**TITLE OF THE THESIS**

**Treball de fi de grau**

**Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona**

**Universitat Politècnica de Catalunya**

**per**

**Ignasi Bernadas Leal**

**GRAU EN ENGINYERIA DE SISTEMES AUDIOVISUALS**

**Tutor: Josep Ramon Casas**

**Barcelona, Juny 2016**

Abstract

Resum

Introducció

Aquest projecte consisteix en la recerca i l’exploració de les possibilitats que pot oferir un sistema incrustat de NVIDIA, la placa Jetson TK1 i, més enllà d’això, fer-la servir com a plataforma per instal·lar-hi un software avançat d’assistència al conductor.

El programa prové d’un projecte de fi de grau d’una estudiant de la Universitat Politècnica de Catalunya amb col·laboració amb l’empresa Arcol i consisteix en un sistema de visió en 360º per a vehicles de grans dimensions.

La visió panoràmica 360º es genera a partir del cosit entre les imatges provinents de 4 càmeres situades al llarg del vehicle, que el dispositiu incrustat ha de processar a temps real per tal de donar suport visual al conductor de l’automòbil.

El problema principal que es va trobar en aquest projecte és que el dispositiu que suportava el programa no tenia prou capacitat computacional com per donar resposta a temps real a l’algorisme de fusió de les imatges.

Davant d'aquest problema es va plantejar la possibilitat d'emprar algun altre dispositiu més potent per tal de millorar els resultats obtingut. Com que l'algorisme de cosit de imatges consisteix en un seguit d'operacions aritmètiques amb matrius, es va pensar que era una bona idea l'ús de una GPU per paral·lelitzar càlculs i augmentar-ne la velocitat.

A partir d'aquí, i aprofitant que la universitat disposava de la placa de NVIDIA, Jetson TK1, i ningú a la universitat l'havia provat com ha eina de desenvolupament, no m'acaba d'agradar el paràgraf.

Desenvolupament del treball

Per tal d'utilitzar la placa Jetson com entorn de desenvolupament i poder executar-hi el software d'Arcol s'ha instal·lat un sistema operatiu amb kernel de Linux (Ubuntu 14.04 LTS). En principi és el que ve pre-instal·lat al dispositiu, però per tal d'obtenir el Know-how de com restaurar el sistema i instal·lar-hi altres llibreries necessàries, s'ha fet un flash del sistema operatiu amb un pack especial per a desenvolupadors proporcionat per NVIDIA. (Jetson Pack).

Un cop configurada la principal eina de treball, també s'ha cregut oportú tenir un entorn de treball més flexible de cara al desenvolupament d'aplicacions, en el qual es poden donar diferents escenaris.

En el primer, i més simple, el mateix dispositiu incrustat serveix com a plataforma de desenvolupament, és a dir, no es requereix de més maquinari extern per programar, compilar i executar el software desenvolupat.

En el segon, la compilació del codi es dur a terme des d’un PC amb una capacitat de càlcul molt més elevada (compilació creuada), cosa que permet treballar amb projectes molt més grans. Per dur a terme la cross-compilació però, s’han instal·lat un seguit de llibreries necessàries per que els softwares compilats siguin compatibles amb l’arquitectura del dispositiu incrustat.

Host – Host

Un cop configurat l’entorn de desenvolupament, s’ha compilat i instal·lat el software.d'Arcol a la placa Jetson.

Després d’instal·lar el software d’Arcol a la placa Jetson s’ha vist que el funcionament sense cap modificació en el codi ja és molt millor que en el hardware original, la placa Xilinx ETC, que funciona amb un processador ARM de 1Gb de memòria Ram.

Tot i la millora, el funcionament del programa encara no és l’adequat, ja que el vídeo sortint mostra al voltant d’una imatge per segon.

A partir d’aquí, s’ha intentat optimitzar el codi de dues maneres; millorant l’algorisme per aprofitar la CPU i, utilitzar la GPU per paral·lelitzar càlculs en les funcions més lentes.

Per fer-ho s'ha fet un "profiling" per determinar quins eren les funcions on es perdia més temps de càlcul.

Degut a la complexitat de tot el projecte, per fer un "profiling" el més real acurat, s'ha extret el la part del software on es fa el cosit de les imatges i s'ha posat en un programa més simple.

En aquest programa s'ha mesurat els temps

Introducció deep learnig??

PROVA, PROVA

PROVA 2, PROVA 2.

Resumen

Dedication: A Dedication page may be included in your thesis just before the Acknowledgments page, but it is not a requirement.

Que és això????

Agraïments

ALBERT GIL!!!!! ALBERT GIL!!!!!!!!

JOSEP RAMON!!!! JOSEP RAMON!!!!

Historial de reversions i registre d'aprovació

|  |  |  |
| --- | --- | --- |
| **Revisió** | **Data** | **Motiu** |
| 0 | 10/05/2016 | Creació del document |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

DOCUMENT DISTRIBUTION LIST

|  |  |
| --- | --- |
| **Nom** | **e-mail** |
| Ignasi Bernadas | ibernadasl.iespf@gmail.com |
| Josep Ramon Casas | josep.ramon.casas@upc.edu |
| Albert Gil | albert.gil@upc.edu |
|  |  |
|  |  |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Escrit per: | | Revisat i aprovat per: | |
| Data | 10/05/2015 | Data |  |
| Nom | Ignasi Bernadas Leal | Nom | Josep Ramon Casas |
| Posició | Autor del projecte | Posició | Supervisor |

Índex

Abstract 2

Resum 2

Resumen 2

Agraïments 2

Historial de reversions i registre d'aprovació 2

Índex 2

Índex de imatges 2

Índex de taules: 2

1. Introducció 2

1.1. Objectius 2

1.2. Requisits i especificacions 2

1.3. Context 2

1.4. Pla de treball 2

2. Tecnologia emprada: 2

2.1. Dispositius Embedded 2

2.2. Computació per GPU 2

3. Desenvolupament del projecte 2

3.1. Configuració entorn de treball 2

3.1.1. NVIDAI Jetson TK1 2

3.1.2. Host (PC) 2

3.1.3. Entorn de compilació i cross-compilació 2

3.1.3.1. ZLIB 2

3.1.3.2. PNG 2

3.1.3.3. TIFF 2

3.1.3.4. JPEG 2

3.1.3.5. Qt 2

3.1.3.6. CUDA 2

3.1.3.7. OpenCv 2

3.2. Porting del Software d’Arcol a la placa Jetson 2

3.2.1. Compilació en el propi dispositiu 2

3.2.2. Compilació creuada 2

3.2.3. Anàlisi i optimització del codi amb CUDA 2

3.2.3.1. Funcions a optimitzar **¡Error! Marcador no definido.**

3.2.3.2. Possibles millores 2

3.3. Deep Learning 2

4. Resultats 2

4.1. Software d’Arcol 2

4.2. Deep Learning 2

5. Pressupost ??????? 2

6. Conclusions 2

Bibliografia: 2

Annexos 2

Glossari 2

Índex de imatges

Each figure in the thesis must be listed in the “List of Figures” and each must be given a page number for its easy location.

