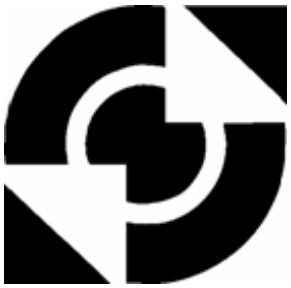
University of Twente

EEMCS / Electrical Engineering
Control Engineering



A software solution for absolute position estimation using WLAN for robotics

Bart Deen

MSc report

Supervisors:
prof. dr. ir J. van Amerongen
dr.ir. J.F. Broenink
ir. E.C. Dertien

June 2008

Report nr. 022CE2008
Control Engineering
EE-Math-CS
University of Twente
P.O.Box 217
7500 AE Enschede
The Netherlands

"I have not set off this little work with pompous phrases, nor filled it with high-sounding and magnificent words, nor with any other allurements of extrinsic embellishments with which many are wont to write and adorn their works; for I wished that mine should derive credit only from the truth of the matter and that the importance of the subject should make it acceptable."

Niccolò Machiavelli - The Prince & The Art of War - 1532

Summary

This project is divided in two parts; Software frameworking for robots and absolute position estimation using the WLAN infrastructure.

With increasing demands in functionality and complexity of a robot, the software complexity increases also. This increase in complexity stresses the limits of conventional software writing used today. To release this stress and give structure to the complexity of the software, a robust framework using a modular approach is needed. After a literature analyse of seven software architectures for robotics, Orocos matched best with the requirements defined in this project.

Tests of Orocos' implementation on the JIWY setup shows Orocos' ability for hard real-time execution of tasks and communication between non real-time and hard real-time task. This without significant performance loss. Drawback of using Orocos are the resources needed.

Orocos is a promising framework and should be implemented on a more complex system than JIWY to explore the real power of this framework

Position estimation is a vital part in developing an autonomous mobile robot. Before a mobile robot can reach its destination it must know its current position. Some dedicated absolute position systems are developed, like GPS and Galileo. These systems work great when in sight of the satellites but are of no use indoors. This project aims to fill this gap and create a method which uses existing WLAN infrastructure for indoor absolute positioning.

Position estimation using propagation models is chosen instead of empirical models. Propagation models are better suitable on an embedded system where recourses are limited. Developing a WLAN positioning method results in a evaluation of four methods; (two radio propagation methods and two trilateration methods) in two different situations; moving from point to point for navigation purposes and stationary on a fixed point for orientation purposes.

In the situation "stationary on a fixed point" the best result was obtained with the ITU path loss model and the linear trilateration method. This method achieved mean error $< 3m$ and can be used for orientation purposes. Orientation means in this case "Am I in the right room", "Which direction is the nearest recharger" etc. Positioning when moving from point to point results in a position dispersion too great to be of any use. Navigation using the WLAN infrastructure in this setting is therefore not possible.

Dedicated hardware, for measuring the signal strength, should result in more reliable measurement with less dispersion. This should result in better position estimations.

Samenvatting

Dit project is opgedeeld in twee delen; Software frameworking voor robots en absolute positie bepaling door middel van WLAN infrastructuur.

Door de toenemende vraag naar functionaliteit en complexiteit van een robot, neemt de complexiteit van de software ook toe. Deze toename in complexiteit zet conventionele methoden voor het schrijven van software onder druk. Om deze druk te verlichten en structuur te geven aan de gecompliceerde software, is er behoefte aan een krachtig modulair software framework. Uit een literatuur onderzoek naar zeven architecturen, ontwikkeld voor robotica, blijkt Orocos het meest geschikt.

Testen die zijn uitgevoerd met de implementatie van Orocos op de JIWY opstelling, tonen aan dat Orocos hard real-time taken kan uitvoeren en kan blijven communiceren met niet real-time taken, zonder veel prestatie verlies te leiden. Nadeel van Orocos is de hoeveelheid resources die Orocos nodig heeft.

Orocos is een veelbelovend framework. Een implementatie op een complexer systeem dan JIWY zal nodig zijn om de echte kracht van Orocos te verkennen en te laten zien.

Positie bepaling is een elementair onderdeel tijdens de ontwikkeling van automatische mobiele robots. Voordat een robot zijn bestemming kan bereiken, moet zijn huidige positie bekend zijn. Enkele speciale absolute positie bepalings systemen zijn ontwikkeld, zoals GPS en Galileo. Deze systemen werken goed wanneer de satelieten in zicht zijn, maar zijn zinloos voor gebruik binnenshuis. Dit project heeft als doel een methode te ontwikkelen die deze overgang mogelijk maakt. Dit, door gebruik te maken van het bestaande WLAN infrastructuur voor absolute positie bepaling binnenshuis.

Er is gekozen voor propogatie modellen in plaats van calculatie intensieve empirische modellen. Propogatie modellen zijn beter geschikt voor een embedded systeem, omdat daar de resources beperkt zijn. De ontwikkeling van de WLAN positie module resulteert in een evaluatie van vier mogelijke methodes (twee propogatie methodes en twee trilateratie methodes) in twee verschillende situaties; verplaatsen van punt naar punt voor navigatie doeleinden en stationair op een vast punt voor oriëntatie doeleinden.

In de situatie “Stationair op één punt” werden de beste resultaten behaald met de ITU path loss model en de linear trilateration methode. Deze methode heeft een gemiddelde fout $< 3m$ en kan gebruikt worden voor oriënterende doeleinden. Oriëntatie betekent hier “Ben ik in de juiste kamer”, “Waar is het dichtstbijzijnde oplaad punt”. Positionering tijdens verplaatsing van punt naar punt levert een te grote positie spreiding op, dat het geen enkel nut heeft. Navigate met behulp van WLAN is daarom niet mogenlijk.

Toegepaste hardware, voor het meten van signaal sterkte, zal betrouwbaarder metingen opleveren met minder spreiding. Dit zal resulteren in een betere positiebepaling.

Contents

1 Introduction

Different sectors are active in developing multi-task autonomous mobile systems. The military is developing unmanned vehicles (seen figure 1.1(a)) and planes. With the "DARPA grand challenge" (DARPA, 2008) the development of unmanned vehicles got a big impulse. The industry uses robots a long time in production and now warehouses are being robotized (Kiva Systems, 2008). A robot from this system is shown in figure 1.1(b). In the domestic sector robots are developed for pleasure and to perform tasks for users with limited abilities, like the El-E (Nguyen et al., 2008) shown in figure 1.1(c).

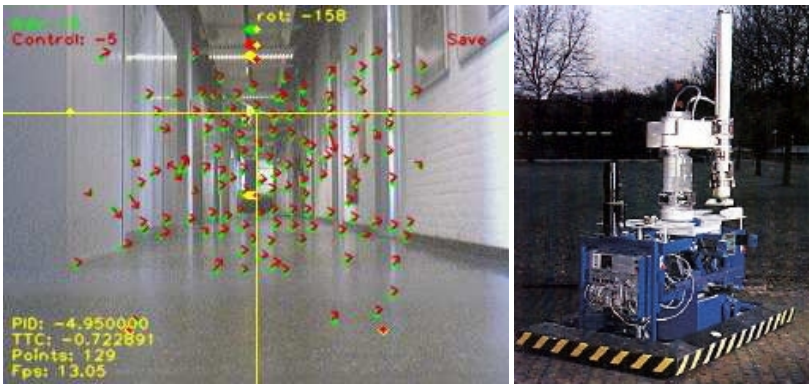
(a) Winner DARPA urban challenge 2007 (b) Kiva systems robot (c) El-E fetchrobot

FIGURE 1.1 - Mobile autonomous robots

Orientation and position estimation are important aspects for autonomous mobility of robots. Position estimation can be classified in two groups; relative and absolute localization. Relative localization uses system information, like odometry, velocity and angles, to derive a position. Absolute localization uses sensors for finding environmental features or beacons to derive a position. GPS is a widely used absolute position system but is not suitable for indoor use because the sensor must be in the line-of-sight of the satellites to be successful. Most indoor absolute localization uses dedicated sensors and beacon networks based on vision, IR or radio beacons or Low Frequency Magnetic Fields (Prigge and How, 2000). At Control Engineering group vision and IR has been succesfully implemented. Usov (2006) implemented navigation with vision for a wheeled robot as is shown in figure 1.2(a). IR was used in positioning MART, shown in figure 1.2(b), by Schipper (2001).

This project investigates the use of WLAN Access-Points for absolute localization. WLAN can be used because WLAN Access-Point have fixed positions within the building and can be seen as radio transmitting beacons. Furthermore, the WLAN infrastructure is present in most of, if not all, buildings and available on robots as communication interface to the outside world. Therefore, using WLAN for positioning is a cheap solution compared to a dedicated beacon network.

With increasing demands in functionality and complexity of a robot, the software complexity increases also. Due to the increasing software complexity, the algorithms and structure for computation are not the main design challenges anymore (Garlan and Shaw, 1994). Designing and specifying the overall system structure emerges as a new kind of problem. Effective software engineering requires a facility in architectural software design. It is important to be able to recognize common paradigms in order to understand high-level relationships among systems and so that new systems can be built as variations on old systems.



(a) Vision based navigation (b) MART

FIGURE 1.2 - Navigation at the Control Engineering group

A software framework for robotics is different from most software frameworks in the way it interacts with physical objects. This demands for stricter timing constraints which the framework must be able to handle. If the timing of a particular control signal is wrong, a robot can damage itself or its environment. A robot is therefore a Hard Real-Time Embedded system.

1.1 The assignment

The assignment for this project can be formalized as: "Develop a method for absolute positioning, within a software framework for robotics of a mobile platform, moving indoors, using the existing WLAN infrastructure. The software framework must be able to cope with the increase software complexity of today's robots".

A software framework is presented which can accommodate the requirements for use in robotics. This framework is used to implement the algorithms needed for absolute position estimation using the WLAN infrastructure. The position estimation of a mobile device is done by measuring the received signal strenght from WLAN Access-Points. From this measurements, a distance to the Access-Point can be calculated. A minimum of three distances to Access-Points and known positions of these Access-Points are needed to calculate the position of the mobile device. Before implementing the algorithms for position estimation, the algorithms will be evaluated with Matlab.

As demonstrator, the JIWY setup will be used. During the demo a pointing device cq. camera will follow a moving target, in this case a person walking with a laptop. This demo will demonstrate:

- The ability of distributed control of the framework by using two computers
- The ability of loop control of the framework by controlling two dc motors
- The working of position estimation using WLAN infrastructure

1.2 Report outline

Chapter two is a literature study on software frameworks used in robotic. This study points out a framework suitable for use in robotics. Designing a method for positioning, using the WLAN infrastructure, is discussed in chapter 3. After a framework is chosen and the development of position estimation is treated, the development of the demonstrator is explained in chapter 4. The conclusion and recommendations of this project are described in chapter 5.

2 Software frameworks for robotics

2.1 Introduction

A software framework facilitates the software engineer with a structured way-of-working. Through a modular approach, provided by the software framework, the complexity of developing software is divided into subsystems. The framework provides the infrastructure of communication and scheduling between the subsystems. Each subsystem can be developed, tested and evaluated before taken part of the total system. This can result in robust, reusable, reliable code modules. It also enables parallel development by different engineers with different expertise. In the end, use of a framework results in robust, reliable, reusable and maintainable source code and shortens the development time.

A software framework for robotics is different from most software frameworks in the way it interacts with physical objects. This demands stricter timing constraints which the framework must be able to handle. If the timing of a particular control signal is wrong, a robot can damage itself or its environment. A robot is therefore a Hard Real-Time Embedded system.

In this chapter, seven software architectures for robotics are presented. They are evaluated for suitability in robotics according the requirements described in section 2.2. Goal is to present a software architecture which enables to simplify and shorten the time of developing robust and reliable software for robots.

2.2 Requirements for a software framework

The software frameworks presented in section 2.3 are evaluated if they satisfy not only classic robotic requirements, like real-time behavior and intensive access to hardware, but also on software engineering aspects as re-usability, maintainability, etc.

The framework must be able to implement new technologies without redefining the software structure. This can be accomplished with various levels of abstraction. A higher level only knows what the underlying layer is able to perform, not how it is achieved. For example: position estimation can be done by relative and/or absolute positioning. By means of abstraction the higher level only needs coordinates of the position, not the way it is estimated. This level of abstraction is also needed for testing the software with virtual hardware, as described by Damstra (2008).

At the Control Engineering group 20-sim (Controllab Products B.V., 2008) is frequently used for creating, simulating and evaluating mechatronic controllers. Automatic code generation is used to implement the controller on a real setup. It is desirable that the proposed software framework is capable of easy implementing these 20-sim generated source code.

Robot behavior is often described in a Finite State-Machine. It is therefore desirable that the framework is able to execute Finite State-Machines on the fly.

A robot can be seen as a Distributed system when, multiple computers are used to control the robot. And the robot can be part of a Distributed system, when multiple robots are used for one goal. In both cases, reliable network communication is essential. The proposed framework is evaluated on availability of network communication possibilities.

The requirements used for evaluation of the software frameworks are shown in table 2.1. Section 2.4 contains the results from the software architectures presented in section 2.3.

Requirements for evaluation	
Requirement	**Description**
Abstraction levels	Availability of various levels of abstraction.
Distributed system	Posability to be part of a distributed system.
Resources	Efficiency of resources use.
Simulation	Suitability for co-simulation/Hardware in the loop.
Real-time	Availability of Hard real-time behavior.
20-sim	Possibility of importing generated code from 20-sim.
Finit State-machine	Possibility of executing Finit state-machines.
Documentation	Availability of up-to-date documentation.

TABLE 2.1 - Requirements for evaluation of software frameworks

Figure 2.1 shows the structure, proposed by Lootsma (2008) for the TULip humanoid robot (Dutch Robotics, 2008) which is being built at this moment at the Control Engineering group. The chosen Framework must be able to accommodate this kind of structure.

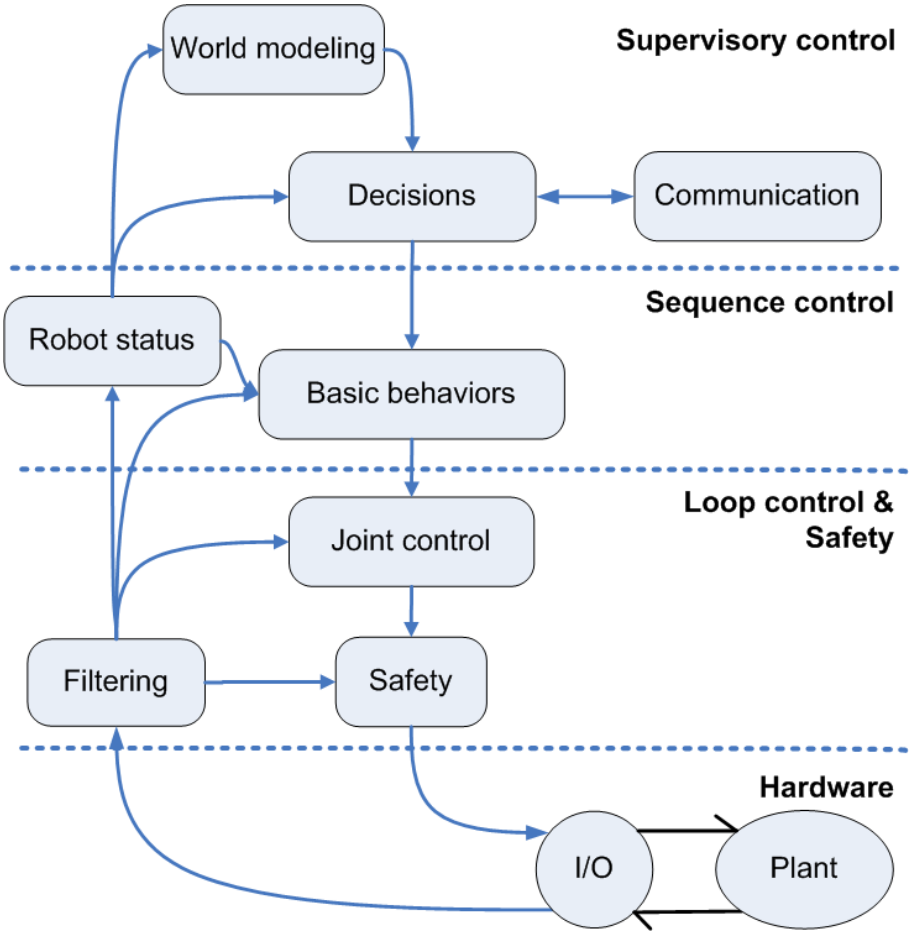


FIGURE 2.1 - Advised structure for the TUlip robot

In the next section a selection of software architectures specialized in robotics are presented.

2.3 Presentation of software architectures for application in robotic

In this section the following software architectures are presented:

- **Yarp**; Yet Another Robot Platform.
- **CLARAty**; Coupled-Layer Architecture for Robotic Autonomy.
- **Orocos**; Open Robot Control Software.
- **URBI**; Universal Real-time Behavior Interface.
- **Orca**; Components for Robotics.
- **RoboFrame**; Robot Framework.
- **gCSP and CTC++**; Graphical tool for Communicating Sequential Processes and Communication Threads for C++.

