UAVCAN-enabled indoor postioning solution for unmanned vehicles

Riccardo Miccini
Technical University of Denmark
Centre for Bachelor of Engineering Studies

June 22, 2016

Abstract

Usual abstract stuff.

The opinions expressed in this document are the author's own and do not reflect the view of UAVComponents and its personnel.

Contents

Chapter 1

Introduction

This introductory chapter will provide background information about Real-time Locating Systems (in short — and throughout the rest of this document — *RTLS*) and the current state of the art, along with a brief description of the partner company, the project description under the form of assignment formulation, and a brief overview of the implementation. The latter will be elaborated further in the following chapters.

1.1 Rationale

The importance of precise location

1.1.1 RTLS

From map-making to post tracking, localization has been a fundamental human endeavor which encompassed several ages and civilizations. Although GPS has proven to be a major leap is this regard, its current limitations causes it to be unsuitable for some purposes. First of all, the accuracy lays in the order of a few meters at best, mainly due to the relativistic implication of orbiting satellites [1] (that is without considering more advanced techniques such as *RTK*). Moreover, signal shadowing and multi-path reflections renders it particularly unreliable indoor or in dense urban areas. In order to fulfill the ever-growing demand for automation, more precise location systems have to be employed.

The term RTLS emerged from such needs, and characterizes systems capable of reckoning, tracking, and showing the position of moving objects and persons within a relative frame of reference, mainly a building or enclosed area. The applications benefiting from those systems mainly lay within the logistical/operational areas, and the interested parties range from manufacturing industries to health care operators (management of flows of goods in a warehouse or production plant; assets retrieval e.g. tools, medical equipment, cattle; surveillance and monitoring; personnel deployment and supervision). Consumer-related applications are also starting to arise in the fields of domotics and electronic devices.

The technologies driving these systems include ultrasounds, infrared radiation, or RSSI (Received signal strength indication) from various sources such

as radio frequency modules, WiFi/WLAN, or even Bluetooth; however, each of them presents its own limitations which prevented them from being used in large scale. [2]

The more recent implementation of previously known technologies into positioning systems spawned a vast array of previously unforeseen applications, such as their use along with drones and other unmanned vehicles for maintenance and inspection of facilities, emergency situation management, and whenever human intervention may pose a safety threat.

1.1.2 UAVCAN

Although drones, copters, rovers, and the like are are experiencing a huge rise in popularity in both commercial and hobby use, integrators and final users often have to cope with a vast amount of different protocols and interfaces, some of which directly inherited from the RC modeling world, thus lacking fault tolerance and scalability. For years, the automotive industry has relied on CAN Bus as a means of communication between different microcontrollers and devices, due to its low cost and resilience against electro-magnetic resilience.

UAVCAN is a lightweight open source protocol aimed to provide fast and reliable communication within drone peripherals. [3] A UAVCAN network is composed of a series of nodes which are independent from the others and require no master/controller or host computer. The protocol sits on top of the transport and physical layers of CAN and relies on a library for easy integration with the user application code. Common standard high-level functions like node health monitoring, network discovery, time synchronization, and firmware update are predefined and taken care of by the library, while message definitions for most drone equipment (speed controllers, navigation data, gimbals, actuators) are standardized and publicly available, making it easy for manufacturers to engineer compliant devices and for integrators to interface with them.

1.2 Company description

UAVComponents ApS is a danish company which develops and supplies components, equipment, and tools for drones and UAVs integrators and hobbyists. Its main products are the *TSLRS* line of transceivers and the *Aeronav One* ground control station. Along with off-the-shelf hardware solutions, the company offers its expertise to clients seeking consultancy or ad-hoc projects. UAVComponents shares its offices and personnel with *Danish Aviation Systems*, a sister company also operating in the drones industry.

The team is composed of roughly eight people, almost all of whom are involved in Research&Development. Besides a solid know-how of piloted and autonomous aerial vehicles, the core competencies include the electronic design of analogic and MCU-based solutions, and software development — both for embedded and computer systems. The company relies on external support for more skill-intensive tasks such as mechanical and RF design. Manufacture is also mostly outsourced, especially for high volume yields.

