

UNIVERSITY OF VAASA

FACULTY OF TECHNOLOGY

TELECOMMUNICATIONS ENGINEERING

Tobias Glocker

**SOFTWARE AND HARDWARE DESIGN OF A MINIATURIZED MOBILE
AUTONOMOUS ROBOT, OPERATING IN A WIRELESS SENSOR
NETWORK**

Master's thesis for the degree of Master of Science in Technology submitted for inspection, Vaasa, 26 May, 2010.

Supervisor

Professor Mohammed Salem Elmusrati

Instructor

M. Sc. (Tech.) Reino Virrankoski

ACKNOWLEDGEMENT

I would like to take the opportunity to thank those people who shared their knowledge for helping me to complete my thesis. First, I would like to thank Professor Mohammed Elmusrati and Reino Virrankoski for the excellent guidance during my thesis work. I am very thankful to our laboratory engineers, Veli-Matti Eskonen and Juha Miettinen, for providing me a nice working environment with all the necessary equipment. Thanks also to Jani Ahvonen for the introduction course of circuit board manufacturing. Lastly, I offer my regards to all of those who supported me in any respect during the completion of this thesis work.

TABLE OF CONTENTS	Page
ACKNOWLEDGEMENT	2
ABBREVIATIONS	5
ABSTRACT	8
1. INTRODUCTION	9
2. THE MAIN PRINCIPLES OF WIRELESS SENSOR NETWORKS	11
2.1. IEEE 802.15.4 Standard	11
2.2. Network Topologies of Wireless Sensor Networks	12
2.3. Beacon Enabled Networks and Non-Beacon Networks	14
2.4. Robustness	15
2.5. Security	16
2.6. Distance Estimation and Localization Methods in WSNs	17
3. WIRELESS SENSOR NODES	20
3.1. Common Requirements	20
3.2. User Datagram Protocol	21
3.3. Internet Control Message Protocol	22
3.4. Internet Protocol version 6	23
3.5. 6LoWPAN and ZigBee	25
4. HARDWARE AND SOFTWARE DESIGN OF THE ROBOT	27
4.1. Hardware	27
4.1.1. Hardware of the Mobile Platform	27
4.1.2. Hardware Platform of the Wireless Node	28

4.1.3.	Hardware Platform of the Embedded PC	29
4.1.4.	Hardware Design of the Robot	30
4.1.5.	Motor Control and Infrared Sensors	34
4.2.	Software	37
4.2.1.	Software Design of the Mobile Platform	37
4.2.2.	Software Design of the Wireless Node	39
4.2.3.	Software Design of the Embedded Linux PC	43
4.2.4.	Memory Mapping	54
4.3.	Computation of the Wheel Speeds	54
5.	GRAPHICAL CONTROL USER INTERFACE	56
6.	EXPERIMENTS	61
6.1.	Indoor RSSI Measurements	61
6.2.	Distance Estimation with the Mobile Robot	64
6.3.	Indoor RSSI Measurements II	66
7.	CONCLUSIONS AND FUTURE WORK	68
	REFERENCES	69
	APPENDIXES	73
	APPENDIX 1. Voltage Regulator for Sensinode	73
	APPENDIX 2. Voltage Regulator for UART connection	74
	APPENDIX 3. Extension Board for Sensinode	75
	APPENDIX 4. Main Board	76
	APPENDIX 5. Shell Scripts	78
	APPENDIX 6. Makefile	82
	APPENDIX 7. Class Diagrams of the Main Program (Embedded PC)	84
	APPENDIX 8. Class Diagrams of the Graphical Control User Interface	86
	APPENDIX 9. Picture of the Mobile Robot	87

ABBREVIATIONS

ACL	Access Control List
ADC	Analog-to-Digital Converter
AES	Advanced Encryption Standard
AoA	Angle of Arrival
ASCII	American Standard Code for Information Interchange
CAP	Contention Access Period
CH	Cluster Head
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA-CA	Carrier Sense Multiple Access with Collision Avoidance
ECCP	Enhanced Capture/Compare/PWM
EEPROM	Electrical Erasable Programmable Read-Only Memory
FFD	Full Function Device
GCUI	Graphical Control User Interface
GPS	Global Positioning System
GTS	Guaranteed Time Slot
HTML	HyperText Markup Language
ICMP	Internet Control Message Protocol
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
I ² C	Inter-Integrated Circuit
IOCTL	Input/Output Control
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
kB	Kilo Byte
LED	Light-Emitting Diode
LPS	Local Positioning System
LSTTL	Low-power Schottky Transistor-Transistor-Logik
MIPS	Million Instructions Per Second

MTU	Maximum Transmission Unit
NFS	Network File System
NwK	Network
OOP	Object Oriented Programming
OS	Operating System
OSI	Open System Interconnection
PAN	Personal Area Network
PCB	Printed Circuit Board (Layout)
PDF	Portable Document File
PIO	Parallel Input/Output
PWM	Pulse Width Modulation
RAM	Random Access Memory
RF	Radio Frequency
RFC	Request for Comments
RFD	Reduced Function Device
RSSI	Radio Signal Strength Indication
RTLS	Real Time Location System
RTOS	Real Time Operating System
RTT	Round Trip Time
SHC	Simple Hierarchical Clustering
SoC	System on Chip
SPI	Serial Peripheral Interface
SRAM	Shadow Random Access Memory
STL	Standard Template Library
ToA	Time of Arrival
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
UI	User Interface
USART	Universal Synchronous and Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
WISM	Wireless Sensor Systems in Indoor Situation Modeling
WPAN	Wireless Personal Area Network

WSN	Wireless Sensor Network
ZC	ZigBee Coordinator
ZDO	ZigBee Device Object
ZED	ZigBee End Device
ZR	ZigBee Router

UNIVERSITY OF VAASA**Faculty of Technology**

Author: Tobias Glocker
Topic of the Thesis: Software and Hardware Design of a Miniaturized Mobile Autonomous Robot, Operating in a Wireless Sensor Network
Supervisor: Professor Mohammed Salem Elmusrati
Instructor: Reino Virrankoski
Degree: Master of Science in Technology
Department: Department of Computer Science
Degree Programme: Degree Programme in Information Technology
Major of Subject: Telecommunications Engineering
Year of Entering the University: 2006
Year of Completing the Thesis: 2010 **Pages: 87**

ABSTRACT

Nowadays wireless nodes are becoming more and more popular in the field of localization. Thanks to the high research effort in this area, wireless sensors become more and more sophisticated. From year to year the accuracy in terms of distance estimation increases. In comparison to other localization devices like a Local Positioning System (LPS) or Global Positioning System (GPS), the wireless nodes are considered as a cheap alternative. The Finnish defence department, police and fire department support current research activities within this area, in the hope that they will get beneficial applications.

The target of this Master's Thesis "Software and Hardware Design of a Miniaturized Mobile Autonomous Robot, Operating in a Wireless Sensor Network" was the construction of miniaturized autonomous robot acting within a Wireless Sensor Network (WSN). The robot consists of an Embedded Linux PC, a wireless node and a mobile platform that are connected with each other. In this Master's Thesis we describe the software and hardware tasks that were necessary for the interaction between the three mentioned components. We also discuss the software implementation for the communication between the wireless nodes and the results of the distance measurements.

KEYWORDS: WSN, Miniaturized Mobile Autonomous Robot, IEEE 802.15.4,
NanoStack

1. INTRODUCTION

In recent years, the development of Wireless Sensor Networks has caused an exponential growth. Wireless Sensor Networks (WSN) have led this trend due to the reduction of the noise level with high sophisticated antennas and better coding techniques, that achieve a better accuracy in distance estimation and localization. In addition, Wireless Sensor Networks are extendable. According to Yu, Prasanna & Krishnamachari (2006: 1) “[they] can be deployed on a global scale for environmental monitoring and habitat study, over a battle field for military surveillance and reconnaissance and in emergent environments for search and rescue”.

Common localization methods used for autonomous robots are Local Positioning Systems (LPS) and Global Positioning Systems (GPS). Local Position Systems and Global Positioning Systems have a quite good accuracy but both are very expensive in comparison to a Wireless Sensor Network consisting of many wireless sensor nodes.

The aim of this thesis was to design and build an miniaturized autonomous robot for the Wireless Sensor Systems in Indoor Situation Modeling (WISM) project. One part of this project deals with distance estimation and localization within a Wireless Sensor Network. To evaluate already developed Algorithms for distance estimation and localization based on their precision, the constructed miniaturized autonomous robot should be guided inside a WSN (see **Figure 1**). The static nodes are deployed in a two dimensional environment and they first localize themselves. When a sensor of a static node detects an event, it will send a message to the robot (mobile node). Since the estimated locations of the static nodes are known, the robot needs to compute a path which leads it to the target node.

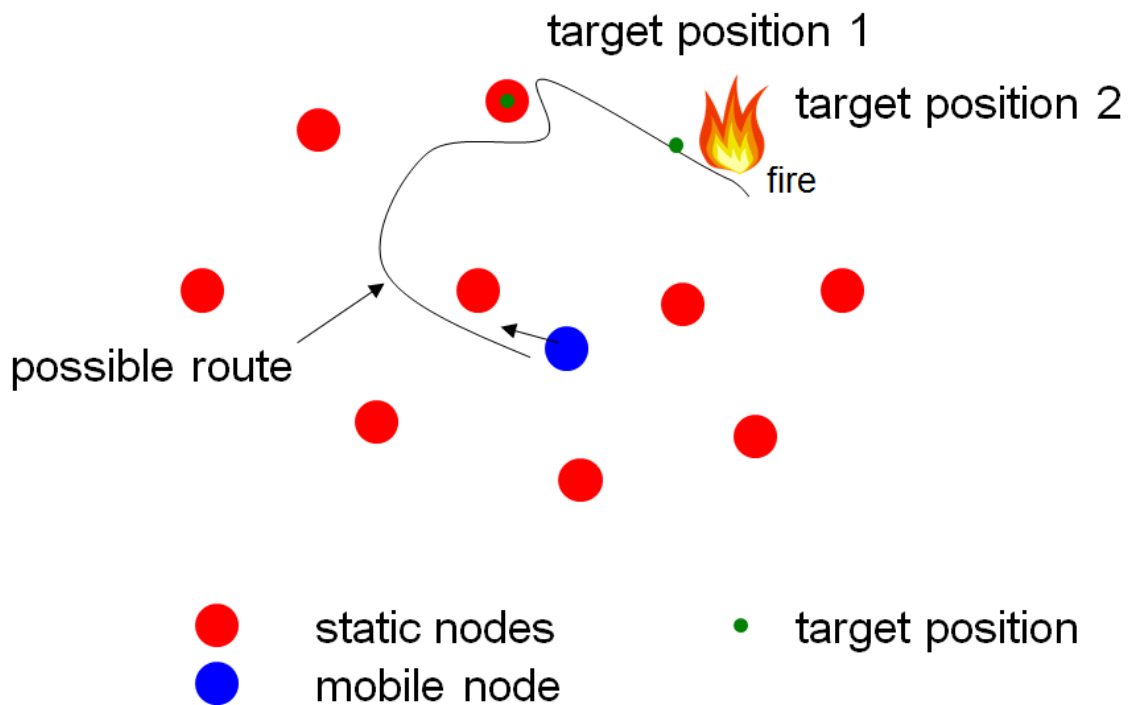


Figure 1. Wireless Sensor Network.

The thesis consists of seven chapters. In the first three chapters, the theory of Wireless Networks is represented. After the theoretical introduction follows the main part of the thesis, that shows the concept of the hardware and software design of the robot. Chapter five describes the Graphical Control User Interface with which the robot can be configured and controlled remotely. The experiments are represented in chapter 6. Opinions and further tasks regarding this project will be given in the last chapter "Conclusions and Future Work".

2. THE MAIN PRINCIPLES OF WIRELESS SENSOR NETWORKS

The main principles of WSNs will be discussed in this chapter. Section 2.1 introduces the IEEE 802.15.4 Standard. An overview of different network topologies is given in section 2.2. The following sections highlight different operation modes in WSNs, methods for the achievement of robustness and security mechanisms. Different localization methods are illustrated in the last section of this chapter.

2.1. IEEE 802.15.4 Standard

The IEEE 802.15.4 standard was finalized by the Institute of Electrical and Electronics Engineers (IEEE) in October 2003. It covers the physical layer and Medium Access Control (MAC) of a low-rate Wireless Personal Area Network (WPAN). This standard was mainly defined for WSNs, home automation, home networking and home security. Most of the previous mentioned applications require low bitrates, delays within certain time intervals and less power consumption. Moreover, the physical layer provides three different bitrates that are shown in **Table 1**. (Karl & Willig 2005: 139–140.)

Table 1. 802.15.4 Frequency Bands.

Frequency	Data Rate
2400 MHz – 2485 MHz	250 kbps
905 MHz – 928 MHz	40 kbps
868 MHz – 868,6MHz	20 kbps

2.2. Network Topologies of Wireless Sensor Networks

In an 802.15.4 network there are two types of devices, Full Function Devices (FFDs) and Reduced Function Devices (RFDs). A FFD can operate as Personal Area Network (PAN) coordinator, as simple coordinator or as device. It must have enough memory to store the necessary routing information of the network. A RFD is intended for small tasks, where no high amount of data is sent to the network. Mostly, they are acting as sensors. (Callaway 2004: 296.)

Every device (RFD) must be assigned to a coordinator (FFD) with which it communicates. A star network consists of one coordinator and several devices. In a star network the devices act as communication endpoints while the coordinator routes the data around the network. Coordinators can communicate in a peer-to-peer fashion, so that multiple coordinators can extend the PAN which is shown in **Figure 2**.

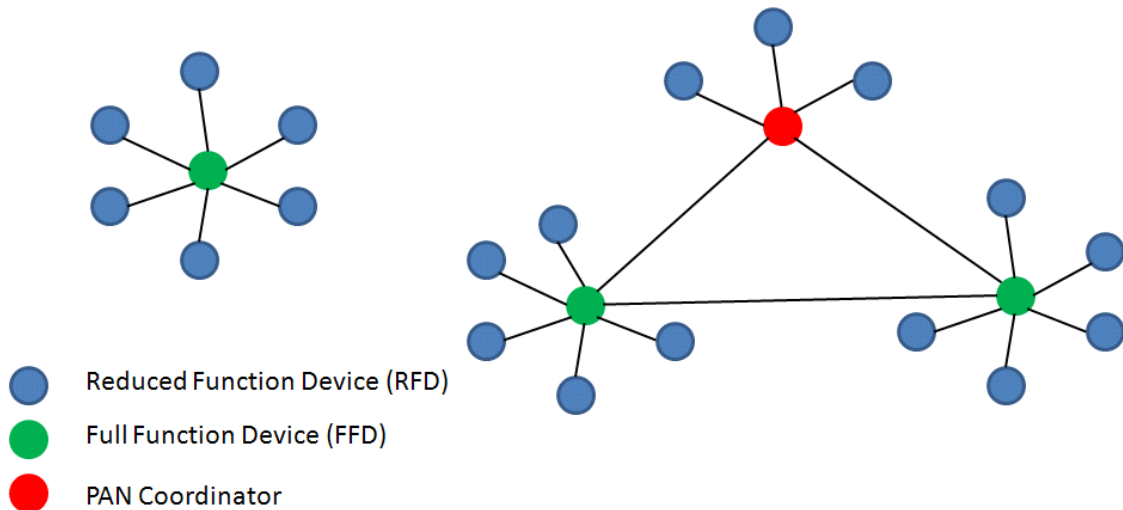


Figure 2. a) PAN with Star Topology. b) Extended PAN with three Star Networks.

Every PAN has a unique 16 bit PAN identifier within a certain radio range, so that more star networks can communicate independently from each other. One of the coordinators is designated to be the PAN coordinator. Once the PAN coordinator has selected the PAN identifier, other FFDs and RFDs can join the network.

Beside small WPANs, also larger networks can be built by using the Cluster-Tree-Topology (see **Figure 3**).

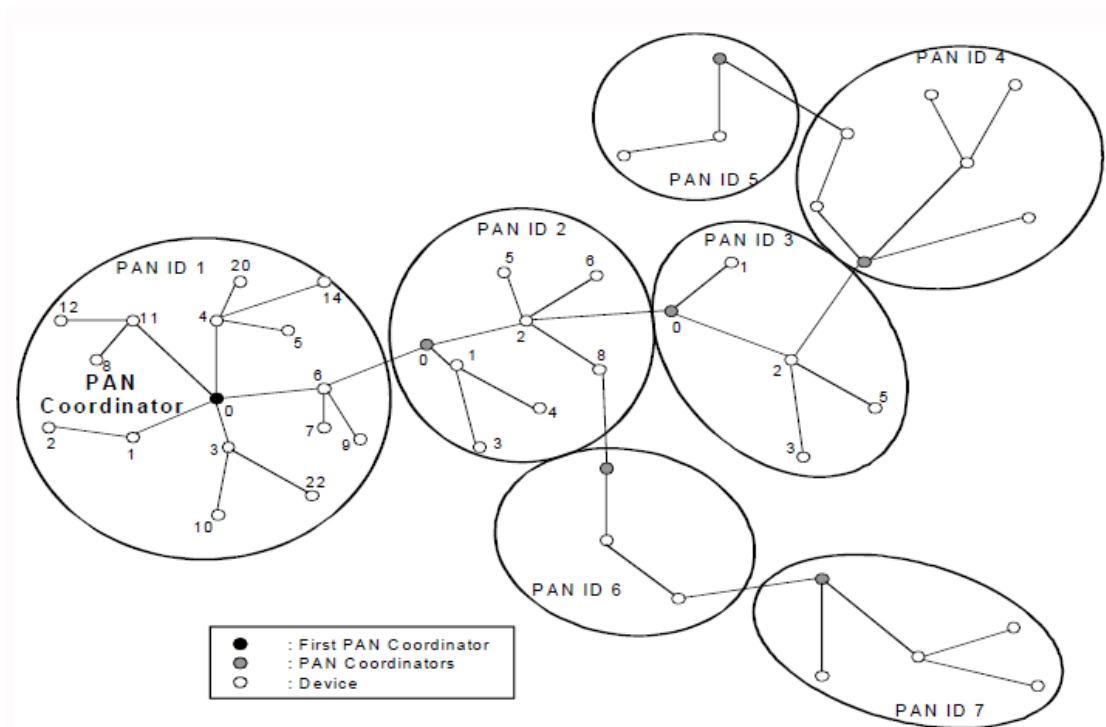


Figure 3. Cluster-Tree Topology (IEEE 2006).

In Simple Hierarchical Clustering (SHC), the PAN coordinator forms the first cluster by broadcasting beacon frames with its PAN ID. When a node receives a beacon it will join the network. The PAN coordinator also called “Cluster Head (CH) 0” adds the new node as a child node to its neighbor list and the joined node adds the PAN coordinator

as a parent node to its neighbor list. The task of the new joined node is to forward the continuously broadcasted beacon frames from the PAN coordinator to other nodes that may join this cluster network. If the PAN coordinator cannot accept any further node due to the fact that the hop count is met, it will instruct a node to become a coordinator (Cluster Head) of a new cluster. This process will proceed until all the nodes have joined the network. (Bandara & Jayasumana 2007.)

2.3. Beacon Enabled Networks and Non-Beacon Networks

There are two operation modes specified in IEEE 802.15.4, a beacon enabled mode and a non-beacon mode. In the beacon enabled mode the coordinator of a star network organizes the channel access with the help of a superframe structure. The size of each superframe is equal. Every superframe starts with a beacon packet that contains a superframe specification describing the length of the active and inactive period. During the inactive period the transceivers from all devices can be switched off.

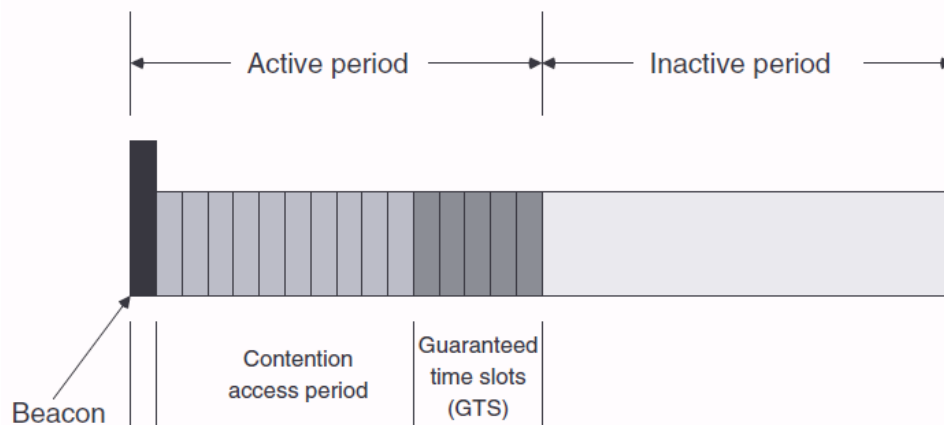


Figure 4. Superframe structure of IEEE 802.15.4 (Karl & Willig 2005: 141).