These software architectures are evaluated on the requirements described in section 2.2. This evaluation is discussed in section 2.4.

2.3.1 Yarp

YARP, Yet Another Robot Platform (Metta et al., 2006), is an open-source project that provides a set of libraries, protocols, and tools to keep software modules and devices drivers decoupled. YARP is written by and for researchers in robotics, particularly humanoid robotics The goal of YARP is to minimize the effort devoted on infrastructure-level software development. This by facilitating code reuse, modularity and to maximize research-level development and collaboration.

The components of YARP can be divided into:

- The libYARP_OS; the interface with the OS to support streaming data across multiple threads on different machines. YARP is written to be OS neutral and has been used on Linux, Microsoft Windows, Mac OSX and Solaris. It uses the open-source ACE (Adaptive Communication Environment) library, which is portable across a very broad range of environments, and YARP inherits that portability.
- The libYARP_sig; performing common signal processing tasks (visual, auditory) in an open manner and can be easily interfaced with other commonly used libraries, for example OpenCV (Open source Computer Vision library).
- The libYARP_dev; containing the interface with common devices used in robotics: framegrabbers, digital cameras, motor control boards, etc.

The three packages of components are maintained separately. Both libYARP_sig and libYARP_dev depends on the core component is libYARP_OS. YARP is currently used and tested on Windows, Linux and QNX6 which are common operating systems used in robotics.

YARP has the ability to redistribute computation load of processes on different CPUs on the fly. This results in a sort of “soft real-time” parallel computation cluster without the more demanding requirements of a real-time operating system (Metta et al., 2006).

2.3.2 CLARAty

CLARAty stands for Coupled-Layer Architecture for Robotic Autonomy (Nesnas et al., 2003). It is a collaborative effort among four institutions: Jet Propulsion Laboratory, NASA Ames Research Center, Carnegie Mellon, and the University of Minnesota. CLARAty develops a framework for generic and reusable robotic components that can be adapted to a number of heterogeneous robot platforms. It also provides a framework that will simplify the integration of new technologies and enables the comparison of various elements. CLARAty consists out of two distinct layers: a Functional Layer and a Decision Layer. The interaction between the layers is

shown in figure 2.2. The Functional Layer defines the various abstractions of the system and adapts the abstract components to real or simulated devices. It provides a framework the algorithms for low- and mid-level autonomy. The Decision Layer provides the system's high-level autonomy, which reasons about global resources and mission constraints. The Decision Layer accesses information from the Functional Layer at multiple levels of granularity.

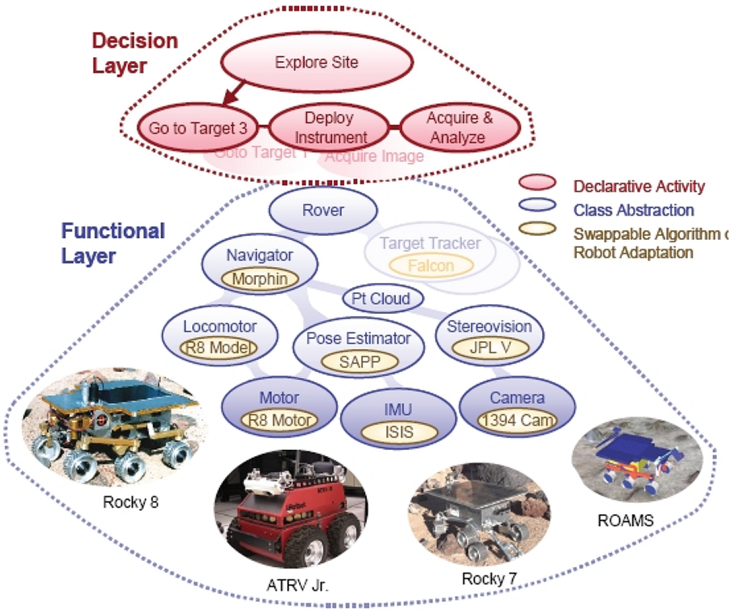


FIGURE 2.2 - Functional Layer and Decision Layer within CLARAty[I]

CLARAty contains a set of public modules and a set of restricted or private modules. The different access privileges dependents on Intellectual Property and ITAR (International Traffic in Arms Regulations) restrictions.

2.3.3 URBI

URBI, described by Baillie (2004) stands for Universal Real-time Behavior Interface and is a robotics platform currently maintained and developed by Gostai (gostai, 2008). URBI is a scripted interface language designed to work over a client/server architecture in order to remotely control a robot or, in a broader definition, any complex system. URBI for robotics is more than a simple driver for the robot, it is a universal way to control the robot. It adds functionalities by plugging software components and develop a fully interactive and complex robotic application in a portable way. The main distinctive qualities of URBI are the following:

- Simplicity: URBI is easy to understand, but with high level capabilities. URBI is suitable both for educational and professional applications.
- Flexibility: URBI is designed to be independent from both the robot and the client system or language (it currently works together with C++, Java, Matlab, on Windows, Mac OSX or Linux). URBI relies on TCP/IP or Inter-Process Communication.
- Parallelism: Parallel processing of commands, concurrent variable access policies, event based programming,...

The platform does not have its own simulation environment, but it has been recently integrated with Webots (cyberbotics, 2008), a popular commercial robot simulation environment.

[I] source `http://claraty.jpl.nasa.gov`

2.3.4 Orocos

Orocos (Bruyninckx, 2001) is the acronym of the Open Robot Control Software project. The goal of this project is to develop a general purpose modular framework for robot and machine control. It is organized in the form four C++ libraries (see figure 2.3):

1 the Real-Time Toolkit(RTT) - provides infrastructure and functionality to build component based real-time application.
2 the Kinematics and Dynamics(KDL) library - provides primitives to calculate kinematic chains.
3 the Bayesian Filtering library(BFL) - provides a framework to perform inference using Dynamic Bayesian Networks.
4 the Orocos component library(OCL) - provides some ready to use components.

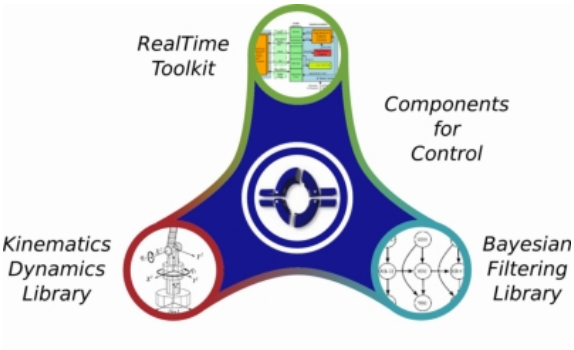


FIGURE 2.3 - Orocos libraries, Bruyninckx (2001)

Orocos components are built using RTT but also can use functionalities provided by other libraries. Users can interface/interact with the component through its predefined set of interfaces. In addition to defining component interface and communication mechanisms, Orocos allows application builders to implement hierarchical state machines using these mechanisms. The state machines can be loaded at runtime in any component. All Orocos components are implemented using the "TaskContext" class. TaskContext defines a "context" in which the component task is executed, where a task is a basic unit of functionality which executed one or more programs(a C function, script or state machine) in a single thread. All the operations performed within this context are free of locks, priority-inversions, i.e. thread-safe deterministic.

Orocos uses CORBA (Common Object Request Broker Architecture) as middleware when distributed control is desirable. CORBA is a standard defined by the Object Management Group ("OMG", 2008).

2.3.5 ORCA

Orca (Makarenko et al., 2006) is an open-source software project which applies Component-Based Software Engineering (CBSE) principles to robotics. CBSE offers developers the opportunity to source existing plug-in software components, rather than building everything from scratch. Orca's main goal is to promote software reuse in robotics targeting the range from single vehicles to distributed sensor networks. ICE (Internet Communications Engine) is used to create a distributed component-based system.

The project contains two distributions:

- Hydro - Libraries of algorithm and driver implementations.
- Orca - Component-based framework for building robotic systems.
- ICE - Internet Communications Engine (Extern)

Orca uses Hydro's libraries to access robotic hardware and process the data. ICE is used for inter-component communication but is developed by ZeroC (2008). Figure 2.4 show the Orca's connection with Hydro and ICE.

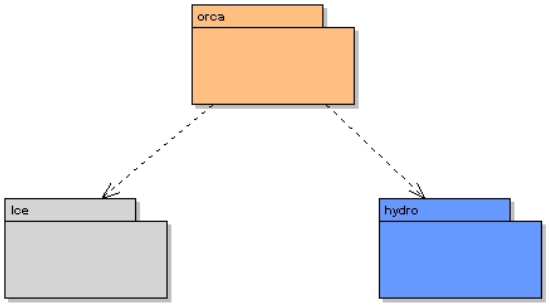


FIGURE 2.4 - Orca connection with Hydro and Ice

2.3.6 Roboframe

RoboFrame (Petters et al., 2007) is a modular software framework for lightweight autonomous robots and created by Technical University of Darmstadt. RoboFrame has been developed and designed to meet the special needs of systems with challenging dynamical locomotion like small size humanoid robots or unmanned aerial vehicles which impose strong restrictions on the available payload and thus the available CPU power. Due to its modular design, the development of control software for different types of robots with a large variety of reactive and deliberative control paradigms, is simplified and can be accomplished in a short time. Debugging, monitoring and control of applications is supported by an expendable and also platform independent graphical user interface.

2.3.7 gCSP and CTC++

gCSP (Jovanovic et al., 2004) is a graphical tool for creating and editing CSP diagrams (Hilderink, 2005). CSP diagrams are dataflow diagrams, connecting processes with channels. Beside the dataflow, the concurrency structure is also indicated. Next to editing diagrams, gCSP does basic consistency checks and can generate code from the diagrams. The generated code needs the CTC++ library for implementing CSP primitives in modern programming languages and for general purpose microprocessors.

A gCSP model is connected with the hardware through linkdrivers. Linkdrivers are objects within the CT library that convert the channel interface to hardware instructions. Linkdrivers abstract away hardware-dependent instructions.

gCSP and CTC++ are used at the Control Engineering group for implementing controllers, simulated and evaluated using 20-sim, on mechatronic setups.

2.4 Evaluation of software frameworks for robotics

Table 2.2 shows the results from the literature study on software architectures for robotics based on the requirements described in section 2.2.

CLARAty is promising but there are different access privileges depending on Intellectual Property and ITAR restrictions. Therefore CLARAty is not a candidate for the proposed framework.

URBI is a client/server approach with its own scripting language. The layered structure as shown in figure 2.1 cannot be achieved with URBI and therefore URBI is dismissed.

Requirement	Roboframe	Orca	Orocos	URBI	CLARAty	YARP	gCSP and CTC++
Abstraction levels	+	+	+	-	+	+	-
Distributed system	+	+	+	-	+	+	-
Resources efficiency	+	?	-	?	+	+	+
Simulation	+	+	+	-	+	+	+
Real-time	-	-	+	-	?	-	+
20-sim	+	+	+	+	+	+	+
Finit State-machine	+	-	+	-	-	-	-
documentation	+	+	+	+	+	+	-
open source	-	+	+	+	-	+	-

TABLE 2.2 - Evaluation of software frameworks

Yarp and Orca scores positive on most of the requirements but lacks a defined structure. Both exists out of set of libraries and protocols. Because a defined structure is missing, the source code becomes unmaintainable when different students, in time, are working on variation of the robot. The Control Engineering group has experience this with 20Works.

Roboframe was proposed by Delft University of Technology in the TUlip project because Technical University of Darmstadt succesfully implemented RoboFrame on humanoid robot. RoboFrame scores positive on most of the requirements but misses Real-Time behavior. Lootsma (2008) proposed a solution to this problem and RoboFrame will be used in the TUlip humanoid.

“gCSP and CTC++” has succesfully implemented different control algorithms for Mechatronic setups. The concept of gCSP and CTC++ is promising but it is not mature enough to implement the layered structure as shown in figure 2.1.

Orocos scores positive and promising on all requirements except "Resourse efficiency". A demonstrator must show if the cost of Resourse inefficiency effects the total system. Beside “gCSP and CTC++” is Orocos the only framework supporting Hard Real-Time behavior.

Orocos is the proposed software framework in this project. The demonstrator (see section 4.3) will be programmed using Orocos and used as testcase.

3 Positioning using WLAN infrastructure

3.1 Introduction

Part goal of this project is to find an absolute positioning system which use Commercial off-the-shelf hardware and uses the existing WLAN infrastructure. This with minimal training and knowledge of the environment. The position retrieved will be used for navigating a mobile robot. Using the WLAN infrastructure is a cheap solution for indoor absolute position estimation. Cheap because all hardware is already in use for communication purposes and because the WLAN infrastructure is already present so no dedicated hardware has to be developed.

The accuracy taken from related work, see table 3.1, is between 1.5 and 8 m. It can be used for navigating or tracking and orientation purposes. Orientation means in this case "Am I in the right room", "Which direction is the Mail room" etc.

3.2 Related work

Several studies were performed on positioning with WLAN, all with different goals and methods. Most had as goal to use the position information to benefit the user with automatic triggered services. For example print to the nearest printer, building navigation etc. Ekahau, Inc provides a commercial Wireless LAN position system called Ekahau Positioning Engine (Ekahau, 2008). The details of the system are patented and not publicly available.

The method of positioning differs between the different projects. An empirical method, based on a grid of known measurements, is used by Chen et al. (2007), de Moraes and Nunes (2006), Ladd et al. (2004) and Bahl and Padmanabhan (2000). With this method a grid with measurements must be made and through the nearest neighbor principle, based on resource intensive probability calculations, a position can be determined. Bahl and Padmanabhan (2000) and Kotanen et al. (2003) use a radio propagation model for determining a distance to an access-point. Through trilateration of these distances and know positions of the access-points, a position can be calculated. This method needs no training before implementing, only known position of the radio beacons. It is therefore more flexible than the empirical method. Downside is that it is less accurate (Bahl and Padmanabhan, 2000). Kwon et al. (2004) use a hybrid method of both a empirical grid and radio propagation. Results on accuracy and computation efficiency of these projects can be found in table 3.1.

	Method		Computation
Project	**Empirical**	**Radio propagation**	**efficiency**
Chen et al. (2007)	2.5[a]		-
de Moraes and Nunes (2006)	1.7[b]		-
Ladd et al. (2004)	1.5		-
Bahl and Padmanabhan (2000)	2.94[c]	8.16[c]	-/+
Kotanen et al. (2003)		1.6[b]	+
Kwon et al. (2004)	5 to 7		–

[a] with 0.9 error quantile
[b] mean error
[c] median 50^{th} percentile

TABLE 3.1 - Accuracy in related projects (reported in meters)

As can be seen from the results in table 3.1 the accuracy differs greatly. This can be explained by the different environments in which the projects took place, more walls and greater distances between the access-points.

In this project, radio propagation models will be used to determine a position. The accuracy of an empirical method is better but is also less resource efficient. Therefore, it is not suitable for an embedded platform used on a robot. Furthermore, with the Radio propagation method the implementation is more flexible and extendable to other floors and buildings because no training is needed. The development of positioning with WLAN is explained in section 3.3.

3.3 Developing the WLAN positioning module

The global architecture of the WLAN positioning module is divided in four parts. This is shown in figure 3.1. Each part is described, implemented and validated in the following sections.

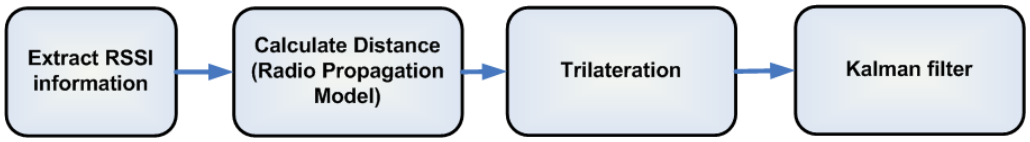


FIGURE 3.1 - Global architecture of the WLAN positioning module

Extracting RSSI information and *Calculating Distance using Radio Propagation Model* are validated using real measurements. *Trilateration* and *Kalman filtering* are verified with simulated positions and distances resembling the real situation. The final validation of the WLAN positioning module use real measurements. During this final validation two situation wil be evaluated; Moving from point A to point B and stationary on a fix point.

From this final validation, the WLAN positioning module is evaluated if it is suitable for use in robotics. This is discussed in section 3.8.

All the measurements (real and simulated) for verification, validation and evaluation where taken at the Control Engineerling group located at floor eight. Figure 3.2 shows the layout of floor eight and the positions of the used Access-Points.