As the vast majority of the employees is quite young, the company can boast a very dynamic and informal atmosphere, which in turn resulted in flexible working hours and a high degree of collaboration among all the colleagues.

1.3 Problem formulation

The aim of the project is to design and develop a UAVCAN-compatible indoor positioning system for aerial unmanned veicles (UAVs) based on the *decaWave* devices. Using these radio modules, an algorithm shall be devised and implemented in order to estimate the UAV position in a 2D coordinate system. The resulting information is to be broadcasted through the UAVCAN bus — a message definition shall be chosen amongst the pool of standard definitions, or designed from scratch.

The performance of the system ought to be in line with decaWave's claims in term of maximum accuracy and range, and the device has to be suitable for battery operation. Moreover, the system shall comply with the UAVCAN specifications and support its major standard functionalities.

The project is carried out in collaboration with *UAVComponents ApS*, which will lend its expertise in hardware design and provide the necessary development tools.

1.4 System description

The system is composed of one (or more) *tags* and up to four *anchors*.

The anchors are to be spread across the area of operation, and should all intersect the same horizontal plane. Their coordinates within the frame of reference shall be know beforehand. The tag device is located inside the drone/vehicle and is connected through the UAVCAN bus.

Both type of devices share the same hardware design, but are flashed with different firmwares. It comprises:

- ARM Cortex-M4 microcontroller
- decaWave DWM1000 module
- Inertial measurement unit
- Barometric sensor
- Power supply unit
- CAN transceiver and connectors
- USB interface
- SWD connector for debugging and flashing

The boards can be powered through either the USB or the UAVCAN connectors.

The software infrastructure governing both firmwares is built upon *ChibiOS*, a real-time operating system for embedded devices. Each device — regardless of its role — can be configured through a command line interface accessible through the USB.

Each anchor runs the same software routine, but with different parameters (coordinates and transmission delay). Their role is to respond to the polling message sent by the tags.

The tag firmware is responsible for initiating new communications and processing it (in a way that is thoroughly described in the Theoretical principles).

Whenever a new response message is parsed and the distance between the tag and that anchor calculated, the position estimation is updated. Lastly, the location broadcaster transmit the current estimation to the UAVCAN bus, in an asynchronous fashion.

Chapter 2

Problem analysis

The following sections aim to elaborate on the Problem formulation and create a more solid base to use as a guideline and template during the actual development. The adopted framework is based upon the one presented during the course of the studies [4] and bears similarities with the *Unified Software Development Process*.

This analysis does not aspire to be exhaustive, but rather satisfactory in the level of detail required to establish the project. Moreover — in order to avoid resorting to a priori assumption, the term *decaWave* is hereby used to denote the product (IC or module) rather than the company.

2.1 Requirements specification

2.1.1 Functional requirements

As per definition, they "define specific behavior or functions", and are the basis for the tests.

2.1.2 Non functional requirements

Other charateristics and qualities of the system.

2.1.3 Use cases

This section describes how the actors can interact with the system. For each there should be: brief description, actors, preconditions, main flow, post conditions, remarks / alternative flows.

2.2 Theoretical principles

This section will present the theoretical aspects of the technologies employed in the project, and provides a knowledge base for better understanding the problem and the employed solution.

2.2.1 Ultra-wideband

Ultra-wideband (UWB in short) is a radio technology that use a very low energy level for short-range, high-bandwidth communications over a large portion of the radio spectrum.

Although this technique has been know for for decades, is a relatively new entry in the RTLS field [5], with the first affordable development kits having entered the market in the last few years. It gained immediate popularity due to its advantages: blablabla

2.2.2 Ranging algorithm

There can be several possible two-way ranging schemes, blabla

Their role is to respond to the polling message sent by the tags after predetermined delay. This response message constains the polling message reception timestamp and its own transmission timestamp.

2.2.3 Position estimation

Explanation of the triangulation algorithms.

2.2.4 CAN Bus

Brief explanation of CAN protocol, its history, and principles of operation. May also be opted out and substituted with notions of ARM Cortex-M architecture.

2.3 Risk management

Risk assessment and management are well-established disciplines amongst many fields of engineering and other areas of human endeavor, and represent a way of minimizing the probability and impact of adverse conditions in a given context.