As it is shown in **Figure 4**, the active period is divided into 16 time slots. The first time slots build the Contention Access Period (CAP) followed by a number of Guaranteed Time Slots (GTSs). While the coordinator is active during the entire active period, a device is only active in the GTS timeslot that is allocated to it. Within the CAP phase a device can go to the sleep mode if it has no data to transmit to the coordinator. It is to mention, that the length of the active and inactive period and the length of a single timeslot is adjustable.

Compared to the beacon enabled mode, the non-beaconed mode works without beacon and GTS frames. Due to the missing beacon frames, no time synchronization is available. For that reason the coordinators must be switched on permanently while the devices can schedule their operating times by themselves. A device wakes up if it has data to transmit or when it has to fetch a packet from the coordinator. (Karl & Willig 2005: 141–145.)

2.4. Robustness

Reliability plays the most essential role in WSNs. Every transmitted packet should be received correctly. To achieve reliability the robustness of the WSN must be increased. There are some methods that provide a better robustness. One of them is the data verification method. In this method a Cyclic Redundancy Check (CRC) is used on every frame to detect bit errors. A better robustness can also be achieved by using optional frame acknowledgement for MAC frames. When a node forwards a frame, it expects to receive an acknowledgement. However, this method is not suitable for small frames due to the high amount of acknowledgements that create a significant overhead. (Karl & Willig 2005: 377–379.)

Another improvement of the robustness can be obtained with the Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) channel access mechanism. Depending on the operation mode, either slotted CSMA-CA or unslotted CSMA-CA is used. Beacon enabled networks work with the slotted CSMA-CA mechanism, while the non-

beacon enabled networks work with the simpler, unslotted CSMA-CA mechanism. (Misic, Fung & Misic 2005.)

2.5. Security

In IEEE 802.15.4, security becomes more important due to the fact that WSNs are more and more used in areas where a high security standard is required. There are four security objectives to consider. One objective is the access control. Every device has an Access Control List (ACL) that contains all valid devices from which it can receive frames. Frames received from unauthorized devices are rejected. Another objective is the data encryption which is achieved by using a symmetric cipher, i.e., Advanced Encryption Standard (AES) provided on beacon payloads, command payloads and data payloads. Only the devices that share the secret key can encrypt and decrypt the network messages. The third objective is the frame integrity used to assure that the received messages haven't been manipulated by an intruder during the transmission. Sequential Freshness, the fourth objective, is a protection against replayed frames. The receiver checks if the sequence number of the current frame is higher than the previous one. If not it will reject the received frame. (Xiao, Sethi, Chen & Sun 2005.)

Sharing a private key in a WSN can be done according to one of the following Keying Models. In the Network Shared Keying model, a single network-wide shared key is used. Although the key management for this method is quite trivial, the Network Shared Key model is more vulnerable than other keying models. A better security provides the Pairwise Keying model where each pair of nodes share a different key. In comparison to the Network Shared Key model, it is more secure but it requires more memory because all the keys for each node pair must be saved. The Group Keying model is a compromise between the previous mentioned keying models. A single key is shared among a set of nodes forming a group. All the nodes within that group, encrypt and decrypt messages with this key. Normally, the nodes will be assigned to the groups based on their location. (Sastry & Wagner 2004.)

2.6. Distance Estimation and Localization Methods in Wireless Sensor Networks

The need to locate objects or humans within a WSN has always been an important part not only for the industry but also for the army, police and health care. Real Time Location Systems (RTLS) consist of a localization engine and a set of nodes that are needed by the localization engine to determine the position of a node. There are several RTLS methodologies which can be used. One of them is the Angle of Arrival (AoA) method that determines the direction of the propagated Radio Frequency (RF) signal. The direction of the RF signal can be measured by using sensitive antennas on the receiver side so that the direction of the transmitter can be obtained. **Figure 5** illustrates how the AoA is ascertained.

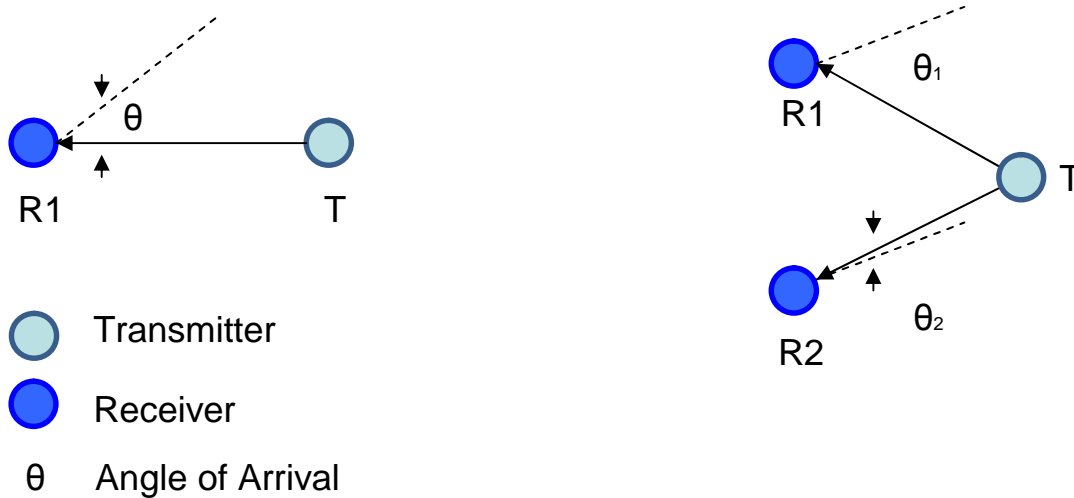


Figure 5. a) AoA method.

b) Determining the position of a transmitter between two receivers with AoA.

To localize a transmitter between two receivers with the AoA method, the positions of both receivers must be known. The transmitter sends a signal to both receivers so that the AoA can be computed for each receiver. After the computation, the results must be

forwarded to a centralized point, where the position of the transmitter is determined. The main advantage of this method is that the computations can be done by using simple triangulation. Nowadays, many nodes have a location engine which takes care of the whole computation process.

Another method that can be applied for measuring the distance between the transmitter and receiver is the Time of Arrival (ToA) method. The distance between transmitter and receiver can be obtained by measuring the propagation delay of the radio signal. Measuring the propagation delay in only one direction requires very accurate clock synchronizations between transmitter and receiver. Due to time deviations it is very difficult to keep the transmitter's clock and receiver's clock synchronized. Time synchronization can be avoided by measuring the Round Trip Time (RTT) displayed in **Figure 6**.

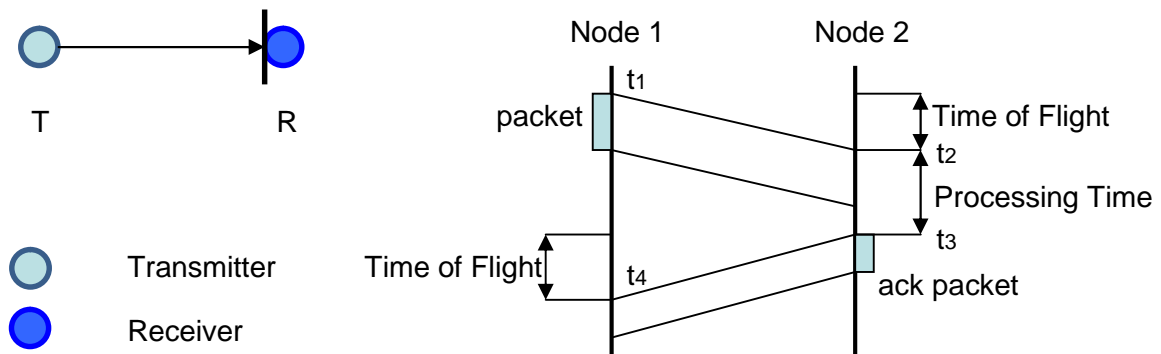


Figure 6. a) ToA method.

b) ToA with RTT.

According to Kuijsten (2008) the RTT is ascertained as follows.

$$d = \frac{RTT}{2} \times c = \frac{(t_4 - t_1) - (t_3 - t_2)}{2} \times c \quad (1)$$

d = Distance between Transmitter and Receiver

c = Speed of Light 300000 km/h

t_1, t_2, t_3 and t_4 = time intervals (see Figure 6)

To localize a node with the ToA method in a two dimensional plane, at least three successful distance measurements to different nodes within the communication range must be done. **Figure 7** represents the localization with the ToA method. Due to the multipath propagation problem, accurate measurements with this method can only be achieved when a line-of-sight connection between transmitter and receiver is given.

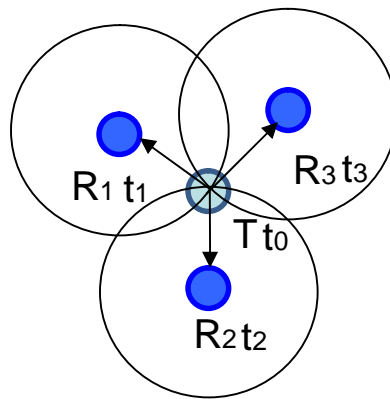


Figure 7. Localization with ToA.

There is also the possibility to determine the distance between two nodes by measuring the value of the signal strength. This value is called Received Signal Strength Indication (RSSI) value. The main advantage of this method is that it requires only a one direction measurement without synchronizing the transmitter's and receiver's clock. However, the problem with RSSI based systems is that an adequate underlying path-loss model must be found for non-light-of-sight and non-stationary environments. In terms of security considerations, RSSI measurements can be manipulated by increasing or decreasing the signal strength. (Nanotron 2007: 1–8.)

3. WIRELESS SENSOR NODES

In this chapter, the first section describes the common requirements for wireless sensor nodes. The sections 3.2, 3.3 and 3.4 explain the User Datagram Protocol, Internet Control Message Protocol, the Internet Protocol version 6 and 6LoWPAN that are parts of the Sensinode's NanoStack. Furthermore, section 3.5 compares 6LoWPAN with ZigBee.

3.1. Common Requirements

Increasing the use of wireless sensor nodes requires a successful design with several unique features. These features must have interesting technical issues that are not found in other wireless networks. An important feature is the size of a wireless sensor node. Smaller nodes are more applicable than larger ones, but they limit also the size and the capacity of the batteries. Hence, it is very essential that the power consumption of a wireless node is low. To reduce the power consumption the node should only be in the active state when it has to process data. According to Callaway (2004: 48–49) the average power consumption is determined by the following formulas.

$$I_{avg} = T_{on} \times I_{on} + (1 - T_{on}) \times I_{stby} \quad (2)$$

$$P_{avg} = U \times I_{avg} \quad (3)$$

I_{avg} = Average current drain

T_{on} = Fraction of time either receiver or transmitter is on

I_{on} = Current drain from the battery when either the receiver or transmitter is on

I_{stby} = Current drain from the battery when both transmitter and receiver are off

P_{avg} = Average power consumption

U = Battery voltage

The manufacturing costs play also a significant role for wireless sensor nodes. To fulfill this objective, the design of the node must avoid the need of high-cost components such as discrete filters. Furthermore, every node should have the capability to configure and maintain itself when it joins a WSN. Another fundamental requirement is to protect the WSN against hackers and sniffers. For that reason it is essential that a wireless sensor node provides security mechanisms to make the transmission of data more secure. (Callaway 2004: 11–15.)

3.2. User Datagram Protocol

For all delivery the User Datagram Protocol (UDP) uses the Internet Protocol (IP). UDP does not detect or correct delivery problems, meaning that messages can be lost, duplicated, delayed, delivered out-of-order or corrupted. By using this protocol the applications running on top of the UDP must be immune against those problems mentioned in the previous sentence or the application programmer needs to take care of it. Four Styles of interaction are supported by UDP. An application can choose a 1-to-1 interaction if it wants to send a message to another application, a 1-to-many interaction for sending a message to multiple recipients, or a many-to-1 interaction for receiving messages from multiple applications. For exchanging messages between a set of applications, a many-to-many interaction can be established. Each UDP message is called a user datagram that consists of two parts. The first part contains the information that is needed for sending and receiving UDP packets, and the second part carries the message data. **Figure 8** illustrates the format of the user datagram. Field “Source Port” holds a 16 bit port number of the sending application and field “Destination Port” holds the 16 bit port number of the receiving application. The total size of the UDP message is stored in the “Message Length” field. To check the correct transmission of the packet a “Checksum” is added to the UDP header. (Comer 2009: 420–424.)

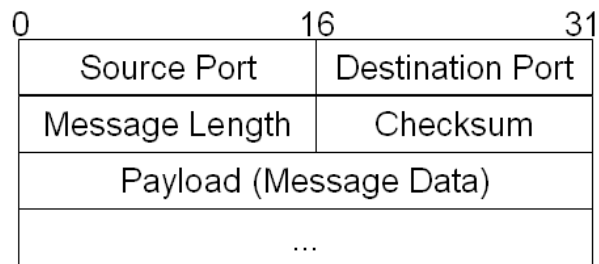


Figure 8. UDP with an 8-octet header.

3.3. Internet Control Message Protocol

From time to time a network error occurs. The notification of a network error is under the responsibility of the Internet Control Message Protocol (ICMP), specified in RFC792. ICMP has the task to transport error and diagnostic data in an Internet Protocol network. Due to the transport of various information, only the basic structure of this protocol is fixed. **In Figure 9** an ICMP protocol header is displayed.

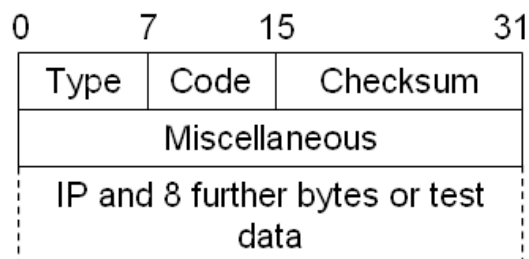


Figure 9. ICMP Protocol Header.

Field “Type” specifies the ICMP packet type. There are more than twenty ICMP messages that have been defined, but only a few of them are used. **Table 2** lists the key ICMP messages and their purposes.

Table 2. Examples of ICMP messages with the message number and purpose (Comer 2009: 390).

Number	Type	Purpose
0	Echo Reply	Used by the ping program
3	Dest. Unreachable	Datagram could not be delivered
5	Redirect	Host must change a route
8	Echo	Used by the ping program
11	Time Exceeded	TTL expired or fragments timed out
12	Parameter Problem	IP header is incorrect
30	Traceroute	Used by the traceroute program

The “Code” field is used to indicate subfunctions within a type. Furthermore, the ICMP packet contains also a 16 bit checksum. Field “Miscellaneous” holds information for miscellaneous purposes like sequence number, Internet Address, and so on. The “Internet Protocol header” contains the triggering and the first eight bits of the transport message. (Santifaller 1994: 45–48.)

3.4. Internet Protocol version 6

The Internet Protocol (IP) is the cornerstone of the Transport Control Protocol (TCP) / IP Architecture (Santifaller 1994: 19). It was developed due to dramatic changes in the hardware technology and extreme increases in scale. To handle heterogeneous networks, the IP provides services that allow applications on different devices to communicate with each other, regardless of the underlying hardware structure. IP applies the “best-effort” method for forwarding packets to the next destination. It is a connectionless protocol with the capability of packet fragmentation and the use of a network-independent addressing scheme. In comparison to the previous version, Internet Protocol version 4 (IPv4), the Internet Protocol version 6 (IPv6) uses larger

addresses and an entirely new datagram header format. In addition, IPv6 divides header information into a series of fixed-length headers. **Figure 10** shows the format of the Internet Protocol version 6 header.

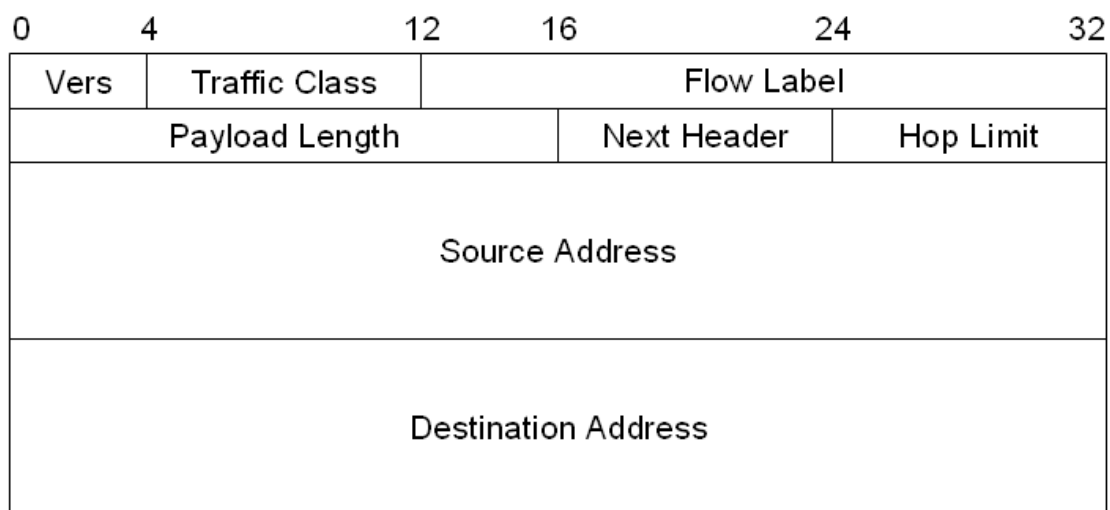


Figure 10. Format of the base header in an IPv6 datagram.

The “Vers” field identifies the version of the Internet Protocol. Field “Traffic Class” specifies the priority of the packet. For giving real-time applications a better service the “Flow Label” field was created. At the moment this field is unused. Compared to IPv4, the “Payload Length” field holds only the size of data being carried without the header size. Field “Next Header” specifies the next encapsulated protocol. To avoid that packets stay in the network forever, a “Hop Limit” field was added to the header. If the hop limited counts down to the value zero, the packet will be discarded. Most of the header space occupies the “Source Address” and “Destination Address” that are needed to identify the original source and the ultimate recipient.

3.5. 6LoWPAN and ZigBee

6LoWPAN is a specification (RFC4944) made by the Internet Engineering Task Force (IETF) group. It was specified to allow the compressed use of IPv6 and UDP standards over IEEE 802.15.4 networks under the consideration of limited bandwidth, memory and energy resources. RFC4944 defines a Mesh Addressing header that is needed to support sub-IP forwarding. Beside the Mesh Addressing header, RFC4944 defines also a Fragmentation header to fulfill the minimum requirements for the Maximum Transmission Unit (MTU), specified in RFC2460. LoWPAN_HC1 and LoWPAN_HC2 are two compression formats that reduce large IPv6 and UDP headers down to several bytes. (Hui & Thubert 2009.)

ZigBee is a protocol stack that is based on the standard Open System Interconnection (OSI) seven-layer model. But it defines only those layers which are relevant for the intended market space. The ZigBee protocol stack consists of a network (NwK) layer and the framework for the application layer, which includes the application support sub-layer, the ZigBee Device Objects (ZDO) and the manufacturer-defined application objects. Three topologies are supported by the NwK layer. One of these topologies is the star topology that is introduced in chapter 2. For the extension of the network, either the tree or the mesh topology can be applied. In a ZigBee network there are three different types of devices, the ZigBee Coordinator (ZC), the ZigBee Router (ZR) and the ZigBee End Device (ZED). Building a network requires a root device with the capability to establish a network. This root device is the so called ZC. The ZigBee Router's tasks are routing of messages, scanning the network and assigning addresses to other ZRs or ZEDs. A ZED is a RFD that can act as a sensor for example. Hence, it is only able to join a network, leave a network and to transmit data to other ZigBee devices. (ZigBee Alliance 2005: 17–18.)

Figure 11 shows a comparison between 6LoWPAN and ZigBee. Both are built on top of the IEEE 802.15.4 protocol that is described in chapter 2.

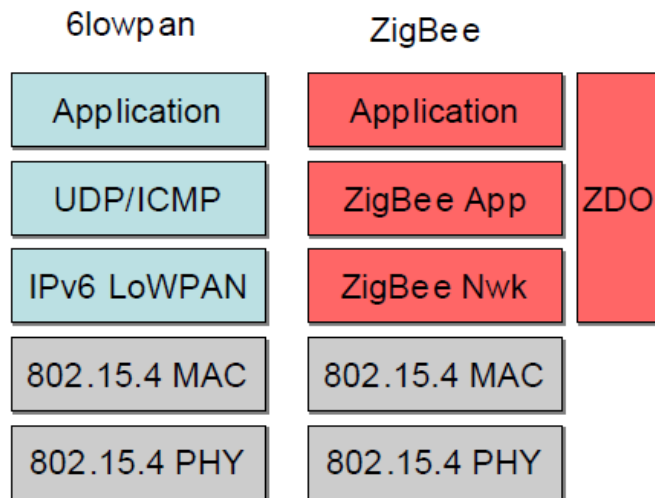


Figure 11. 6LoWPAN versus ZigBee stack comparison (Sensinode 2008).