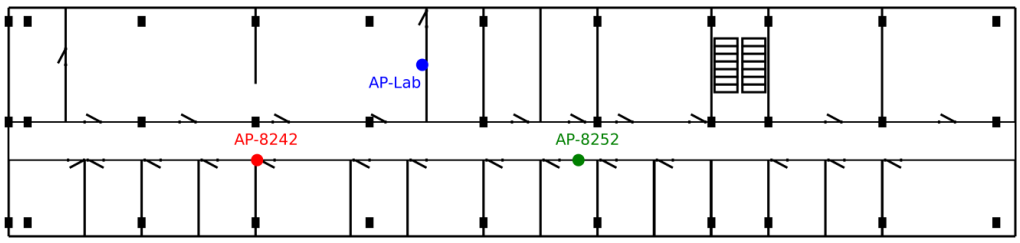


FIGURE 3.2 - Layout of floor eight

3.4 Received Signal Strength Indicator (RSSI)

3.4.1 Background

The Received Signal Strength Indicator (RSSI) is part of the Physical layer (PHY) in the Open Systems Interconnection (OSI) Model for IEEE 802.11 wireless LAN. The OSI Model (see table 3.2) is a layered, abstract description for communications and computer network protocol design. The RSSI is a parameter that has a value of 0 through *RSSI Max*. This parameter is a measurement by the PHY of the energy observed at the antenna during data transfer. RSSI is

intended to be used in a relative manner. Absolute accuracy of the RSSI reading is not specified (IEEE Std., 2003).

OSI Model			
	Data unit	**Layer**	**Function**
Host Layers	Data	7. Application	Network process to application
		6. Presentation	Data representation and encryption
		5. Session	Interhost communication
	Segment	4. Transport	End-to-end connections and reliability (TCP)
Media Layers	Packet/Datagram	3. Network	Path determination and logical addressing (IP)
	Frame	2. Data link	Physical addressing (MAC & LLC)
	Bit	1. Physical	Media, signal and binary transmission

TABLE 3.2 - OSI Model

The Linux Wireless Extension (WE) (Tourrilhes, 2007) is an Open Source project sponsored by Hewlett Packard since 1996 and build with the contribution of many Linux users all over the world. The Wireless Extension is a generic Application Programming Interface (API) allowing the WLAN driver to be exposed to user-space. It enableling on the fly configuration and specific statistics to Wireless LANs, like Signal Strength, noise, frequency, AP MAC addresses etc.

3.4.2 Implementation

Within the driver for Wireless LANs an option exists to perform an environment scan of finding Access-Poits (AP) within range. When this option is executed, the driver returns a list of founded APs and there specific Wireless statistics, like MAC address, Extended Service Set ID (ESSID), noise indicator and RSSI. This option has as purpose roaming profiling, eg. switching between access-point when moving. The WE API provides functions and data structures for execution of this option in the driver.

The DriverWOM (Driver Wireless Orientation Module) program is written to perform an environment scan of nearby APs. It saves the retrieved information in a readable format, currently supported are; Comma-Separated Values (CSV) and Extensible Markup Language (XML). The DriverWOM program uses the WE API functions and datastructures to retrieve Wireless specific statistics of these nearby APs. The RSSI and operating frequency are needed to calculate the distance to the AP. The MAC address of the AP is used to identify the AP for position purposes.

The DriverWOM program needs a configuration file and the WLAN device (eth0, wlan0 etc) to operate correctly. The configuration file is a XML file containing AP specific information, like ID, position, antennegain. An empty template in XML format is shown in listing 3.1. During the environment scan by the DriverWOM program all known information, Wireless specific and AP specific, is saved when an AP is found which is described in the Config file.

Listing 3.2 shows the command line options for the DriverWOM program. The DriverWOM has an output Channel option “Network”. This is not implemented altough a structure for implementation is ready.

```
<?xml version="1.0" ?>
<AP-config>
   <AP>                              <!-- Next AP  -->
      <ID> </ID>                     <!-- MAC Address of AP used as ID -->
      <Name> </Name>                 <!-- ESSID of AP -->
      <Desc> </Desc>                 <!-- Description -->
      <X> </X>                       <!-- x-position of AP in cm -->
      <Y> </Y>                       <!-- y-position of AP in cm -->
      <Z> </Z>                       <!-- z-position of AP in cm -->
      <AntenneGain> </AntenneGain>   <!-- Gain of antenna in dBi -->
      <Power> </Power>               <!-- Transmit power in mW -->
   </AP>
</AP-config>
```

LISTING 3.1 - DriverWOM config file in XML format

```
linux:\$ ./DriverWOM -h
Help
Use ./DriverWOM
        -c <config file>
        -w <WLAN device>
        -C Channel =  1: screen; 2: Network; 4: XML file; 8: CSV file
        -o <Channel output file>
        -p <UDP port number>
        -a <UDP Address>
        -l <loops>
```

LISTING 3.2 - DriverWOM program options

3.4.3 Verification extracting RSSI

For this verification the DriverWOM performs an environment scan and saves the retrieved information in a CSV,- or XML file. Listing B.1 shows the output in CSV format. Listing B.2 shows the same information in XML format. From the timestamps in these listing can be seen that each 0.1 second an environment scan is performed and the retrieved data saved. This results in 10 measurements/sec.

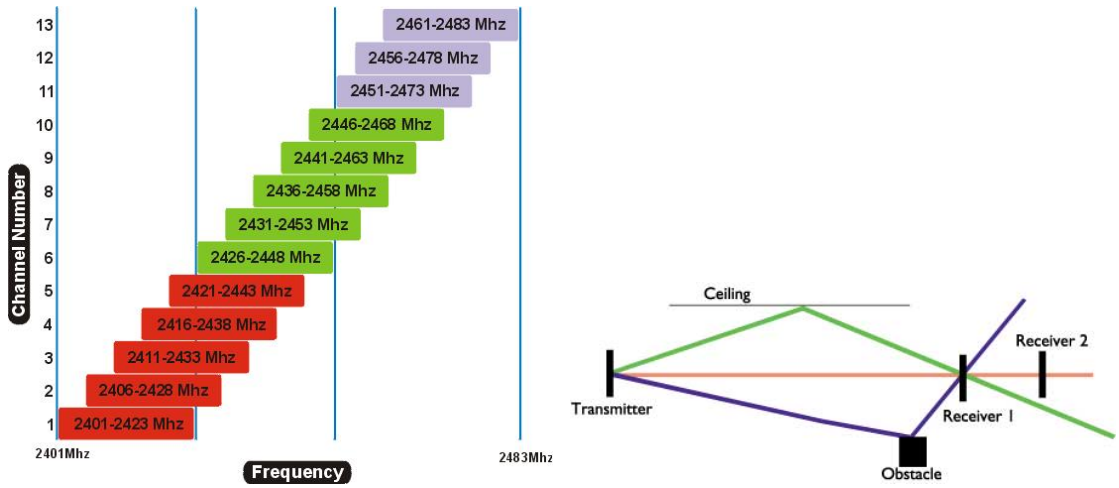
During this project the CSV output is used to evaluate WLAN positioning with use of Matlab.

3.5 Wave propagation models

3.5.1 Background

The IEEE 802.11b standard uses radio frequencies in the 2.4-GHz band. Accurate prediction of signal strength is a complex and difficult task, since the signal propagates by unpredictable means. The 2.4-GHz frequency band is subject to interference from microwave ovens, Bluetooth devices, 2.4-GHz cordless phones and overlap from IEEE 802.11 channel (figure 3.3(a)). Also signals of this frequency are absorbed by water, and consequently, people will also absorb signal since human bodies are almost 70% water. Furthermore the signal strength varies due to multipath fading (see figure 3.3(b)). This is caused by reflection, refraction and absorption of radio waves by masses standing in the wave area. Although many efforts have been made to model radio-signal distribution in an indoor environment a general model is not jet found. One of the main challenges for this project will be finding an approximation of radio-signal distribution which suits best for estimating a position.

In this project two radio wave propagation models will be evaluated. In general the radio wave propagation model can be described as equation 3.1.



(a) WLAN channels and overlap (b) Radio signal multipath fading

FIGURE 3.3 - Interference on Radio waves

$$P_{rx} - G_{rx} = P_{tx} + G_{tx} - L \tag{3.1}$$

Where:
P_{rx} = received power at receiver [dB].
G_{rx} = Gain of the receiver antenna[dBi].
P_{tx} = Power of transmitter[dB].
G_{tx} = Gain of transmitter antenna.
L = Power loss defined by Path loss model. L contains all the interference, losses and distance information.

The decibel (dB) is a logarithmic unit of measurement that expresses the magnitude of a physical quantity (usually power or intensity) relative to a specified or implied reference level. Since it expresses a ratio of two quantities and therefore it is a dimensionless unit.

- **dBm**: dB(1 mW) - power measurement relative to 1 milliwatt.
- **dBi**: dB(isotropic) - the forward gain of an antenna compared to the hypothetical isotropic antenna, which uniformly distributes energy in all directions.

3.5.2 Implementation

ITU indoor path loss model

The International Telecommunication Union (ITU) is an international organization established to standardize and regulate international radio and telecommunications. The ITU Indoor Propagation Model (ITU, 2005), also known as ITU Model for Indoor Attenuation, is a radio propagation model that estimates the path loss inside a room or a closed area inside a building delimited by walls of any form. Suitable for appliances designed for indoor use, this model approximates the total path loss an indoor link may experience.

The ITU indoor path loss model is expressed in equation 3.2.

$$L = 20\log(f) + N\log(d) + P_f(n) - 28 \tag{3.2}$$

where,
L = The total path loss. Unit: decibel [dB].
f = Frequency of transmission. Unit: megahertz [MHz].
d = Distance [m].
N = The distance power loss coefficient.

n = Number of floors/walls between the transmitter and receiver.
$P_f(n)$ = The floor/wall loss penetration factor.

Table 3.3 shows representative values for the power loss coefficients, N, as given by the ITU and table 3.4 gives values for the floor penetration loss factor,$P_f(n)$.

Frequency Band	Residential Area	Office Area	Commercial Area
900 MHz	N/A	33	20
1.2 GHz	N/A	32	22
1.3 GHz	N/A	32	22
1.8 GHz	28	30	22
4 GHz	N/A	28	22
5.2 GHz	N/A	31	N/A
60 GHz[a]	N/A	22	17

[a]60 GHz is assumed to be in the same room

TABLE 3.3 - Power Loss Coefficient Values, N. ITU (2005)

Frequency Band	Residential Area	Office Area	Commercial Area
900 MHz	N/A	$9(n=1)$	N/A
	N/A	$19(n=2)$	N/A
	N/A	$24(n=3)$	N/A
1.8 - 2 GHz	$4n$	$15+4(n-1)$	$6+3(n-1)$
5.2 GHz	N/A	$16(n=1$ only)	N/A

n is the number of floors penetrated ($n \geq 1$)

TABLE 3.4 - Floor loss penetration factor, $P_f(n)$. ITU (2005)

Log Distance Path Loss Model
The Log Distance Path Loss Model is an indoor radio propagation model that predicts the path loss a signal encounters inside a building over distance and is formally expressed in equation 3.3b. $PL(d_0)$ can be measured empirically at distance d_0 or calculated with the free space equation shown in equation 3.3a. The log-distance path loss model is a modified power law with a log-normal variability, similar to log-normal shadowing.

$$PL(d_0) = 20\log_{10}(f) + N \cdot \log_{10}(d_0) - 27.54 \tag{3.3a}$$

$$L_{total} = PL(d_0) + N \cdot 10\log_{10}\frac{d}{d_0} + X_g \tag{3.3b}$$

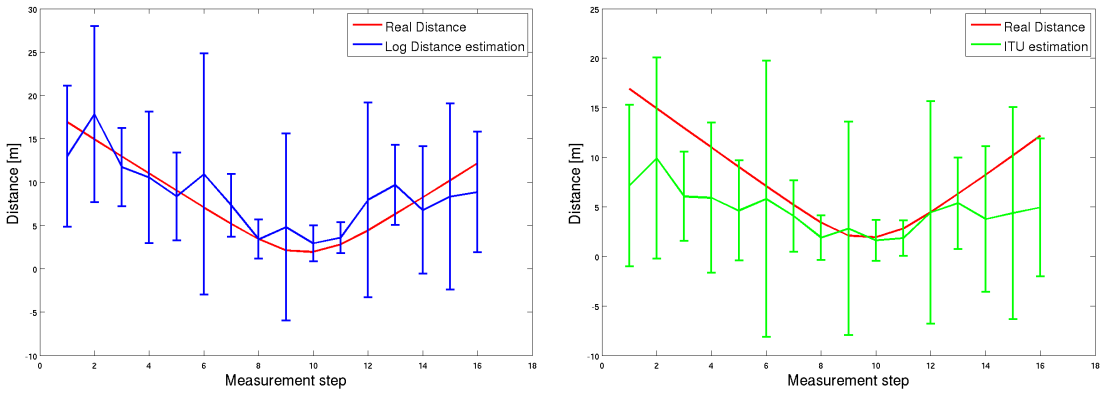
where,
$PL(d_0)$ = The free space loss at reference distance d_0. Unit: Decibel [dB]
f = Frequency of transmission. Unit: megahertz [MHz].
d_0 = The reference distance. Unit: Metre [m]
L_{total} = The total path loss inside a building. Unit:Decibel [dB]
d = The length of the path. Unit: Metre [m]
N = The path loss distance exponent.
X_g = A Gaussian random variable with zero mean and σ dB standard deviation.

3.5.3 Evaluation Path Loss models

Both algorithms, “ITU indoor path loss model” and “Log Distance Path Loss Model” are evaluated using Matlab. For this evaluation a grid with known positions and known distances to the

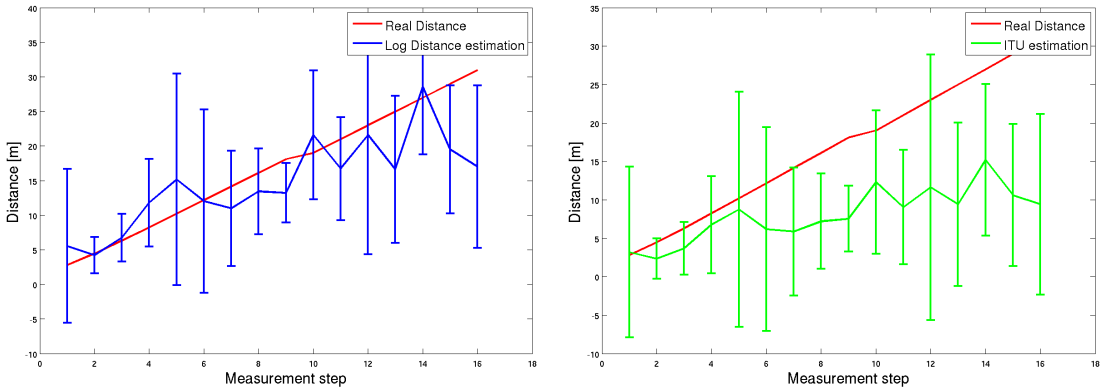
AP, at the hallway of the Control Group has been created. The DriverWOM program is used to scan the environment and retrieve RSSI information. The DriverWOM program run on laptop. At each point in the grid, 150 RSSI measurements where taken and saved in a CSV file witch could can be imported by a Matlab script.

In the hallway two Acces-Point are present; AP-8242 and AP-8252, see figure 3.2. The Matlab script read CSV file and calculate the disctance to the two APs using the "ITU indoor path loss model" and "Log Distance Path Loss Model". The results are compared with the real distances and shown in figures 3.4 and 3.5.



(a) Result using the Log Distance model (b) Results using the ITU model

FIGURE 3.4 - Distance estimation when passing by AP-8242



(a) Result using the Log Distance model (b) Results using the ITU model

FIGURE 3.5 - Distance estimation when moving from AP-8252

Figure 3.4 shows shows the results when passing by "AP-8242" and figure 3.5 shows shows the results moving from "AP-8252". In the figures red shows the real distance to the Access-Points. Blue the estimated distance using the "Log Distance Path Loss Model". Green represent the estimated distances using the "ITU indoor path loss model". The figures shows that the estimated distances follows the real distances but the error increases when the distance from the Access-Points increases. The standard deviation, shown in the error bar, is large and could give problems in estimation positions using trilateration.

The increasing error when the distance to the Access-Point increased can be partly explained by the fact that the models include wall attenuations and these measurements where taken in line-of-sight of the APs. The high standard deviation can be explained by the environment

in whitch this experiment was taken. The experiment was taken on a typical workday. This means lots of computers,laptop and people witch can/are interfering with the measurements. Speaking from experience; the standard deviation will decrease on a quit day. This has however less impact on the mean value.

3.6 Trilateration

3.6.1 Background

Trilateration is a method of determining the relative positions of objects using the geometry of triangles in a similar fashion as triangulation. Unlike triangulation, which uses angle measurements to calculate a location, trilateration uses distances to known locations of tree or more reference points. To accurately and uniquely determine a location on a 2D plane at least 3 reference points are needed. The basic problem is to find the intersection of 3 or more spheres with radius l_i. This can be formalize in finding a solution $[x\ y\ z]^T$ to the three or more nonlinear equations shown in equations 3.4.

$$\begin{cases} (x-x_1)^2+(y-y_1)^2+(z-z_1)^2 = l_1^2 \\ (x-x_2)^2+(y-y_2)^2+(z-z_2)^2 = l_2^2 \\ (x-x_i)^2+(y-y_i)^2+(z-z_i)^2 = l_i^2 \end{cases} \tag{3.4}$$

Where $P_i = (x_i, y_i, z_i), i = 1,2,3$ are coordinates of access-point i and l_i is the distance estimation associated with it, see figure 3.6(b).