In the Risk assessment subsection, possible sources of hindrance are categorized and listed. Preventive measures are then suggested in the Prevention and mitigation subsection.

2.3.1 Description

This section acts as a legend, providing the necessary information for interpreting the subsequent risk assessment. Risks are evaluated according to their composite risk index (CRI):

$$\text{CRI} = \text{Severity} \times \text{Probability of occurrence} \tag{2.1}$$

The CRI represents the degree of importance of each entry. Entries with a high (ie. red-leaning) CRI shall have maximum priority.

Traditionally, this kind of analysis involves assigning a numeric value from an arbitrarily chosen range to each variable — e.g. 1–5 for the severity and 0–100% for the probability. However, educated estimations require solid data and a tracked history of previous occurrences, all of which were either not available or difficult to obtain with the necessary degree of confidence. [6] Therefore, the final value of the CRI has been neglected in favor of a color-mapped matrix composed

of four sub-ranges. Whilst selecting these sub-ranges, higher "granularity" has been given to the more adverse scenarios.

		Probability of occurrence			
		Negligible (1)	Low (2)	Medium (3)	High (4)
Severity	Small (1)				
	Moderate (2)				
	Large (3)				
	Critical (4)				

Table 2.1: Risk impact matrix

2.3.2 Risk assessment

Follows the lists of threats (divided by category) with associated likelihood, impact, and overall CRI. The risk index is accompanied with the affected resource(s), chosen amongst:

- Product **F**unctionality
- Project **C**ost
- Development **T**ime

Whenever possible or necessary, a brief explanation of the predicted consequences is given.

Hardware

The estimations are based on company personnel's knowledge and previous experience with the given components and communication protocols governing them. For the purpose in this analysis the board components have been divided in primary (microcontroller, decaWave module, CAN transciever) and secondary (power supply unit, other sensors, USB interface), based on their importance for the fulfillment of the project.

ID	Risk	Prob.	Sev.	CRI	Res.	Contingencies/Notes
SD	*Schematics design issues (wrong/missing parts, connections...)*					
SD.1	Primary components	3	4	🔴	TC	New PCB manufacturing
SD.2	Secondary components	2	3	🟡	TF	
PL	*PCB layout issues (wrong footprints, overlapping traces...)*					
PL.1	Primary components	3	3	🟠	TC	New PCB manufacturing
PL.2	Secondary components	1	2	🟢	TF	
SC	*Supply chain issues*					
SC.1	Components lead time	1	3	🟡	T	Wait until restocked
SC.2	PCB lead time	3	3	🟠	T	Negotiation/bargaining
SC.3	Defective PCB	1	4	🟡	T	New PCB delivery
PU	*Performance/usability issues*					
PU.1	MCU doesn't boot up	1	4	🟡	T	
PU.2	MCU not detected by SWD	2	4	🟠	T	Tedious troubleshooting
PU.3	High packet loss (USB, CAN)	1	2	🟢	F	
PU.4	High packet loss (decaWave)	2	4	🟠	F	

Table 2.2: Risk assessment: Hardware-related threats

Software

The estimations are based on personal previous development experience, online research, and relevant documentation. The table sub-categories loosely reflect the different software layers described in the Software section.

ID	Risk	Prob.	Sev.	CRI	Res.	Contingencies/Notes
DE	*Development environment issues*					
DE.1	No compiler available	1	4	●	C	Switch MCU vendor/arch. or host system
DE.2	No flashing tools available	2	4	●	C	Switch MCU vendor/arch. or host system
DE.3	Loss of data	2	3	●	T	
LL	*Low-level support issues (peripherals, data busses...)*					
LL.1	No MCU peripherals drivers	2	3	●	TF	Implement from scratch
LL.2	No MCU peripherals docs.	1	4	●	TC	Switch MCU vendor/arch.
LL.2	No comm. with ext. devices	2	4	●	TF	
MW	*Middleware issues (libraries, device APIs...)*					
MW.1	Restrictive/costly license	1	3	●	CF	Purchase; Implement from scratch
MW.2	Lack of platform support	3	2	●	TF	Implement port
MW.3	Poor documentation	2	3	●	TF	
MW.4	Unstable/buggy code	2	2	●	F	
UA	*User application-level issues*					
UA.1	Reached computational limit	1	3	●	F	
UA.2	Middlewares conflict	2	3	●	TF	
UA.3	Poor system accuracy	2	3	●	F	
UA.4	Unimplemented use cases	2	4	●	T	Increased time-to-market
UA.5	Unmet requirements	2	4	●	T	Increased time-to-market

