Systems*. PhD thesis, NTechnische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2002.

W. Elmenreich, W. Haidinger, R. Kirner, T. Losert, R. Obermaisser, and C. Trödhandl. TTP/A Smart Transducer Programming - A Beginner's Guide. Research Report 33/2002, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2002a.

W. Elmenreich, W. Haidinger, H. Kopetz, T. Losert, R. Obermaisser, M. Paulitsch, and C. Trödhandl. Initial Demonstration of Smart Sensor Case Study. *DSoS Project (IST-1999-11585) Deliverable PCE3*, Apr. 2002b.

W. Elmenreich, W. Haidinger, P. Peti, and L. Schneider. New Node Integration for Master-Slave Fieldbus Networks. In *Proceedings of the 20th IASTED International Conference on Applied Informatics (AI 2002)*, pages 173–178, Feb. 2002c.

W. Elmenreich, L. Schneider, and R. Kirner. A Robust Certainty Grid Algorithm for Robotic Vision. In *Proceedings of the 6th IEEE International Conference on Intelligent Engineering Systems (INES)*, pages 25–30, Opatija, Croatia, May 2002d.

R. Frank. *Understanding Smart Sensors*. Artech House Inc., 2nd edition, 2000.

J. Gait. A Probe Effect in Concurrent Programs. *Softw. Pract. Exper.*, 16(3):225–233, 1986.

P. Grossmann. Multisensor Data Fusion. *The GEC Journal of Technology*, 15:27–37, 1998.

J. Hähniche and L. Rauchhaupt. Radio Communication in Automation Systems: the R-Fieldbus Approach. In *Proceedings of 3rd IEEE International Workshop on Factory Communication Systems (WFCS 2000)*, Porto, Portugal, Sep. 2000.

D. Hearn and M. P. Baker. *Computer Graphics with OpenGL*. Prentice Hall, 2003.

D. J. T. Heatley and I. Neild. Optical Wireless – The Promise and the Reality. In *IEE Colloquium on Optical Wireless Communication*, pages 1/1–1/6. IEE, London, UK, June 1999.

B. Huber and W. Elmenreich. Wireless Time-Triggered Real-Time Communication. In *Proceedings of the 2nd Workshop on Intelligent Solutions in Embedded Systems*, Graz, Austria, Jun. 2004.

IEEE. *Information Technology – Telecommunications and Information Exchange between Systems – Local and Metropolitan Area Networks – Specific requirements; Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE, ISO/IEC 8802.11:1999(E); ANSI(IEEE Sd. 802.11) edition, 1999.

J. Joyce, G. Lomow, K. Slind, and B. Unger. Monitoring distributed systems. *ACM Transactions on Computer Systems*, 5 (2), May 1987.

H. Kopetz. Should Responsive Systems be Event-Triggered or Time-Triggered? *Institute of Electronics, Information, and Communications Engineers Transactions on Information and Systems*, E76-D(11):1325–1332, 1993.

H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston, Dordrecht, London, Jan. 1997. ISBN 0-7923-9894-7.

H. Kopetz, M. Holzmann, and W. Elmenreich. A Universal Smart Transducer Interface: TTP/A. *International Journal of Computer System Science & Engineering*, 16(2), March 2001.

H. Kopetz et al. Specification of the TTP/A Protocol V2.00. Research Report No 61/2001, Department of Computer Science, Vienna University of Technology, Vienna, Austria, September 2002.

S. Krywult and C. Steiner. TTP/A Monitoring CORBA Gateway. Unpublished, 2004.

A. Kutlu, H. Ekiz, and E. Powner. Wireless Control Area Network. In *IEE Colloquium on Networking Aspects of Radio Communication Systems*, pages 3/1–3/4. IEE, London, UK, March 1996.

J.-C. Laprie. Dependability: Basic Concepts and Terminology. *Dependable Computing and Fault Tolerant Systems*, 5:257–282, 1992. Springer Verlag, Vienna.

C. H. LeDoux and D. S. Parker, Jr. Saving traces for Ada debugging. In *Proceedings of the 1985 annual ACM SIGAda International Conference on Ada*, pages 97–108, 1985.

J. J. Leonard and H. F. Durrant-Whyte. Mobile robot localization by tracking geometrc beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, 1991.

T. Losert and R. Obermaisser. Wireless Real-Time Communication Technologies: A Comparative Study. In *Proceedings of the IEEE Workshop on Real-Time Embedded Systems*, London, United Kingdom, Dec. 2001.

C. E. McDowell and D. P. Helmbold. Debugging Concurrent Programs. *ACM Computing Surveys*, 21, No.4, Dezember 1989.

V. Mhatre and C. Rosenberg. Design Guidelines for Wireless Sensor Networks: Communication Clustering and Aggregation. *Ad Hoc Networks*, 2(1):45–63, Jan 2004.

H. P. Moravec. Sensor Fusion in Certainty Grids for Mobile Robots. *AI Magazine*, 9(2): 61–74, 1988.

P. Morel and A. Croisier. A Wireless Gateway for Fieldbus. In *Proceedings of 6th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, volume 1, pages 105–109, Toronto, Ont. Canada, Sep 1995.

OMG. *Smart Transducers Interface V1.0*. Object Management Group, Needham, MA, US, 2003. available at `http://doc.omg.org/formal/2003-01-01`.

A. W. Paeth. A fast Algorithm for General Raster Rotation. In *Proceedings on Graphics Interface '86/Vision Interface*, pages 77–81. Canadian Information Processing Society, 1986.

P. Peti. Monitoring and Configuration of a TTP/A Cluster in an Autonomous Mobile Robot. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2001.

P. Peti, R. Obermaisser, W. Elmenreich, and T. Losert. An Architecture supporting Monitoring and Configuration in Real-Time Smart Transducer Networks. In *Proceedings of the IEEE Sensors 2002*, volume 2, pages 1479–1484, June 2002.

T. S. Rappaport. *Wireless communication - Principles and Practice*. Prentice Hall Communications Engeneering and Emerging Technologies Series. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1996.

W. D. Rencken. Concurrent Localisation and Map Building for Mobile Robots using Ultrasonic Sensors. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2129–2197, Yokohama, Japan, July 1993.

D. Roberts. 'OLCHFA' a Distributed Time-Critical Fieldbus. In *IEE Colloquium on Safe Critical Distributed Systems*, pages 6/1–6/3. IEE, London, UK, Oct 1993.

R. Schlatterbeck and W. Elmenreich. TTP/A: A Low Cost Highly Efficient Time-Triggered Fieldbus Architecture. *SAE World Congress 2001, Detroit, Michigan, USA*, March 2001.

L. Schneider. Real Time Robot Navigation with a Smart Transducer Network. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2001.

C. Taylor. Building representations for the environment of a mobile robot from image data. In *Proceedings of the 1991 SPIE Conference on Mobile Robots*, pages 331–339, Boston, MA, Nov 1991.

C. Trödhandl. Architectural Requirements for TTP/A Nodes. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, May 2002.

J. M. Trojer. Implementierung eines Kalman Filters zur Positionsbestimmung eines Modellautos. Bachelor's thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2004.