Índex de taules:

Each table in the thesis must be listed in the “List of Tables” and each must be given a page number for its easy location.

# Introducció

An Introduction that clearly states the rationale of the thesis that includes:

1. Statement of purpose (objectives).
2. Requirements and specifications.
3. Methods and procedures, citing if this work is a continuation of another project or it uses applications, algorithms, software or hardware previously developed by other authors.
4. Work plan with tasks, milestones and a Gantt diagram.
5. Description of the deviations from the initial plan and incidences that may have occurred.

## Objectius

El primer objectiu que es va proposar per aquest treball, va ser millorar i optimitzar el funcionament d’un software d’assistència al conductor desenvolupat anteriorment en un projecte de col·laboració entre la Universitat Politècnica de Catalunya i l’empresa d’accessoris per a l’automoció Arcol.

El software original va ser dissenyat i implementat sobre un hardware “embedded”, fabricat per Xilinx. Aquest hardware havia de processar les imatges procedents de 4 càmeres situades en diferents punts d’un vehicle gran, com podria ser un autobús, i fer-ne un cosit a temps real. El cosit proporciona una visió 360º del vehicle, ajudant així al conductor a tenir una visibilitat completa al seu voltant i a millorar la visió en angles morts del retrovisor.

Malauradament, aquest projecte mai va acabar de tenir èxit ja que el hardware no era capaç de realitzar el cosit a temps real i, al final, el que havia de ser una imatge 360º a temps real, va acabar sent una senyal de vídeo que mostrava un frame cada 3 segons.

Com que el principal problema d’aquest projecte va ser que el dispositiu que havia de suportar el programa no era l’adequat, es va creure que utilitzant-ne un amb millors prestacions es podria obtenir un millor resultat.

respecte el que tenia en hardware pel que es va desenvolupar.

Aprendre el funcionament del dispositiu NVIDIA Jetson TK1

Instal·lar el software d’Arcol a la placa Jetson

## Requisits i especificacions

Amb aquest treball s’espera tenir implementat el software d’Arcol sobre el dispositiu incrustat Jetson TK1 de NVIDIA.

Obtenir una millora en la processat de la senyal per tal d’obtenir un resultat final amb almenys entre 5 i 10 frames de vídeo per segon.

## Context

## Pla de treball

Per dur a terme el treball, s’ha seguit el següent pla de treball:

La feina ha estat repartida en quatre grans paquets de treball, dintre dels quals hi ha un seguit de tasques a realitzar.

|  |  |  |
| --- | --- | --- |
| Paquet de treball 1 | | |
| Major constituent: aprenentatge de la plataforma Nvidia Jetson i del seu entorn de desevolupament | | |
| Petita descripció: Familiarització posada a punt de l’entorn de treball (NVidia Jetson TK1). | Inici: 29/02/2016  Fi: 11/03/2016 | |
| Tasca interna T1: Posar en marxa el dispositiu  Tasca interna T2: Fer un “backup” del sistema operatiu per tal de poder recuperar l’estat inicial en cas que fos necessari.  Tasca interna T3: Fer un primer programa de prova “Hello world” | Entregables:  HelloWorld | Dates: |

|  |  |  |
| --- | --- | --- |
| Paquet de treball 2 | | |
| Major constituent: Creació d’un entorn de treball i Desenvolupament d’una app per a Jetson | | |
| Petita descripció: Crear un entorn on desenvolupar els primer executables de prova. Les aplicacions s’han de poder compilar i executar des d’un PC i a la pròpia Jetson. (cross-compilació) | Inici: 11/03/2016  Fi: 03/05/2016 | |
| Tasca interna T1: Instal·lació de les llibreries OpenCV, Qt, CUDA...  Tasca interna T2: Fer un programa senzill utilitzant la arquitectura CUDA (Utilitzar la GPU en lloc de la CPU) “Hello CUDA”  Tasca interna T3: Fer un programa senzill utilitzant les llibreries de OpenCV “Hello OpenCV”  Tasca interna T4: Fer un programa senzill utilitzant Qt: “Hello Qt”  Tasca interna T5: Integració de totes les llibreries de suport en un sol programa: “Hello Qt, OpenCV and CUDA”  Tasca interna T6: Compilar i executar els programes realitzats tant en una *work station* com en la placa Jetson. | Entregables:  HelloCUDA  HelloOpenCv  HelloOpenCvCUDA  HelloQt  HelloAll  HelloCrossCUDA  HelloCrossOpenCv  HelloCrossOpenCvCUDA  HelloCrossQt  HelloCrossAll | Dates: |

|  |  |  |
| --- | --- | --- |
| Paquet de treball 3 | | |
| Major constituent: Porting d’Arcol a la placa Jetson | | |
| Petita descripció: Realitzar el canvi de plataforma del programa d’Arcol de la placa Xillinx's Zynq a la NVidia Jetson TK1 | Inici: 18/04/2016  Fi: | |
| Tasca interna T1: Aconseguir compilar i executar el programa sobre la placa Jetson  Tasca interna T2: Cross-compilar el sofware d’Arcol des de un PC 64 bits i executar-lo sobre la placa Jetson  Tasca interna T3: Optimitzar el funcionament del programa. | Entregables: | Dates: |