Table 2.3: Risk assessment: Software-related threats

2.3.3 Prevention and mitigation

After having discerned and evaluated as many threats as possible, a number of preemptive measures are suggested and summarized on a sub-category basis. In order to characterize possible alternative methods, a

ID	Description	Approach	Applies to
HW.1	Adopt previously employed components and parts	Avoidance	SD, PL
HW.2	Follow design resources (datasheets, application notes...) from vendors	Reduction	SD, PL, PU
HW.3	Allow multiple people to verify the schematics	Reduction	SD
HW.4	Adopt established naming conventions for pins and signals	Reduction	SD
HW.5	Employ EDA tool's electrical and design rule check (ERC, DRC)	Reduction	SD, PL
HW.6	Verify parts availability and costs beforehand	Avoidance	SC
HW.7	Keep a local stock of commonly used components	Reduction	SC
HW.8	Generate BOM from EDA tool	Avoidance	SC
HW.9	Provision from multiple suppliers	Reduction	SC
HW.10	Pay for expedited delivery	Retention	SC
HW.11	Employ EDA tool's differential pair routing	Reduction	PU
HW.12	Limit bitrate to safer value	Retention	PU
HW.13	Base solution of DWM1000 (integrated module) instead of DW1000 (IC)	Retention	PU
SW.1	Verify tools availability and support	Avoidance	DE
SW.2	Adopt revision control system and host project on dedicated server	Reduction	DE
SW.3	Assess MCU capabilities through vendor's evaluation kit	Reduction	DE, LL, UA
SW.4	Switch to platform/vendor with more extensive support	Retention	DE, LL
SW.5	Provide redundant interfaces whenever possible (e.g. SPI and I2C)	Reduction	LL
SW.6	Connect pin to MCU when in doubt about its configuration	Reduction	LL
SW.7	Employ open-source software solutions	Reduction	DE, MW
SW.8	Acquire fluency with the necessary tools	Reduction	DE, LL, MW
SW.9	Use most tested and stable library version	Reduction	MW
SW.10	Employ reusable components from own/company source base	Reduction	LL, MW, UA
SW.11	Evaluate and test different ranging schemes	Reduction	UA
SW.12	Periodically assess development progress	Reduction	UA
SW.13	Consult online community (user forums, mailing lists, GitHub...)	Reduction	DE, LL, MW, UA

Table 2.4: Risk management: threats preventions and mitigation proposal

2.4 Milestone plan

Here is the project milestone plan as formulated during the *Inception* and *Elaboration* phases. This version of the plan lists a series of achievements deemed necessary to accomplish the major functionalities. It is arranged in a progressive way, with a time axis roughly flowing vertically. Each task depends from its

sub-tasks and — more indirectly — on the previous ones sharing its hierarchy level.

Although it has been used — together with the previously discussed Use cases — as blabla, casini vari

Please refer to the Possible improvements for the final version of the milestone plan based on the actual project development.

This section should be based on the following (sorted out) milestone plan. It should also refer to the conclusions and appendices for remarks and detailed logs.

1. get the modules to work (Arduino, mbed)
   (a) simple tx/rx
   (b) ranging
   (c) localization
2. schematics design
3. software development
   (a) implement basic peripherals
   (b) port working localization code to ARM
   (c) integrate libUAVCAN
4. integration and validation tests
5. documentation
   (a) final report
   (b) product datasheet
   (c) production files

Chapter 3

Design and implementation

This section focuses on the design and implementation of the proposed solution into a physical product. It namely consists in the microcontroller-driven circuit board design, the firmware for the beacons/anchors, and that of the tag.

