



UNIVERSITY OF PISA

MASTER DEGREE THESIS

**Shooter Localization in Wireless
Acoustic Sensor Networks: experiments,
design and algorithm implementation on
a centralized gateway**

Author:
Andrea Simone PINNA

Supervisor:
Prof. Stefano GIORDANO
First Co-Supervisor:
Ing. Gregorio PROCISSI
Second Co-Supervisor:
CF AN Ing. Carlo ROATTA

*A thesis submitted in fulfilment of the requirements
for the degree of Telecommunication Engineering*

in the

Networking Research Group
Department of Information Engineering

April 2015

"If a man isn't willing to take some risk for his opinions, either his opinions are no good or he's no good."

"Se un uomo non si rivela disposto a lottare per le sue idee, o le sue idee non valgono nulla, o non vale nulla lui."

Ezra Pound

UNIVERSITY OF PISA

Abstract

Telecommunication Engineering
Department of Information Engineering

Telecommunication Engineering

Shooter Localization in Wireless Acoustic Sensor Networks: experiments, design and algorithm implementation on a centralized gateway

by Andrea Simone PINNA

In modern warfare, from Stalingrad to Hue, from Sarajevo to Fallujah, sniper attacks have become a more and more critical tactical issue. For this reason, in the last years many studies have been focused on acoustic sensor networks for shooter localization. Nowadays some western armies use mobile sensors equipped with three or four microphones with a FPGA programmed to detect the TOAs (Time of Arrivals) of the two acoustic phenomena that characterize a supersonic bullet gunshot: Shockwave and Muzzle Blast. This task is very challenging because of the very fast rise times of the associated acoustic signals and because they can be easily masked by reverberations and noise. Consequently, the sensors currently deployed adopt a time domain analysis based detection, which is very precise but requires a huge sampling frequency, in the order of MS/s. This thesis proposes the design of a distributed low cost network of single channel static sensors. The signal analysis technique is a Joint Time Frequency (JTF) technique. The choice of a JTF technique arises from the necessity of measuring TOAs with low cost sensors and ADCs (Analog to Digital Converters) which cannot afford very high sampling rates. Various JTF techniques for this application have been already proposed in literature, the STFT is the less computationally expensive one. Hence it was chosen, being the sensors cheapness and low computational load two of the most important design specifications for the work of this thesis. In the idea of the network, the sensors deliver TOA informations to a centralized gateway which performs the shooter position estimation with proper algorithms. A ZedBoard was thought as the centralized gateway, because it is provided with a Zynq SoC, which appears to be one of the most promising architectures in the class of embedded systems. Real data acquisition and signal analysis are provided with an Ultramic microphone, then a shooter-microphone distance is computed with a single sensor approach. The good experimental results encourage further investigations: the error is confined between 0.5 and 3.1 meters.

Acknowledgements

Si ringraziano:

il Prof. Stefano Giordano per avermi permesso di svolgere la tesi su un argomento che mi ha molto appassionato e per il supporto non solo tecnico.

Il Comandante Carlo Roatta, le cui gentilezza e competenza sono state eccezionali ed utilissime, soprattutto per aver profuso grandi sforzi per permettermi di porre in essere la campagna sperimentale e per avermi introdotto a Linux.

L' Ing. Franchi e l' Ing. Michele Di Cosmo per aver messo a disposizione la strumentazione necessaria e per essersi impegnati in prima persona per fornire informazioni necessarie ed apprezzatissimo supporto sul campo.

L' Ing. Giuseppe Portaluri che sempre mostratosi interessato e contento di fornire fondamentali consigli tecnici.

Il mio carissimo collega d'oltreoceano Alferes Davd Alberto Lau Gastelo che ha generosamente speso tempo per colmare tutte le lacune che avevo in informatica e per avermi aiutato ad entrare nella filosofia Linux.

Tengo poi a ringraziare affettuosamente:

tutta la mia famiglia che da sempre mi supporta e soprattutto mi sopporta.