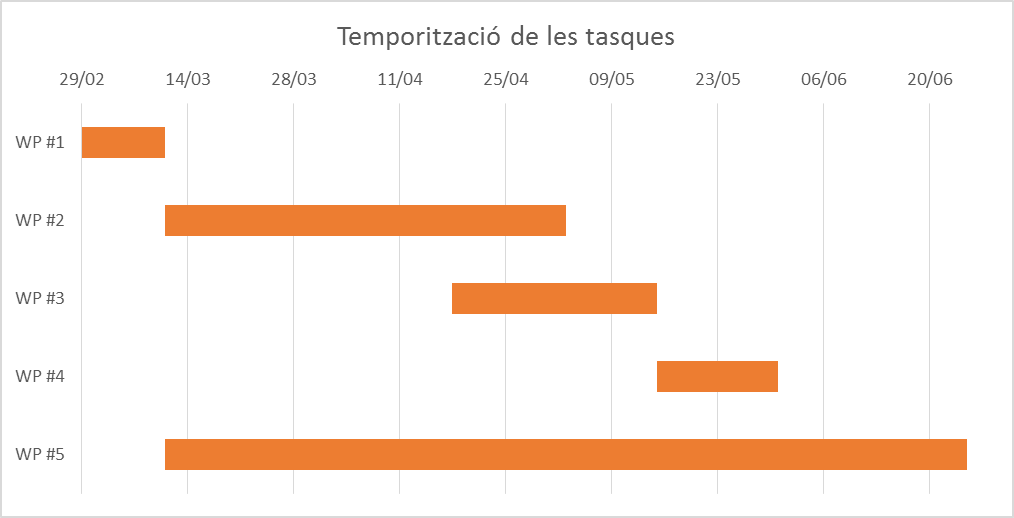
|  |  |  |
| --- | --- | --- |
| Paquet de treball 4 | | |
| Major constituent: Explorar noves tècniques | | |
| Petita descripció: Fer recerca sobre els possibles nous usos que se li podrien donar a la placa Jetson. | Inici:  Fi: | |
| Tasca interna T1: Buscar informació sobre *Deep Learning* i plataformes orientades a l’ús d’aquestes tècniques sobre GPU  Tasca interna T2: Desenvolupar un programa amb els coneixements adquirits. | Entregables: | Dates: |

|  |  |  |
| --- | --- | --- |
| Paquet de treball 5 | | |
| Major constituent: Memòria escrita | | |
| Petita descripció: Redactar tot el procés seguit en la elaboració del treball, així com els coneixements adquirits, els resultats, conclusions, etc. | Inic: 11/03/2016  Fi: 25/06/2016 | |
| Tasca interna T1: Redactar la memòria  Tasca interna T2: Prepara la exposició del treball | Entregables:  Critical Review  Memòria del treball  Presentació final | Dates: |

**Fites**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| WP# | Task# | Títol | Fita | Data finalització |
| 1 | 1 | Posar en marxa el dispositiu | Engegar la Jetson i veure que funciona | 02/03/2016 |
| 1 | 2 | Fer un “backup” del sistema | Tenir una còpia del sistema operatiu per si s’ha de restaurar el sistema | 04/03/2016 |
| 1 | 3 | “Hello world” | Ser capaç d’executar un programa simple en el dispositiu | 11/03/2016 |
| 2 | 1 | Instal·lació de les llibreries OpenCV, Qt, CUDA | Tenir totes les llibreries necessàries per al projecte instal·lades correctament | 13/03/2016 |
| 2 | 2 | “Hello CUDA” | Saber realitzar un programa utilitzant les llibreries CUDA | 18/03/2016 |
| 2 | 3 | “Hello OpenCV” | Saber realitzar un programa utilitzant les llibreries OpenCV | 25/03/2016 |
| 2 | 4 | “Hello Qt” | Saber realitzar un programa utilitzant les llibreries Qt | 31/03/2016 |
| 2 | 5 | “Hello Qt, OpenCV and CUDA” | Saber realitzar un programa integrant totes les llibreries esmentades (CUDA, OpenCv i Qt) | 18/04/2016 |
| 2 | 6 | Desenvolupament dual i creuat | Poder compilar i executar des d’una workstation i a la pròpia Jetson els programes de “Hello X” fets | 03/05/2016 |
| 3 | 1 | Porting d’Arcol a la placa Jetson | Aconseguir fer funcionar el programa d’arcol a la placa Jetson | 25/04/2016 |
| 3 | 2 | Optimització d’Arcol | Fer que el programa funcioni més ràpidament que sobre la plataforma Xilinx Zynq | 15/05/2016 |
| 4 | 1 | Recerca sobre *Deep Learning* | Obtenir informació sobre *Deep Learning* i les possibles aplicacions que podria tenir en la placa Jetson | 22/05/2016 |
| 4 | 2 | Desenvolupar un programa amb *Deep Learning* | Reflectir els coneixement adquirit amb una aplicació concreta. | 31/05/2016 |
| 5 | 1 | Redactar la memòria | Tenir un informe complert de tot el treball realitzat | 25/06/2016 |
| 5 | 2 | Prepara la exposició del treball | Fer l’exposició final del TFG | 08/07/2016 |

### Diagrama de Gantt



# Tecnologia emprada:

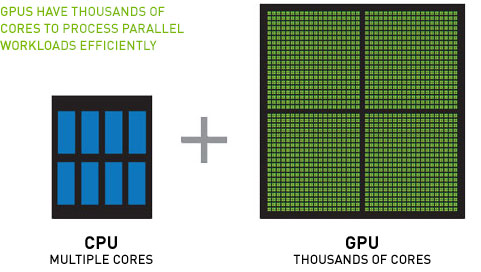
## Dispositius incrustats o “Embedded”

Els dispositius incrustats o “embedded”, son petites computadores dissenyades en general per dur a terme una funció força específica. N’existeixen per una gran varietat d’usos i, és per això que en podem trobar en

A diferència del ordinadors convencionals ETC explicació

## Computació per GPU

Les GPUs (Unitats gràfiques de processat) són microprocessadors emprats normalment per al processat de gràfics i altres operacions de coma flotant. Estan pensades principalment per alleugerir la càrrega del processador central (CPU) en programes amb gràfics potents com videojocs i, permetre així, que aquest pugui dedicar recursos a altres tipus de càlculs.