During the course of the development there have been two major hardware design iterations, with the latter completely breaking compatibility with previous code. This controversial measure has been deemed necessary after having invested a considerable amount of the allocated time trying to troubleshot the decaWave module transmission failures. (refer to Risk assessment ID# PU.4, SC.2 and Prevention and mitigation ID# SW.4)

Nevertheless, the upcoming sections will present an exhaustive description of the system, which may serve as a vademecum for the continuation of the project (outside of its academic scope). Such solution is based on the second, more successful iteration of the project. Notions about the former design are included in the Possible improvements. Adequate considerations about the development process will be drawn in section 5.2.

Amongst the third-party tools and software/hardware components employed in the making of the project, only those directly embedded into the implementation of the aforementioned designs (e.g. UAVCAN stack, ChibiOS, the onboard sensors, etc...) will be thoroughly described, whilst the others are only briefly mentioned (see subsection 3.1.1 and subsection 3.2.1).

3.1 Hardware

This section will present the hardware architecture of the system, focusing on both large-scale aspect and individual components/subsystems.

The board design consists in a single PCB built around the decaWave DWM1000 ultra-wideband radio module. It also includes a with a microcontroller, CAN and USB interfaces, and secondary sensors. At least four of them are necessary for a complete system and since the anchors are stationary, they do not require the secondary sensors to be populated.

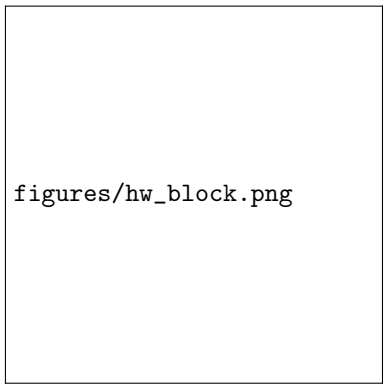

Figure 3.1: Hardware design: block diagram

3.1.1 Setup

The schematic capture, PCB layout, and subsequent export of production files (Gerber,) has all been performed with the aid of
emphKiCad, an open-source ECAD software. Most of the necessary components were available out of the box, with the missing ones imported from the company's *Eagle* libraries. KiCad also include a component library editor for both symbols and footprints.

For hardware troubleshooting and testing, a quad-channel Tektronix digital oscilloscope was used.

Connection of the board to the host system for firmware upload and debug was possible thanks to an *ST-LINK/V2*-compatible SWD in-circuit debugger. Before the custom-made PCB were available, development has been carried out on a *STM32F4DISCOVERY* board, which features an the aforementioned debugger probe, an MCU from a compatible family, and pin headers.

3.1.2 decaWave DWM1000 module

The DWM1000 is an UWB wireless transceiver module designed for easy adoption in two-way ranging schemes. It is based around the DW1000, an integrated low-power, single chip CMOS RF transceiver IC compliant with the *IEEE802.15.4-2011 UWB* standard, and has been chosen because it required no RF design: the antenna and associated RF circuitry are already on the module.

Along with typical RF modules capabilities such as configurable power modes and data transmission rates, the module can use custom, proprietary leading-edge detection algorithm which — along with configurable preamble size and optional, non-standard frame delimiter — can provide accurate timestamping of incoming and departing messages.

The module contains an 38.4 MHz reference crystal used by the SPI interface and the transmitter for signal modulation and timestamps generation. Its accuracy is ensured by a factory trimming process which brings the frequency offset down to about 2 ppm. The on-chip crystal trimming circuit and temperature sensor allows for dynamic temperature compensation.

Interfacing with the device is achieved through an SPI bus that can run at up to 20 MHz. Other data connections include the interrupt (IRQ) and reset (RST) signal.

3.1.3 Microcontroller

After the previous failed attempt with the *LPC1549* ARM Cortex-M3 microcontroller running at 72 MHz, it has been deemed necessary to move towards a more powerful MCU with faster serial communication ports.

A better device for the project has been found in the *STMicroelectronics STM32F405RG*. It is based upon the 32-bit ARM Cortex-M4 core running at up to 168 MHz, and feature a single-precision FPU, 1 MB of FLASH memory, 192 kB of static RAM, and a wide array of internal peripherals including CAN, SPI, USART, I2C, and USB.

Most these peripherals are commonly available in modern, high-performances MCUs. However, the STM32F405RG has the advantage of being shipped in a 64-pin LQFP package which makes it tiny, inexpensive, and easy to solder manually.