Ambra che ha saputo darmi un nuovo motivo per stringere i denti, andare avanti e superare la crisi che mi aveva colpito prima di conoscerla e che rappresenta ora meglio prima la ragione per cui dare sempre il massimo.

La famiglia di Ambra che mi ha trattato come un terzo figlio.

Tutti i miei camerati che dal primo giorno di accademia sopportano il mio caratteraccio e la mia tirannide, che con me hanno superato innumerevoli avventure, non tutte positive, e che sono sempre stati il bastone su cui appoggiarmi quando iniziavo a zoppicare, il libro da consultare quando mi mancavano le conoscenze necessarie, gli amici che mi hanno fatto divertire anche con una semplice chiacchierata davanti ad una birra.

Tutti i miei carissimi amici di Vicenza, che ogni volta che torno a casa mi fanno sentire come non fossi mai partito.

Contents

Abstract	ii
Acknowledgements	iii
Contents	iv
List of Figures	vii
List of Tables	x
1 Sensor Network Design	1
1.1 Introduction	1
1.2 Target Application	3
1.3 Network General Features	5
1.3.1 Wireless Protocol Selection	5
1.3.2 ZigBee Topologies	8
1.3.2.1 Star Topology	9
1.3.2.2 Peer-to-Peer (Mesh) Topology	10
1.3.2.3 Cluster-Tree Topology	10
1.3.3 ZigBee Channel Access Policy	11
1.3.4 The Topology chosen for this Network Design	12
1.4 Synchronization Issues	13
1.4.1 GPS Clock Synchronization	13
1.5 Hardware description	14
1.5.1 Sensor Nodes	14
1.5.2 The Centralized Gateway	16
1.5.2.1 Why Zynq SoC	17
1.5.2.2 Programming ZedBoard with System Generator for DSP: a failed attempt	19
2 Gunshot Acoustic Parameters Estimation	22
2.1 Overview on gunshot acoustic signals	22
2.1.1 Muzzle Blast	24
2.1.2 Bullet Shock Wave	25
2.2 Environmental Effects on Gunshot Acoustical Features	28

2.2.1	Reverberant Environment: Sound Reflection and Multi-Path	29
2.2.2	Effect of Wind	30
2.2.3	Temperature and Humidity Effects on the Speed of Sound	30
2.2.3.1	Temperature Dependence	31
2.2.3.2	Effect of Variable Temperature	32
2.2.3.3	Humidity Effects	33
2.2.3.4	Frequency Dependent Sound Absorption due to Humidity	33
2.3	Shock Wave and Muzzle Blast Discrimination	34
2.3.1	Discrimination Issues	34
2.3.2	Approaches presented in Literature	34
2.3.2.1	A Time Domain Analysis: a State Machine fed by Zero Crossing encoding	35
2.3.2.2	Joint Time-Frequency and Wavelet Analysis	37
2.3.2.3	Final Considerations and Signal Analysis Technique Choice	40
3	Shooter Localization Problem	41
3.1	The Geometry of the problem and derivation of Shooter Equations	41
3.2	Introduction to different algorithm typologies	43
3.2.1	Multi-Channel Acoustic Sensor Networks	43
3.2.2	Single-Channel Acoustic Sensor Networks	44
3.2.3	Final Observations	44
3.3	Synchronous sensors network shooter localization Algorithm	45
3.3.1	Direction of Arrival	46
3.3.2	Worst Case shooter position estimation	47
3.4	Asynchronous sensors network shooter localization Algorithm	49
3.5	A special case: Single (Single-Channel) Sensor	50
4	Experimental Validation	53
4.1	Experimental Setup	53
4.1.1	The Microphone	54
4.1.1.1	Specifications and Hardware Description	54
4.1.1.2	Ultramic's Software Description	56
4.1.2	Sensor Configuration	57
4.1.3	The Weapons	57
4.1.4	The Measurements	59
4.2	Signals Analysis and Features Extraction	61
4.2.1	Considerations and TDOA calculation	64
4.3	Final Shooter Ranging	66
4.3.1	Results Interpretation	66
4.4	Error Sources	67
4.4.1	TDOA estimation Error Sources	67
4.4.1.1	Microphone Performances	68
4.4.1.2	Sampling Rate	68
4.4.1.3	Shockwave and Muzzle Blast discrimination Signal Anal- ysis Technique	68
4.4.2	Ranging Algorithm Error Sources	69
4.5	Conclusions	70