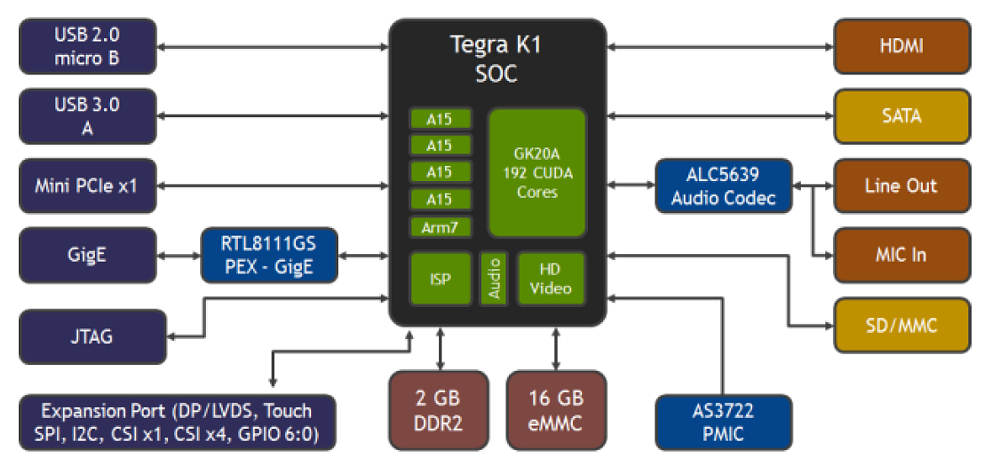
<http://www.nvidia.com/object/gpu-accelerated-computing.html>

# Desenvolupament del projecte

## Configuració entorn de treball

### NVIDAI Jetson TK1

#### Característiques



<http://hackerboards.com/nvidia-jetson-tk1-most-advanced-hacker-sbc/>

La placa Jetson TK1 és un dispositiu incrustat que té com a motor principal el processador Tegra de Nvidia K1. Aquest processador compta amb una arquitectura formada per una CPU ARM de quatre nuclis Cortex A15 i una GPU Nvidia Kepler de 192 nuclis GK20a, que és capaç de treballar a més de 300 gigaflops, tot això acompanyat amb 2 GB de memòria DRAM DDR3L 933MHz i més de 16 GB d'emmagatzematge EMMC.

A més de la gran capacitat de processat també conté una sèrie de ports d'entrada i sortida: USB, Ethernet, i JTAG, HDMI, port sèrie, SD / MMC, Mini-PCIe, SATA, àudio (una línia de sortida i una entrada de micròfon), i un port d'expansió 125 pins. Aquest port d'expansió es pot utilitzar per connectar altres perifèrics com càmeres, pantalles LCD i pantalles tàctils, etc.

Tot això fa que la Jetson TK1 sigui una dels sistemes desenvolupament integrat més ràpid i econòmic del mercat.

#### Configuració

La placa Jetson porta de sèrie un sistema operatiu Ubuntu instal·lada. Es tracta de Linux4Tegra, que és bàsicament un sistema Ubuntu 14.04 amb els drivers necessaris pre-instal·lats.

Partint de que ja hi ha un sistema operatiu, la feina de configuració és més senzilla. En la placa utilitzada per el projecte d’Arcol, la Xilnx, no hi havia la possibilitat de tenir un sistema operatiu instal·lat en el propi dispositiu, amb la qual cosa, totes les llibreries necessàries per a l’execució del software es van haver de compilar des d’una estació de treball exterior (PC o servidor), amb els compiladors específics, i llavors copiar-les.

Malgrat venir configurada amb un sistema operatiu, s’ha cregut convenient tornar-lo a instal·lar per tenir clar el procediment en cas que fos necessari. Així doncs, s’ha utilitzat un pac de software proporcionat per NVidia (JetPack) per instal·lar de nou el SO i altres eines que es mostren a continuació:

* Llibraries
  + CUDA 6.0
  + VisionWorks
  + OpenCV4Tegra (Llibreries de OpenCV especials per la Jetson TK1)
  + cuDNN v2 for Jetson TK1 Developer Kit.
* Eines per al desenvolupament
  + Tegra Graphics Debugger
  + [Tegra System Profiler](https://developer.nvidia.com/tegra-system-profiler)
  + PerfKit
* Mostres d’exemple
  + NVIDIA GameWorks OpenGL Samples

A part d’això el pac també inclou kit d’eines per el desenvolupament creuat que s’instal·len en el Host.

* CUDA Toolkit for Host (Ubuntu with cross-development support)

El Jetpack es pot descarregar en el següent enllaç. [Download JetPack for LT4](https://developer.nvidia.com/embedded/downloads#?tx=$software,l4t-tk1).

Per instal·lar el software s’han seguit les instruccions de la guia d’instal·lació proporcionada a la mateixa web.

La descàrrega i instal·lació s’han de dur a terme des d’un PC al que anomenarem Host d’ara en endavant, el qual es connecta a la placa a través d’un cable USB per fer la instal·lació.

A part del JetPack, per tal de poder compilar i executar el software d’Arcol a la placa Jetson, també s’ha hagut d’instal·lar les llibreries de Qt, ja que també són utilitzades i no les proporciona el pac.

S’ha instal·lat la versió de Qt 4.8, que és la que es va fer servir en el seu dia en la realització del projecte.

sudo apt-get install qt4-dev-tools libqt4-dev libqt4-core libqt4-gui

### Host (PC)

El host utilitzat en aquest treball ha sigut un ordinador Alienware bla, bla, bla. Degut a un tema de compatibilitat amb el JetPack de NVIDIA el sistema operatiu que s’ha instal·lat en el host és un Ubuntu 14.04 LTS, ja que és l’únic compatible amb aquest software.

Tot seguit, com s’ha explicat en el punt anterior, s’ha instal·lat el JetPack proporcionat per NVIDIA, el qual conté els compiladors i les llibreries de CUDA necessàries per poder compilar aplicacions per la placa Jetson.

També s’ha instal·lat Git per descarregar el codi font de les llibreries. No és estrictament necessari ja que es pot descarregar el codi amb un fitxer .tar, però és més còmode.

imatge@imatge:~/ibernadas$ sudo apt-get install git

### Entorn de cross-compilació

#### Eines de compilació

##### Make

Explicació ./configure i flags

CMake

CMake ( abreviatura per "cross platform make") és una eina multiplataforma i opensource dissenyada per a construir, provar i empaquetar software. CMake s'utilitza per controlar el procés de compilació de programari utilitzant una plataforma simple i arxius de configuració del compilador independents, els quals s’utilitzen per generar makefiles nadius.

S’ha instal·lat Cmake ja que algunes de les llibreries que s’han instal·lat l’utilitzen com a eina de compilació.

sudo apt-get install cmake

Scons

sudo apt-get install scons

#### Llibreries necessàries per compilar Arcol

El software d’Arcol fa servir algunes llibreries que no estan instal·lades de sèrie en el sistema operatiu Ubuntu. Principalment són dues les llibreries més importants que s’han d’instal·lar.

Per una banda, el programa utilitza una interfície gràfica basada en Qt bla, bla bla... Per altra banda, s’utilitza OpenCV, que és una eina molt potent utilitzada en el processat de imatge.