4. HARDWARE AND SOFTWARE DESIGN OF THE ROBOT

4.1. Hardware

The robot consists of a mobile platform, a wireless sensor node and an Embedded Linux PC. The hardware of the mobile platform is described in section 4.1.1. In the following sections the hardware of the wireless sensor node and the hardware of the Embedded PC will be explained. Section 4.1.4 shows how the hardware parts are connected with each other. In section 4.1.5 the motor control and the reading of the infrared sensors are discussed.

4.1.1. Hardware of the Mobile Platform

The mobile platform (see **Figure 12**) used in this thesis is a product of Matrix Multimedia. It mainly consists of a PIC18F4455 microcontroller, a motor driver chip (L293D), a microphone with sound level amplifier circuit, three distance sensors, a light sensor, eight user definable LEDs, an external 5 V power supply and a Universal Serial Bus (USB) interface for programming the microcontroller. Four AA batteries inside the plastic chassis supply the two motors and the circuit board with power. The maximum speed that can be reached is 20 cm per second. (Matrix Multimedia.)

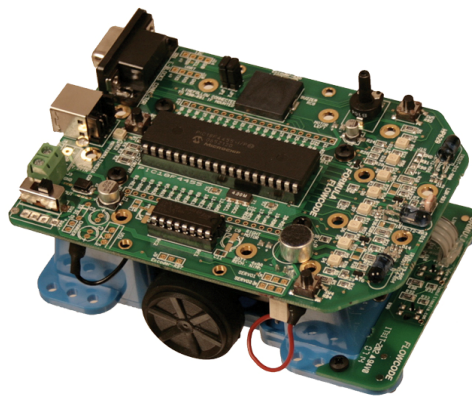


Figure 12. Formula Flowcode Buggy (Matrix Multimedia).

The PIC18F4455 is a microcontroller produced by microchip that handles up to 12 Million Instructions Per Second (MIPS). It provides an enhanced USART interface, a Serial Peripheral Interface (SPI) interface and an Inter-Integrated Circuit (I²C) interface. Four Timer modules, an Enhanced Capture/Compare/Pulse Width Modulation (ECCP) module and a 10 bit Analog-to-Digital Converter (ADC) are integrated in this chip. (Microchip 2009: 1.)

4.1.2. Hardware Platform of the Wireless Node

The wireless nodes used in this thesis are manufactured by Sensinode in Oulu, Finland. A wireless node consists of an integrated Radiocrafts RC2301AT module with the size of 12.7 x 25.4 x 2.5 mm. This module is complete ZigBee-ready and has an IEEE 802.15.4 transceiver. It is a so called System on Chip (SoC) module containing a high performance 8051 microcontroller core with 128 kilo byte (kB) flash memory, 8 kB Shadow Random Access Memory (SRAM), 4kB Electrically Erasable Programmable Read-Only Memory (EEPROM) and an 8 channel 14 bit Analog-to-Digital Converter (ADC). For the communication with the outer world, the module provides 19 digital and analog Input/Output pins, a Universal Asynchronous Receiver/Transmitter (UART), SPI and a debug interface. (Radiocrafts 2007.)

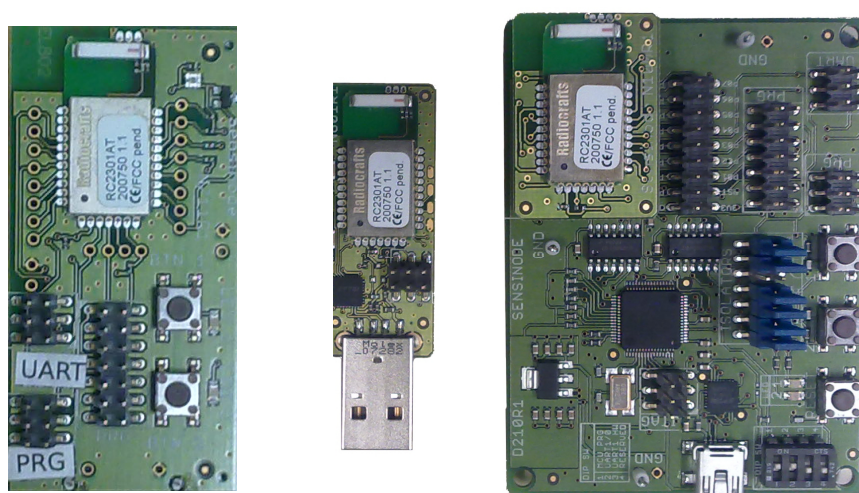


Figure 13. Sensinode NanoSeries Node Designs. N711, N601 and D210 with N100.

Sensinode offers different types of node designs for the NanoSeries which are shown in **Figure 13**. The N100 module consists only of the Radiocrafts RC2301AT chip which can be used in own manufactured circuit boards. The NanoSensor N711 board includes besides the RC2301AT chip a temperature sensor, light sensor, battery holder, two Light-Emitting Diodes (LEDs) and two pushbuttons. For logging wireless network data on the computer, Sensinode provides the NanoRouter N601 board with Universal Serial Bus (USB) interface. Moreover, data can be transmitted from the computer to the wireless network. The wireless nodes are programmed with the D210 development board.

4.1.3. Hardware Platform of the Embedded PC

Portux920T is a single board computer with an AT91RM9200 CPU from Atmel. CPU-Modules with ARM architecture are suitable for embedded systems with high performances and low power consumption. The size of the Portux board is 100 x 71 mm. With a clock rate of 180 MHz the ARM processor can solve complex computations within a short time interval. It also offers a variety of integrated peripherals such as USB 2.0, Ethernet, SPI and four Universal Synchronous and Asynchronous Receivers/Transmitters (USARTs). Both flash memory (16 MB) and Random Access Memory (RAM) (64 MB) are big enough to run Embedded Linux on this ARM processor. The baseboard comes with a 10/100 Mbit/s Ethernet interface and with two serial interfaces. It contains a Secure Digital (SD)/Multimedia (MMC) Card Slot for memory extension. Additional peripherals can be connected with the Portux Extension Bus (PXB). **Figure 14** gives an overview of the main interfaces on the base board. (Taskit.)

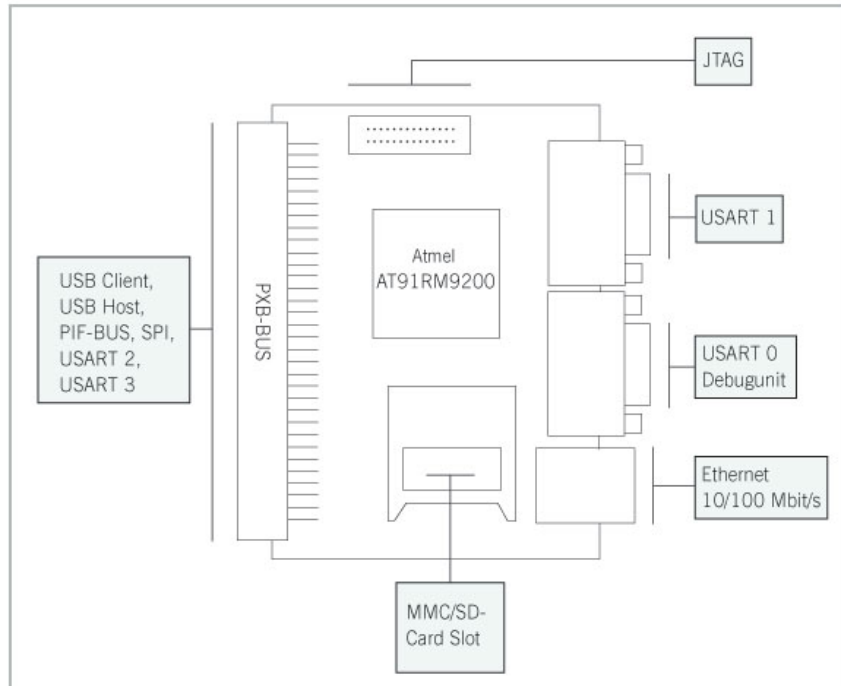


Figure 14. Overview of the main interfaces on the base board Portux920T (Taskit).

4.1.4. Hardware Design of the Robot

One task of this thesis was to design a circuit board that connects the hardware components mentioned in the previous sections. The Embedded PC acts as Master Device that communicates with the wireless node and the microcontroller on the mobile platform via USART. An overview of the circuit board is given in **Figure 15**. USART 3 of the Embedded PC is directly connected with the UART of the wireless sensor node. Both devices work with the same voltage level (3.3 V). USART 2 is used for the connection between Embedded PC and the microcontroller (PIC18F4455) on the mobile platform. Due to the different voltage levels on both sides, the transmitter of the microcontroller cannot be connected directly with the receiver of the Embedded PC. Thus it was necessary to build a voltage regulator that regulates the microcontroller's transmitter output from 5 V to 3.3 V. An 11.1 V Li-PO battery with 1300 mAh supplies the circuit board with power. The Embedded PC has its own voltage regulator but not the wireless node. For that reason a voltage regulator board needs to be developed to

reduce the battery's 11.1 V output voltage to 3.3 V. There are four push buttons on the board that are connected with the Embedded PC.

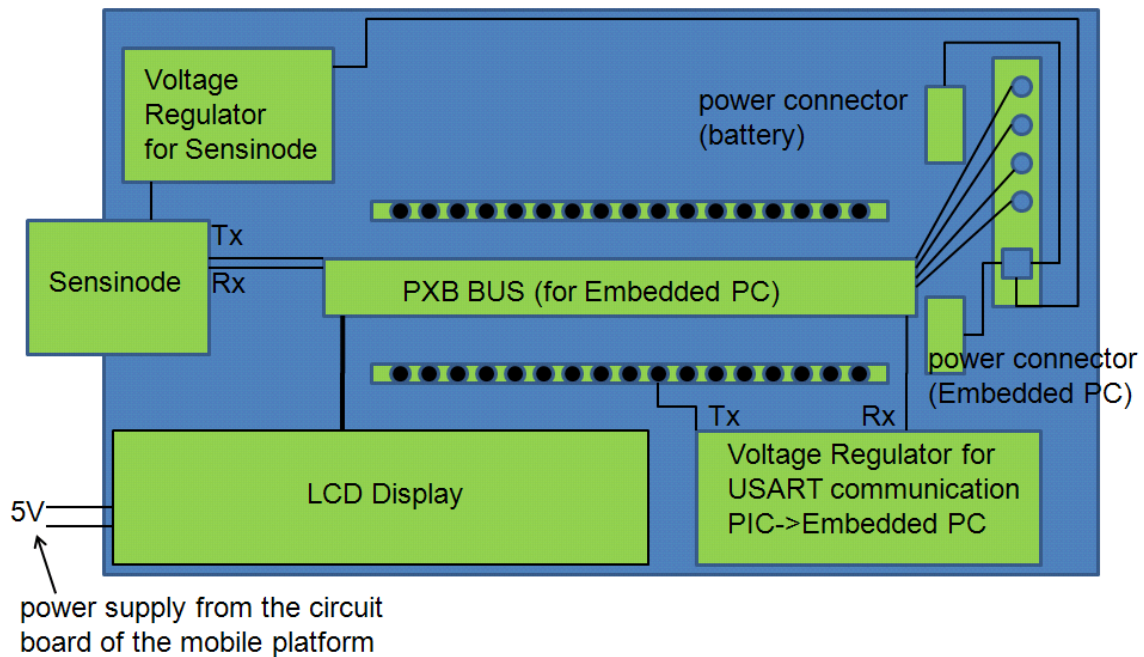


Figure 15. Overview of the designed circuit board.

The voltage regulator board for the wireless node consists of a LM2937ET voltage regulator, a ceramic capacitor with a capacity of 1 μF at the input circuit and an electrolytic capacitor with a capacity of 1 μF at the output circuit. The LM2937 is a positive voltage regulator capable of supplying up to 500 mA of load current. It can be supplied with a continuous input voltage up to 26 V (National Semiconductor 2000). In

Figure 16 the Schematic of the voltage regulator board is illustrated.

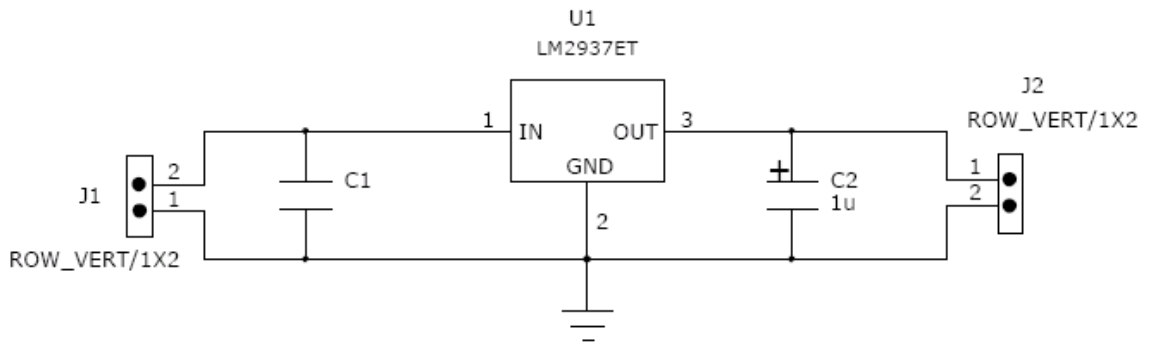


Figure 16. Schematic of the UART voltage regulator board.

The voltage regulator board for the USART communication between the microcontroller and the Embedded PC consists of the same components as the voltage regulator board for the wireless node and of a 74HC244 chip which regulates the voltage from 5 V to 3.3 V.

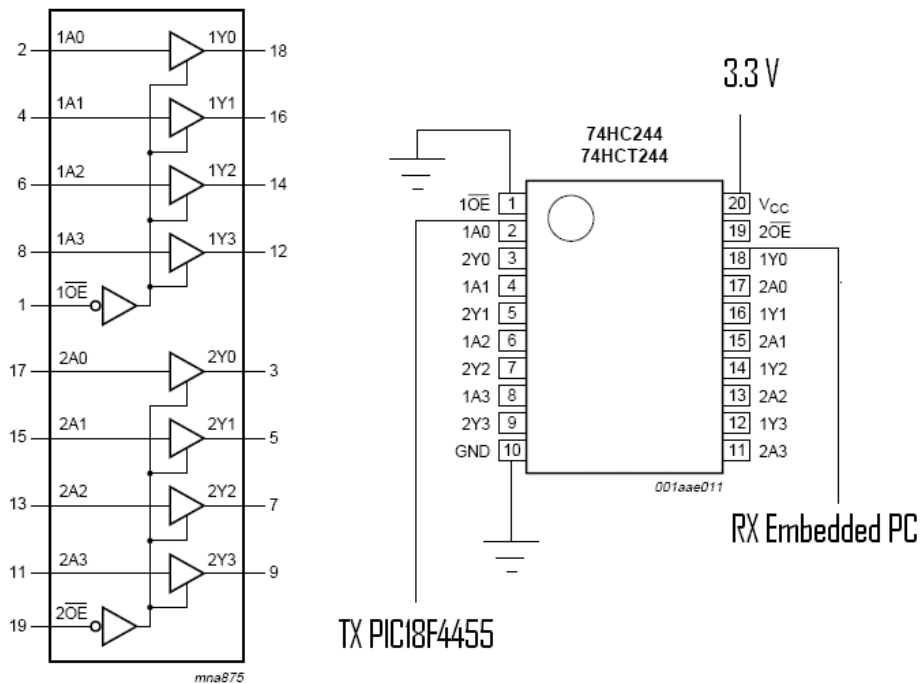


Figure 17. 74HC244 (Philips 2005).

The 74HC244 chip is a high-speed Si-gate Complementary Metal Oxide Semiconductor (CMOS) device and is pin compatible with Low-power Schottky Transistor-Transistor-Logik (LSTTL). It has octal non-inverting buffer/line drivers with 3-state outputs. These 3-state outputs are controlled by the output enable inputs 1OE and 2OE (see **Figure 17**). A high OE causes the outputs to assume a high-impedance. The output voltage (pin 18) corresponds to the operating voltage. (Philips 2005.)

To display messages and errors the circuit board contains also a Liquid Crystal Display (LCD). This display is a 16 character, 2-line alphanumeric LCD device which is connected to an upstream E-block board. It requires data in a serial format on five data inputs. **Figure 18** represents the timing diagram. The numerical representation of character follows the American Standard Code for Information Interchange (ASCII). When a character is sent to the LCD, the character must be sent in two steps. First the four most significant bits are transmitted then the remaining four least significant bits. As each half byte is sent Pin 6 must be set to high then to low to acknowledge the half byte. The upstream board must wait for at least the length of the execution time, before the next half byte can be sent. (Matrix Multimedia 2005.)

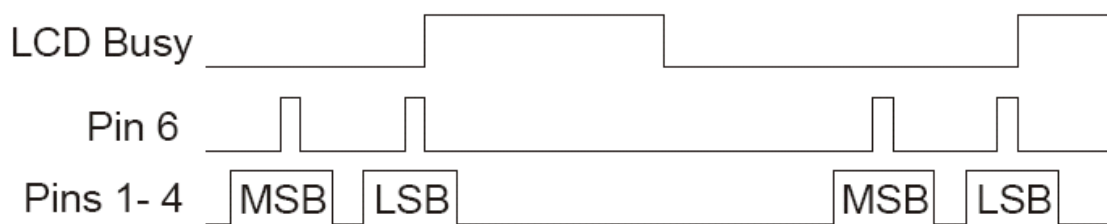


Figure 18. Overview of the designed circuit board (Matrix Multimedia 2005).

4.1.5. Motor Control and Infrared Sensors

The microcontroller (PIC18F4455) on the mobile platform (see section 4.1.1.) is responsible for sending the converted analog values from the infrared sensors to the Embedded PC and for the control of the motor drivers that give the power to the motors.

Three infrared sensors are connected with the input channels of the Analog-Digital-Converter (ADC) that converts each analog input signal to a corresponding 10 bit digital number. Before a value of an infrared sensor can be read the corresponding ADC channel must be selected and configured as an analog input channel. Furthermore the acquisition time which is computed according to the following formula must be set. It is required that the channel must be sampled for at least the minimum acquisition time before the analog-to-digital conversion can be started.

$$T_{ACQ} = T_{AMP} + T_C + T_{COFF} \quad (4)$$

T_{ACQ} = Acquisition Time

T_{AMP} = Amplifier Settling Time

T_C = Holding Capacitor Charging Time

T_{COFF} = Temperature Coefficient

The amplifier settling time is the time after that an output signal remains within a given error band according to some input stimulus. In the datasheet the amplifier settling time is specified with 0.2 microseconds. Besides the amplifier settling time, the acquisition time depends also on the temperature coefficient that can be neglected when the temperature is below 25 degree Celsius.

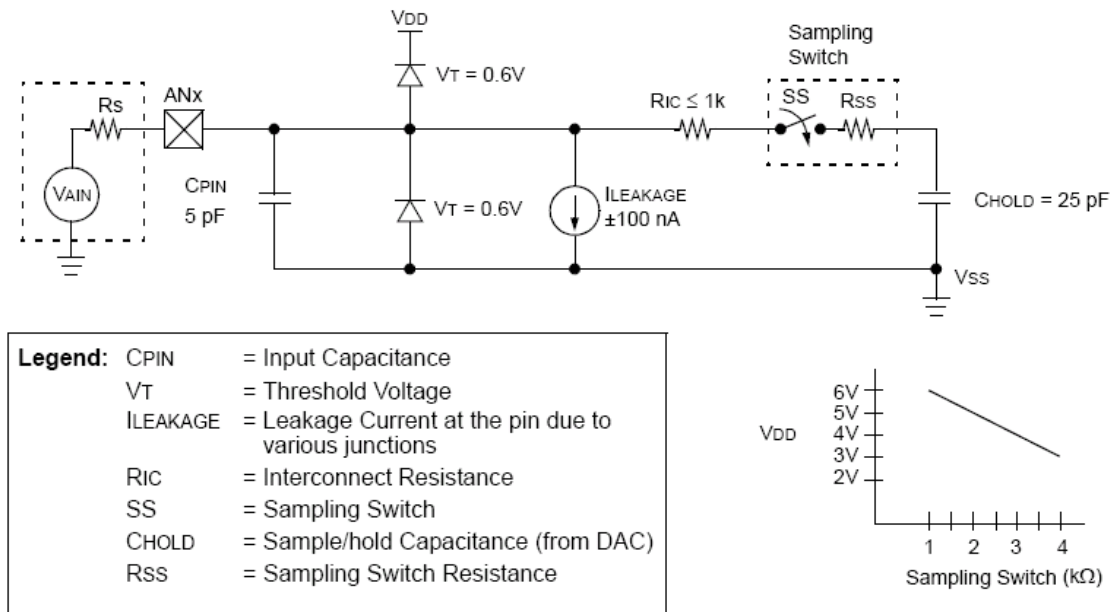


Figure 19. Analog Input Model of the ADC (Microchip 2009: 269).