3.6.2 Implementation

Linearization Method

From the problem of finding the intersection of several spheres, formalized in equation 3.4, a linear system, 3.5, can be constructed. The linearization process converts the problem into finding the point of intersection of several planes. This linearization process is in detail described in appendix C and results in 3.5 with A, $\overrightarrow{x}$ and $\overrightarrow{b}$ defined by 3.6.

$$A\overrightarrow{x} = \overrightarrow{b} \tag{3.5}$$

with,

$$A = \begin{pmatrix} (x_2-x_1) & (y_2-y_1) & (z_2-z_1) \\ (x_3-x_1) & (y_3-y_1) & (z_3-z_1) \\ \vdots & \vdots & \vdots \\ (x_n-x_1) & (y_n-y_1) & (z_n-z_1) \end{pmatrix}, \quad \overrightarrow{x} = \begin{pmatrix} x-x_1 \\ y-y_1 \\ z-z_1 \end{pmatrix}, \quad \overrightarrow{b} \begin{pmatrix} b_{21} \\ b_{31} \\ \vdots \\ b_{n1} \end{pmatrix} \tag{3.6}$$

where,
x_i, y_i, z_i = The coordinates of access-point i. $i = 1,2,\ldots,n$.
n = Number of Acces-Points found.

The linear system 3.5 has $(n-1)$ equations with three unknowns in xyz-plane or two unkowns in the xy-plane. Therefore, theoretically three reference points ($n = 3$) are needed to determine a position in a xy-plane. Position ($\overrightarrow{x}$) can derived with equation 3.7.

$$\overrightarrow{x} = A^{-1} * \overrightarrow{b} \tag{3.7}$$

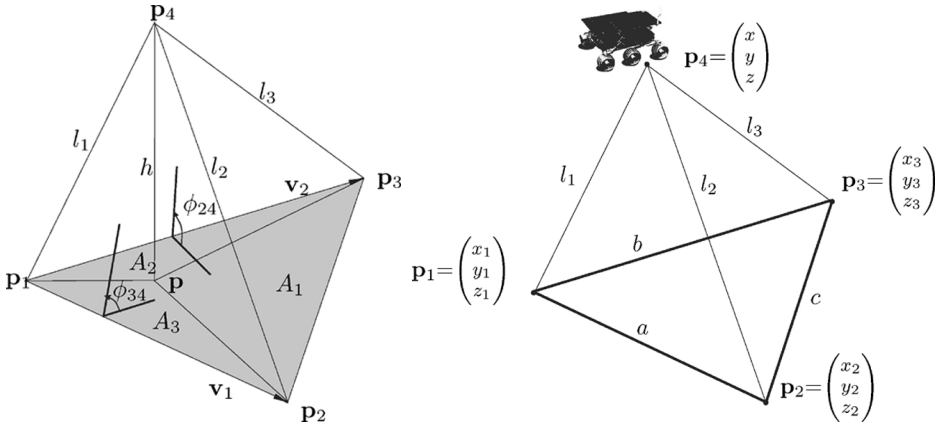
Coordinate-free method

A coordinate-free solution for trilateration is written by Thomas and Ros (2005). The propose method is an alternative closed-form formulation for trilateration based on constructive ge-

ometric arguments. By using barycentric coordinates (see figure 3.6(a)) a formula is derived (equation 3.8) containing a few number of Cayley – Menger determinants. Cayley – Menger determinants are used to characterize euclidean spaces in terms of distances between points. The formulation given by Thomas and Ros (2005) is coordinate-free because it only deals with inter-point distances. Hence, its numerical conditioning is independent from the chosen reference frames.

$$\begin{aligned} p_4 = & p_1 + \frac{1}{D(p_1,p_2,p_3)} \\ & \cdot (-D(p_1,p_2,p_3;p_1,p_3,p_4)\cdot v_1 \\ & + D(p_1,p_2,p_3;p_1,p_3,p_4)\cdot v_2 \\ & \pm \sqrt{D(p_1,p_2,p_3,p_4)}\cdot (v_1 \times v_2)) \end{aligned} \tag{3.8}$$

where:
$D(...)$ is a Cayley – Menger determinant.
p_i are point in the tetrahedron, see figure 3.6(b)



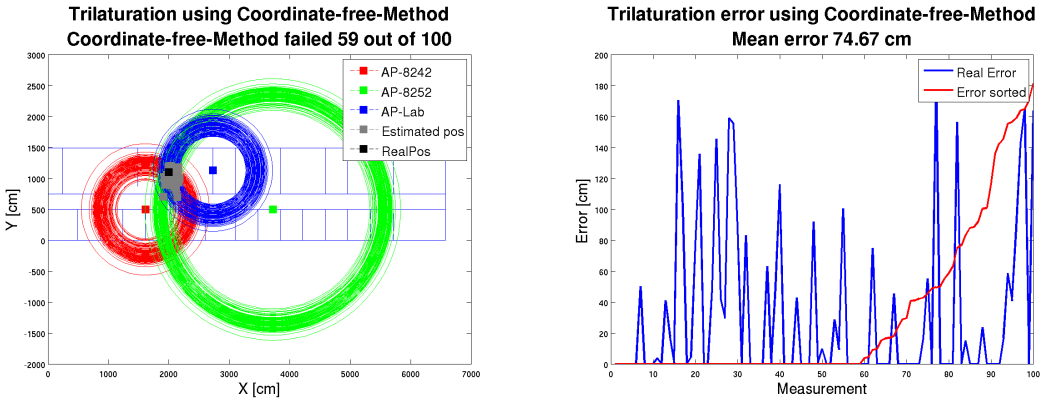
(a) Use of barycentric coordinates (b) Trilateration

FIGURE 3.6 - Coordinate-free solution for trilateration. source Thomas and Ros (2005)

3.6.3 Evaluation trilateration methods

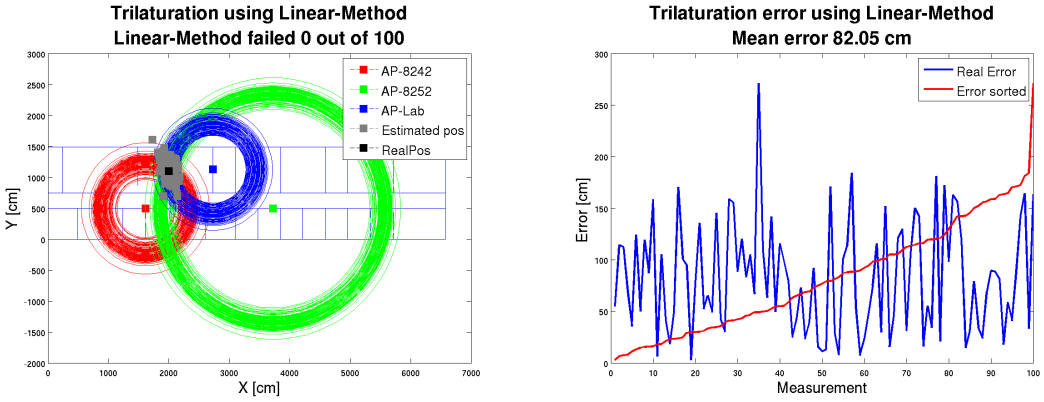
In section 3.5, two models for calculating distances from RSSI measurements where presented. The evaluation shows big variations in the calculated distance. These distances are needed in calculating a position using trilateration. The two trilateration models must be able to cope with these variations in distances. During this evaluation 100 random variations are applied on a simulated real distance. These noisy simulated distances are feeded to the trilateration models. The results can found in figures 3.7 and 3.8.

Figure 3.7(a) and 3.8(a) shows 3 circles. These circles resambles the distance associated with that Access-Point. The location of the Access-Points are the colored square in the center of the cicles. Positions estimated by trilateration are shown in grey. The black square is the real postion (in this case simulated).



(a) Dispersion Coordinate-free method (b) Mean error

FIGURE 3.7 - Evaluation Coordinate-free Method



(a) Dispersion Linearization method (b) Man error

FIGURE 3.8 - Evaluation linearization method

Figure 3.7(a) show the dispersion of the estimated positions using the Coordinate-free trilateration method. This method has an avarage error from the real position of 74 cm, see figure 3.7(b). Besides an avarage error of 74 cm the method was 59 times unable to calculate a position. A calculation failed when the lengths are not compatible with the size of the base.

Figure 3.8(a) show the dispersion of the calculated positions using the linearization method. This method has a avarage error from the real position of 82 cm and 100% successfully, see figure 3.8(b). The Coordinate-free method has a lower mean error but it is a false assumption to think it is beter then linearization method. The Coordinate-free method fails 59 times, each time it fails the error is zero. This has big impact on the mean value.

It can be concluded that the lineare trilateration method results in relaible positions because it never fails.

3.7 Kalman filtering

3.7.1 Background

Kalman filtering is a widely used filtering method to estimate states in a dynamic system from noisy measurements. It uses an internal system model to make a prediction of the next measurement where all the noise vectors are neglected since they are unknown. As soon as the new

measurement is available a correction is made on the estimated state vector based on the difference between the predicted and actual measurement. The weight of this correction, called the Kalman gain, is determined by the ratio between the covariance of the measurement and process noise.

The model used in Kalman filters is a state space description of the model added with noise processes. Equation 3.9 show the general form of a linear state space model with noise factors.

$$\begin{aligned} x_k &= Ax_{k-1} + Bu_{k-1} + w_{k-1} \\ y_k &= Cx_k + v_k \end{aligned} \tag{3.9}$$

where:
w_{k-1} is the system noise with normal probability distribution: $p(w) \sim N(0, Q)$
v_k is the measurement noise with normal probability distribution: $p(v) \sim N(0, R)$

The kalman filter algorithm is given in equation 3.10 and resembles a predictor – corrector algorithm. Equations 3.10a and 3.10b as predictor and 3.10c, 3.10d and 3.10e the corrector.

$$\hat{x}_k = Ax_{k-1} + Bu_{k-1} \tag{3.10a}$$

$$\hat{P}_k = AP_{k-1}A^T + Q \tag{3.10b}$$

$$K_k = \hat{P}_k C^T \left(C\hat{P}_k C^T + R\right)^{-1} \tag{3.10c}$$

$$x_k = \hat{x}_k + K_k \left(z_k - C\hat{x}_k\right) \tag{3.10d}$$

$$P_k = \left(I - K_k C\right)\hat{P}_k \tag{3.10e}$$

A kalman filter can be tuned by adjusting Q and R. When the measurement error covariance R in equation 3.10c approaches to zero, the gain K_k weights the $(z_k - C\hat{x}_k)$ more heavily. This means that the actual measurement z_k is "trusted" more and more, while the predicted measurement $C\hat{x}_k$ less and less. When $\hat{P}_k$ approach to zero the gain K_k weights the $(z_k - C\hat{x}_k)$ less heavily.

3.7.2 Implementing

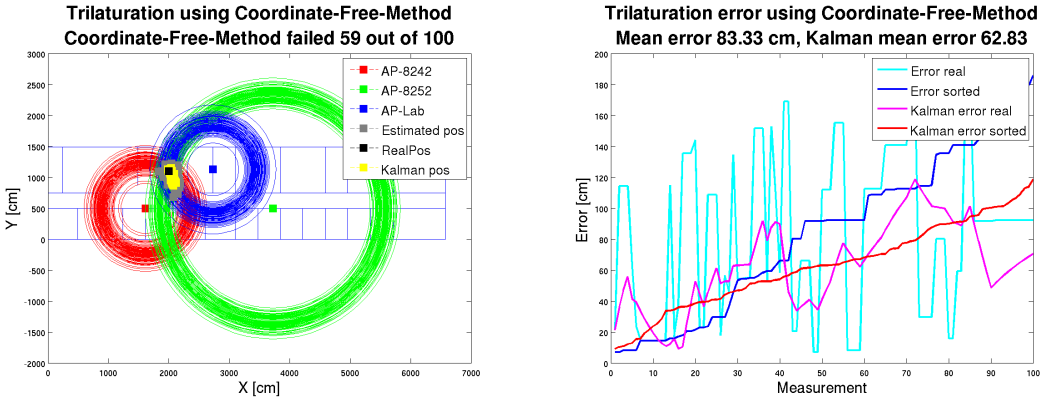
When the movement of the target is unknown it can be modeled as a Brownian motion. Equation 3.10a act as predicter for the Kalman filter. It wil predict unknown movement of the target using Brownian motion. The state space equation for Brownian motion is shown in equation 3.11.

$$\begin{pmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{pmatrix}_k = \begin{pmatrix} 1 & \triangle T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \triangle T \\ 0 & 0 & 0 & 1 \end{pmatrix}_{k-1} \cdot \begin{pmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{pmatrix}_{k-1} + \begin{pmatrix} 0 \\ w_{v_x} \\ 0 \\ w_{v_y} \end{pmatrix}_{k-1} \tag{3.11}$$

3.7.3 Evaluation kalman filter

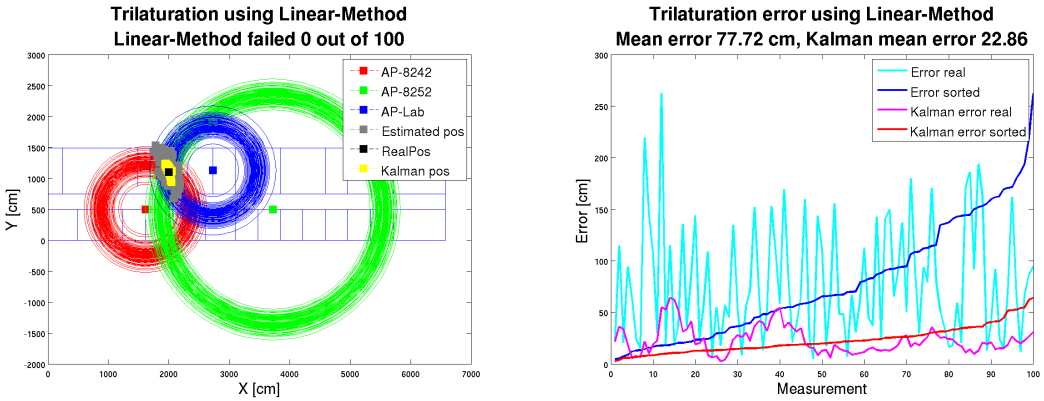
The Kalman filter is evaluated on the performance using the simulated noisy positions from trilateration evaluation. This evaluation is discribed in section 3.6.3 on page 19. The results of feeding these simulated positions to the Kalman filter are shown in figures 3.9 and 3.10.

Figure 3.9(a) and 3.10(a) shows 3 circles. These circles resambles the distance associated with that Access-Point. The location of the Access-Points are the colored square in the center of the cicles. Positions estimated by trilateration are shown in grey. Posistions filtered with the Kalman filter are shown in yellow. The black square is the real postion (in this case simulated).



(a) Dispersion with Coordinate-free method (b) Mean error of Coordinate-free method

FIGURE 3.9 - Evaluation Coordinate-free method and Kalman filtering



(a) Dispersion with linearization filtering (b) Man error

FIGURE 3.10 - Evaluation linearization method and Kalman filtering

It can be seen from both figures that the mean error is reduced when filtered by the kalman filter. The reduction in the mean error is biggest using the linearization method. This can be explained due to the fact that the dispersion on the linearization method positions resembles with White Noise and the dispersion in Coordinate-free method not. The kalman filter performs optimal when the noise is White Noise. The Coordinate-free method has no White Noise because it failed half the time and when the method failed it saves the last know location. If the method saves an error instead of the last know location the Kalman filter will fail totally.

3.8 Evaluating WLAN positioning

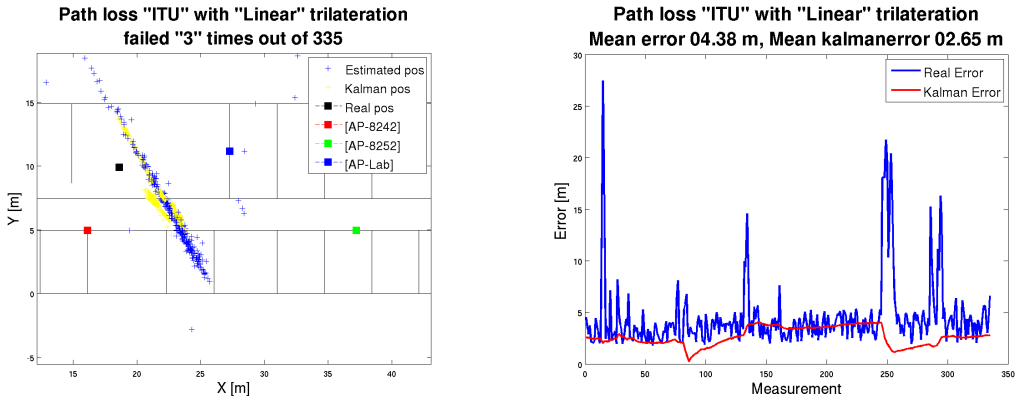
This section describes the evaluation of the WLAN position module. The global architecture for this module is shown in figure 3.1 are performed on a fixed point target and on a moving target, both with known real positions. Details about the results are shown in appendix D.