Per cross-compilar ambdues llibreries però, es necessari fer alguns passos previs. OpenCV i Qt utilitzen llibreries externes com Zlib, Png, Tiff, Jepg per poder llegir aquests formats o, en el cas d’OpenCV també utilitza CUDA, que és la que permet utilitzar funcions per la GPU. OpenCV també ofereix la possibilitat de incloure Qt en la seva compilació, donant-li així algunes funcionalitats extres. En el projecte d’Arcol no s’utilitzen funcions d’OpencCV amb funcionalitats de Qt però igualment s’ha decidit compilar amb Qt per si acaba sent necessari. Així doncs, l’ordre que s’ha seguit per instal·lar les diferents llibreries és el plasmat en aquest document.

Per instal·lar totes les llibreries necessàries per a la cross-compilació, hem definit una carpeta d’instal·lació.

imatge@imatge:~/ibernadas$ sudo mkdir /usr/local/arm

imatge@imatge:~/ibernadas$ export ARM\_SDK=/usr/local/arm

Com que la carpeta on s’instal·laran totes les llibreries està en un path del sistema, per evitar haver de donar permisos d’administrador cada cop que es vulgui fer un canvi, s’ha canviat els permisos d’accés amb la següent comanda:

Escriure la comanda que no macuerdo

Un cop fet això s’ha procedit a compilar les diferents llibreries.

ZLIB

Primer s’ha instal·lat Zlib, ja que a part de ser necessària per compilar OpenCV i Qt, també és pre-requisit per compilar PNG, TIFF i JPEG??.

S’ha baixat el codi font de la pàgina de GitHub i s’ha compilat la versió més recent amb el compilador arm-linux-gnueabihf-gcc.

imatge@imatge:~/ibernadas$ git clone https://github.com/madler/zlib.git

imatge@imatge:~/ibernadas/zlib$ cd zlib

imatge@imatge:~/ibernadas/zlib$ git checkout v1.2.8

imatge@imatge:~/ibernadas/zlib$ ./configure CC=arm-linux-gnueabihf-gcc \

--prefix=$ARM\_SDK

imatge@imatge:~/ibernadas/zlib/zlib$ make && make install

PNG

vbfv

imatge@imatge:~/ibernadas/libpng$ ./configure \

--host=arm-linux-gnueabihf \

CC=arm-linux-gnueabihf-gcc \

CPPFLAGS=-I$ARM\_SDK/include \

LDFLAGS=-L$ARM\_SDK/lib \

--prefix=$ARM\_SDK \

--enable-shared=yes

imatge@imatge:~/ibernadas/libpng$ sudo make && make install

TIFF

imatge@imatge:~/ibernadas$ git clone https://github.com/LuaDist/libtiff.git

imatge@imatge:~/ibernadas$ cd libtiff/

imatge@imatge:~/ibernadas/libtiff$ CC=arm-linux-gnueabihf-gcc \

CXX=arm-linux-gnueabihf-g++ \

CPPFLAGS="-I$ARM\_SDK/include \

CXXFLAGS="-I$ARM\_SDK/include \

CFLAGS="-I$ARM\_SDK/include \

LDFLAGS=-L$ARM\_SDK/lib -mfpu=neon"

./configure \

--prefix=$ARM\_SDK \

--enable-shared=yes \

--host=arm-linux-gnueabihf \

--enable-cxx \

--disable-jbig \

--disable-lzma

imatge@imatge:~/ibernadas/libtiff$ sudo make && make install

JPEG

imatge@imatge:~/ibernadas$ git clone https://github.com/LuaDist/libjpeg.git

imatge@imatge:~/ibernadas/libjpeg$ ./configure \

CC=arm-linux-gnueabihf-gcc \

CPPFLAGS=-I$ARM\_SDK/include \

LDFLAGS=-L$ARM\_SDK/lib \

--prefix=$ARM\_SDK \

--host=arm-linux-gnueabihf \

--enable-shared=yes

imatge@imatge-Alienware-14:~/ibernadas/libjpeg$ sudo make && make install

Qt

Per instal·lar Qt hi ha hagut més dificultats. Primer de tot s’ha hagut d’editar el fitxer qt-4.8.2/mkspecs/qws/linux-arm-gnueabi-g++/qmake.conf, canviant el compilador que hi va per defecte “arm-linux-none-\*” per “arm-linux-gnueabihf-g\*”.

#

# qmake configuration for building with arm-none-linux-gnueabi-g++

#

include(../../common/linux.conf)

include(../../common/gcc-base-unix.conf)

include(../../common/g++-unix.conf)

include(../../common/qws.conf)

# modifications to g++.conf

QMAKE\_CC = arm-linux-gnueabihf-gcc

QMAKE\_CXX = arm-linux-gnueabihf-g++

QMAKE\_LINK = arm-linux-gnueabihf-g++

QMAKE\_LINK\_SHLIB = arm-linux-gnueabihf-g++

# modifications to linux.conf

QMAKE\_AR = arm-linux-gnueabihf-ar cqs

QMAKE\_OBJCOPY = arm-linux-gnueabihf-objcopy

QMAKE\_STRIP = arm-linux-gnueabihf-strip

QMAKE\_CFLAGS="${CROSS\_COMPILE\_CFLAGS} -I$(ARM\_SDK)/include -DZYNQ"

QMAKE\_CXXFLAGS="${CROSS\_COMPILE\_CPPFLAGS} -I$(ARM\_SDK)/include -DZYNQ"

QMAKE\_LFLAGS="-L$(ARM\_SDK)/lib"

load(qt\_config)

Embedded (revisar i explicar els problemes de cross-compilació trobats)

imatge@imatge:~/ibernadas$ gunzip qt-everywhere-opensource-src-%VERSION%.tar.gz

imatge@imatge:~/ibernadas$ tar xvf qt-everywhere-opensource-src-%VERSION%.tar

imatge@imatge:~/ibernadas$ cd qt

imatge@imatge:~/ibernadas/qt$ ./configure \

-embedded arm \

-xplatform qws/linux-arm-gnueabi-g++ \

-little-endian -opensource \

-host-little-endian \

-qtlibinfix -UPC-SDK-EMB \

-prefix $ARM\_SDK \

-no-webkit \

-no-phonon \

-no-dbus \

-no-opengl \

-no-pch \

-nomake demos \

-nomake examples \

-system-zlib \

-system-libpng \

-system-libtiff \

-system-libjpeg \

-qt-gfx-transformed \

-qt-gfx-qvfb \

-qt-kbd-qvfb \

-qt-mouse-qvfb -v

imatge@imatge-Alienware-14:~/ibernadas/qt make -j6

imatge@imatge-Alienware-14:~/ibernadas/qt sudo make install

CUDA

Comprovar que el JetPack instal·la les llibreries de CUDA. Em va semblar veure-ho a /usr/arm-linux-gnueabihf/lib.