To guarantee that the robot works also in areas up to 85 degree Celsius the pre-computed time value of 1.2 microseconds has been taken from the datasheet as well as the time value of the holding capacitor charging time which is 1.05 microseconds. The holding capacitor charging time depends on the components of the analog input model and is computed as follows.

$$T_C = -(C_{HOLD}) \times (R_{IC} + R_{SS} + R_S) \times \ln\left(\frac{1}{2048}\right) \times \mu s \quad (5)$$

Figure 19 shows the Analog Input Model of the ADC. Finally the total acquisition time can be determined. The program needs to wait at least 2.45 microseconds until the converted values from the ADC can be read. (Microchip 2009: 266–273.)

Pulse Width Modulation is used to control the speed of the motors. It is a very efficient way of providing intermediate amounts of electrical power between fully on and fully off (see **Figure 20**). The speed of the motor is determined by the duty cycle which describes the proportion of on time to the period of time. A low duty cycle corresponds

to a low speed of the motor because the power given to the motor during one period is low. For setting the duty cycle 8 bit variables are used. This means that the motor can be controlled with 256 different speeds.

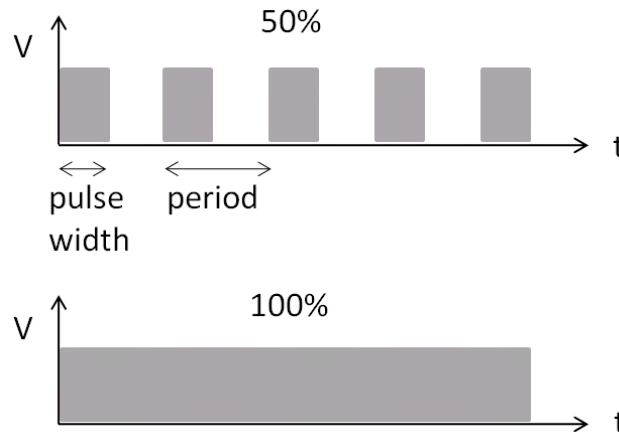


Figure 20. Pulse Width Modulation.

The H-Bridge is needed to run a motor forward and backward. A Classic Bipolar H-Bridge consists of resistors, NPN transistors, PNP transistors and of Schottky diodes. **Figure 21** illustrates a Classic Bipolar H-Bridge. For spinning the motor clockwise the resistor R2 is connected to the ground and the PNP transistor Q2 is switched on. When resistor R3 is supplied with a positive voltage then Q3 is switched on and a current can flow from the power source through the transistor Q2, through the motor, and to the ground. To run the motor counterclockwise the resistor R4 must be connected to the ground, so that the PNP transistor Q4 is switched on. The resistor R1 must be supplied with a positive voltage to switch on the NPN transistor Q1. Then the current can flow from the power source through the transistor Q4, through the motor, through the transistor Q1 and to the ground. The Schottky diodes protect the transistors against overvoltage that can be caused by the motor. Besides the control of the motor direction, the H-Bridge can be used for slowing down the speed with an electronic brake. (Cook 2004: 158–161.)

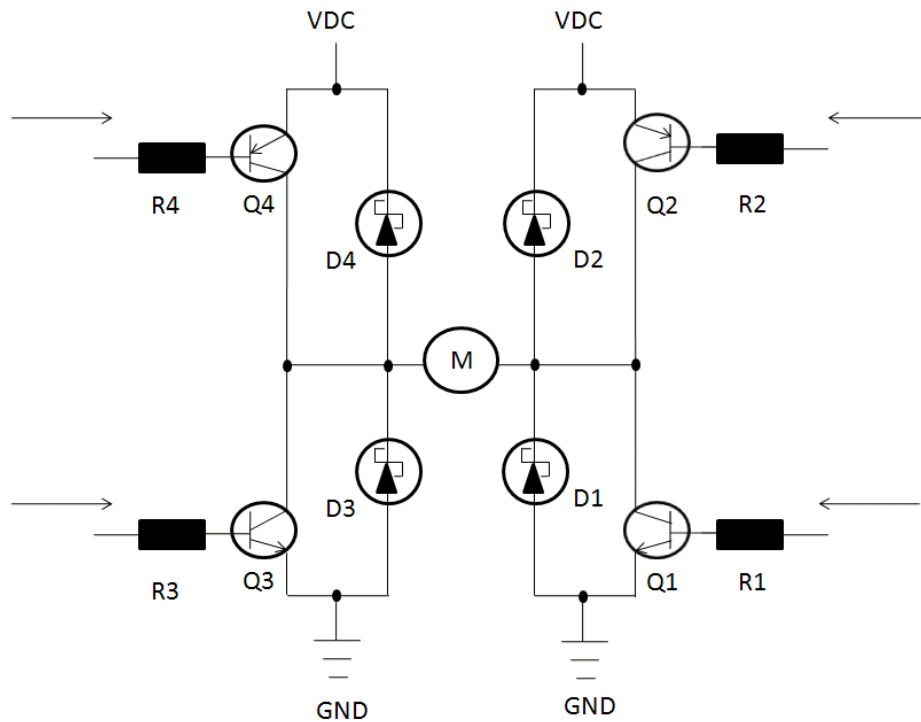


Figure 21. Classic Bipolar H-Bridge.

4.2. Software

4.2.1. Software Design of the Mobile Platform

In comparison to the Embedded PC, the microcontroller (PIC18F4455) on the mobile platform is quite slow. For that reason the program running on that microcontroller must be kept simple. Simple means that the program does not contain heavy computations, so that the execution time of one loop iteration is short. As already mentioned in section 4.1.4 the microcontroller is connected with the Embedded Linux PC via USART and it receives periodically a packet that contains a start byte, direction byte, two bytes for the speed of the motors and one byte for the checksum (see **Figure 22**). After receiving a correct packet the program sets the duty cycle of the PWM according to the desired wheel speed and runs the motors. Before the robot starts to move the value of the infrared sensors will be read to ensure that there are no obstacles in front of the robot. **Figure 23** displays the flowchart of the program.

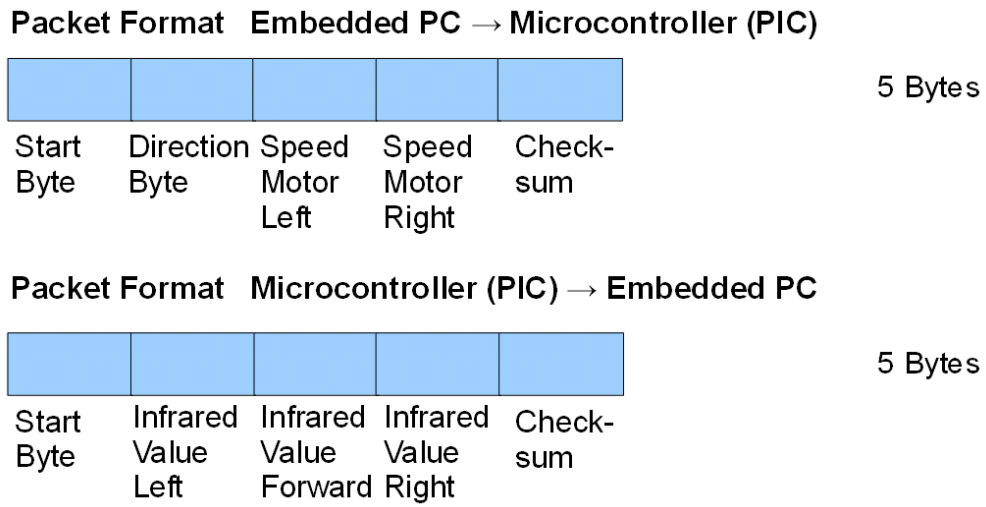


Figure 22. Packet Format for message exchange between Embedded PC and microcontroller.

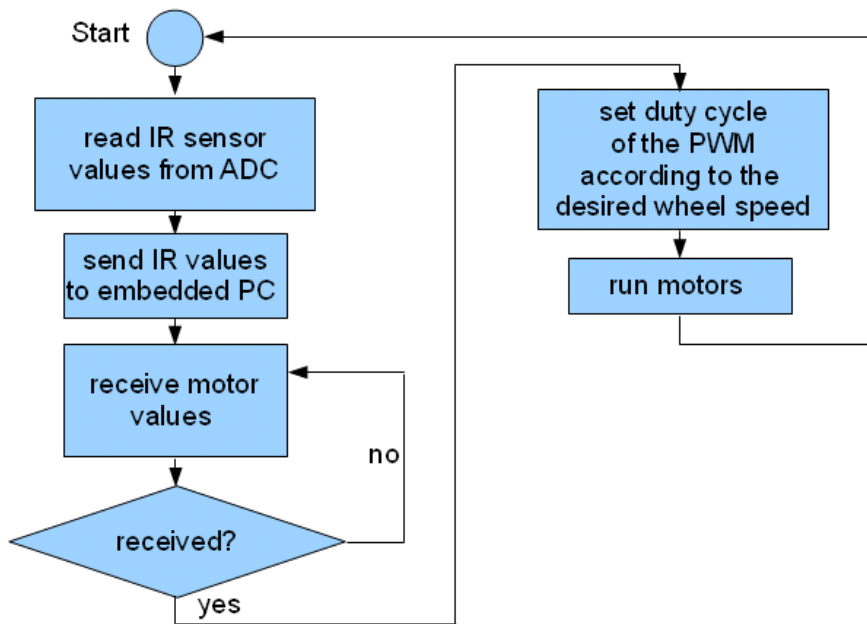


Figure 23. Flowchart for the microcontroller (PIC18F4455) software.

4.2.2. Software Design of the Wireless Node

Sensinode provides its own software called NanoStack. NanoStack is built upon FreeRTOS which is a portable real-time Operating System. FreeRTOS provides a microkernel with a scheduler, memory allocation, queues, semaphores and system timer functionality. The system timer functionality of FreeRTOS is needed by NanoStack for additional implementations, such as asynchronous timer service.

To reduce the usage of the Random Access Memory (RAM) and to guarantee an effective way of flow control, NanoStack runs as a single Task in the realtime Operating System (FreeRTOS). In addition, the protocol modules do not use direct function calls between each other so that the stack usage analysis is simplified. The buffer handling is done in a way that the user application is not blocked during the protocol stack operation. **Figure 24** illustrates the architecture of the NanoStack.

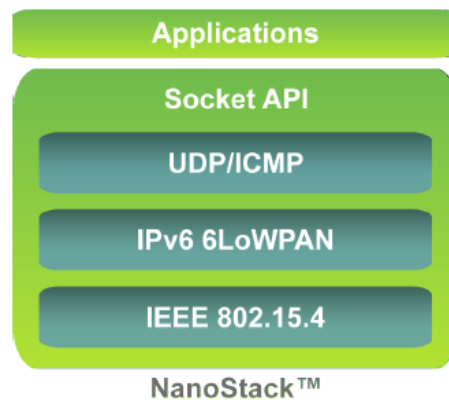


Figure 24. The NanoStack Architecture (Sensinode 2008).

The main task of the program running on the wireless node located on the robot's circuit board is to establish a communication with other wireless nodes. First the program creates a socket so that it can communicate with a program running on another wireless node. After the socket has been successfully created and bound the program is in the

communication state. In this state the program will forward the received packets from other wireless nodes via UART to the Embedded PC. It also sends packets received from the Embedded PC to the corresponding wireless node or to all the wireless nodes. Packets with different lengths can be received and sent. The first byte must contain the start character (0x55) and the second byte the packet length. **Figure 25** gives a flowchart overview of the program. **Figure 26** represents the flowchart of the transmit function and **Figure 27** illustrates the flowchart of the receive function.

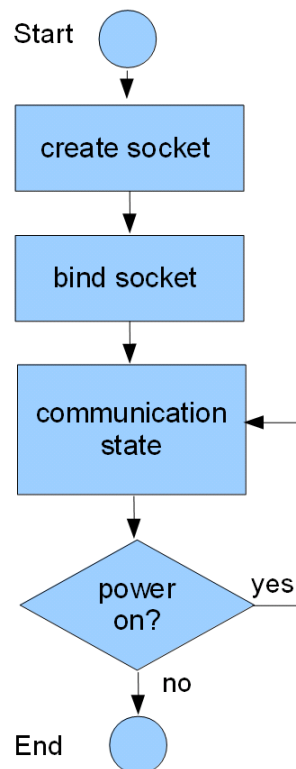


Figure 25. Flowchart Overview.

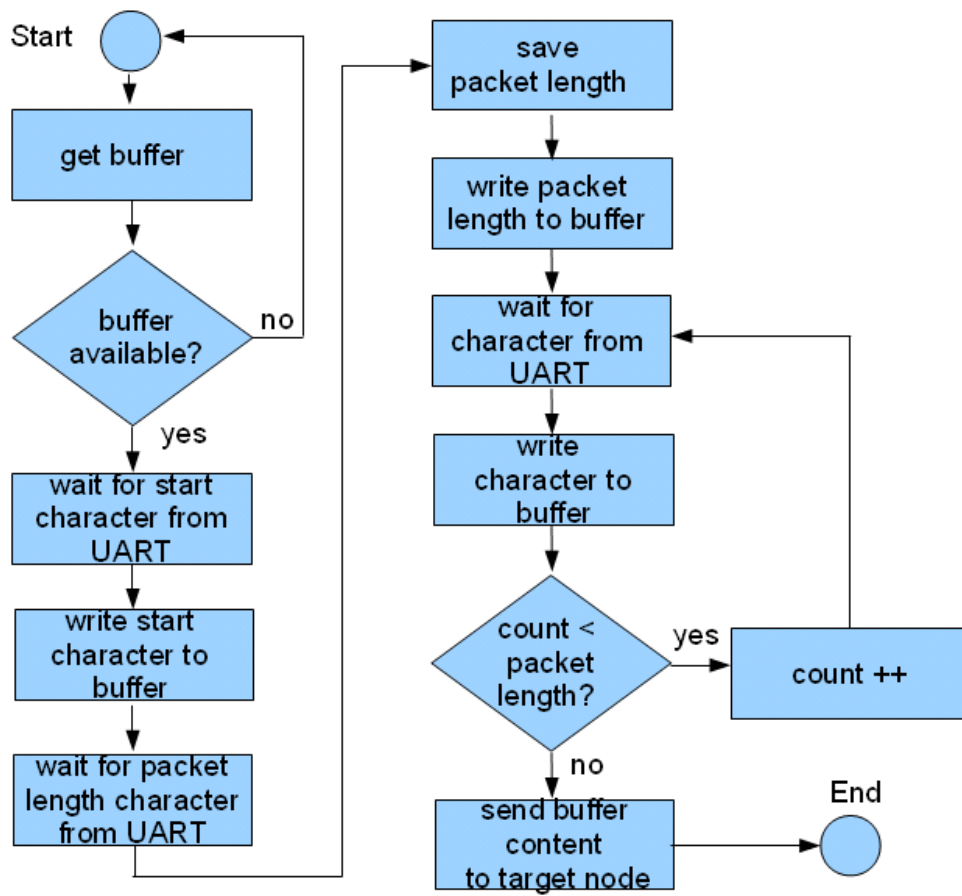


Figure 26. Flowchart of the transmit function.

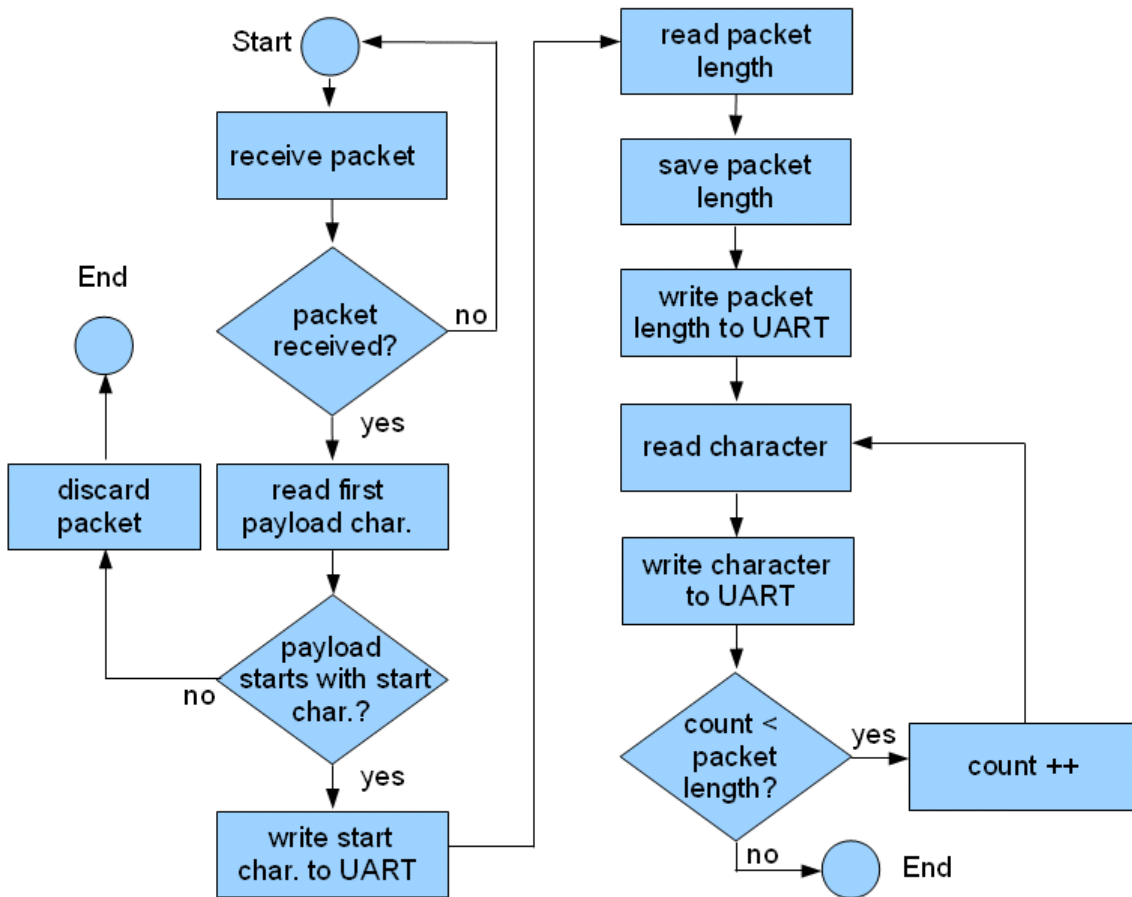


Figure 27. Flowchart of the receive function.

4.2.3. Software Design of the Embedded Linux PC

On the Embedded PC (Portux920T) runs an Embedded Linux Version developed by the Embedded PC manufacturer Taskit. Nowadays Embedded Linux becomes more and more famous due to its advantages. The source code for Linux is completely open and can be changed by anyone. Programmers can develop Linux applications and sell it without paying any sort of royalty. It is also possible to make changes to Linux but all the changes made to the Linux kernel must be shared. Linux is also famous because of its stability and good documentation. Many documents are available on the internet and dozens of books can be found in book stores.

For developers, nothing is more frustrated than tracing the own code all the way to a piece of code without source. As already mentioned the source code of Linux is completely present. Many times the problem is in the own code but sometimes the problem is in the code without source. It does not matter where the problem is. Fact is that without source code it is much more time consuming to figure out mistakes. If the source code to the function which is called from the own program is available, the programmer might quickly see the problem. This is one of the biggest advantages of open source software. (Lombardo 2002: xiv–xxvi.)

Besides the above mentioned advantages, Embedded Linux provides also benefits related to the application development with concurrency. Concurrency means that several computations are executed simultaneously. In Embedded Linux concurrency can be achieved by creating multiple threads. A further advantage of Embedded Linux is that the application can be programmed with an Object Oriented Programming (OOP) language such as C++ or Java.

When the Embedded PC is switched on, the boot loader loads the Embedded Linux Operating System. As soon as this process is finished a bash script will be executed that starts the configuration program. It reads the configuration commands sent from the Graphical Control User Interface and sets the parameter in the local configuration file. After the configuration is done the program can be terminated by pressing the inner push button on the circuit board. Then the main program will be executed. **Figure 28** shows the flowchart of the process on the Embedded PC.