3.1.4 Sensors

Two supplementary sensors are present on the board: a *BMP-280* pressure sensor and *MPU-9250* inertial measurement unit.

The first one measures atmospheric pressure, which can be used for altitude estimation. The latter provides measurements of inertial acceleration, angular rate, and magnetic field along along each of the three axes. It operates thanks to internal *MEMS*-based structures which can move relative to each other, and whose change in capacitance is then sampled and processed. IMUs are often used to measure the heading of a vehicle in space, and can also be part of a *dead-reckoning system* for position estimation.

As shown on **??**, the devices are connected through both a SPI and an I2C bus (as per SW.5, SW.6). An additional `INT` pin is available on the IMU, and can be used

3.2 Software

This section discusses the implementation details of the two firmwares involved in the project. The two firmwares share most of the code base, with only the application-level parts performing different operations. Their main components are:

ChibioOS An open-source real-time operating system (*RTOS*) with portable peripherals support and high-performance kernel

Zubax-ChibioOS Middleware providing a more comprehensive command-line interface over USB, watchdog support, and FLASH-based configuration storage

libuavcan Reference implementation of the UAVCAN protocol written in C++

decadriver APIs for controlling the decaWave products

application code High-level board functionalities and RTLS implementation

Due to the difficulties presented at the beginning of chapter 3, at the time of writing only the anchors software is fully developed. For a detailed report of the missing features and requirements, refer to Acceptance tests.

The following subsections will cover the shared project layout, informations and integration details of the third-party modules, An overview of the host system used for the development is provided for reference purposes.

3.2.1 Setup

Figure 3.2: Development environment: typical session

Development environment

The choice of an integrated development environment has fallen on *Ac6 System Workbench for STM32*, an Eclipse-based IDE with built-in support for the STM32 line of ARM microcontrollers. The distribution is based on open-source software, and requires little to no configuration. The components shipped with it are:

GCC ARM Embedded GCC-based cross compiler and debugger for ARM Cortex platform

OpenOCD On-chip debugging tool supporting a wide range or generic JTAG/SWD interfaces

Eclipse CDT An extensible multi-language IDE

All of these packages could be installed manually by fetching them from their respective sources, but doing so with a single archive is undoubtedly more convenient. On top of this, it also comes with its own project management facilities like build setting manager, linker script editor, STM evaluation boards library, and so forth.

The latter tools are not taken advantage of due to the choice of managing the project building through *make* and *Makefile*, as it is the supported build system for both ChibiOS and libuavcan. Building projects using Makefile is fully supported withing Eclipse and there

SublimeText or *Atom* text editors are sometimes used too, especially when large amount of code has to be refactored or restructured.