OpenCv

Un cop instal·lades totes les llibreries necessàries, s’ha instal·lat OpenCv. S’ha baixat el codi font de GitHub i s’ha utilitzat la versió 4.12 del software.

imatge@imatge:~/ibernadas$ git clone https://github.com/Itseez/opencv.git

imatge@imatge:~/ibernadas/opencv$ git checkout 2.4.12

imatge@imatge:~/ibernadas/opencv$ mkdir build && cd build

To seguit, en el fitxer arm-gnueabi.toolchain.cmake s’ha canviat la versió del compilador de 4.6 a 4.8.

imatge@imatge-Alienware-14:~/ibernadas/opencv/build$ cmake \

-DCMAKE\_INSTALL\_PREFIX=$ARM\_SDK \

-DCMAKE\_TOOLCHAIN\_FILE=../platforms/linux/arm-gnueabi.toolchain.cmake \

-DBUILD\_SHARED\_LIBS=ON \

-D BUILD\_opencv\_nonfree=ON \

-DCMAKE\_BUILD\_TYPE=RELEASE \

-DWITH\_OPENCL=OFF \

-DEIGEN\_INCLUDE\_PATH=$ARM\_SDK/include/eigen \

-D ENABLE\_NEON=ON \

-D ENABLE\_VFPV3=ON \

-DWITH\_QT=ON \

-DWITH\_CUDA=ON \

-DWITH\_JASPER=OFF \

-DZLIB\_INCLUDE\_DIR=$ARM\_SDK/include \

-DTIFF\_INCLUDE\_DIR=$ARM\_SDK/include \

-DTIFF\_LIBRARY=$ARM\_SDK/lib/libtiff.so \

-DJPEG\_INCLUDE\_DIR=$ARM\_SDK/include \

-DJPEG\_LIBRARY=$ARM\_SDK/lib/libjpeg.so \

-DPNG\_PNG\_INCLUDE\_DIR=$ARM\_SDK/include \

-DPNG\_LIBRARY=$ARM\_SDK/lib/libpng.so \

-DCMAKE\_EXE\_LINKER\_FLAGS="-L$ARM\_SDK/lib -lQtGui -lQtNetwork -lQtTest -lQtCore -ltiff -ljpeg -lpng -lz" \

-DENABLE\_PRECOMPILED\_HEADERS=OFF \

-DWITH\_V4L=OFF \

-DWITH\_FFMPEG=OFF \

-DWITH\_GSTREAMER=OFF \

-DBUILD\_PERF\_TESTS=OFF \

-DBUILD\_TESTS=OFF \

-DBUILD\_opencv\_python=OFF \

-DBUILD\_opencv\_apps=OFF \

-DBUILD\_ZLIB=OFF \

-DBUILD\_TIFF=OFF \

-DBUILD\_JPEG=OFF \

-DBUILD\_PNG=OFF \

-DCUDA\_ARCH\_BIN="3.2" \

-DCUDA\_ARCH\_PTX="" \

-DQT\_QMAKE\_EXECUTABLE=$ARM\_SDK/bin/qmake \

-DQT\_QTCORE\_INCLUDE\_DIR=$ARM\_SDK/include/QtCore \

-DQT\_QTCORE\_LIBRARY=$ARM\_SDK/lib/libQtCore-UPC-SDK-EMB.so \

-DQT\_QTCORE\_LIBRARY\_RELEASE=$ARM\_SDK/lib/libQtCore-UPC-SDK-EMB.so \

-DQT\_MOC\_EXECUTABLE=$ARM\_SDK/bin/moc -DQT\_UIC\_EXECUTABLE=$ARM\_SDK/bin/uic \

-DQT\_RCC\_EXECUTABLE=$ARM\_SDK/bin/rcc -DCMAKE\_AR=/usr/bin/arm-linux-gnueabihf-ar -DQT\_QTGUI\_LIBRARY=$ARM\_SDK/lib/libQtGui-UPC-SDK-EMB.so \

-DQT\_QTGUI\_INCLUDE\_DIR=$ARM\_SDK/include/QtGui \

-DQT\_QTGUI\_LIBRARY\_RELEASE=$ARM\_SDK/lib/libQtGui-UPC-SDK-EMB.so \

-DQT\_QTGUI\_LIBRARY\_DEBUG=$ARM\_SDK/lib/libQtGui-UPC-SDK-EMB.so \

-DQT\_QTTEST\_LIBRARY=$ARM\_SDK/lib/libQtTest-UPC-SDK-EMB.so \

-DQT\_QTTEST\_LIBRARY\_RELEASE=$ARM\_SDK/lib/libQtTest-UPC-SDK-EMB.so \

-DQT\_QTTEST\_LIBRARY\_DEBUG=$ARM\_SDK/lib/libQtTest-UPC-SDK-EMB.so \

-DQT\_QTTEST\_INCLUDE\_DIR=$ARM\_SDK/include/QtTest ..

## Porting del Software d’Arcol a la placa Jetson

Per fer la compilació del software d’Arcol abans s’ha descarregat el codi font del servidor del Grup de Processar de Imatge de la UPC.

Commanda!!!!!!!!!

### Compilació i execució en el propi dispositiu

La compilació del software d’Arcol en la placa Jetson no ha provocat gaires complicacions.

El software ve preparat amb una sèrie d’arxius de configuració per la compilació de Scons. Gràcies a això només s’ha hagut de canviar els paths d’algunes de les llibreries i s’han deshabilitat algunes funcionalitats no compatibles amb el sistema.

Especificar els canvis realitzats (Repassar amb l’Albert)

Després de canviar els arxius necessaris s’ha cridat la següent comanda:

ubuntu@ubuntu:~ibernadas/arcol$ scons bin/arcol-gui-calibration

Amb això, s’ha creat un executable dins de la carpeta bin/ amb el nom “arcol-gui-calibration”.

Per executar el programa s’ha hagut de fer un pas més. Com que el software necessita una senyal de vídeo provinent de les quatre càmeres situades a les cantonades d’un autobús i no es disposa d’un autobús ni de les quatre càmeres, s’ha utilitzat una senyal pre-gravada per fer l’execució.