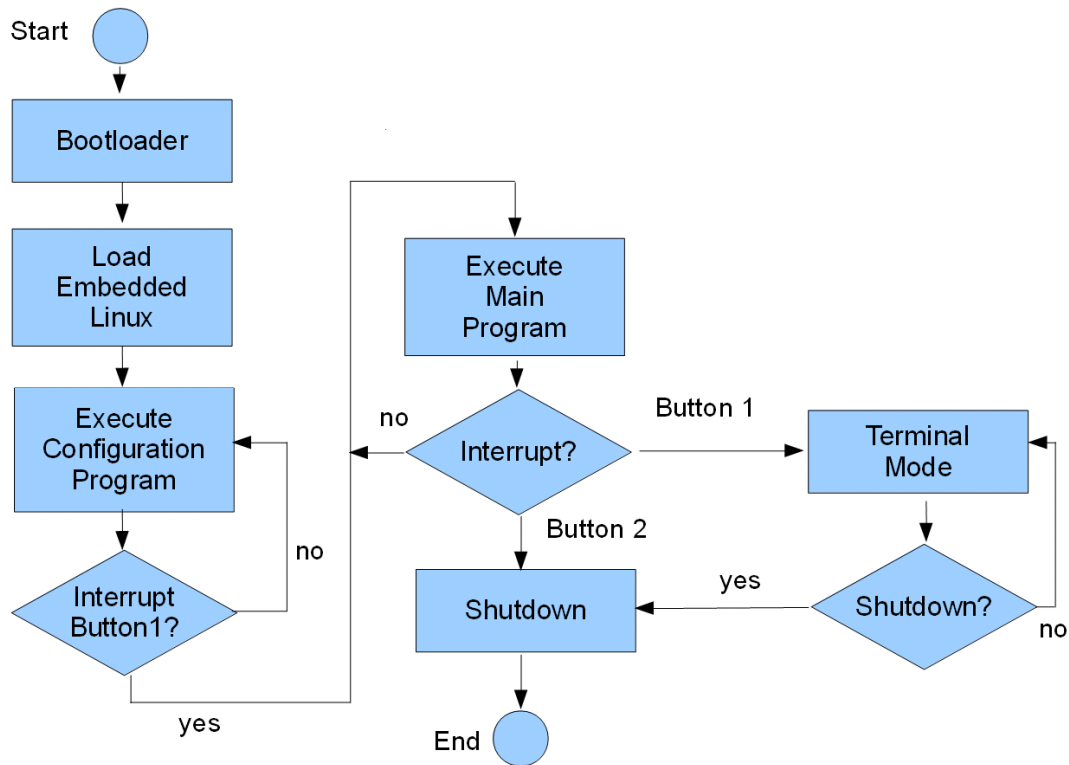


Figure 28. Flowchart of the process on the Embedded PC.

The Embedded Linux needs to be configured before the software runs according to the above shown Figure. As already known, configuration is always tricky and time consuming. For that reason the developed program package contains some bash scripts that configure Embedded Linux automatically. The source code for the configuration

program and the main program is written on a normal PC. With a cross compiler the executable file for the ARM processor on the Embedded PC will be generated. A programmed Makefile takes care about the whole compilation and linking process (see **Figure 29**). It finds all the source files and header files automatically and it can generate a binary file with and without debugging output. The library files required to run the program are statically linked.

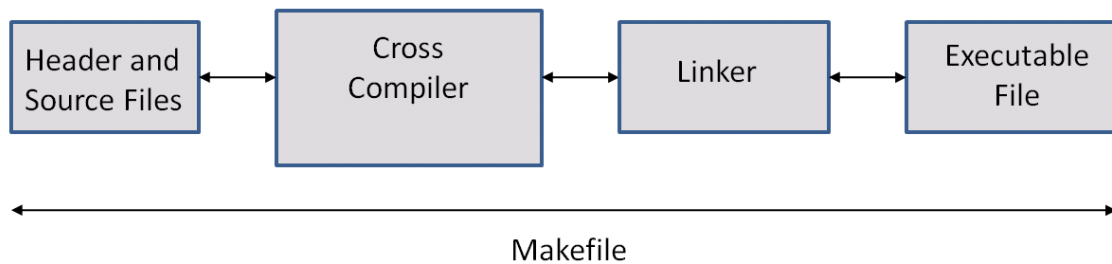


Figure 29. Compilation and Linking Process.

To upload the developed programs to the Embedded PC a terminal connection is required. C-Kermit, a combined network and serial communication software package is used to establish a terminal connection between PC and Embedded PC. It is possible but not recommended to upload the program to the Embedded PC by using the serial interface because of its low transmission rate. A better solution is to upload the program files by using the Ethernet connection. This demands a Network File System (NFS) Server installation and configuration on the PC side. With a NFS server files can be accessed by mounting a file tree on the server computer into the local file tree of the client computer (Santifaller 1994: 176). The program files on the PC can then be transferred from the PC to the Embedded PC. **Figure 30** represents the communication between PC and Embedded PC.

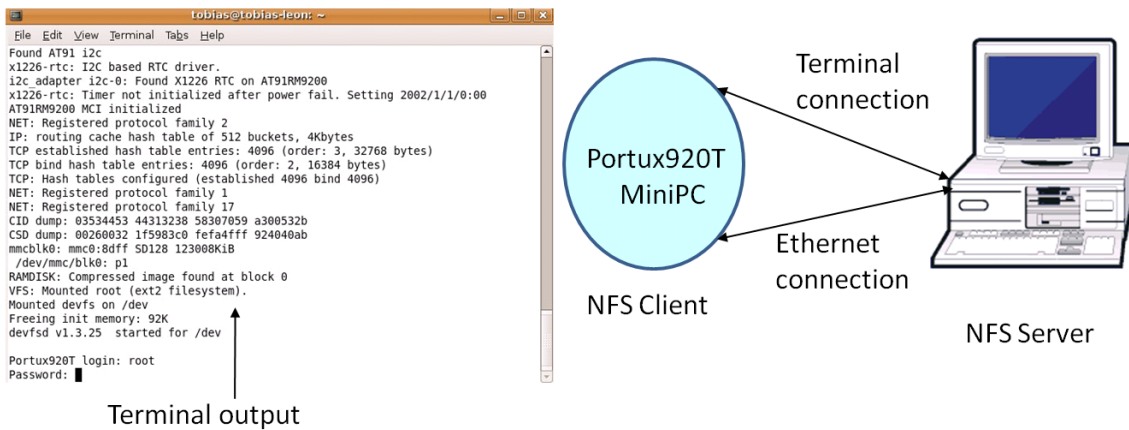


Figure 30. Communication between PC and Embedded PC.

The configuration program consists of nine classes (see **Table 3**). Its task is to communicate with the wireless node on the robot board that might receive configuration packets sent by the wireless node communicating with the Graphical Control User Interface. When a control packet is received the program checks first if the packet was received correctly by checking the checksum before it starts to parse the packet. After the packet was successfully received and parsed the program sets the corresponding configuration parameter in the configuration file.

Table 3. Classes and their tasks.

Class Name	Tasks
FileParser	<ul style="list-style-type: none"> - reads the values of the parameters saved in the configuration file - writes the value modifications of the parameters to the configuration file
InterruptHandler	<ul style="list-style-type: none"> - configures the interrupt registers - contains the interrupt service routines

Lcd	<ul style="list-style-type: none"> - includes the whole display driver - writes a message to the LCD
MemoryAccess	<ul style="list-style-type: none"> - maps the program memory to the device memory
Serial	<ul style="list-style-type: none"> - provides methods for reading and writing data from/to the serial interface
SerialSensinode	<ul style="list-style-type: none"> - reads the input buffer of the UART connected with the wireless node. - writes data to the output buffer of the UART connected with the wireless node.
SharedBuffer	<ul style="list-style-type: none"> - this class contains the shared buffer used for Inter-thread Communication
Thread	<ul style="list-style-type: none"> - sets the priority of a thread - starts a thread - cancels a thread
ThreadObject	<ul style="list-style-type: none"> - this class contains a virtual method run and is used for making generic calls

When the configuration program is terminated the main program is loaded. The main program communicates with the wireless node and with the microcontroller (PIC18F4455) on the mobile platform. It reads the received packets sent from the static wireless sensor nodes or from the wireless node that forwards the joystick commands from the GCUI. After the packet was received correctly the program checks the packet type and computes the wheel speed according to the desired direction. Before the robot starts to move it will read the values of the infrared sensors that are delivered from the microcontroller (PIC18F4455). If there are no obstacles in front of the moving direction the robot starts to move. Otherwise a new direction must be computed. The main program consists of all the classes of the configuration program and of five additional classes, shown in **Table 4**. For the recording of the robot's moving path the methods of the DataLogger class can be used to save all the desired positions in a text file. The

RemoMath class contains all the computation methods. Later on, this class should be extended with additional methods that include the implementation of prediction filters.

Table 4. Classes and their tasks.

Class Name	Tasks
DataLogger	- logs data to a text file
MotionPlanner	- is the coordinator of the main program - collects data received from the wireless node - collects data received from the microcontroller - calls the necessary computation methods - sends motor control data to the microcontroller
RemoMath	- contains the computations of the wheel speed - includes the computation of the distance estimation
SerialMotor	- reads the input buffer of the UART connected with the microcontroller on the mobile platform that converts the analog values of the infrared sensors to digital values - writes data to the output buffer of the UART connected with the microcontroller on the mobile platform that controls the motor driver
Timer	- provides methods related to time

Both programs are implemented so that several tasks are being executed in a parallel fashion. This can be achieved by using multiple threads in the application program. In traditional Operating Systems (OSs) every program has its own address space and at least one thread, the so called “main thread”. In many situations it is desired that more threads run in a parallel fashion within the same address space. This leads to the advantage that more tasks can be handled at the same time. When a process with multiple threads runs on a system with a single Central Processing Unit (CPU) then

only one thread can be executed at any one time. **Figure 31** illustrates three threads that run in the same address space. (Tanenbaum 2002: 96–98.)

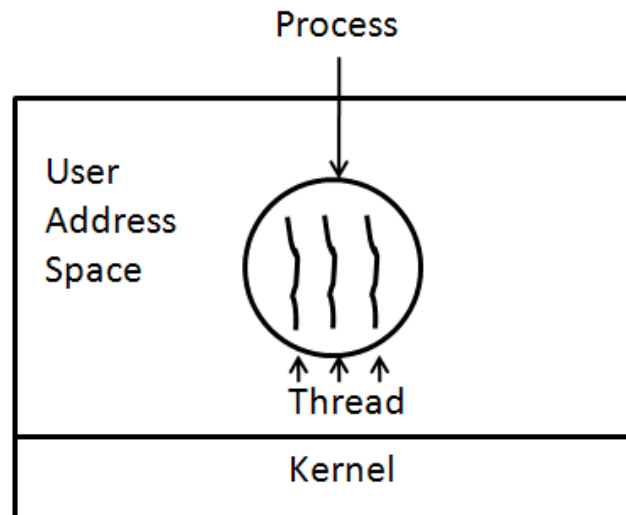


Figure 31. Process with three Threads.

The order of how the threads are being executed depends on the scheduling algorithm. One scheduling algorithm is called “Round Robin” (see **Figure 32**) where each thread gets a predetermined timeslot to do its task before the next thread is scheduled. In this algorithm, it is assumed that every thread has the same priority.

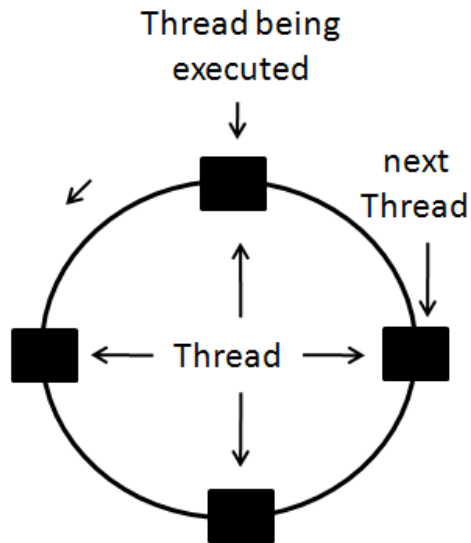


Figure 32. Round Robin Scheduling

It is not always requested that all the threads have the same priority. Sometimes it is necessary that a thread with an urgent task must be accomplished before the other threads or it must be processed immediately due to an emergency. For that reason, preemptive Operating Systems have been developed. The developed configuration program and main program contain a thread class that allows assigning each thread a priority number. Threads with a higher priority number are served first. In case a CPU processes a task of a lower prioritized thread and an event occurs in a higher prioritized thread, the current thread will be interrupted and its current state will be saved. Then the higher prioritized thread will be accomplished. After that, the CPU will continue with the previous interrupted thread if there are no tasks of higher prioritized threads available. **Figure 33** displays a priority based scheduling of three threads with different priorities. (Tanenbaum 2002: 159–166.)

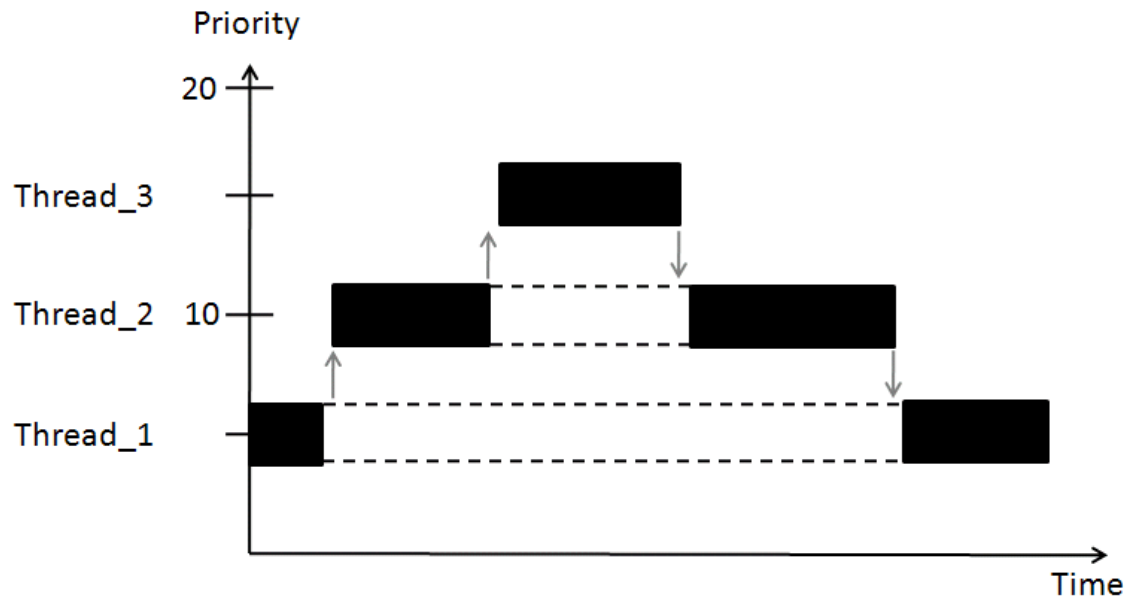


Figure 33. Priority based scheduling of three threads.

The implementation of multithreaded programs is quite difficult and challenging, especially when multiple threads exchange messages over a common resource which is also known as “Inter-thread Communication”. Threads can communicate with each other via pipes, shared buffers and message queues. In case two threads communicate with each other it is important that no other thread disturbs their communication. Inter-thread communication is not easy to handle and it requires many attentions to avoid “Race Conditions”. According to Ippolito (2004) “[a] race condition often occurs when two or more threads need to perform operations on the same memory area, but the results of computations depends on the order in which these operations are performed”. To avoid collisions between threads that share those states it is necessary that only one thread can enter a shared state at any one time. Protecting a common resource against simultaneous use is also known as “Mutual Exclusion”. Parts of a program in which an access to a shared resource happens are named “Critical Regions”. **Figure 34** represents a mutual exclusion with two threads.

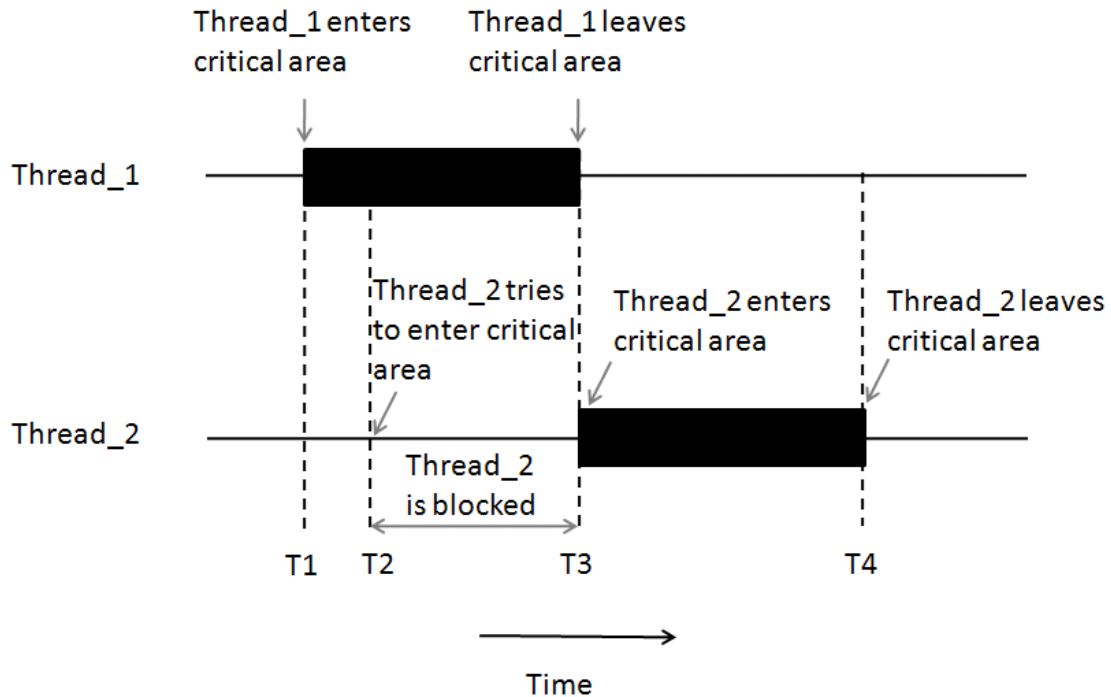


Figure 34. Mutual Exclusion with two threads.

According to Tanenbaum (2002: 119) a multi-threaded program can only work right and efficient when the following conditions are fulfilled.

1. Only one thread can enter a critical region.
2. It is forbidden to make assumptions about the CPU speed and the amount of CPUs.
3. A process outside the critical region may not block other threads.
4. No thread should wait a long time to enter a critical region.

There are several methods that can be used to fulfill these four conditions. One method to prevent that multiple threads enter a critical region, is to switch off all interrupts. A thread entering a critical region switches off all interrupts until it leaves the critical region. This method is very simple but not attractive because it would not be clever to allow the user switching off all the interrupts. A better solution can be achieved with a Mutex variable that protects a common resource from multiple accesses at any one time.

A Mutex variable has two different states either one or zero. Normally, the initialization of the Mutex variable with a value of one means that the common resource is in use, otherwise it is free. Before a thread accesses the common resource it checks by calling `mutex_lock` if the Mutex is in use or not. In case the Mutex is not free the thread needs to wait until the resource will be released from the other thread which currently holds the resource. (Tanenbaum 2002: 120–131.)

Both programs developed for the Embedded PC use a shared buffer with a Mutex variable for exchanging messages between threads. The buffers can store numbers as well as strings.

The configuration program and the main program are both developed with the Object Oriented Programming (OOP) language C++. This leads to the advantage that the programmer can also make use of the powerful Standard Template Library (STL). This library contains many commonly used data structures and algorithms. It is applicable to nearly any type of data because the STL is constructed from template classes and functions. The STL defines various routines such as sorting, searching and transforming and it includes support for vectors, lists, queues and stacks. Every software component is reusable and self-contained. Hence, the integrity of a component is unaffected by errors or misuse. (Schildt 1999: 2–5.)

The source code is documented with Doxygen which was developed by Dimitri van Heesch. Doxygen is a documentation system for multiple programming languages such as C++, C or Java. It has the capability to generate an online documentation in HyperText Markup Language (HTML) format and/or a reference manual in LATEX from multiple documented source files. Doxygen can also generate output files in PostScript format and hyperlinked Portable Document File (PDF) format. Dependency graphs, inheritance diagrams and collaboration diagrams are generated automatically. (Van Heesch 2010.)

4.2.4. Memory Mapping

Linux is a virtual memory system. The addresses seen by the user do not directly correspond to the physical addresses used by the hardware. Using virtual memory leads to the advantage that running programs can allocate more memory than physically available. The virtual address space of a single process can be larger than the system's physical memory. Virtual memory allows the mapping of the program memory to the device memory. (Corbet, Rubini & Kroah-Hartman 2005: 412–424.)

There are four Parallel Input/Output (PIO) controllers inside the Atmel AT91RM9200 CPU. One PIO controller handles up to 32 fully programmable input/output lines. Each I/O Line controlled by a PIO controller is associated with a bit in each of the 32 bit PIO Controller User Interface registers. (Atmel 2009: 345–355.)

To access the Controller User Interface registers inside the program it is required that the device memory file which gives access to the Controller User Interface registers is mapped to the program memory . Linux provides a function called mmap that returns an address at which the mapping is placed.

4.3. Computation of the Wheel Speeds

The motors used in this robot are gear box motors that can be controlled with 256 different speeds. As already mentioned in section 4.1.5 the motors are controlled with PWM. To move the robot to the computed target the following direction-based control method is used (Hong, Shim, Jung, Lee & Hong 2001).

$$S_L = k \times (K_y \times C_y + K_x \times C_x) \quad (6)$$

$$S_R = k \times (K_y \times C_y - K_x \times C_x) \quad (7)$$

S_L = Speed of the left motor

S_R = Speed of the right motor

K_x, K_y = Constants related to adjust the turning speed of the system

C_x, C_y = Target direction

k = Scaling is the scaling vector to generate maximum PWM

5. GRAPHICAL CONTROL USER INTERFACE

To calibrate and control the robot remotely a Graphical Control User Interface (GCUI) has been developed with QT and C++. QT is a cross-platform application and User Interface (UI) framework. It contains integrated development tools with a cross-platform Integrated Development Environment (IDE). The main task of the GCUI is to calibrate the robot.