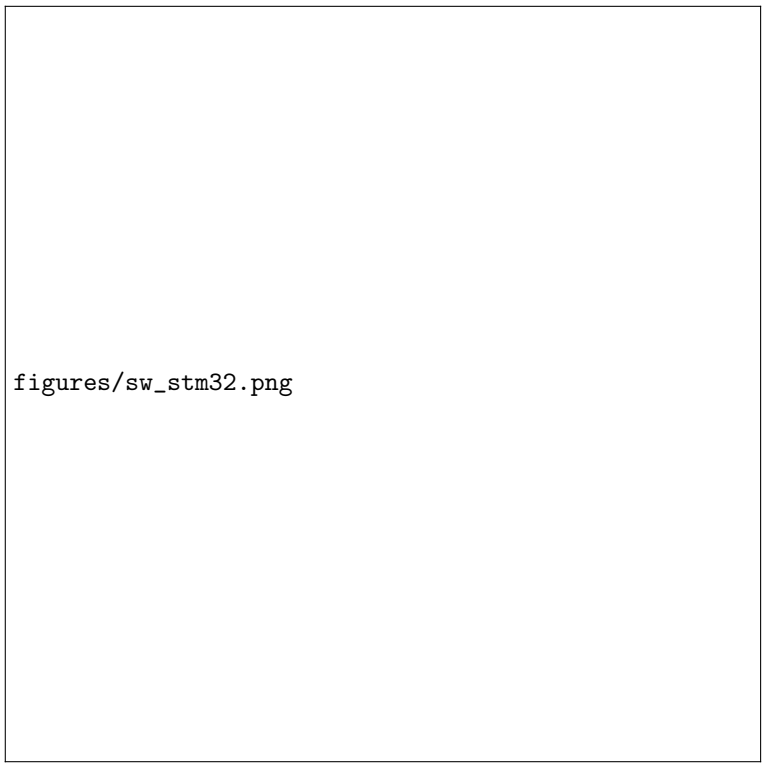

Figure 3.3: Development environment: typical session

Project structure

Explain code subdivision (HAL, modules, processes, other user app code, and libraries).

Other tools

Sublime Text, git, GitHub, debug server, UAVCAN tools.

3.2.2 ChibiOS

HAL

RT

configuration

3.2.3 UAVCAN

Protocol

Description and explanation of the protocol (only uppermost layers).

Library

Explanation of the different implementations and functionalities of the reference one.

Platform driver Description of driver interface, microcontroller CAN interface, and necessary adaptation.

3.2.4 Software modules

Board layer

decaWave drivers

The bus configuration (sampling polarity and phase) can be customized by manipulating

Configuration manager

Stores and retrieves settings of arbitrary type. Needed by pretty much anything. Will be implemented on top of EEPROM since it's the only available permanent, user-writtable memory (Flash is too, but on a page-basis).

Command line interface

Used for providing runtime infos and changing settings without the need of a debugger probe. Will read from UART.

3.2.5 Threads

(There may be more processes).

UAVCAN Node

Broadcasting of node information, server for handling configuration changes, firmware updates, and so on.

Location publisher

As per title.

Location estimator

As per title.
others?

Idle task

Updates watchdog, blinks LEDs.

Chapter 4

Testing

4.1 Acceptance tracing

This section should be an overview of the tests conducted. A table shall correlate the requirements with the tests that prove their fulfillment, as well as the outcome (passed, not passed, partially passed, feature not implemented).

4.2 Acceptance tests

Detailed view of the hardware-related tests. Should show the following: purpose (which requirements it tests), preconditions, test actions wth relative expected results.

Chapter 5

Conclusions

5.1 Product assessment

Conclusions based on the product, e.g. if it fulfilled all the requirements, if it yielded the desired performances etc ...

5.2 Process assessment

Conclusions based on the development, e.g. how lean it has been, any useful tool discovered, any cumbersome aspect of the workflow that coud have been improved ...

use risk management to explain escalation of issues: e' stato veritiero? e' servito a un cazzo?

5.3 Possible improvements

Suggestions on improvements based on missing features, company and supervisors feedbacks, etc ...

Bibliography

[1] 2016 — Richard W. Pogge — Real-World Relativity: The GPS Navigation System http://www.astronomy.ohio-state.edu/~pogge/Ast162/Unit5/gps.html

[2] 2009 — Joost Koppers — Location Based Services: A general model for the choice between positioning techniques (Dutch) Radboud University Nijmegen

[3] 2014 — Pavel Kirienko — UAVCAN Specification http://uavcan.org/Specification/1._Introduction/

[4] 2013 — DTU Course Base — Object oriented software engineering compendium http://www.kurser.dtu.dk/2013-2014/62432.aspx

[5] 2005 — Faranak Nekoogar — Ultra-Wideband Communications: Fundamentals and Applications ISBN-10: 0-13-146326-8

[6] 2004 — David Hillson & David Hulett — Assessing Risk Probability: Alternative Approaches PMI Global Congress Proceedings

Appendix A

Glossary

MCU Microcontroller Unit

NXP A semiconductor designer and manufacturer company

RTLS Real-time locating system

UAV Unmanned aerial vehicle, commonly known as drone

term definition

another term another definition

Appendix B

Quickstart guide

Setup, CLI and stuff.

Appendix C

Iteration 1

This section should explain how to set up and use the device, the way a user manual normally would.

Appendix D

Further material

Further content to be found in the hand-in support, with explanation of its folder structure. It may include:

- Source code
- Repository commit logs
- Matlab scripts
- Data and pictures from tests
- Production files
- Bill of materials