El fet de utilitzar una senyal gravada ha generat alguns problemes. Primer de tot, el vídeo de quatre càmeres ocupa massa com per guardar-ho a la memòria interna de la placa Jetson, que només té 16 Gb. Per solucionar això, s’han copiat els vídeos en una targeta de memòria SD.

Per altra banda, el fet d’haver de llegir de la targeta SD ha provocat que el software funciones a una velocitat inferior de la que és capaç de processar. Més endavant en el treball s’explica amb detall aquest problema.

Deixant de banda això, s’ha executat el programa amb la següent comanda:

ubuntu@ubuntu:~ibernadas/arcol$ bin/arcol-gui-calibration /media/ubuntu/3931-3032/arcol/calibration/2015-07-16/2015-07-16-calib/ /media/ubuntu/3931-3032/arcol/calibration/2015-07-16/2015-07-16-stitching/

La comanda executa el binari creat anteriorment, que correspon a una interfície de calibració. Aquesta interfície permet navegar i veure el vídeo proporcionat per les 4 càmeres i, tot seguit, marcant una sèrie de punts en les imatges fer una calibració del sistema.

Per fer la calibració són necessàries unes imatges especials, per això en la comanda d’execució del programa es passen com a paràmetres la carpeta amb les imatges de calibració primer, i les imatges per processar després.

Aquí potser hi aniria bé una captura de pantalla de la Gui de Calibració i una altre de la pantalla amb l’Stitching

### Compilació creuada

S’ha de provar des del PC

### Anàlisi i optimització del codi

L’algorisme de cosit o “stitching” consta de dues parts principals. En la primera, s’agafen les imatges provinents de les 4 càmeres situades al voltant del vehicle i, mitjançant una calibració feta prèviament es re-mapegen per tal de poder-les superposar.

Tot seguit, utilitzant una matriu amb un seguit de pesos per cada bit de cada imatge, es superposen les quatre imatges per obtenir-ne una de sola, que correspon a la visió 360º.

Explicar millor i posar imatge

#### PROFILING

Per dur a terme un bon anàlisi dels temps d’execució que consumeix cada funció del programa s’ha fet un perfilat.

El perfilat es pot obtenir afegint el flag “-pg” en la compilació. Amb aquest flag habilitat i una eina de perfilat, com per exemple Gprof, es pot obtenir un anàlisi acurat de quants cops es crida una funció al llarg de l’execució del software, quan temps dura cada crida, o quin és el temps total que s’ha estat executant una funció.

Així doncs, per utilitzar-lo en un programa senzill només cal seguir els següents passos:

Instal·lar Gprof en cas de no tenir-lo instal·lat:

sudo apt-get install gprof

Compilar el programa amb l’opció de profiling habilitada

gcc -Wall -pg test\_gprof.c test\_gprof\_new.c -o test\_gprof

Executar el programa per tal d’obtenir dades.

./test\_gprof

I utilitzant el programa Gprof extreure els resultats.

gprof test\_gprof gmon.out > analysis.txt

<http://www.thegeekstuff.com/2012/08/gprof-tutorial/>

En el cas d’Arcol, per habilitar l’opció de profiling, com que la compilació es fa a traves de Scons, en comptes del flag “-pg” hem d’afegir al final de la comanda “profile=true”. Amb aquesta opció habilitada, el flag “-pg” serà afegit en la compilació de totes les classes.

Així doncs, la comanda per habilitar el perfilat en el software d’Arcol és la següent:

scons bin/asfsdflsnfksdfj profile=true

En el programa d’Arcol però, la complexitat del software és molt gran i ASDakjdfADS.

Per tal d’obtenir un millor resultat en perfilat, s’ha aïllat la part de l’algorisme de cosit de les imatges en un programa més senzill, sense interfície gràfica i, utilitzant únicament les funcions imprescindibles. (Veure annex N)

#### Optimització per CPU

Per tal d’intentar millorar el rendiment del programa s’ha intentat canviar l’algorisme del blending.

L’algorisme de blending estava implementat amb dos bucles for, que recorrien un a un, tots els píxels de les imatges de les quatre càmeres, i els multiplicaven per els pesos corresponents, per després sumar-los tots en la imatge final de sortida.

Càmera 1

Càmera 2

Càmera 3

Càmera 4

Matriu de pesos 1

Matriu de pesos 2

Matriu de pesos 3

Matriu de pesos 4

Imatge 360º

**x**

**x**

**x**

**x**

Com que, tant les imatges de les 4 càmeres, com les matrius de pesos, en el programa són objectes cv::Mat i, les llibreries d’OpenCv proporcionen funcions per a la multiplicació i la suma de matrius, s’ha provat de fer el següent.

En comptes d’utilitzar un for que recorri píxel per píxel cada imatge, s’ha fet la multiplicació de les imatges per les matrius de pesos utilitzant la funció cv::multiply i, posteriorment cv::add per fer la suma de les imatges ponderades.

El codi ha quedat de la següent manera:

void arcolcv::CustomBlender::blend(std::vector<cv::Mat\*> &images, cv::Mat &out){

out.create(this->weights[0].size(), images[0]->type());

out.setTo(cv::Scalar(0,0,0));

for (size\_t i = 0; i < images.size(); ++i){

cv::Mat temp;

temp.create(this->weights[0].size(), images[0]->type());

temp.setTo(cv::Scalar(0,0,0));

cv::Mat Afc3;

cv::Mat t\_weights[] = {this->weights[i], this->weights[i], this->weights[i]};

merge(t\_weights, 3, Afc3);

cv::multiply(\*images[i], Afc3, temp);

cv::add(out, temp, out);

}

}

Abans de poder fer la multiplicació de les imatges per les matrius de pesos s’ha hagut de fer un pas previ. Com que la matriu de pesos és una matriu de tipus (CV\_8UCH1, CV\_32FC1)?? I les imatges son del tipus CV\_32FC3... acabar d’explicar

#### Optimització per GPU (CUDA)

La optimització de l’algorisme per GPU ... explicació

OpenCv disposa de llibreries que utilitzen CUDA amb el propòsit de utilitzar la GPU per realitzar els càlculs de forma més ràpida. Algunes d’aquestes funcions són les que s’han utilitzat per fer l’optimització del codi.

Si fins al moment tant les imatges com els mapes i els pesos eren objectes del tipus cv::Mat, per fer els càlculs a través de la targeta gràfica és necessari fer la conversió a cv::gpu::GpuMat. Aquesta conversió implica carregar l’objecte cv::Mat a la GPU mitjançant la funció cv::gpu::GpuMat::upload(). En el següent fragment de codi es pot veure un exemple de com es fa la càrrega, processat i descàrrega de una imatge a la GPU.