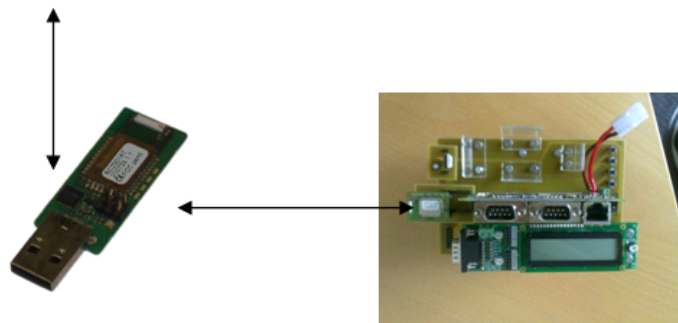
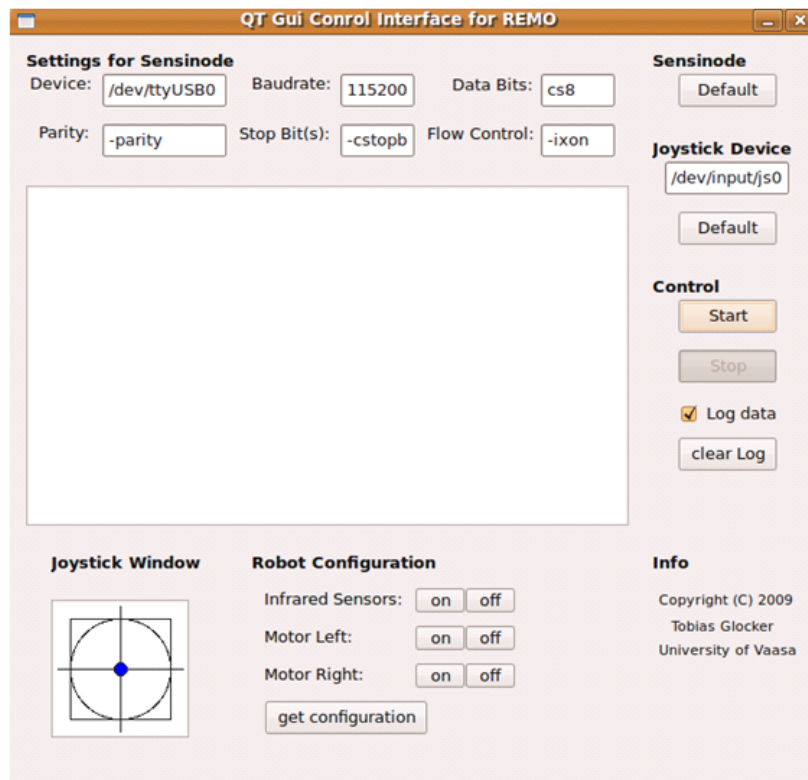


Figure 35. Graphical Control User Interface.

Furthermore, the GCUI provides also a joystick control feature with that the robot can be controlled and which can be helpful by adjusting the motor values. There are also features for receiving error messages and control messages from the robot in the calibration state. In addition, some configuration parameters can be set remotely. **Figure 35** displays the Graphical Control User Interface.

The GCUI communicates with the Sensinode NanoSeries N601 router via USB interface. On the circuit board is a FT232RQ chip from FTDI that converts the USB signals to serial signals and vice versa. The FT232RQ chip is connected with the UART1 of the 8051 microcontroller integrated in the Radiocrafts RC2301AT System on Chip module. From there the data will be sent to or received from the wireless node located on the robot's circuit board.

As already mentioned the robot is first in the configuration mode after it is being switched on. In this mode the robot can be configured remotely with the GCUI. **Figure 36** represents the packet format used for exchanging messages between robot and GCUI in the configuration mode.

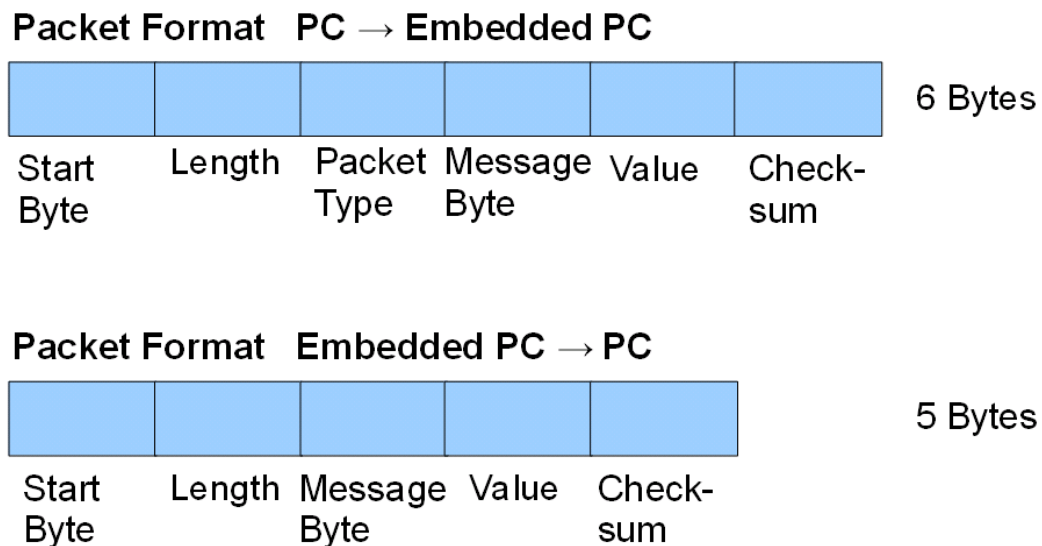


Figure 36. Packet Format for message exchange between PC and Embedded PC in configuration mode.

Every packet starts with a start character followed by the packet length and the packet type which defines if it is a configuration packet 'C', a joystick packet 'J' or another type of packet. The message byte is an index to the parameter that needs to be initialized with the value stored in the value byte. The packet ends with the checksum byte necessary to check if the received packet is ok.

In the run state the robot can be also controlled with the joystick that is connected with the USB interface with the Laptop or Desktop PC. To access the joystick features, Linux provides the input/output control (ioctl) system call for device-specific operations. This system call can return the number of buttons, number of axes and the name of the joystick. Linux offers also a read function with that the joystick signals can be read. The values returned from the read function for the X-axis and Y-axis are between -32767 and +32767 (see **Figure 37**).

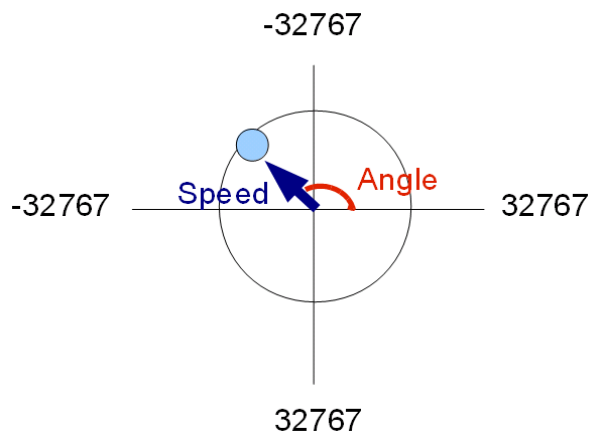


Figure 37. Joystick Control.

Figure 38 represents the packet format of the packet that is sent from the GCUI to the robot when the robot is controlled with the joystick. The values of the axes returned from the read function must be scaled because they are too big to be stored in a byte.

Eight bits can be used to control the speed of the motor. The direction byte determines the directions of the motors.

Packet Format PC → Embedded PC

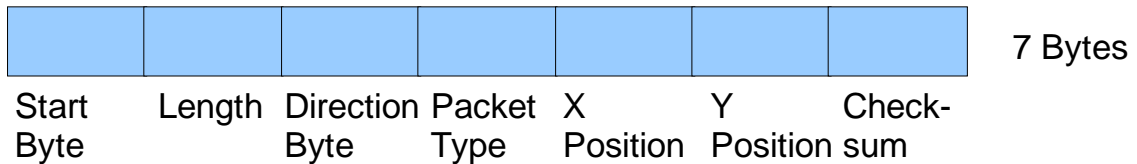


Figure 38. Packet Format for message exchange between PC and Embedded PC.

QT provides a QT Designer with that the Graphical User Interface has been built. It can compose and customize widgets or dialogs in a what-you-see-is-what-you-get manner. The GCUI program consists of five classes. **Table 5** describes their tasks.

Table 5. Classes and their tasks.

Class Name	Tasks
CommPort	- opens, initializes and closes the communication port with Sensinode's NanoRouter N601
ControlGuiThread	- takes care about the joystick control - reads joystick commands - updates the joystick window in the User Interface - scales the joystick parameters - makes method calls, necessary to prepare and to send the joystick packet
ControlGuiWidget	- handles everything related to the graphical representation of the Graphical User Interface

FileParser	<ul style="list-style-type: none">- reads the values of the parameters saved in the configuration file- writes the value modifications of the parameters to the configuration file
Sensinode	<ul style="list-style-type: none">- prepares the joystick packet sent to the robot- prepares the configuration packet sent to the robot- reads data from Sensinode's NanoRouter N601- writes data to Sensinode's NanoRouter N601

6. EXPERIMENTS

6.1. Indoor RSSI Measurements

In this experiment the RSSI values of different distances between two wireless nodes were measured in a computer class room at Technobothnia. The RSSI value can only be determined when there is a message exchange between the two wireless nodes. To keep the communication ongoing ICMP messages were sent until the RSSI could be read.

First the RSSI measurements between the two wireless nodes were taken on the ground. **Table 6** shows the results of the measured RSSI values according to the following distances. At each position 15 RSSI values have been measured.

Table 6. RSSI Measurements between two wireless nodes taken on the ground.

Distance in m	0	0.15	0.25	0.5	1	1.5	2	2.5	3	3.5
Minimum RSSI value in dBm	-13	-70	-68	-74	-82	-81	-85	-91	-92	-91
Maximum RSSI value in dBm	-12	-69	-66	-72	-80	-79	-82	-90	-91	-90

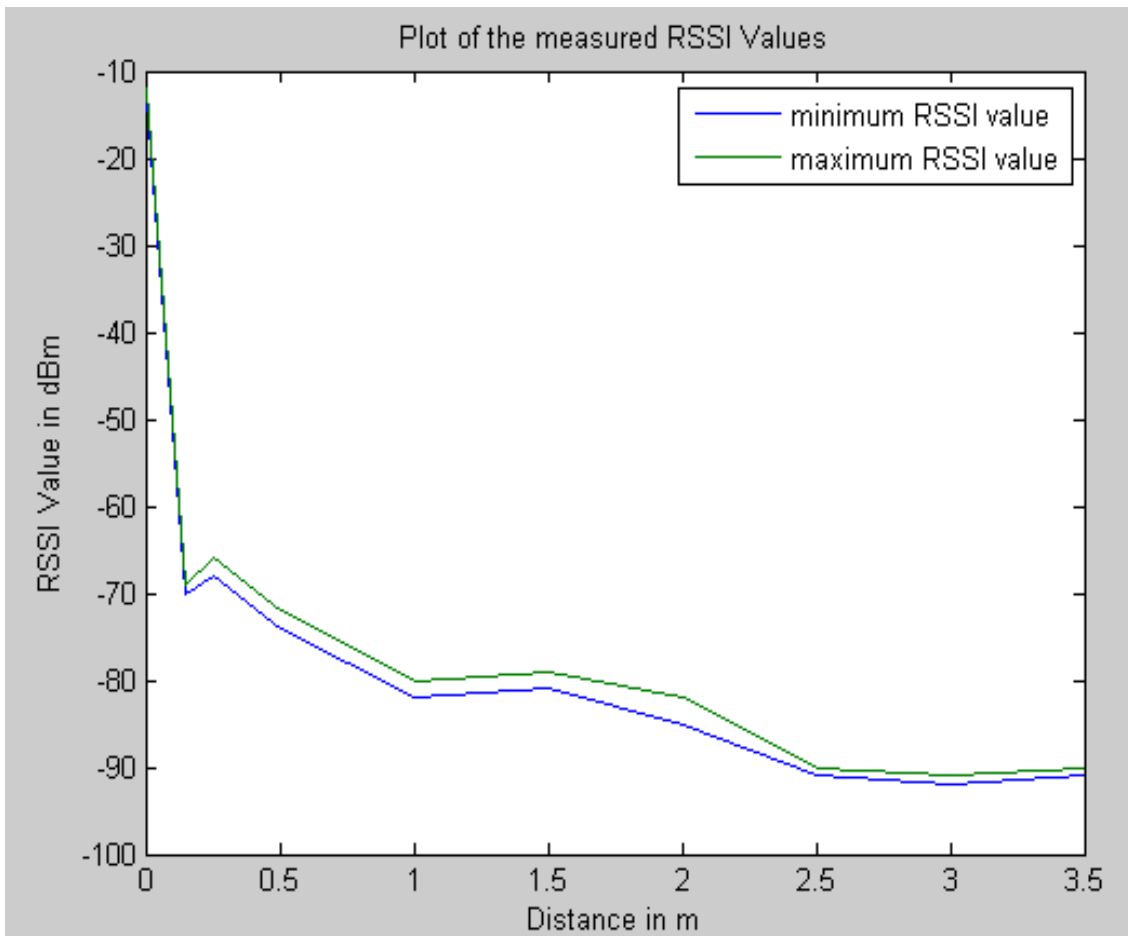


Figure 39. RSSI Measurements between two wireless nodes taken on the ground.

Figure 39 shows that the RSSI values decrease when the distance increases. This corresponds also to the theory but in an ideal case the RSSI values would decrease linearly. Between some distances the RSSI values do not change so much. In the above Figure the RSSI value measured at the distance of 1.5 m is equal to the value measured at a distance of 0.95 m. Good distance estimations are achieved between the distances of 1.5 m to 2.5 m because the RSSI values change almost linearly. After a distance of 3.5 m no signal strength is received.

In the second part of this experiment, the RSSI values between the two wireless nodes were measured 50 cm above the ground.

Table 7. RSSI Measurements between two wireless nodes taken 50cm above the ground.

Distance in m	0	0.15	0.25	0.5	1	1.5	2	2.5	3
Minimum RSSI value in dBm	-16	-74	-66	-72	-86	-89	-85	-81	-80
Maximum RSSI value in dBm	-15	-71	-65	-70	-83	-87	-83	-80	-79
Distance in m	3.5	4	4.5	5	5.5	6	6.5	7	7.5
Minimum RSSI value in dBm	-83	-88	-91	-93	--	-93	-90	-90	-90
Maximum RSSI value in dBm	-82	-85	-88	-91	--	-92	-89	-88	-89

Table 7 and **Figure 40** represent the results of the RSSI measurements. It can be observed that a much higher distance can be reached because the power of the signal will not be absorbed so much by the ground. Compared to the measurements on the ground the RSSI values do not decrease continuously with an increasing distance. Instead of decreasing, the RSSI values between the distances of 1.5 m to 3 m increase a lot so that it not possible to decide if the second wireless node is 80 cm away or 3 m. In both cases, the deviation between the minimum and maximum value is low.

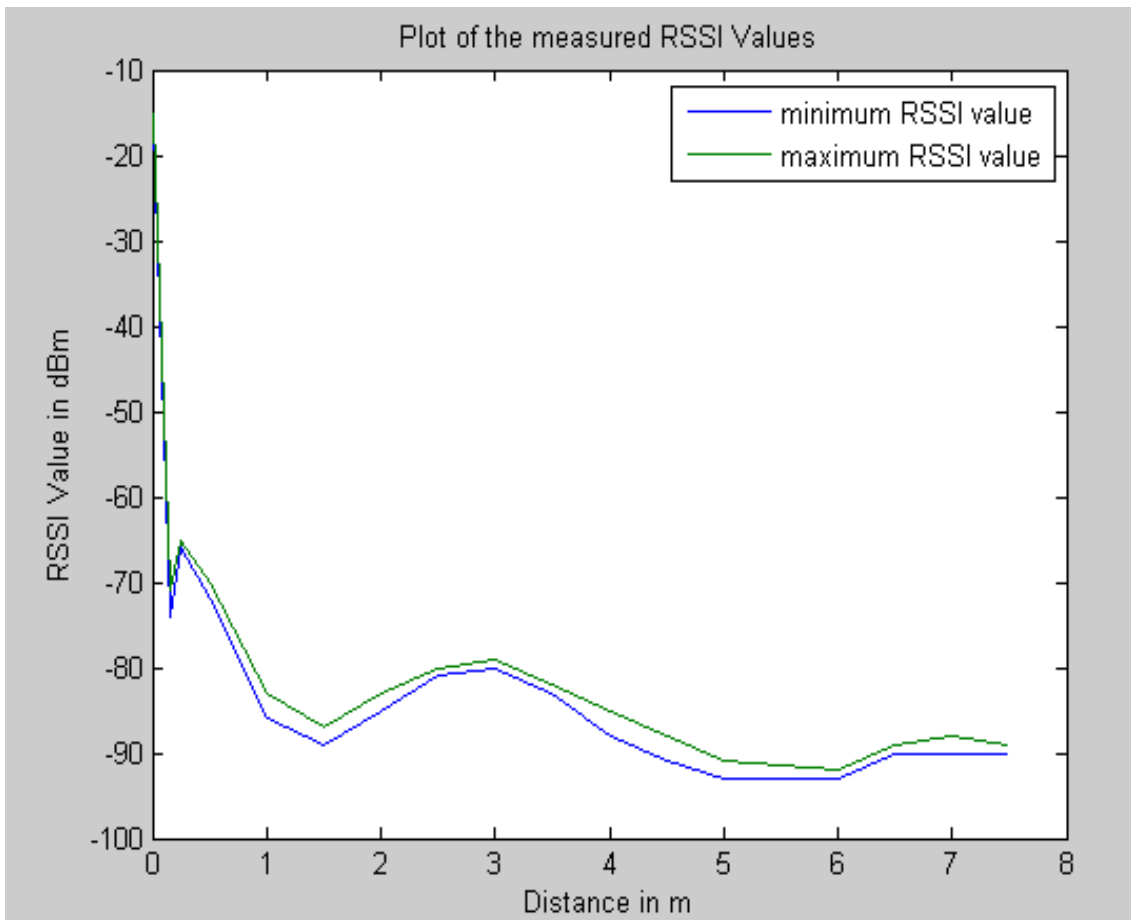


Figure 40. RSSI Measurements between two wireless nodes taken 50 cm above the ground.

6.2. Distance Estimation with the Mobile Robot

A further experiment has been done with the mobile robot in the same class room as the RSSI measurements in section 6.1. The robot was placed 3 m away from the wireless node put on the ground from where it started to move until a certain RSSI value stored in the configuration file has been reached or exceeded (see **Figure 41**). This experiment is essential because it shows the behaviour of the measured RSSI values in a non-static environment.

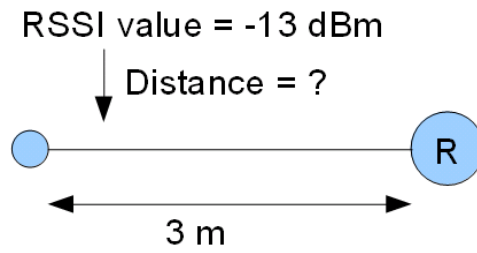


Figure 41. Distance Measurement with a predetermined RSSI value.

Table 8 includes the results of the distance measurements for a number of predetermined RSSI values between the wireless node located on the robot and another wireless node put on the ground. By comparing these results with the RSSI measurements in section 6.1 (see Table 6) it can be observed that the robot stops almost at the expected positions.

Table 8. Distance Measurements between the mobile robot and a wireless node with predetermined RSSI values.

RSSI value in dBm	-50	-70	-75	-80	-85	-90
Distance in m	0.1	0.2	0.35	0.62	0.9	2.7

6.3. Indoor RSSI Measurements II

In this experiment the RSSI values of different distances between two wireless nodes were measured in the entrance hall of the Tritonia library. The RSSI measurements were taken on the ground. **Table 9** contains the results of the measured RSSI values according to the following distances. At each position 15 RSSI values have been measured.

Table 9. RSSI Measurements between two wireless nodes taken on the ground.

Distance in m	0	0.15	0.25	0.5	1	1.5	2	2.5	3
Minimum RSSI value in dBm	-11	-66	-77	-82	-88	-92	-89	-88	-93
Maximum RSSI value in dBm	-11	-66	-76	-80	-85	-91	-88	-87	-91

It can be noticed that the RSSI values in this experiment decrease faster for distances up to 1.5 m than in the previous experiment (see section 6.1/Table 6). Good distance estimations can be achieved up to a distance of 1.5 m because the RSSI values decrease constantly within that range (see **Figure 42**).