3.8.1 Fixed point target

During this experiment the DriverWOM program scans the environment while fixed on a de-

fined point. The results are shown in appendix section D.1.

Figures in appendix section D.1 and 3.11(a) shows the location of the Access-Points with the colored squares. Positions estimated by trilateration are shown with a blue + and the Kalman filtered with the yellow +. The black square is the real postion.



(a) Positions (b) Errors

FIGURE 3.11 - ITU path loss and linear trilateration

Estimating a position on a fixed point only get good results when using the ITU path loss model and the linear trilateration method, see figure 3.11. The other test, see figures D.2, D.3 and D.4, have a mean error too large to be of any use. The Coordinate-free trilateration method, see figures D.2(a) and D.4(a), fails to calculate a position in 99% of the time and with a mean error above the 13 meters (figures D.7(b) and D.9(b)) the position are also very unreliable.

3.8.2 Moving

During this experiment the DriverWOM program scans the environment while moving on a defined path acting as a moving target. The results are shown in appendix section D.2.

Following a moving target has bad results on all four tests, see pictures D.7, D.8, D.9 and D.10. With linear trilateration method the dispersion is to high that even a slow kalman filter cannot fix it, see figures D.8(a) and D.10(a). The mean error (figures D.8(b) and D.10(b)) of the kalman filtered positions is between the 6 and 8 meters is too high, even for orientation purposes. The Coordinate-free trilateration method, see figures D.7(a) and D.9(a), fails to calculate a position in 90% of the time and with a mean error between 7 and 8 meters (figures D.7(b) and D.9(b)) the position are also very unreliable.

3.8.3 Conclusion evaluation WLAN positioning

The conclusion of this evaluation is that positioning using the ITU path loss model and the linear trilateration can be used for position estimation. With a mean error under the 3 meters (see figure 3.11(b)) and little dispersion (see figure 3.11(a)) after kalman filtering, it is able to locate a position on room basis. This can be used for orientation and room specific information.

The conclusion of positioning when moving is that due to the dipersion of estimated postions Navigation and orientation are not possible.

4 Implementation

4.1 Introduction

The demonstrator must show the capabilities of the chosen framework and the findings of positioning using the WLAN infrastructure. Orocos is chosen to be best suitable as software framework for robotics, see section 2.4. In this chapter the findings of implementing and the evaluation of the implementation are discussed. For the implementation the JIWY (section 4.3 and figure 4.2(a)) is chosen as test platform. Information of terminology used in chapter can be found in appendix F.

Orocos will be used in mechatronic setups like robots, interacting and controlling the mechanical setup. The performance of low level cq. loop control is therefore an import aspect. The performance of low level cq. loop control relies heavily on timing constraints due to discreet nature of using computers. The Orocos framework will be compared on these timing constraints with the existing method, gCSP and CT, used at the Control Lab. This is explained in section 4.4 and 4.5. The results are discussed in section 4.6.

4.2 Positioning using WLAN

With the result from positioning of a moving target, discussed in section 3.8, it is not feasible to track a moving target. This excludes demonstrating tracking a moving target using the WLAN positioning as proposed in section 1.1.

The results of positioning on a fix position are much better and can be used for orientation purposes, like “In which room am I”, “Where is the nearest recharge station” etc.... The positioning becomes part of world-modeling as can be seen in figure 4.1. This figure shows the top level (supervisory control) in the advised structure (figure 2.1) for the TUlip robot

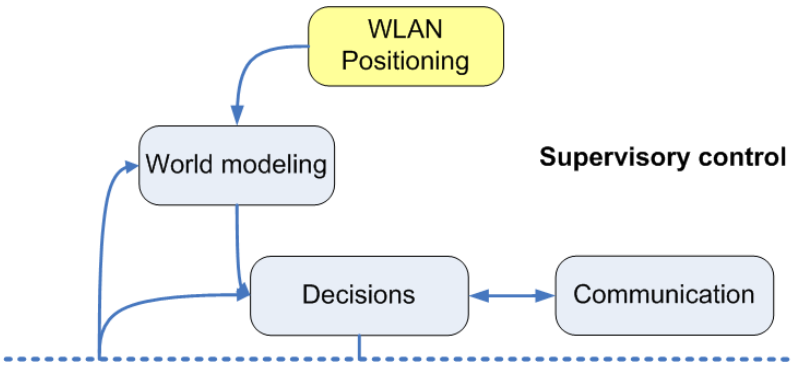
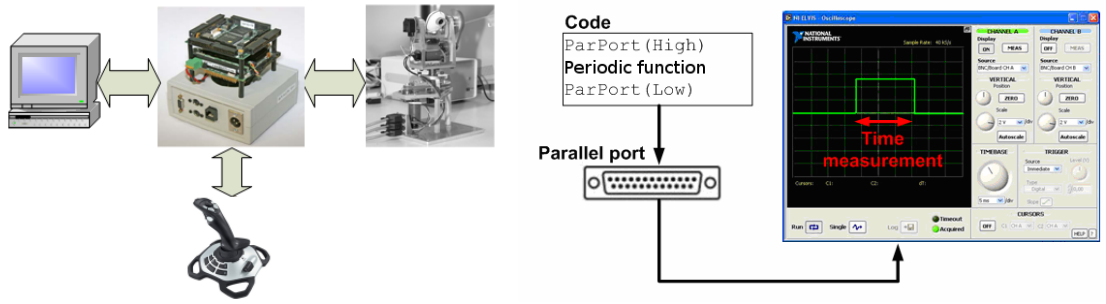


FIGURE 4.1 - Position WLAN positioning in framework

4.3 Test platform

JIWY is a mechatronic setup for holding a camera. The construction contains two joints that allow the camera to rotate on a horizontal axis and a vertical axis. The maximum swing is limited by mechanical end stops. The maximum angles between the end stops are respectively 300^o and 150^o. These end stops prevents full swings so that the wires cannot be twisted or damaged. Each joint is equipped with one DC motor and one incremental encoder. The JIWY is connected to a PC-104 embedded PC with an anything I/O card, see figure 4.2(a). On this stack Linux with RTAI 3.5 is installed.



(a) Test setup with a PC/104 and JIWY (b) Timing measurement method

FIGURE 4.2 - Test setup and timing measurement method

Timing measurement setup

Timing performance will be measured using an oscilloscope and the parallel port. Figure 4.2(b) shows how to measure the operation time. Writing a value to the parallel port add a fixed latency of $2\mu s$. This implies that the measured time on the oscilloscope includes $2\mu s$ switching time of the parallel port. With the option "persist on" on the oscilloscope the variation in start time can be shown and measured.

Timing performance will be measured in two situations; *idle load* and *heavy load*. In idle load, a minimum of background processes are active, all the systemload is from its own execution. Under heavy load, resource intensive processes are running next to the process own execution. The hard real-time process must operate without performance loss due to these resource intensive processes.

4.4 JIWY with Orocos

This section will explain how the software for controlling the test setup is build using the Orocos framework. Extra information about the basics using Orocos can be found in appendix E.

The software for the JIWY setup, shown in figure 4.3, contains two Orocos components. The "Joystick" component generates setpoints for the "JIWY Controller" component which calculate the control signals. Both components are build using the "TaskContext" primitive (figure E.3) and connected with each other through the ports Pan and Tilt. The "JIWY controller" is connected with the hardware through the abstract layer for device Interfaces. This layer abstracts away hardware specific code which implies that a Orocos component has the same set of functions for different hardware. Appendix A contains a detailed structure of the software for the JIWY setup.

The source of "Joystick" contains code for reading from a native Linux device driver. Interfacing with native Linux drivers cannot take place in RTAIs hard Real-Time environment. Therefore, this component is executed as a Periodic task in Soft Real-Time. The periodic task reads each period from the Linux HID (Human Interface Device) driver. Within Orocos this Periodic task is given the lowest priority. Time variation in this task has minimal effect on the total system because the "Joystick" generates setpoints and is therefore not part of the Loop-Control, which run in Hard Real-Time. Drawback from Soft Real-Time and Hard Real-Time constructions is the possibly of starvation of the Soft Real-Time task. This mean that the task is not executed because processes with higher priorities consumes all the CPU recourses.

The "JIWY Controller" calculates the control signals for the JIWY setup. Timing is an important aspect for a stable system. Therefore this "JIWY Controller" is executed in Hard Real-Time and

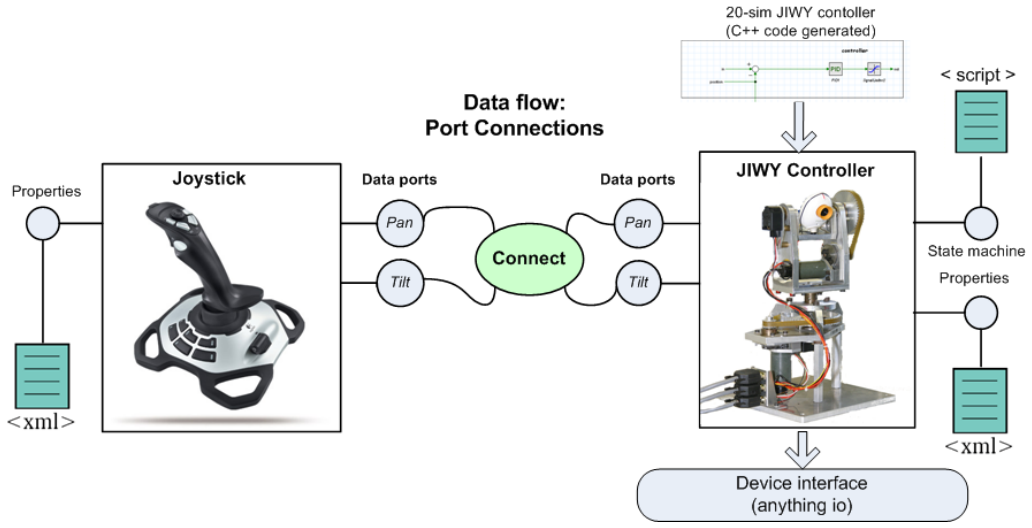


FIGURE 4.3 - Structure JIWY setup in Orocos

given Orocos' highest priority. By using Orocos' lock-free port interface, hard real-time components can exchange data with non real-time components and still maintain hard real-time.

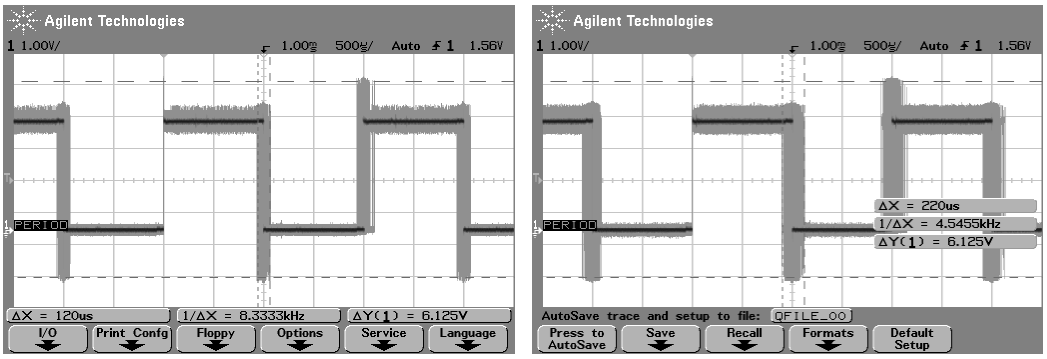
The main code for the JIWY setup is shown in listing 4.1. Here the Components are created and connected with each other, furthermore the behavior of the components are determined. Periodic activities need a priority level and refresh time (in seconds). The RTT activity interface activates each period the periodic function of the Orocos component. A template of a Orocos component is shown listing E.1.

```
/* Create the components */
Jiwy jiwyController("JIWY_controller");
usbHid joystick("Joystick");

/* Connect ports and peers */
jiwyController.addPeer(&joystick);
connectPorts(&joystick, &jiwyController);

/* Add component engine to Activity interface */
PeriodicActivity periodicActivityA(OS::HighestPriority, 0.001, jiwyController.engine());
PeriodicActivity PeriodicActivityB(OS::LowestPriority, joystick.engine());

/* Confige the component */
joystick.configure();
jiwyController.configure();

/* Start the component */
joystick.start();
jiwyController.start();
```

LISTING 4.1 - Main Orocos source code

Timing variance in the start time of a periodic task can be measured using the timing measurement setup shown in figure 4.2(b). Each time the periodic task starts a bit on the parallel port will toggle. Figure 4.4 show the result of this timing test under idle (figure 4.4(a)) and heavy load (figure 4.4(b)).

In figure 4.4 each rising and failing flank resembles a start of the periodic task. The update frequency during this test is 1KHz and should results in pulses of $1ms$. Variations in start time



(a) Idle load (b) Heavy load

FIGURE 4.4 - Timing test of the Orocos framework

are shown in gray around the falling flank. Because the oscilloscope triggers on a rising edge no variation are seen on start of the pulse. Variation in start time are shown in variations of the start time of the next update. If this task is late and the next update is on time, the pulswidth is $< 1ms$. If this update is on time en the next update is late the pulswidth is $> 1ms$.

Figure 4.4(a) shows the timing variations in a system with idle load. The variations are measured using measurement tools on the oscilloscope and has value of $120\mu s$. The timing variations in a system with heavy load (see figure 4.4(b)) are measured $220\mu s$.

4.5 JIWY with gCSP and CTC++

gCSP (Jovanovic et al., 2004) is a graphical tool for creating and editing CSP diagrams (Hilderink, 2005). CSP diagrams are dataflow diagrams, connecting processes with channels. Beside the dataflow, the concurrency structure is also indicated. Next to editing diagrams, gCSP does basic consistency checks and can generate code from the diagrams. The generated code needs the CTC++ library for implementing CSP primitives in modern programming languages and for general purpose microprocessors.

The gCSP-model for the JIWY setup is shown in figure 4.5. This model is used during the course Real-time software development [i]. The setup is controlled by three processes. Pan and Tilt calculates the control signal for the mechanical setup using the control models from 20-sim. Process SanityCheck is a safety layer and checks if the control signals are within defined range to prevent damaging the mechanical setup. A gCSP model is connected with the hardware through linkdrivers. Linkdrivers are objects within the CT library that communicate with hardware using the channel interface of CT. Linkdrivers abstract away hardware-dependent instructions. In figure 4.5 linkdrivers are represented by the blue dots with a “?” or “!”.

All three processes operate with an 1KHz frequency. To measure and evaluate the performance of gCSP and CTC++, the same timing test as with Orocos are performed. One under an idle system load and one under heavy system load. The results are shown in figure 4.6.

[i] info see: `http://www.ce.utwente.nl/bnk/`

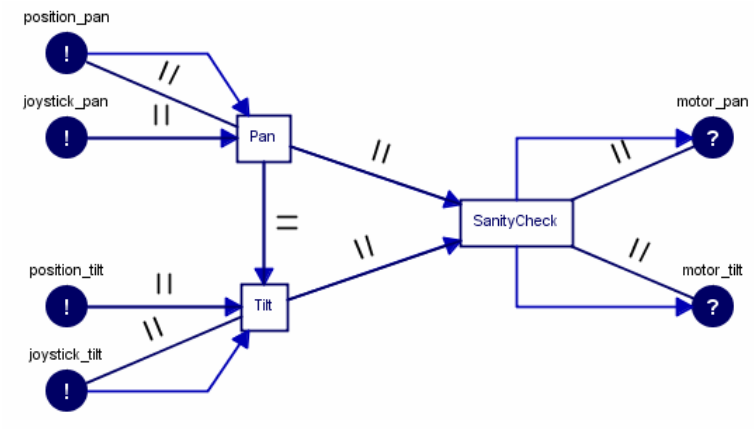
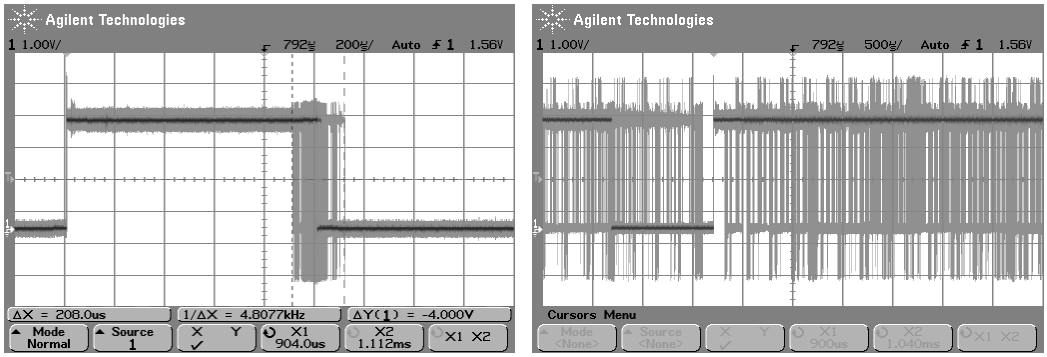


FIGURE 4.5 - Structure JIWY setup in gCSP



(a) Idle load (b) Heavy load

FIGURE 4.6 - Timing test of gCSP and CT

Figure 4.6(a) shows the timing variations in a system with idle load. The variations are measured using measurement tools on the oscilloscope and has value of 208μs. The timing variations in a system with heavy load (see figure 4.6(b)) are of the chart. No measurement where taken because no defined variation can be found. These results cause an instable system.