A Getting Started Guide to ZedBoard and Linaro Ubuntu OS for Linux users	72
A.1 Purpose of the Paper	72
A.2 Boot Linaro Ubuntu on ZedBoard	73
A.2.1 Formatting the SD Card	73
A.2.1.1 Command Line Approach	75
A.2.1.2 GParted GUI Approach	75
A.2.2 Linux File System	78
A.2.2.1 Copy the Linaro File System to the ext4 partition of the SD Card	82
A.2.2.2 Build the Linux Kernel	83
A.2.2.3 Configure the Kernel	85
A.2.2.4 Build the Kernel	85
A.2.2.5 Obtain the BOOT.BIN File	87
A.2.2.6 Compile the Device Tree	87
A.2.2.7 Boot the SD card onto the ZedBoard	88
A.2.3 Boot Linaro using Minicom	90
A.2.4 Booting Linaro using GNU Screen	94
A.3 How to share a PC Internet Connection with ZedBoard	95
A.4 ZedBoard Remote Control	95
A.4.1 Connecting to ZedBoard with SSH	97
A.4.1.1 SSH with X11: a GUI for applications	99
A.4.2 Controlling ZedBoard with (tight) VNC	99
A.5 Bug Fixed	102
B Script in C for Single Channel Single Sensor shooter ranging	104
Bibliography	106

List of Figures

1.1	Dragunov soviet sniper rifle	4
1.2	Data rate vs Range for different wireless protocols	6
1.3	Comparison of the normalized energy consumption for each protocol	7
1.4	ZigBee star topology	10
1.5	ZigBee peer to peer topology	10
1.6	ZigBee cluster-tree topology	11
1.7	SAM R21 photo	15
1.8	SAM R21 peripherals	15
1.9	UC530 Fastrax GPS Antenna Module	17
1.10	ZedBoard block diagram	18
1.11	Zynq-7000 AP SoC Overview	19
1.12	System Generator Model for Single Sensor approach	20
1.13	Xilinx System Generator for DSP Blockset	20
2.1	gunshot	22
2.2	91meters	23
2.3	352meters	23
2.4	549meters	23
2.5	732meters	23
2.6	muzzleblast	25
2.7	shockwave	27
2.8	Nshape	28
2.9	soundreflection	29
2.10	sth	32
2.11	humidity	33
2.12	shockwave detection state machine	36
2.13	muzzle blast state machine	37
3.1	geometry	41
3.2	muzzle shock fusion	48
3.3	geometry2	51
4.1	ultramic1	54
4.2	ultramic2	54
4.3	mems frequency response	55
4.4	Ultramic internal electronics	55
4.5	Ultramic gain configuration switches	56
4.6	bat recording with SeaWave	57