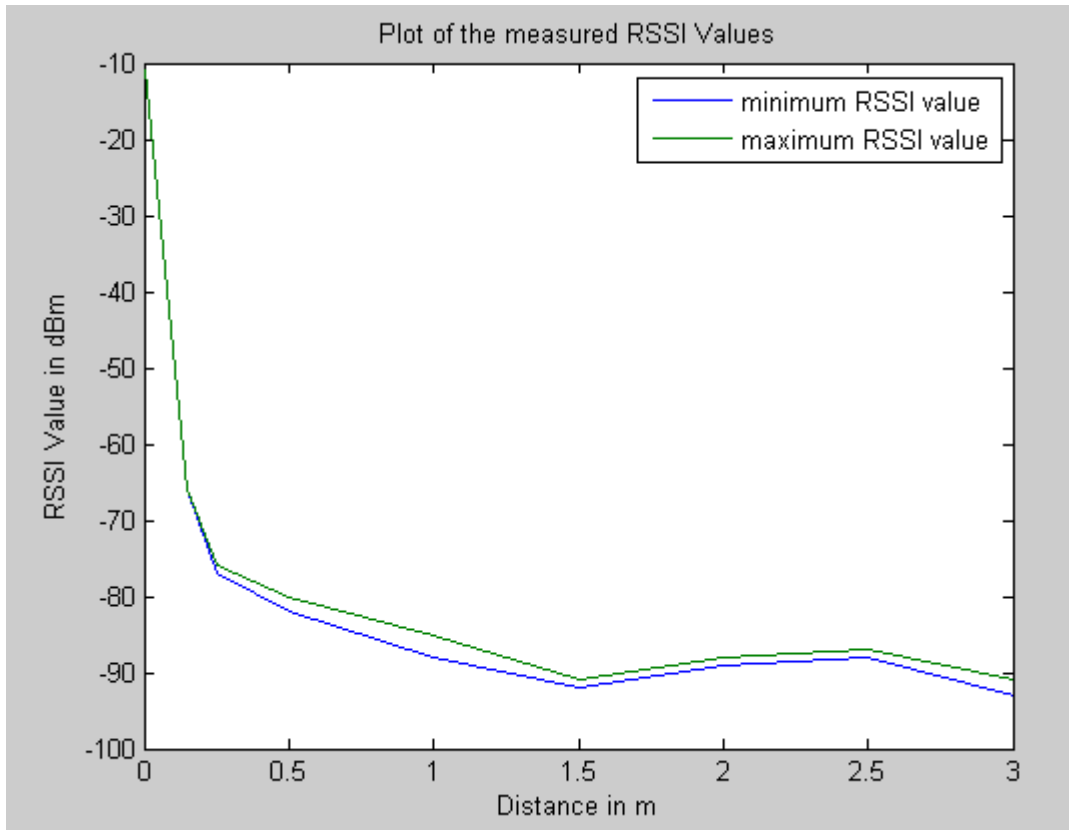


Figure 42. RSSI Measurements between two wireless nodes taken on the ground.

Generating a common distance map from a number of measured RSSI values is difficult because the measured signal strength is influenced by the surrounded noise. The surrounded noise is not deterministic, meaning that different RSSI values for certain distances are measured in several environments. The results of the RSSI measurements taken in two different buildings confirm that.

7. CONCLUSIONS AND FUTURE WORK

In this thesis work, a mobile miniaturized autonomous robot with the capability to operate in a Wireless Sensor Network has been developed for the Wireless Sensor Systems in Indoor Situation Modelling (WISM) project. The objective of this thesis project is to have a robot that is able to move to predetermined positions within a Wireless Sensor Network, so that already developed algorithms for distance estimation and localization can be tested under real conditions. As already mentioned the costs of Global Positioning Systems and Local Positioning Systems are expensive compared to a Wireless Sensor Network consisting of several wireless sensor nodes. Therefore it is an interested research topic not only for the technology side but also for the economical side to realize localization with new low cost communication control concepts.

A number of state of the art Wireless Sensor Network technologies have been reviewed and their capabilities have been analyzed. Considerations in terms of security and reliability are investigated. To build this robot it was necessary to have electronic skills and programming skills in low-level and high-level programming. Four circuit boards were developed to connect the wireless sensor node and the microcontroller on the mobile platform with the Embedded PC. For each device a software implementation was required. A Graphical Control User Interface has been implemented to control and configure the robot remotely. RSSI values have been measured and evaluated for certain distances in different environments.

Based on the observations and evaluations the robot is not driving exactly straight forward or backward due to the inaccuracy of the motors. The motors have no wheel encoders with them a feedback signal can be generated. Thus it is necessary to replace the current motors with more sophisticated motors that contain an optical encoder with a high resolution. Then an accurate motor control could be achieved with a Proportional-Integral-Derivative Controller.

REFERENCES

- Atmel (2009). ARM920T-based Microcontroller. AT91RM9200. Rev. 1768I-ATARM-09-Jul-09 [online]. Available from the Internet: <URL: http://www.atmel.com/dyn/resources/prod_documents/doc1768.pdf/>.
- Bandara, H. M. N. Dilum & Anura P. Jayasumana (2007). An Enhanced Top-Down Cluster and Cluster Tree Formation Algorithm for Wireless Sensor Networks. Department of Electrical and Computer Engineering. Colorado State University, Fort Collins. Second International Conference on Industrial and Information Systems, ICIIS 2007.
- Callaway, Edgar H (2004). *Wireless Sensor Networks. Architectures and Protocols*. Auerbach. ISBN 0-8493-1823-8.
- Comer, Douglas E. (2009). *COMPUTER NETWORKS and INTERNETS*. Fifth Edition. Pearson Education. ISBN-10: 0-13-504583-5.
- Cook, David (2004). *Intermediate ROBOT BUILDING*. United States of America: APRESS. ISBN 1-59059-373-1.
- Corbet, Jonathan, Alessandro Rubini & Greg Kroah-Hartman (2005). *Linux Device Drivers*. Third Edition. O'REILLY. ISBN 1-56592-292-1.
- Hong, J.P., H.M. Shim, S.B. Jung, E.H. Lee & S.H. Hong (2001). A STEERING ALGORITHM OF THE MCU BASED CONTROLLER FOR TWO-WHEEL DRIVE VEHICLES. 0-7803-7090-2/01 ©2001 IEEE.
- Hui, J. & P. Thurbert (2009). Compression Format for IPv6 Datagrams in 6LoWPAN Networks. Draft-ietf-6lowpan-hc-06 [online] [cited 21 February 2010]. Available from the Internet: <URL: <http://tools.ietf.org/pdf/draft-ietf-6lowpan-hc-06.pdf>>.

IEEE (2006). IEEE Standard for Information technology – Telecommunication and information exchange between systems – Local and metropolitan area networks – Specific requirements. Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs).

Ippolito, Greg (2004). POSIX thread (pthread) libraries [online] [cited 13 May 2010]. Available from the Internet: <URL: <http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html>>.

Karl, Holger & Andreas Willig (2005). *Protocols and Architectures for Wireless Sensor Networks*. England: John Wiley & Sons Ltd. ISBN 0-470-09510-5.

Kuijsten, Dirk (2008). Sensor Measurements for Wi-Fi location with Emphasis on Time-of-Arrival Ranging. TU Delft.

Lombardo, John (2002). *Embedded Linux*. New Riders. ISBN 0-7357-0998-X.

Matrix Multimedia. Formula Flowcode. User guide. Available from the Internet: <URL: <http://www.matrixmultimedia.com/datasheets/HP794-30-1.pdf>>.

Matrix Multimedia (2005). LCD board datasheet. EB005-00-3 [online]. Available from the Internet: <URL: <http://www.matrixmultimedia.com/datasheets/EB005-30-3.pdf>>.

Microchip (2009). PIC18F2455/2550/4455/4550 Data Sheet. 28/40/44-Pin, High-Performance, Enhanced Flash, USB Microcontrollers with nanoWatt Technology [online]. Available from the Internet: <URL: <http://ww1.microchip.com/downloads/en/DeviceDoc/39632e.pdf>>.

Misic, Vojislav B., Jun Fung & Jelena Misic (2005). MAC Layer Security of 802.15.4-Compliant Networks. Department of Computer Science. University of Manitoba, Canada. 0-7803-9466-6/05 ©2005 IEEE.

Nanotron (2007). Real Time Location Systems (RTLS). A White Paper from Nanotron Technologies GmbH [online] [cited 07 March 2010]. Available from the Internet: <URL: http://www.nanotron.com/EN/pdf/WP_RTLS.pdf>.

National Semiconductor (2000). LM2937 500mA Low Dropout Voltage Regulator [online]. Available from the Internet: <URL: <http://www.datasheetcatalog.org/datasheet2/3/07p1guu8945hgdlzf2wh2ac5llpy.pdf>>.

Philips (2005). 74HC244; 74HCT244. Octal buffer/line driver; 3-state. Product data sheet. Rev. 03 – 22 December 2005.

Radiocrafts (2007). ZigBee – Ready RF Transceiver Modules. RC230x Data Sheet (rev. 1.2) [online] [cited 02 March 2010]. Available from the Internet: <URL: http://www.radiocrafts.com/uploads/rc230x_data_sheet_1_2.pdf>.

Santifaller, Michael (1994). *TCP/IP and ONC/NFS. Internetworking in a UNIX Environment*. Second Edition. Addison-Wesley (Deutschland) GmbH. ISBN 0-201-42275-1.

Sastry, Naveen & David Wagner (2004). Security Considerations for IEEE 802.15.4 Networks. University of California, Berkley. ACM 1-58113-925-X/04/0010 © 2004.

Schildt, Herbert (1999). *STL PROGRAMMING from the GROUND UP. Learn to Program Using the C++ Standard Template Library*. Osborne/McGraw-Hill. ISBN 0-07-882507-5.

Sensinode (2008). NanoStack Manual v1.0.2.

Tanenbaum, Andrew S. (2002). *Moderne Betriebssysteme*. Zweite überarbeitete Auflage. Pearson Studium. ISBN 3-8273-7019-1.

Taskit. Taskit Portux920T Overview [online] [cited 18 March 2010]. Available from the Internet: <URL: <http://www.taskit.de/en/products/portux/index.htm>>.

Van Heesch, Dimitri (2010). Doxygen. Source code documentation generator tool [online] [cited 19 April 2010]. Available from the Internet: <URL: <http://www.stack.nl/~dimitri/doxygen/>>.

WISM. Wireless Sensor Systems in Indoor Situation Modeling [online] [cited 6 February 2010]. Available from the Internet: <URL: <http://teg.uwasa.fi/projects/wism/>>.

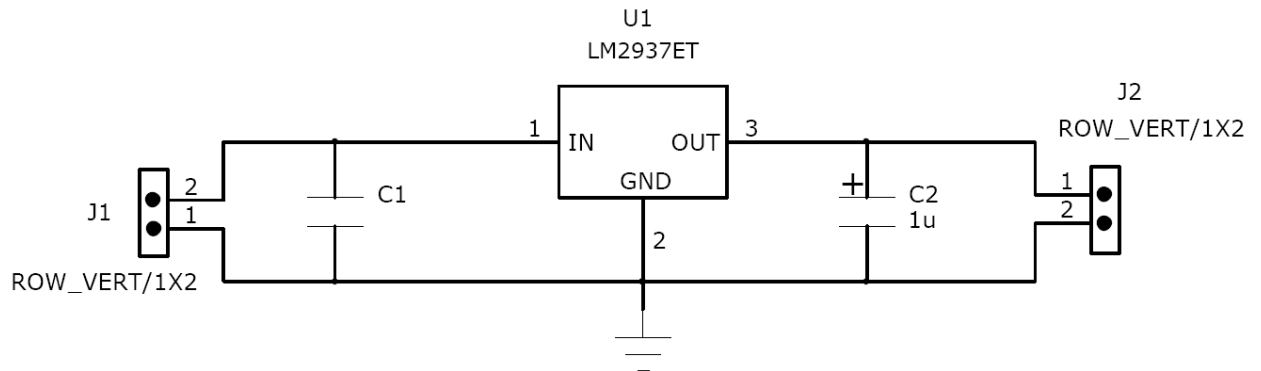
Xiao, Yang, Sakshi Sethi, Hsiao-Hwa Chen & Bo Sun (2005). Security Services and Enhancements in IEEE 802.15.4 Wireless Sensor Networks. 0-7803-9415-1/05 © 2005 IEEE.

Yu, Yang, Viktor K. Prasanna & Bhaskar Krishnamachari (2006). *Information Processing and Routing in Wireless Sensor Networks*. World Scientific Publishing Co. Pte. Ltd. ISBN 981-270-146-X

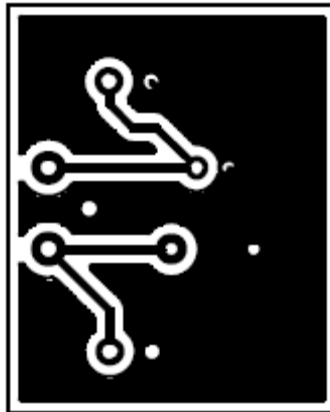
ZigBee Alliance (2005). ZigBee Specification. ZigBee Document 053474r06, Version 1.0.

APPENDIXES

APPENDIX 1. Voltage Regulator for Sensinode

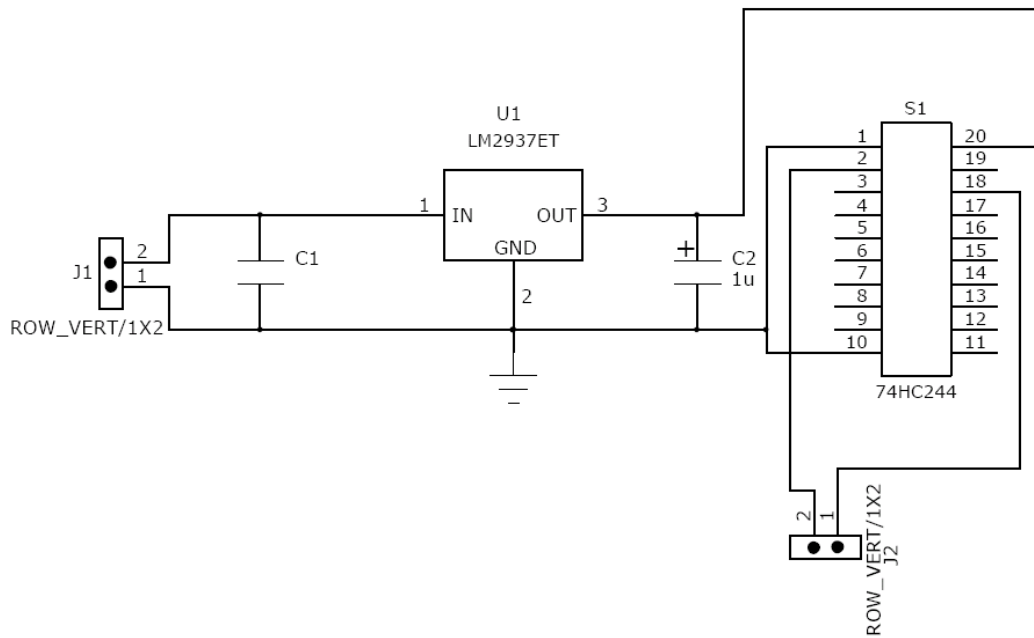


Schematic Design.

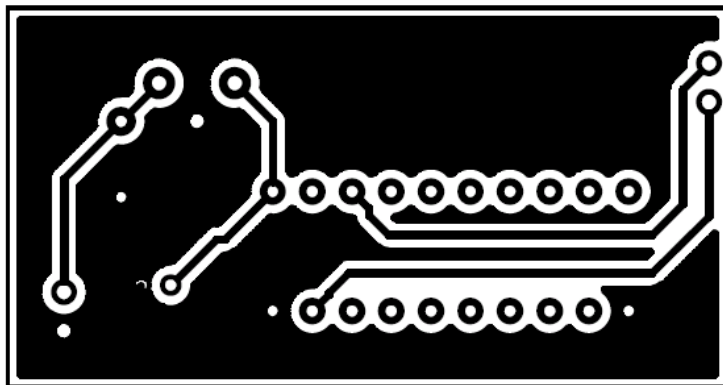


Printed Circuit Board (PCB) Design.

APPENDIX 2. Voltage Regulator for UART connection (PIC18F4455 / Embedded PC)

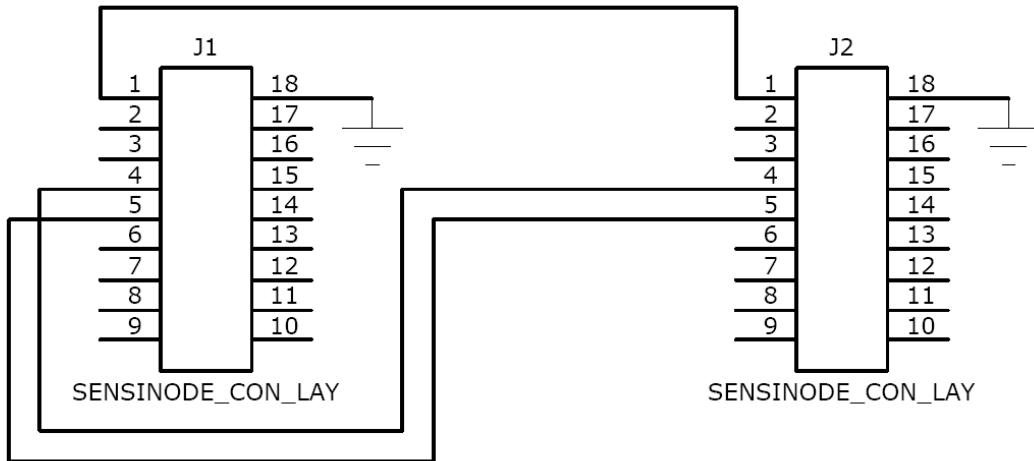


Schematic Design.

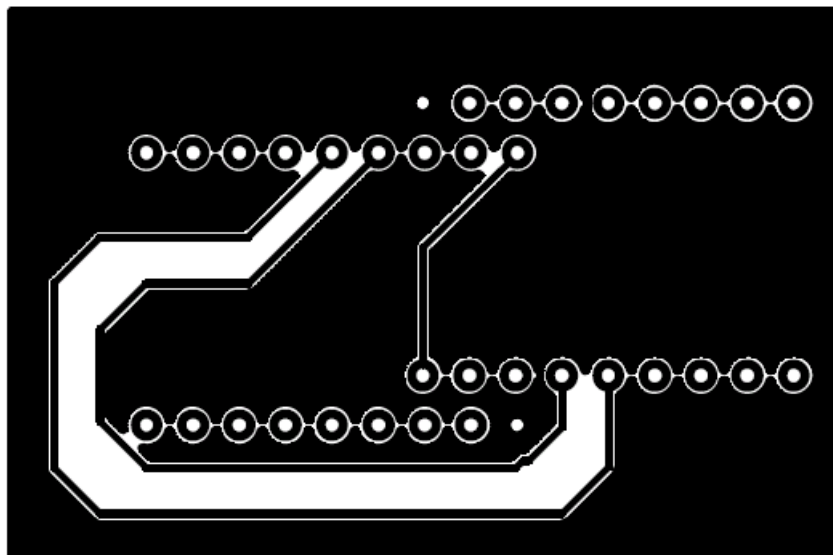


PCB Design.

APPENDIX 3. Extension Board for Sensinode

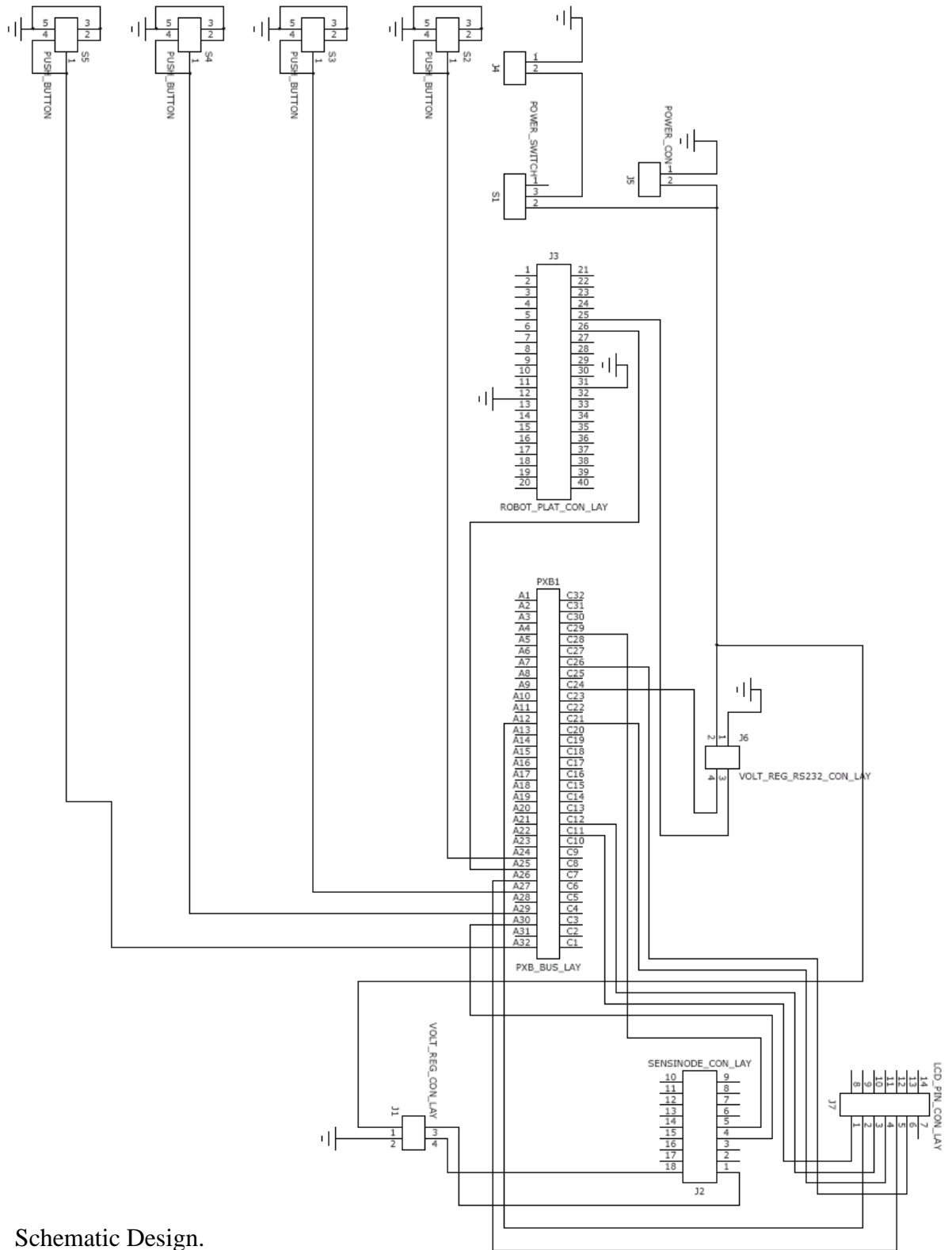


Schematic Design.