4.6 Evaluation Orocos versus gCSP and CTC++

Orocos and gCSP-CTC++ both target the development of building software for mechanical setups. Both are able to connect multiple processes in a timed critical environment through an abstract device interface with the hardware. But the implementation differs greatly. gCSP uses CSP principles for parallel operating processes. Inter communication is done with channels. The CTC++ library is used for implementing these CSP principles on a general purpose processor. Orocos uses a port based component structure on top of the RTT middleware. Data exchange is done through (free-lock) ports and inter communication done by commands, method and events.

Orocos is lager, more configurable and flexible than gCSP-CTC++. Furthermore Orocos has three extensive libraries on top of the RTT middleware.

To evaluate the timing constrains force upon a real-time system both “gCSP and CTC++” and Orocos are tested on timing variation of a periodic task. Both “gCSP and CTC++” and Orocos are stable, with large timing variations, in a system under idle load. In a system with heavy load “gCSP and CTC++” becomes unstable and misses its timing demand big time. Orocos stays

stable, although the timing variance increases.

These big difference on a system with heavy load can be explained by how the process are created and placed in RTAI's hard real time environment. The Orocos Activity interface create for each component a RTAI thread. These threads are placed in the Hard real-time environment of RTAI. RTAI will schedule these thread according its priority. The "JIWY controller" has the highest priority and therefore will be served first. The "Joystick" has the lowest priority and because of the Linux driver cannot be part of the Real-time environment. Through the lock-free port the "JIWY controller" is able to communicate with the "Joystick" and stay in hard real-time. This results in minimal interference from the heavy load.

All the process in "gCSP and CTC++" are placed in one Hard-real time RTAI thread. The different processes are scheduled with its own scheduler according the Occam principles. Because the current CTC++ implementation of the "Joystick" is incorrect, the whole "gCSP and CTC++" thread cannot be part of the Hard-Real time environment. Because the thread is not hard real-time anymore the heavy load will interfere with the Controller. This results in the unstable system shown in figure 4.6(b).

5 Conclusion and recommendations

5.1 conclusion

Software framework

The evaluation (section 2.4) of the literature study on software frameworks points out Orocos as best suitable for use in robotics. Orocos' is the ony framework with the ability for executing in hard-real time. A timing test concludes that Orocos' real-time behavior is better than the current used method of gCSP and CTC++.

The implementation on the JIWY setup shows that Orocos real-time process are able to communicate with Non real-time process without performance loss. Also real-time processes stays stable when the system load increases. This makes Orocos suitable for use in robotics. Non real-time tasks can be used for robot behavior and real-time tasks for controlling algorithms of the robot.

Drawback of using Orocos is that is not a lightweight framework and consumes more resources then code that is being built from scratch.

Positioning using WLAN infrastructure

The development for positioning using the WLAN infrastructure (chapter 3) results in an evaluation of four methods in two situations. The situations are "moving from point to point" and "stationary on a fixed point". It can be concluded that positioning stationary on a fixed point is possible within a range $< 3m$ only if enough measurements are taken. Positioning when moving from point to point results in a position dispersion too large to be of any use. This was not expected after reading the results of similar projects.

The high position dispersion comes from the environment and from unreliable measurements using Commercial off-the-shelf hardware. The RSSI value in this hardware is implemented as an indicator and not a real measurement for use in calculations. The environment of walls (scattering and absorption), people(absorption), laptops and other APs (radio overlap) has a great interference on the Received Signal strength. Furthermore path loss models are developed to calculate signal coverage of an area by the radio network. In this case the distance is a parameter and the Signal strength the goal. In this project the signal strength is the parameter and the distance the goal. If the models are linear is was not problem but they are not.

5.2 recommendations

Use dedicated hardware for signal strength measurements. During this project Commercial off-the-shelf hardware is used in measuring Signal Strength. The purpose of the Commercial off-the-shelf hardware is not measuring signal-strenght but communication. The signal strength is only used as indicator during roaming. When dedicated hardware is used for measuring the signal strength the measurements are more reliable and position estimations should results in less dispersion.

investigate the use of empirical method for position estimations. Empirical methods are used in commercial WLAN position systems. This method relies heavily on probability methods and are calculation intensive. Furthermore, a grid of measured signal strengths on known location must be made before the empirical methods can be used. This method is not investigated

in this project because the resources needed for calculation are not suitable in a embedded system.

Implement Orocos on setup with more complexity than JIWY. Orocos is successfully implemented on the JIWY setup but the JIWY setup is a simple mechatronic setup. It only touches the capabilities of Orocos. To explore the real power of this framework, a more complex system is needed; for example the TUlip robot or the Production Cell. Orocos has a powerful state-machine engine which can start, stop and reload on run-time and statemachines are very useful in describing behaviors patterns. Furthermore the Kinematics and Dynamics Library (KDL) and the Bayesian Filtering Library (BFL) are not investigated but looks very promising.

Develop a template for 20-sim in the Orocos component format. During this project the control algorithm consist of generated code from 20-sim. The default C++ template is used and the generated code is incorporated by hand in the component structure of Orocos. A nicer way is to create a template for 20-sim which is able to generate code from 20-sim in the Orocos component format. The defined structure of an Orocos component makes auto generated code possible. The 20-sim C++ template uses array for input and output. Orocos components use port with individual names rather than TV. Some investigation is needed if 20-sim is able to generate the code.

A Detailed Orocos structure of JIWY

Figure A shows a detailed structure for the JIWY setup using Orocos.

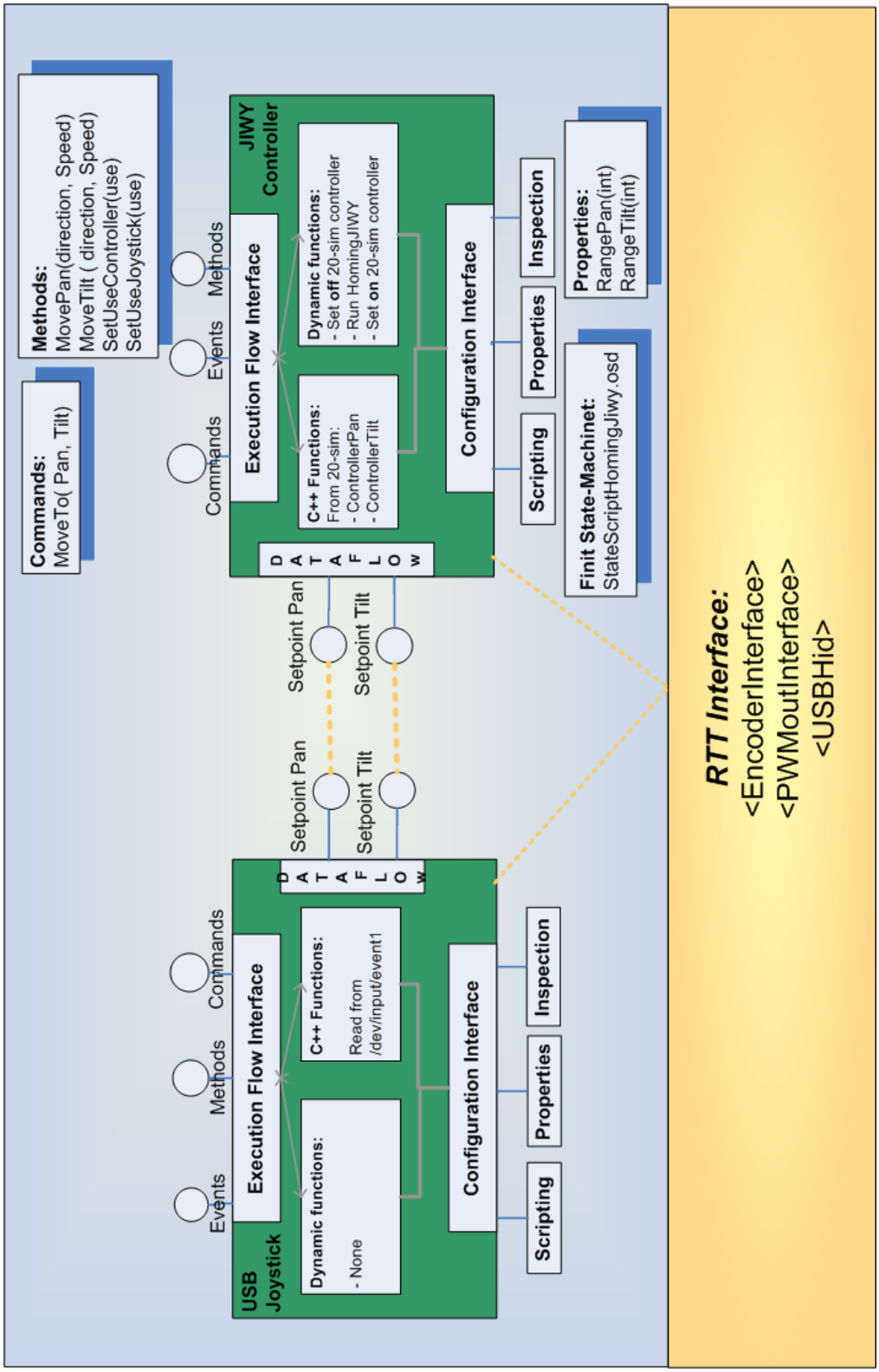


FIGURE A.1 - Detailed Orocos structure JIWY setup

B Export DriverWOM

The DriverWOM is able to export the gathered information in CSV format and XML format. Listing B.1 shows the CVS output. Listing B.2 shows the XML output.

```
# CVS output of AP scan created at: Fri May 16 14:50:22 2008 Comment: "Moving from 8252 to 8242"
# 00:07:0E:15:A4:90 [AP-8242] 00:07:0E:15:A0:00 [AP-8252] 00:17:31:D6:94:F5 [AP-Lab]
TimeStamp, x_0,y_0,z_0, GainTx_0, PowerTx_0, Freq_0, RSSI_0, GainRx_0,
x_1,y_1,z_1, GainTx_1, PowerTx_1, Freq_1, RSSI_1, GainRx_1,
x_2,y_2,z_2, GainTx_2, PowerTx_2, Freq_2, RSSI_2, GainRx_2 #endLine
1210228150.536509, 1612, 992, 200, 2, 20, 2437000000, -54, 2,
3720, 992, 200, 2, 20, 2462000000, -27, 2,
2728, 372, 120, 2, 20, 2412000000, -52, 2 #endLine
1210228150.636637, 1612, 992, 200, 2, 20, 2437000000, -52, 2,
3720, 992, 200, 2, 20, 2462000000, -27, 2,
2728, 372, 120, 2, 20, 2412000000, -52, 2 #endLine
.......
```

LISTING B.1 - DriverWOM CSV export file

```
<?xml version="1.0" ?>
<ScanEvent>
        <!-- Options of this ScanEvent -->
        <ScanOptions>
                <Time>Fri May 16 14:50:22 2008</Time>
                <FileName>export.xml</FileName>
        </ScanOptions>
        <!-- Scan measurements from here -->
        <Measurement timestamp="1210228150.536509">
                <AP ID="00:07:0E:15:A4:90">
                        <RSSI>-54</RSSI>
                        <Freq>2437000000</Freq>
                </AP>
        </Measurement>
        <Measurement timestamp="1210228150.536509">
                <AP ID="00:07:0E:15:A0:00">
                        <RSSI>-27</RSSI>
                        <Freq>2462000000</Freq>
                </AP>
        </Measurement>
        <Measurement timestamp="1210228150.536509">
                <AP ID="00:17:31:D6:94:F5">
                        <RSSI>-52</RSSI>
                        <Freq>2412000000</Freq>
                </AP>
        </Measurement>
        <Measurement timestamp="1210228150.536509">
                <AP ID="00:07:0E:15:A4:90">
                        <RSSI>-54</RSSI>
                        <Freq>2437000000</Freq>
                </AP>
        </Measurement>
        <!-- Next measurement -->
        ........
</ScanEvent>
```

LISTING B.2 - DriverWOM XML export file

C Linearization process

The intersection at $[x\ y\ z]^T$ of i spheres is described by i equations shown in C.1.

$$(x-x_i)^2+(y-y_i)^2+(z-z_i)^2=l_i^2 \tag{C.1}$$

Where $P_i = [x_i\ y_i\ z_i]^T$, $i = 1, 2, 3$ are coordinates of accespoint i and l_i is the distance estimation associated with it, see figure 3.6(b).

The j^{th} constraint is used as a linearizing tool. Adding and subtracting x_j, y_j and z_j in C.2 result in C.2.

$$\begin{gathered}(x-x_j+x_j-x_i)^2+(y-y_j+y_j-y_i)^2+(z-z_j+z_j-z_i)^2=l_i^2\\ \text{with}\big(i=1,2,..,j-1,;\quad j+1,...,n\big)\end{gathered} \tag{C.2}$$

Expanding and regrouping the term in known and unknown variables leads to C.3.

$$\begin{gathered}(x-x_j)(x_i-x_j)+(y-y_j)(y_i-y_j)+(z-z_j)(z_i-z_j)\\ =\frac{1}{2}\left[(x-x_j)^2+(y-y_j)^2+(z-z_j)^2-r_i^2+(x_i-x_j)^2+(y_i-y_j)^2+(z_i-z_j)^2\right]\\ =\frac{1}{2}\left[l_j^2-l_i^2+d_{ij}^2\right]=b_{ij}\end{gathered} \tag{C.3}$$

Where,

$$d_{ij}=\sqrt{(x_i-x_j)^2+(y_i-y_j)^2+(z_i-z_j)^2} \tag{C.4}$$

is the distance between AP's P_i and P_j.

The choice of the linearization tool is arbitrarily, so for easy programming the first ($j = 1$) constrain is choossen. This is analogous to selecting the first accespoint. This result in C.5:

$$\begin{aligned}(x-x_1)(x_2-x_1)+(y-y_1)(y_2-y_1)+(z-z_1)(z_2-z_1)&=\frac{1}{2}\left[l_1^2-l_2^2+d_{21}^2\right]=b_{21}\\ (x-x_1)(x_3-x_1)+(y-y_1)(y_3-y_1)+(z-z_1)(z_3-z_1)&=\frac{1}{2}\left[l_1^2-l_3^2+d_{31}^2\right]=b_{31}\\ (x-x_1)(x_n-x_1)+(y-y_1)(y_n-y_1)+(z-z_1)(z_n-z_1)&=\frac{1}{2}\left[l_1^2-l_n^2+d_{n1}^2\right]=b_{n1}\end{aligned} \tag{C.5}$$

C.5 can easily written in matrix form C.6 with A, $\vec{x}$ and $\vec{b}$ defined by C.7.

$$A\,\vec{x}\ =\ \vec{b} \tag{C.6}$$

with,

$$A=\begin{pmatrix}(x_2-x_1) & (y_2-y_1) & (z_2-z_1)\\ (x_3-x_1) & (y_3-y_1) & (z_3-z_1)\\ \vdots & \vdots & \vdots\\ (x_n-x_1) & (y_n-y_1) & (z_n-z_1)\end{pmatrix},\quad \vec{x}=\begin{pmatrix}x-x_1\\ y-y_1\\ z-z_1\end{pmatrix},\quad \vec{b}\begin{pmatrix}b_{21}\\ b_{31}\\ \vdots\\ b_{n1}\end{pmatrix} \tag{C.7}$$

D Evaluation positioning using WLAN

The execution flow shown in figure 3.1 are performed on a fixed point target and on a moving target, both with known real positions. The evaluation took place on floor 8 at the Control Engineering group. A layout with the positions of the Access-Points of this floor is shown in figure 3.2. Before evaluating, the DriverWOM program scans the environment for known Access-Points and saves the result in a CSV file. This file can be imported in matlab and the measurement are used during this evaluation. Section D.1 shows the result of the measurement taken on fixed point. Section D.2 shows the result of a moving target.

The figures shown in this appendix are rather small and can be best viewed in the digital version. The digital version can be download from the Control Engineering website: `http://www.ce.utwente.nl`.

D.1 Fixed target evaluation

First the DriverWOM program scan for Access-Point while on a fixed point and save the results in a CSV file. Importing the file in matlab results in Matlab output:

```
Read Data from file
    'Lab at Workplace by door (8242)'
Found the following AccessPoints:
    [AP-8242] 00:07:0E:15:A4:90
    [AP-8252] 00:07:0E:15:A0:00
    [AP-Lab] 00:17:31:D6:94:F5
```

After the importing the dB measurement from the file, distances, using both methods, to the access-point are calculated. Figure D.1 show these distances.