//Declaració de les variables

cv::Mat imatge, imatge\_out;

cv::gpu::GpuMat imatge\_gpu, imatge\_out\_gpu;

//Lectura de disc

imatge = cv::imread(“imatge1.png”);

//Càrrega de la imatge a la GPU

imatge\_gpu.upload(imatge);

//Processat amb GPU

funció\_processat\_gpu(imatge\_gpu, imatge\_out\_gpu);

//Descàrrega a la CPU

imatge\_out\_gpu.download(imatge\_out);

Per fer l’optimització amb GPU, primer de tot s’ha intentat agafar les funcions que composen el stitching (remap i blending) i passar la seva execució a la GPU, però com que el procés de càrrega en memòria de la GPU és un procés lent, aquest pot esdevenir un coll d’ampolla.

Finalment aquesta opció s’ha desestimat ja que, en cada crida de les funcions remap i blend era necessari carregar i descarregar a la memòria de la GPU els mapes, les imatges i les matrius de pesos, amb la qual cosa, l’únic que aconseguíem era enlentir el software.

Fent uns càlculs ràpids; si s’agafa un vídeo de 100 mostres per fer el stitching, la quantitat de transferències a la memòria de la GPU és el següent:

* 100 crides a Stitching:
  + 4 crides a remap:
    - Càrrega a GPU de: Mapx, Mapy, imatge entrada
    - Descàrrega imatge de sortida
  + 1 crida a blend
    - Càrrega a GPU de: 4 imatges, 4 matrius de pesos
    - Descàrrega imatge de sortida

En total es fan 100\*((4\*3)+ 8) = 2000 càrregues i 100\*(4 + 1) = 500 descàrregues a la GPU.

Com que els mapes per fer el remap de les imatges, al igual que els pesos per fer el blending, són els mateixos al llarg de tota l’execució, s’ha solucionat el problema fent la transferència a la GPU d’aquests un sol cop al principi de l’execució del programa.

Fer-ho utilitzant les funcions de les classes originals però, requeria un canvi important en tot el codi, amb la qual cosa s’ha decidit fer el codi de nou en el programa utilitzat per fer el profiling. (Veure codi Annex N)

Després dels canvis esmentats el nombre de càrregues a la memòria de la GPU per 100 frames de vídeo és el següent:

* Càrrega dels mapx i mapy per les 4 càmeres
* Càrrega de les 4 matrius de pesos
* 100 crides a Stitching
  + Càrrega a GPU de: 4 imatges
  + Descàrrega imatge de sortida

Això fa un total de 4\*2 + 4 + 100\*4 = 452 càrregues i 1\*100 = 100 descàrregues a la memòria de la GPU, però tenint en compte que els mapes i les matrius de pesos es carreguen abans de fer el processat, tenim que durant aquest només hi ha 400 càrregues i 100 descàrregues a la GPU.

Un cop tenim les imatges i les altres variables carregades a la GPU, s’han de processar. Per fer-ho, s’han utilitzat un seguit de funcions d’OpenCv equivalents a les utilitzades a l’apartat anterior.

Per fer el remap, s’ha utilitzat cv::gpu::remap() en comptes de cv::remap() i, per fer el blending s’han utilitzat les funcions cv::gpu::multiply() i cv::gpu::add() en comptes de cv::multiply() i cv::add().

Buscar blendLinear a:

<http://docs.opencv.org/2.4/modules/gpu/doc/image_processing.html>

#### Possibles millores

## Deep Learning ??

# Resultats

## Software d’Arcol

Els resultats obtinguts són força satisfactoris.

En la plataforma original, el vídeo construït a partir de les imatges proporcionades per les quatre càmeres es mostrava a un frame cada 2 segons.

En el dispositiu Jetson, després de solucionar els problemes de lectura de les imatges en disc i sense optimitzar el codi, el resultat final ja era d’uns 3 frames per segon.

Finalment, després d’optimitzar les funcions del stitching per GPU, s’ha aconseguit que la imatge reconstruïda es mostri a uns 8 frames per segon.

Tenint en compte que la placa Jetson és al voltant de 10 vegades més econòmica que la Xilinx i, que la velocitat de processat és fins a 16 vegades més ràpida

## Deep Learning

# Pressupost ???????

NVIDIA Jetson TK1 EEUU 🡪 192 $

UK 🡪268 €

UK 🡪132.22 lliure = 168.58 €

# Conclusions

This should include your summary, conclusions and recommendations.

Bibliografia:

A thorough reference list as in the following examples: Conference paper [1], journal paper [2], book [3], standard-1 [4], standard-2 [5], online reference [6], patent [7], M.S. thesis [8] and Ph.D. dissertation [9].

1. J. Polastre, R. Szewczyk, D. Culler. "Telos: enabling ultra-low power wireless research". *In Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, IPSN 2005*, 25-27 April 2005, Los Angeles, USA. pp. 364-369. doi: 10.1109/IPSN.2005.1440950.
2. V.C. Gungor, B. Lu, G.P. Hancke. "Opportunities and challenges of Wireless Sensor Networks in Smart Grid". *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 3557-3564, October 2010. DOI: 10.1109/TIE.2009.2039455.
3. R. Faludi. *Building Wireless Sensor Networks: with ZigBee, XBee, Arduino, and Processing*, 1st ed. Sebastopol, USA: O'Reilly Media, 2010.
4. *Internet Protocol, Version 6 (IPv6) Specification*. IETF RFC 2460, December 1998.
5. *IEEE Standard for Information technology. Telecommunications and information exchange between systems Local and metropolitan area networks. Specific requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE Std 802.11-2012.
6. T. Tarun, P. Viswanathan, S. Suman. "Wireless Sensor Network White Paper". *Tetcos Engineering*, 2012. [Online] Available: http://www.tetcos.com/Enhancing\_Throughput\_of\_WSNs.pdf. [Accessed: 23 October 2012].
7. J. P. Wilkinson, “Nonlinear resonant circuit devices,” U.S. Patent 3 624 125, July 16, 1990.
8. M.V. Alvarez Fernández. "Feasibility study and design for Wireless Sensor Networks in a space environment". M.S. thesis, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, The Netherlands, 2011.
9. J. O. Williams, “Narrow-band analyzer,” Ph.D. dissertation, Department of Electrical Engineering, Harvard University, Cambridge, MA, USA,1993.

Annexos

Appendices may be included in your thesis but it is not a requirement.

Glossari

A list of all acronyms and the meaning they stand for.