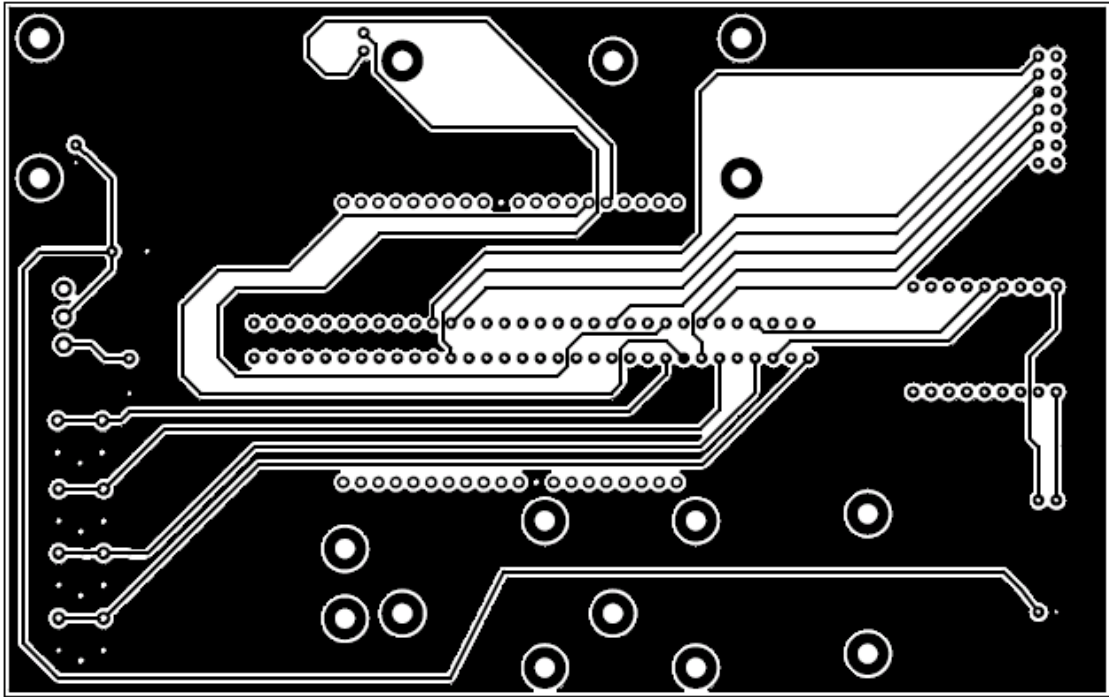


PCB Design.

APPENDIX 4. Main Board



Schematic Design.



PCB Design.

APPENDIX 5. Shell Scripts

```
#####
###
### rc.local script: startup script (contains cammands which need
### to be executed during the boot process)
###
### Copyright (C) 2009, Tobias Glocker (p87915@student.uwasa.fi)
###
### University of Vaasa
###
#####

#!/bin/sh

# hostname
/bin/hostname -F /etc/hostname

# link skript for hotplug
ln -s /config/hotplug /sbin/hotplug
devfsd /dev

# mount nfs on mnt
#/bin/mount -t nfs -o nolock 192.168.4.238:/develop/portux /mnt

# mount SD Card on data
mount -t vfat /dev/mmc/blk0/part1 /data

# run the configuration program
/data/runREMOconfig

# shutdown (halt) the OPS
#/data/shutdown.sh

#####
###
### Parameter script for Installation/Uninstallation
###
### Copyright (C) 2009, Tobias Glocker (p87915@student.uwasa.fi)
###
### University of Vaasa
###
#####

DATA_FOLDER="/data"
CONFIG_FOLDER="/config"
CURRENT_LOCAL_FILE="rc.local"
LOCAL_FILE_BACKUP="rc.local_backup"
MOUNT_NFS_FILE="mountNFS.sh"
SHUTDOWN_FILE="shutdown.sh"
REMO_CONFIG_FILE="runREMO.conf"
BINARY_FILE="runREMO"
CONFIG_LOCAL_FILE="$CONFIG_FOLDER/$CURRENT_LOCAL_FILE"
CONFIG_LOCAL_FILE_BACKUP="$CONFIG_FOLDER/$LOCAL_FILE_BACKUP"
```

```
STRING_FILE_NAMES="$CURRENT_LOCAL_FILE $MOUNT_NFS_FILE
                  $SHUTDOWN_FILE $REMO_CONFIG_FILE
                  $BINARY_FILE"
```

```
#####
###
### Mount script (mounts the Client with the NFS Server on the
###                Desktop PC)
###
### Copyright (C) 2009, Tobias Glocker (p87915@student.uwasa.fi)
###
### University of Vaasa
###
#####
```

```
#!/bin/sh
mount -t nfs -o nolock 193.166.111.170:/home/tobias/shared/mnt/
```

```
#####
###
### Shutdown script (shuts down the Embedded PC)
###
### Copyright (C) 2009, Tobias Glocker (p87915@student.uwasa.fi)
###
### University of Vaasa
###
#####
```

```
#!/bin/sh
halt
```

```
#####
###
### Uninstallation script
###
### Copyright (C) 2009, Tobias Glocker (p87915@student.uwasa.fi)
###
### University of Vaasa
###
#####
```

```
#!/bin/sh
```

```
# include paramters file necessary for the installation
. ./parameter.sh
```

```
# write to screen "starting uninstallation"
echo "starting uninstalltion!"
```

```

# check if data directory exists
if [ ! -d $DATA_FOLDER ]; then
    echo "Directory $DATA_FOLDER does not exist"
    exit 1
fi

# check if all the files needed for the installation exist
for filename in $STRING_FILE_NAMES
do
    if [ -f "$DATA_FOLDER/$filename" ]; then
        rm -f "$DATA_FOLDER/$filename"
    fi
done

# check if "rc.local_backup" file exists
# if yes rename it to "rc.local"
if [ -f $CONFIG_LOCAL_FILE_BACKUP ]; then
    mv $CONFIG_LOCAL_FILE_BACKUP $CONFIG_LOCAL_FILE
fi

# write to screen "uninstallation was successful"
echo "uninstallation was successful!"

#####
###
### Installation script
###
### Copyright (C) 2009, Tobias Glocker (p87915@student.uwasa.fi)
###
### University of Vaasa
###
#####

#!/bin/sh

# include paramters file necessary for the installation
. ./parameter.sh

# write to screen "starting installation"
echo "starting installation!"

# check if data directory needed for the installation exists
if [ ! -d $DATA_FOLDER ]; then
    echo "Directory $DATA_FOLDER does not exist"
    exit 1
fi

# remove the existing files
for filename in $STRING_FILE_NAMES
do
    if [ ! -f $filename ]; then
        echo "File $filename does not exist"
        exit 1
    fi
done

```

```
# rename the current "rc.local" file in the "/config" directory
# to "rc.local_backup", if it exists and if there is no backup
# file available
if [ -f $CONFIG_LOCAL_FILE ]; then
    if [ ! -f $CONFIG_LOCAL_FILE_BACKUP ]; then
        mv $CONFIG_LOCAL_FILE $CONFIG_LOCAL_FILE_BACKUP
    fi
fi

# copy the updated (new) "rc.local" to the "/config" directory
cp $CURRENT_LOCAL_FILE $CONFIG_FOLDER

# copy the "mountNFS.sh" script to the "/data" directory
cp $MOUNT_NFS_FILE $DATA_FOLDER

# copy the "shutdown.sh" script to the "/data" directory
cp $SHUTDOWN_FILE $DATA_FOLDER

# copy the executable file "runRemo" to the "/data" directory
cp $REMO_CONFIG_FILE $DATA_FOLDER

# copy the config file "runRemo.conf" to the "/data" directory
cp $BINARY_FILE $DATA_FOLDER

# write to screen "installation was successful"
echo "installation was successful!"
```

APPENDIX 6. Makefile

```
#####
###
### Makefile for REMO
###
### Copyright (C) 2009, Tobias Glocker (p87915@student.uwasa.fi)
###
### University of Vaasa
###
#####

# compiler
CC = arm-linux-3.4.2-g++

# compiler options
CFLAGS = -Wall -Werror -D_REENTRANT

# static linked libraries directory/directories
STATIC_LIB_DIRS := ../staticLib

# linker options
LDFLAGS = -lpthread -static-libgcc

# source file extensions
SOURCE_FILE_EXTENSION = cpp

# collect all the files ending with SOURCE_FILE_EXTENSION
# (also source files in the subfolders will be considered)
SRCS := $(shell find . -name '*.$(SOURCE_FILE_EXTENSION)')

# name of the executeable file
OUTPUT_FILE = runREMO

# get the list of object files from the source files
OBS := $(SRCS:%.$(SOURCE_FILE_EXTENSION)=%.o)

# depend file to generate dependencies with header files
DEPENDFILE = .depend

# if TARGET=debug then add debugging information to the compiler
# flags
ifeq ($(TARGET),debug)
    CFLAGS += -g -DDEBUG -DDEBUG_MOTOR -DDEBUG_SERIAL_SENSINODE
                -DDEBUG_REMO_MATH -DDEBUG_PACKET
endif

# compile and link the program
#link
prog: $(OBS)
    $(CC) $(CFLAGS) $(LDFLAGS) -L $(STATIC_LIB_DIRS) -o
    $(OUTPUT_FILE) $(OBS)

# compile
# every object file depends on the corresponding
(SOURCE_FILE_EXTENSION) file
```

```
%.o: %.$(SOURCE_FILE_EXTENSION)
    $(CC) -o $@ -c $(CFLAGS) $<

# generate the dependency file
dep: $(SRCS)
    $(CC) -MM $(SRCS) > $(DEPENDFILE)

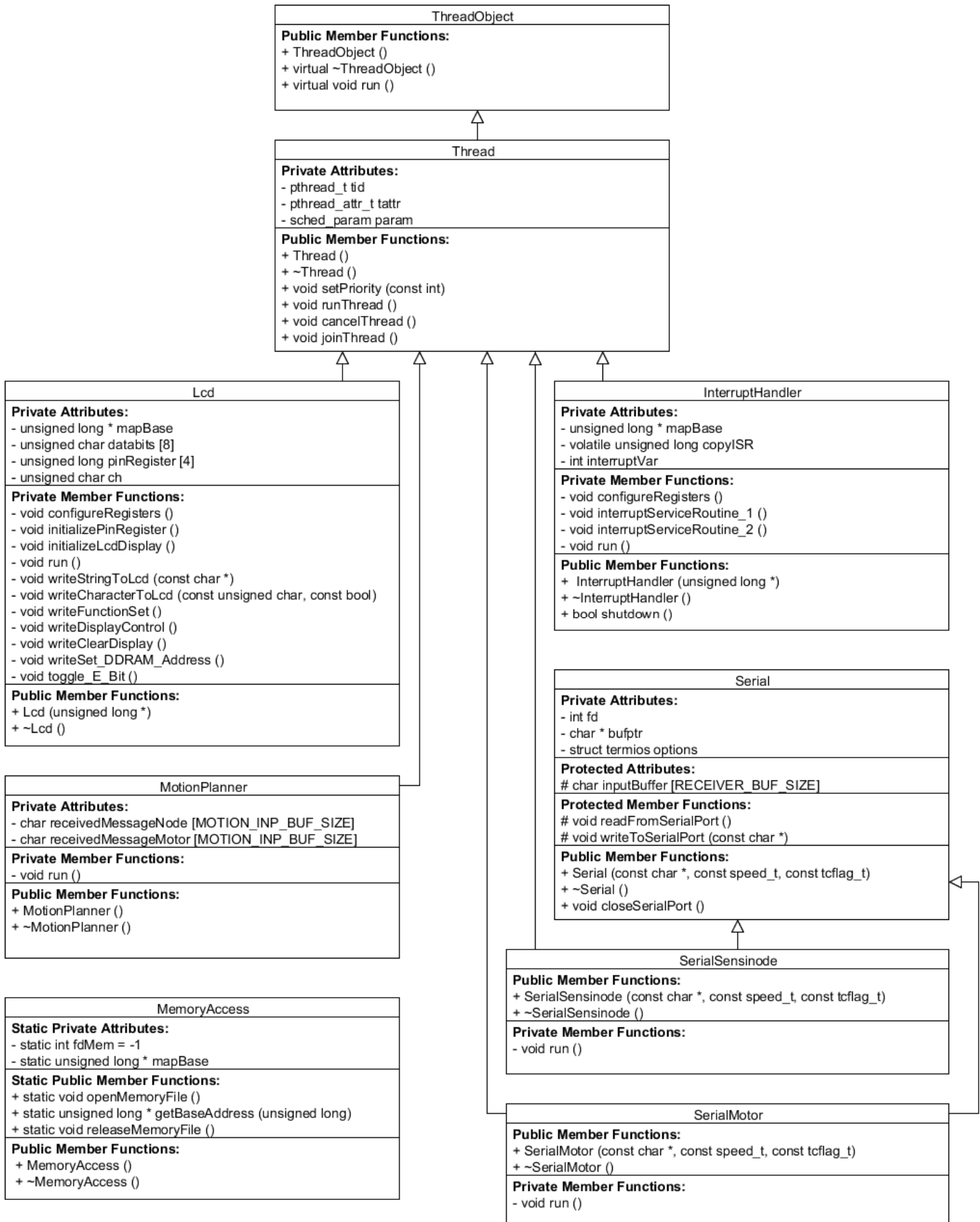
-include $(DEPENDFILE)

# generate the dependency file, compile and link the program (with
# debugging output)
all: dep prog

# erase the executable file and the object files
clean:
    -rm -f $(OUTPUT_FILE) $(OBS)
    @echo "make clean done."

# erase the executable file, the object files and the
# dependency file
cleanall: clean
    -rm -f $(DEPENDFILE)
    @echo "make cleanall done."
```

APPENDIX 7. Class Diagrams of the Main Program (Embedded PC)



FileParser
Static Private Attributes: - static char filename [STRING_BUFFER_SIZE] - static char variableValue [STRING_BUFFER_SIZE] - static char inputLine [STRING_BUFFER_SIZE]
Static Private Member Functions: - static inline bool parseVariableString (const char *, const char *) - static inline bool findVariableString (std::ifstream &, const char *)
Static Public Member Functions: + static void setFileName (const char *) + static const char *getStringValue (const char *, const char *) + static inline float getFloatValue (const char *, const char *) + static inline int getIntValue (const char *, const char *) + static void writeStringValue (const char *, const char *, const char *)

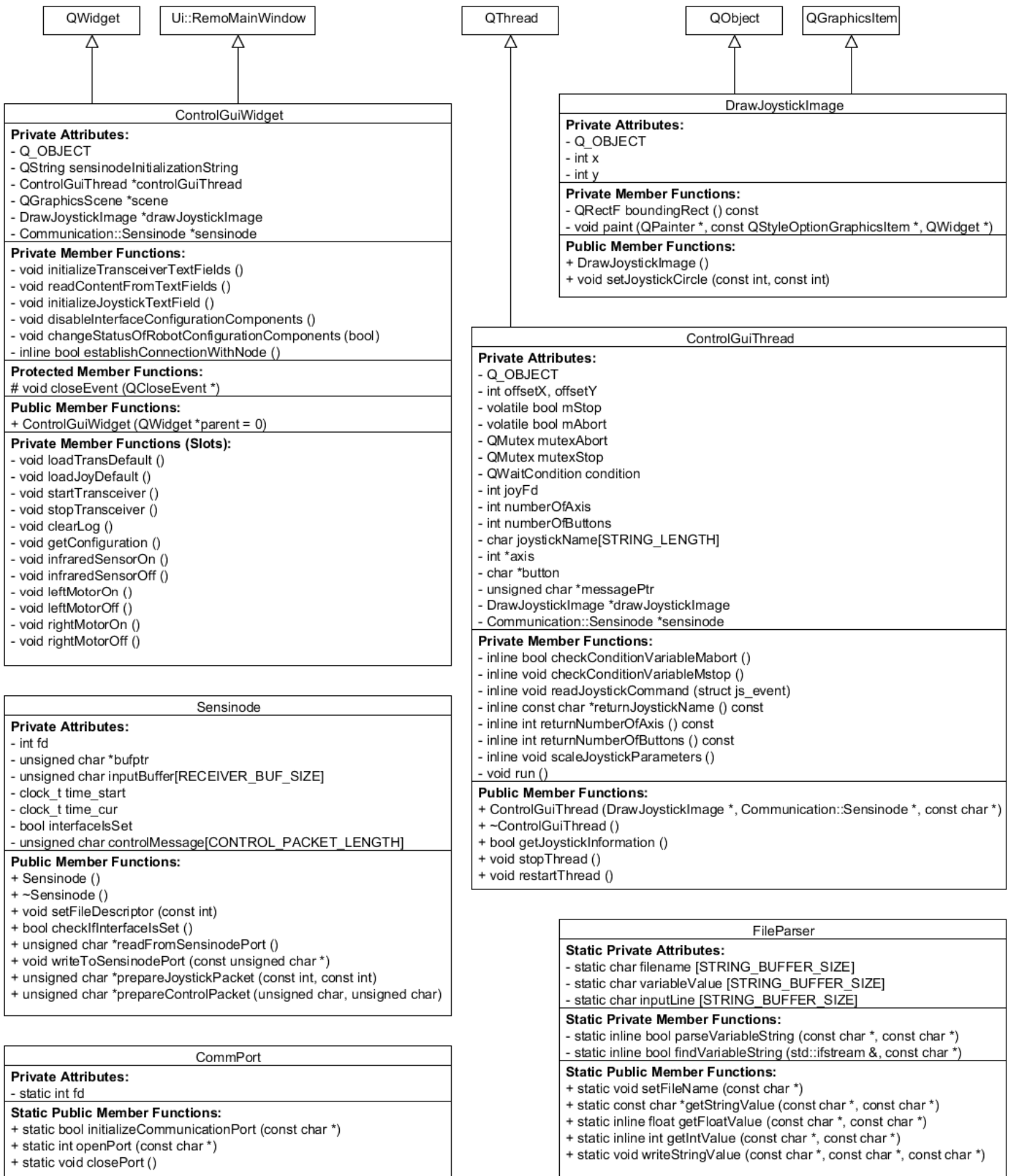
SharedBuffer
Private Attributes: - pthread_mutex_t mutexString - pthread_mutex_t mutexInt - char sharedStringVariable [SHARED_BUFFER_SIZE] - int sharedIntVariable
Public Member Functions: + SharedBuffer () + ~SharedBuffer () + void writeToBuffer (const char *) + void writeToBuffer (const int) + void readFromBuffer (bool &, char *) + void readFromBuffer (int *)

RemoMath
Static Public Member Functions: + static void computeWheelSpeeds (const char *, const int, int &, int &) + static int moveUntilRSSIValuesReached (const int, const int)
Public Member Functions: + RemoMath () + ~RemoMath ()

Timer
Private Attributes: - struct tm * _time - time_t sec
Public Member Functions: + Timer () + ~Timer () + void getCurrentTime () + unsigned int getSecond () const + unsigned int getMinute () const + unsigned int getHour () const

DataLogger
Private Attributes: std::ofstream logfile
Public Member Functions: + DataLogger (const char *) + ~DataLogger () + writeDataToFile (const char *) + closeFile ()

APPENDIX 8. Class Diagrams of the Graphical Control User Interface



APPENDIX 9. Picture of the Mobile Robot