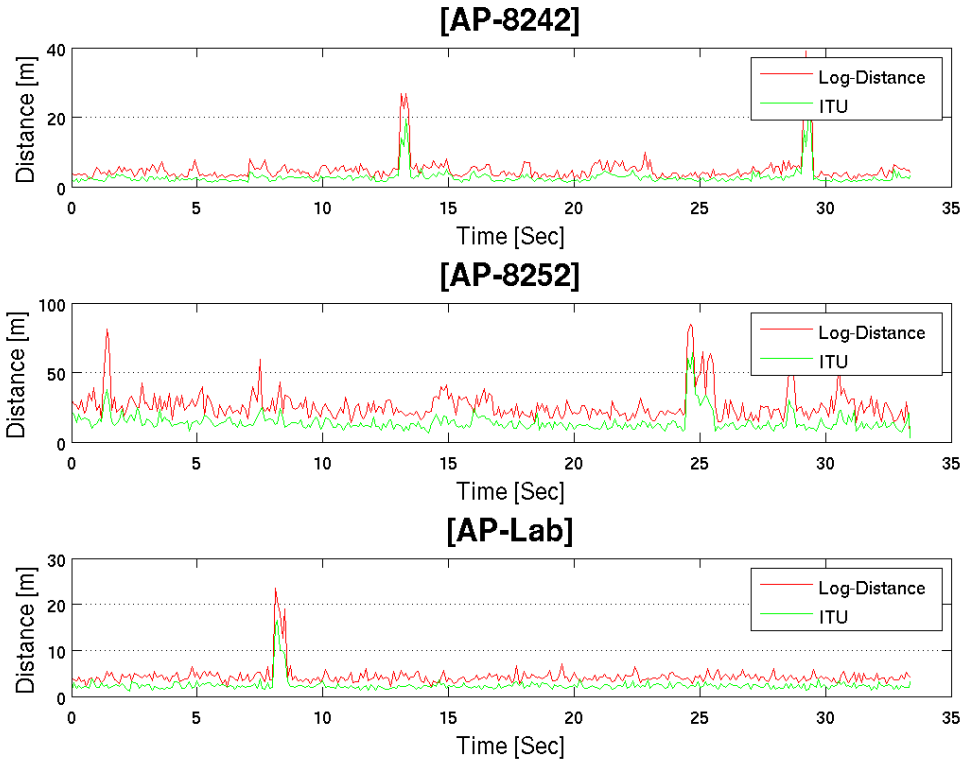


FIGURE D.1 - Calculated distances over time of both Path loss methods

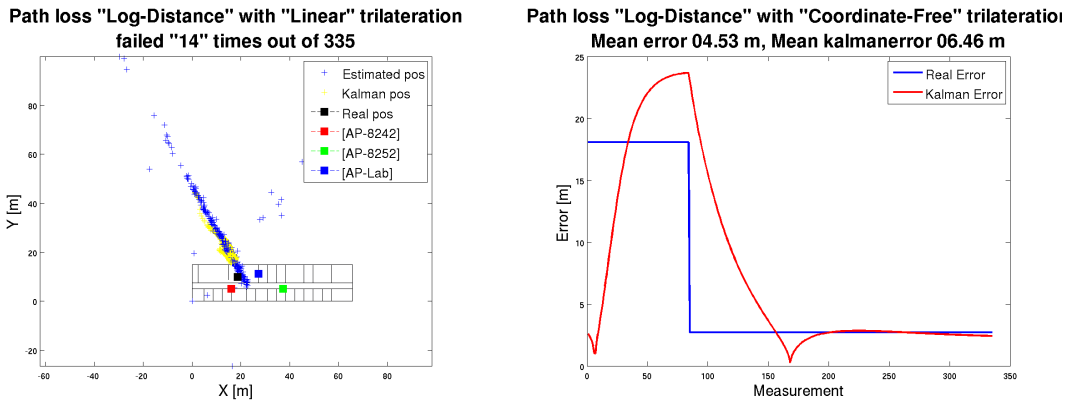
The distances to and the positions of the access-point are fed to the two trilateration methods. The calculations results in Matlab output:

```
Position estimation using trilateration

Positioning using distance estimation method "Log-Distance"
Method Coordinate-Free results in:
    334 where out of reach
    1 point where calculated out of a data set of 335
Method Linear results in:
    14 where out of reach
    321 point where calculated out of a data set of 335

Positioning using distance estimation method "ITU"
Method Coordinate-Free results in:
    334 where out of reach
    1 point where calculated out of a data set of 335
Method Linear results in:
    3 where out of reach
    332 point where calculated out of a data set of 335
```

remarkable are the bad results from the “Coordinate-free Method”. Clearly the noise on the distances was too big too get proper results. The Linear approach worked well.

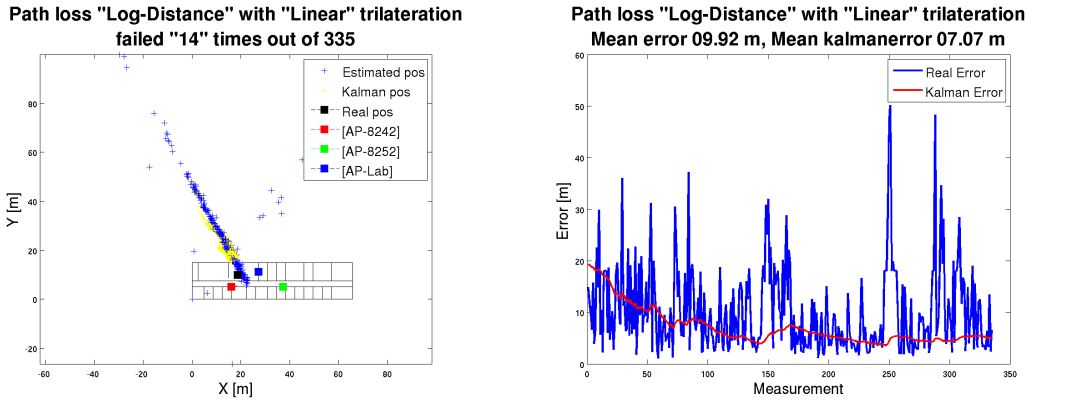
The positions are now filtered using a Kalman filter and the error to the real position is calculated. The results are plotted in the figures below. The Matlab output is:

```
Plot the position estimations

Positioning using distance estimation method "Log-Distance"
    Coordinate-Free; had 84 null positions
    Coordinate-Free; had 0 null positions using Kalman
    Linear; had 14 null positions
    Linear; had 0 null positions using Kalman
Positioning using distance estimation method "ITU"
    Coordinate-Free; had 131 null positions
    Coordinate-Free; had 0 null positions using Kalman
    Linear; had 3 null positions
    Linear; had 0 null positions using Kalman
```



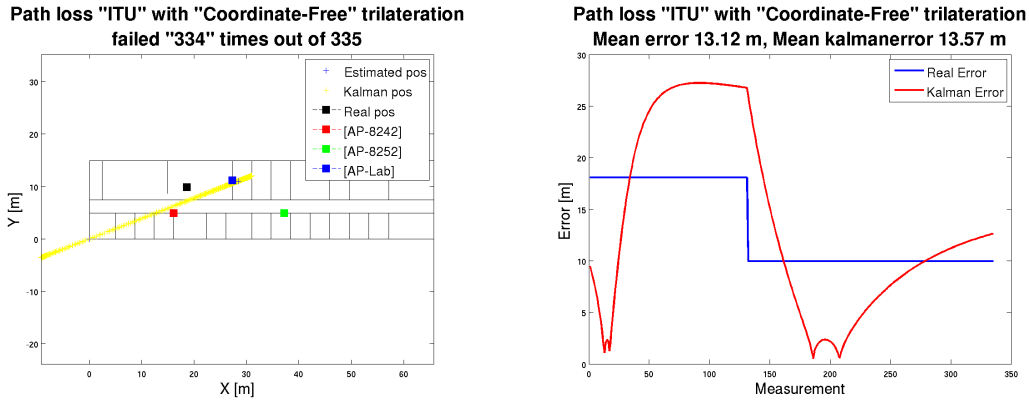
(a) Positions (b) Errors

FIGURE D.2 - Log Distance path loss and Coordinate-free trilateration



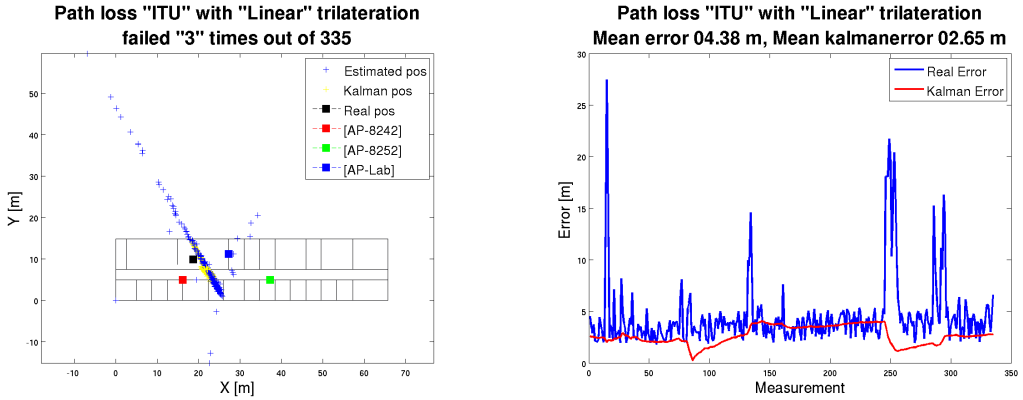
(a) Positions (b) Errors

FIGURE D.3 - Log Distance path loss and linear trilateration



(a) Positions (b) Error

FIGURE D.4 - ITU Path loss and Coordinate-free trilateration



(a) Positions (b) Errors

FIGURE D.5 - ITU path loss and linear trilateration

Estimating a position on a fixed point only get good results when using the ITU path loss model and the linear trilateration method, see figure D.5. The other test, see figures D.2, D.3 and D.4, have a mean error too hight to be of any use. The Coordinate-free trilateration method, see figures D.2(a) and D.4(a), fails to calculate a position in 99% of the time and with a mean error above the 13 m (figures D.7(b) and D.9(b)) the position are also very unreliable.

The conclusion of this evaluation is that positioning using the ITU path loss model and the linear trilateration can be used for position estimation. With a mean error under the 3m it is able to locate a position on room basis. This can be used for orientation and room specific information.

D.2 Moving target evaluation

First the DriverWOM scan the environment when moving on a defined path and save the result in a CSV file. Importing the file in matlab result in Matlab output:

```
Read Data from file
    'door too door in hallway'
Found the following Access-Points:
    [AP-8242] 00:07:0E:15:A4:90
```

```
[AP-8252] 00:07:0E:15:A0:00
[AP-Lab] 00:17:31:D6:94:F5
```

After the importing the dB measurement from the file, distances, using both methods, to the access-point are calculated. Figure D.2 show these distances.

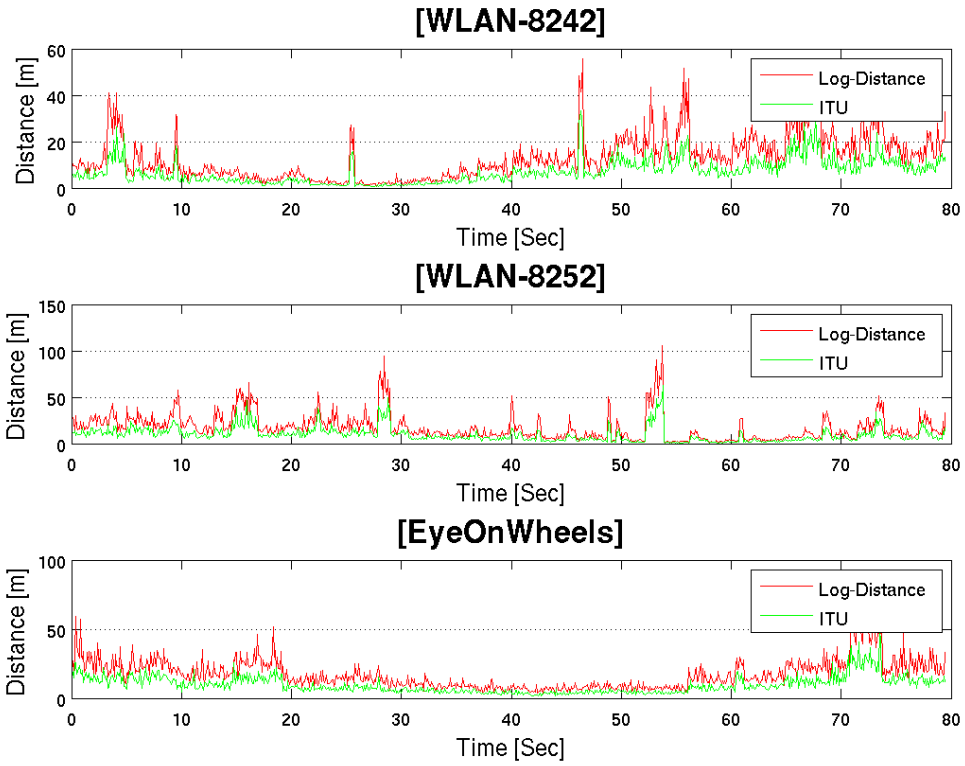


FIGURE D.6 - Calculated distances over time of both Path loss methods

The distances to the access-points and the positions of the aces-point are fed to the two trilateration methods. The calculations results in Matlab output:

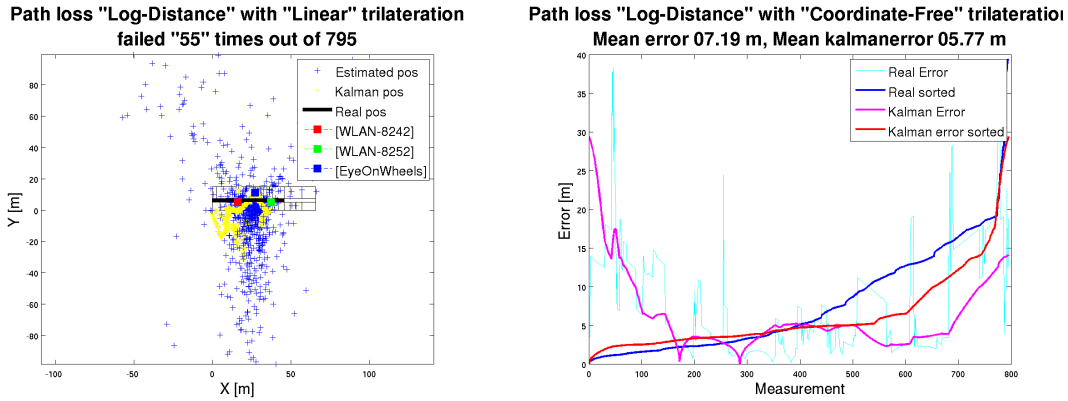
```
Position estimation using trilateration

Positioning using distance estimation method "Log-Distance"
Method Coordinate-Free results in:
    681 where out of reach
    114 point where calculated out of a data set of 795
Method Linear results in:
    55 where out of reach
    740 point where calculated out of a data set of 795

Positioning using distance estimation method "ITU"
Method Coordinate-Free results in:
    750 where out of reach
    45 point where calculated out of a data set of 795
Method Linear results in:
```

```
    6 where out of reach
    789 point where calculated out of a data set of 795
```

Remarkable are the bad results from the “Coordinate-Free trilateration Method”. Clearly the noise on the distances was too big to get proper results. The Linear approach worked well.

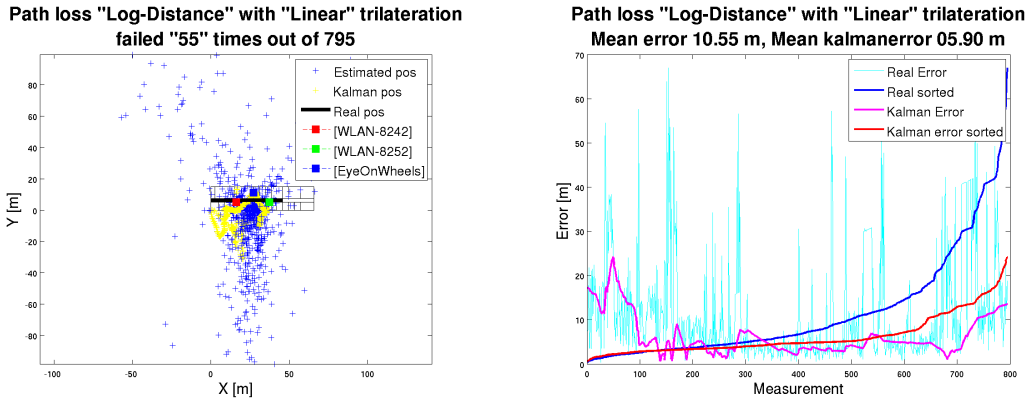
The positions are now filter using a Kalman filter and the error to the real position is calculated. The results are plotted in the figures below. The Matlab output is:

```
Plot the position estimations

Positioning using distance estimation method "Log-Distance"
    Coordinate-Free; had 2 null positions
    Coordinate-Free; had 0 null positions using Kalman
    Linear; had 55 null positions
    Linear; had 0 null positions using Kalman
Positioning using distance estimation method "ITU"
    Coordinate-Free; had 26 null positions
    Coordinate-Free; had 0 null positions using Kalman
    Linear; had 6 null positions
    Linear; had 0 null positions using Kalman
```



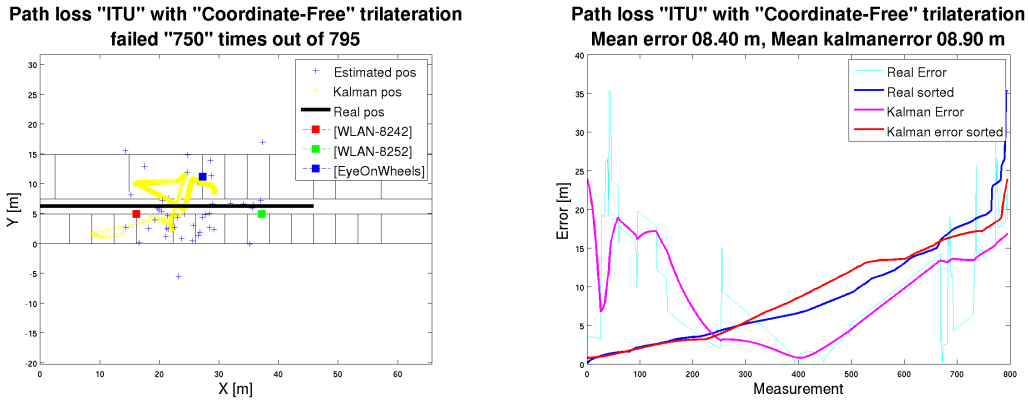
(a) Positions (b) Errors

FIGURE D.7 - Log Distance path loss and Coordinate-free trilateration



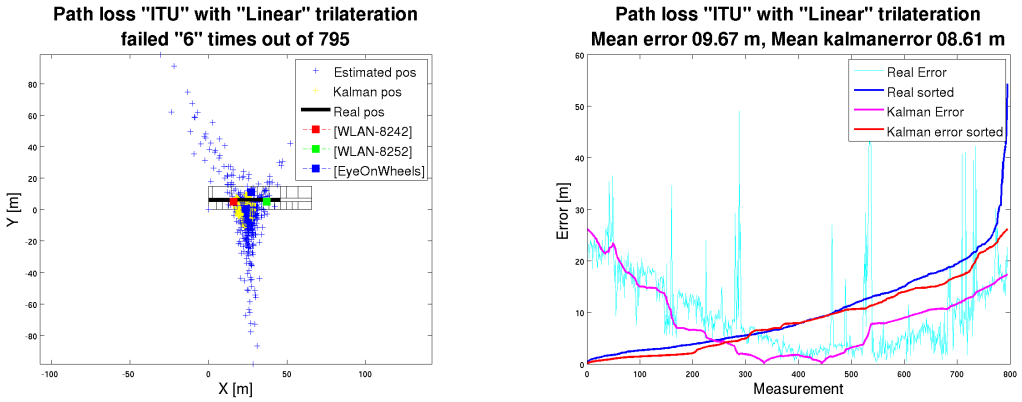
(a) Positions (b) Errors

FIGURE D.8 - Log Distance path loss and linear trilateration



(a) Positions (b) Error

FIGURE D.9 - ITU Path loss and Coordinate-free trilateration



(a) Positions (b) Errors

FIGURE D.10 - ITU path loss and linear trilateration

Following a moving target has bad results on all four tests, see pictures D.7, D.8, D.9 and D.10. With linear trilateration method the dispersion is too high that even a slow kalman filter cannot fix it, see figures D.8(a) and D.10(a). The mean error (figures D.8(b) and D.10(b)) of the kalman filtered positions is between the 6 and 8 meters is too hight, even for orientation purposes. The

Coordinate-free trilateration method, see figures D.7(a) and D.9(a), fails to calculate a position in 90% of the time and with a mean error between 7 and 8 meters (figures D.7(b) and D.9(b)) the position are also very unreliable.

The conclusion of this evaluation is that positioning of a moving target is not possible. The errors and dispersion are to great for tracking and also for orientation purposes.

E The Orocos framework

This appendix is a summary of the "Orocos Component Builder's Manual v 1.4.0 [i]" and contains only some trivial information needed to understand the basics.

E.1 Introduction

The Orocos Real-Time Toolkit (RTT) provides a C++ framework, targeting the implementation of real-time and non-realtime control systems. The Real-Time toolkit act as middleware, allowing components to run on (real-time) operating systems and offers real-time scripting capabilities, the component communication and distribution API and XML configuration. This is illustrated in figure E.1[ii].

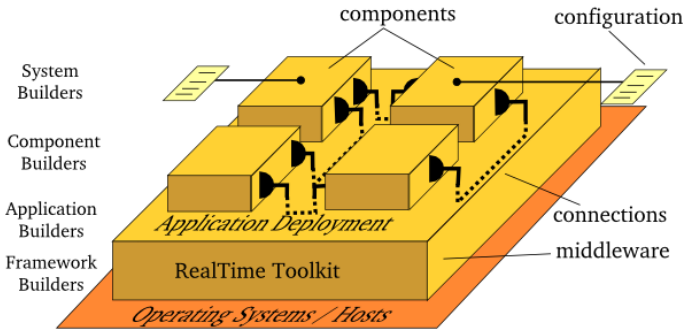


FIGURE E.1 - Orocos RTT as middleware

The Real-Time toolkit allows components to run on (real-time) operating systems and offers real-time scripting capabilities, the component communication and distribution API and XML configuration. This structure is illustrated in figure E.2[ii].

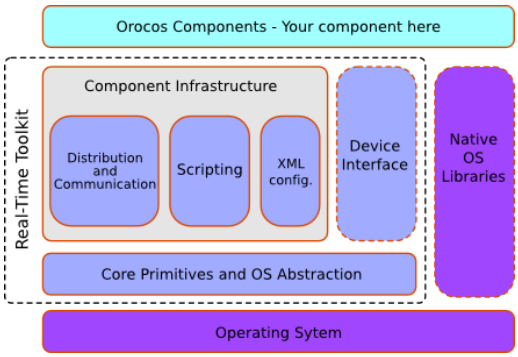


FIGURE E.2 - Orocos structure

E.2 Orocos components

Each compononent, see figure E.3, is built using the "TaskContext" primitive. The "TaskContext" primitive offers:

- efficient (lock-free) port based data exchange.

- Interfacing through events, process commands and applying methods.
- An execution engine for executing Finite State Machines or scripts in hard real-time.
- On-line configuration of properties by “set” and “get” commandoś or by importing a XML file.
- An abstract interfaces to common hardware.

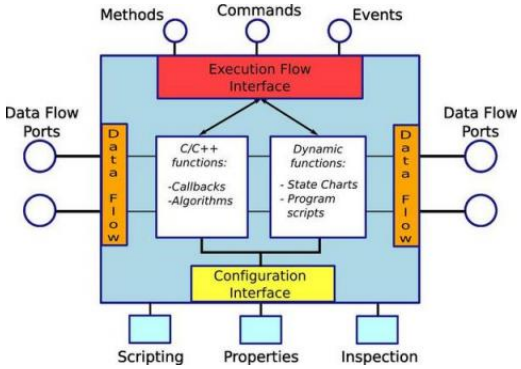


FIGURE E.3 - Component structure

A Component can interact with other components, state machines and scripts by its interfaces: events, commands, methods, properties and data ports. A Component is run by a “PeriodicActivity” or “NonPeriodicActivity” which attaches a thread to the Execution Engine. The RTT::ExecutionEngine is the beating heart of each component which executes the the application code, reacts to events, processes commands etc .... Scripting is enabled by the RTT::ScriptingAccess.

E.2.1 Component C++ template

Listing E.1 shows a template for building components. The component structure is filled in by inheriting from the TaskContext and implementing the ’Hook’ functions. There are five such functions which are called when a TaskContext’s state changes. Use of configureHook() for configuration-time setup/cleanup code (read XML, print status messages etc.) and cleanupHook() (write XML, free resources etc.) are mandatory. The run-time (or: real-time) application code belongs in the startHook(), updateHook() and stopHook() functions. The run-time functions are called oppun by the Activity Interface (PeriodicActivity of NonPeriodicActivity).

[i] source `http://www.orocos.org/stable/documentation/rtt/v1.4.x/doc-xml/orocos-components-manual.html`

[ii] source `http://www.orocos.org`

```
class MyTask
   : public TaskContext
  {
   public:
       MyTask(std::string name)
         : TaskContext(name)
       {
          // Constructor
          // see later on what to put here.
       }

       /**
        * This function is for the configuration code.
        * Return false to abort configuration.
       */
       bool configureHook() {
          // ...
          return true;
       }

       /**
        * This function is for the application's start up code.
        * Return false to abort start up.
        */
       bool startHook() {
          // ...
          return true;
       }

       /**
        * This function is called by the Execution Engine.
        */
       void updateHook() {
          // Your component's algorithm/code goes in here.
       }

       /**
        * This function is called when the task is stopped.
        */
       void stopHook() {
          // Your stop code after last updateHook()
       }

       /**
        * This function is called when the task is being deconfigured.
        */
       void cleanupHook() {
          // Your configuration cleanup code
       }
  };
```

LISTING E.1 - Orocos component template

F Glossary

CSP: The CSP - Communicating Sequential Processes (Hoare, 1985) - language is a process algebra for describing and analyzing concurrent systems. A CSP based design methodology encourages partitioning of the design and allows for good reasoning about the design at every level. A CSP based design trajectory is described in PhD report of Hilderink (2005).

gCSP: gCSP (Jovanovic et al., 2004) is a graphical tool for creating and editing CSP diagrams. It is based on the Graphical Modeling Language (GML) developed by Hilderink (2005). CSP diagrams are dataflow diagrams, connecting processes with channels. Beside the dataflow, the concurrency structure is also indicated. Next to editing diagrams, gCSP does basic consistency checks and can generate code from the diagrams.

CT-library: The CT library is inspired by occam (INMOS Ltd., 1988). It is a kernel library for implementing occam primitives in modern programming languages and for general purpose microprocessors. Occam is a concurrent programming language that builds on the Communicating Sequential Processes (CSP) formalism, and shares many of its features.

Real-Time A real-time system is one in which the correctness of the system depends not only on the logical results, but also the time at which the results are produced. Next to timing, predictability is a keyword in a real-time system. Predictability means that the system has to react to all possible events in a predictable way. The classical conception is that in a hard real-time system, the completion of an operation after its deadline is considered useless - ultimately, this may lead to a critical failure of the complete system (e.g., airplane control system). A soft real-time system on the other hand will tolerate such lateness, and may respond with decreased service quality (e.g., dropping frames while displaying a video).

RTAI: RTAI (RTAI, 2008) is an abbreviation of Real Time Application Interface. It is based on the Linux kernel, providing the ability to make it fully pre-emptable. The RTAI kernel offers the same services as the Linux kernel core, adding some features of an industrial real time operating system. RTAI considers Linux as a background task, running when no real time activity occurs.

LXRT: LXRT is the user space interface with RTAI modules (inside the Linux kernel). The usage of LXRT benefits from the 2 modes: user and kernel. LXRT switches automatically between the two modes if necessarily. For example from “kernel space” to “user space” when user space instructions are needed, like a library call and file access. These switches result in time losses but overall these losses benefit the use of user-space instructions.

middleware: Middleware is computer software that connects software components or applications. In Orocos, the middleware is provided by the Real-Time Toolkit, which allows (among other things) that components can communicate over a network. It does this using the Corba middleware software in turn.

Bibliography

Bahl, P. and V. Padmanabhan (2000), RADAR: an in-building RF-based user location and tracking system, *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, **2**, pp. 775–784 vol.2.

Baillie, J.-C. (2004), URBI: towards a universal robotic body interface, *Humanoid Robots, 2004 4th IEEE/RAS International Conference on*, **1**, pp. 33–51 Vol. 1.

Bruyninckx, H. (2001), Open robot control software: the OROCOS project, *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, **3**, pp. 2523–2528 vol.3, ISSN 1050-4729.

Chen, A., C. Harko, D. Lambert and P. Whiting (2007), An algorithm for fast, model-free tracking indoors, *SIGMOBILE Mob. Comput. Commun. Rev.*, **11**, 3, pp. 48–58, ISSN 1559-1662.

Controllab Products B.V., . (2008), 20-sim, the software for modeling dynamic systems, `http://www.20sim.com/`.

cyberbotics (2008), Cyberbotics Webots, `http://www.cyberbotics.com/`.

Damstra, A. (2008), *Virtual prototyping through co-simulation in hardware/software and mechatronics co-design*, Master's thesis, Control Laboratory, University of Twente.

DARPA (2008), DARPA Grand Challenge, `http://www.darpa.mil/GRANDCHALLENGE/`.

Dutch Robotics, . (2008), Dutch Robotics, `http://site.dutchrobocup.com/`.

Ekahau (2008), Ekahau Positioning Engine, `http://www.ekahau.com/`.

Garlan, D. and M. Shaw (1994), An Introduction to Software Architecture, Technical Report CMU-CS-94-166, Carnegie Mellon University.

gostai (2008), The creators of URBI, `http://www.gostai.com/`.

Hilderink, G. H. (2005), *Managing complexity of control software though concurrency*, Ph.D. thesis, University of Twente.

Hoare, C. (1985), Communicating Sequential Processes, *Prentice Hall*.

IEEE Std., . (2003), Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, *ANSI/IEEE Std 802.11, 1999 Edition (R2003)*, pp. i–513.

INMOS Ltd., . (1988), occam 2 Reference Manual, *Prentice Hall*.

ITU, . (2005), Propagation data and prediction methods for the planning of indoor radio communication systems and radio local area networks in the frequency range 900 MHz to 100 GHz, Technical report, ITU-R RECOMMENDATION.

Jovanovic, D., B. Orlic, G. Liet and J. Broenink (2004), gCSP: A Graphical Tool for Designing CSP systems, *Communicating Process Architectures 2004*.

Kiva Systems, . (2008), The Mobile Fulfillment System, `http://www.kivasystems.com/mobile-fulfillment-system.html`.

Kotanen, A., M. Hannikainen, H. Leppakoski and T. Hamalainen (2003), Positioning with IEEE 802.11b wireless LAN, *Personal, Indoor and Mobile Radio Communications, 2003. PIMRC 2003. 14th IEEE Proceedings on*, **3**, pp. 2218–2222 vol.3.

Kwon, J., B. Dundar and P. Varaiya (2004), Hybrid algorithm for indoor positioning using wireless LAN, *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th*, **7**, pp. 4625–4629 Vol. 7, ISSN 1090-3038.

Ladd, A., K. Bekris, A. Rudys, D. Wallach and L. Kavraki (2004), On the feasibility of using wireless ethernet for indoor localization, *Robotics and Automation, IEEE Transactions on*, **20**, 3, pp. 555–559, ISSN 1042-296X.

Lootsma, M. (2008), *Implementation of controllers for the 3TU soccer robot*, Master's thesis, Control Laboratory, University of Twente.

Makarenko, A., A. Brooks and T. Kaupp (2006), Orca: Components for Robotics, *Intelligent Robots and Systems, (IROS'06). IEEE/RSJ International Conference on*.

Metta, G., P. Fitzpatrick and L. Natale (2006), YARP: Yet another robot platform, *International Journal on Advanced Robotics Systems*, **3**, pp. 43–48.

de Moraes, L. F. M. and B. A. A. Nunes (2006), Calibration-free WLAN location system based on dynamic mapping of signal strength, in *MobiWac '06: Proceedings of the 4th ACM international workshop on Mobility management and wireless access*, ACM, pp. 92–99, ISBN 1-59593-488-X.

Nesnas, A., A. Wright, M. Bajracharya, R. Simmons, T. Estlin and W. Kim (2003), CLARAty: An architecture for reusable robotic software, *SPIE Aerosense Conference*.

Nguyen, H., Cressel, Anderson, A. J. Trevor, A. Jain, Z. Xu and C. C. Kemp (2008), El-E: An Assistive Robot that Fetches objects from Flat Surfaces, in *Robotic Helpers: User Interaction, Interfaces and Companions in Assistive and Therapy Robotics*, 3rd ACM/IEEE International Conference on Human-Robot Interaction (HRI), pp. 79–86.

"OMG" (2008), Common Object Request Broker Architecture, `http://www.corba.org/`.

Petters, S., D. Thomas and O. von Stryk (2007), RoboFrame - A Modular Software Framework for Lightweight Autonomous Robots - Workshop, Workshop at IROS 2007.

Prigge, E. and J. How (2000), An indoor absolute positioning system with no line of sight restrictions and building-wide coverage, *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, **2**, pp. 1015–1022 vol.2.

RTAI (2008), RealTime Application Interface for Linux, `https://www.rtai.org/`.

Schipper, D. (2001), *Mobile Autonomous Robot Twente, a mechatronics design approach*, Ph.D. thesis, University of Twente.

Thomas, F. and L. Ros (2005), Revisiting trilateration for robot localization, *Robotics, IEEE Transactions on [see also Robotics and Automation, IEEE Transactions on]*, **21**, 1, pp. 93–101, ISSN 1552-3098.

Tourrilhes, J. (2007), Wireless Tools for Linux, `http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/`.

Usov, M. (2006), *Vision Based Mobile Robot Navigation*, Master's thesis, Control Laboratory, University of Twente.

ZeroC (2008), The Internet Communications Engine (Ice), http://www.zeroc.com/ice.html.