4.7	Sensor Configuration	58
4.8	SeaWave running on Windows 7 virtual machine on a Linux PC	58
4.9	5.56mm NATO bullet	59
4.10	.308 Winchester bullet	59
4.11	5.56mm NATO and .308 Winchester comparison	59
4.12	Geometry of the experimental campaign	60
4.13	1 st Shot Spectrogram	62
4.14	2 nd Shot Spectrogram	62
4.15	3 rd Shot Spectrogram	63
4.16	4 th Shot Spectrogram	63
4.17	5 th Shot Spectrogram	63
4.18	6 th Shot Spectrogram	64
4.19	7 th Shot Spectrogram	64
A.1	output of <code>lsblk</code> command.	73
A.2	output of <code>lsblk</code> command with sd device.	74
A.3	output of <code>df</code> command with sd device.	74
A.4	unmounting the partitions.	75
A.5	deleting the partitions.	76
A.6	creating new partitions (first part)	77
A.7	creating new partitions (second part)	78
A.8	checking new partitions with <code>lsblk</code> command	78
A.9	formatting the two partitions with <code>mkfs</code> utility	79
A.10	error reported after the first attempt to install GParted	79
A.11	starting GParted and selecting the SD card device node	80
A.12	No partition onto the SD card	80
A.13	One existing partition onto the SD card	81
A.14	symbol to click for executing actions on GParted	81
A.15	editing the fat32 partition labeled BOOT	81
A.16	symbol to click for editing partitions on GParted	81
A.17	editing the ext4 partition labeled rootfs	82
A.18	Unzip the Linaro image in a temporary folder named "linaro"	82
A.19	Mount the SD card to <code>/tmp/sdext4</code>	83
A.20	Umount the SD card	83
A.21	Set ARM GNU Toolchain path	84
A.22	Configure Linux Kernel for ZedBoard	85
A.23	Kernel configuration Menu	86
A.24	The zImage present in the folder <code>linux-digilent/arch/arm/boot/zImage</code>	86
A.25	This error occurs when the ARM GNU Toolchain is not in the path	87
A.26	Line containing the "bootargs" in the <code>digilent-zed.dts</code> file at the beginning	88
A.27	Line containing the "bootargs" in the <code>digilent-zed.dts</code> file after the modifications in order to boot ZedBoard with a Linaro file system	88
A.28		89
A.29	Check ZedBoard device name with <code>dmesg — grep tty</code> , it will probably be <code>ttyACM0</code>	90
A.30	Start minicom	91
A.31	Minicom settings window	91

A.32 Minicom serial port setup	92
A.33 Window displayed when closing minicom after serial port set up	92
A.34 Stop ZedBoard auto boot process	93
A.35 Linaro boot process finished	93
A.36 Linaro Ubuntu desktop appears on the monitor	94
A.37 Edit a new connection	95
A.38 Add a new connection	96
A.39 Choose an ethernet connection	96
A.40 Share the connection to other computers	96
A.41 First ssh ZedBoard configuration	97
A.42 Second ssh ZedBoard configuration	97
A.43 Finding ZedBoard IP address	98
A.44 Creating the SSH connection with ZedBoard	99
A.45 Using x11 with SSH	99
A.46 a x11 GUI for gedit	100
A.47 Starting a VNC server session on the ZedBoard	100
A.48 Starting <i>VNCviewer</i> on the host PC	101
A.49 VNCviewer GUI	101
A.50 VNCviewer GUI authentication	101
A.51 VNC lxde remote Linaro desktop	101
A.52 Xterm running on a VNC lxde remote linaro desktop	102
A.53 warning message identifying the bug	103

List of Tables

4.1	ultramic gain configuration	56
4.2	ultramic os compatibilities	56
4.3	Weapons Characteristics	59
4.4	Shots Acquisition Settings	61
4.5	computed TDOAs	66
4.6	Estimated Shooter Distances	67

*Alla mia famiglia,
ad Ambra
e ai miei nuovi fratelli Alessandro, Alessio, Antonio, Cosimo
(Mimmo), David e Mattia, senza i quali non avrei mai avuto la
forza di raggiungere un tale obiettivo.*

Chapter 1

Sensor Network Design

1.1 Introduction

In modern warfare, from Stalingrad to Hue, from Sarajevo to Fallujah, sniper attacks have become a more and more critical tactical issue. The importance of countersniper systems is underscored by the constant stream of news reports coming from the Middle East. For example, in Afghanistan our soldiers are often target of well hidden and expert snipers, who know the territory much better than us. In October 2006 CNN reported on a new tactic employed by insurgents. A mobile sniper team moves around busy city streets in a car, positions itself at a good standoff distance from dismounted US military personnel, takes a single well-aimed shot and immediately melts in the city traffic. By the time the soldiers can react, they are gone. For this reason, in the last years many studies have been focused on acoustic sensor networks for shooter localization. When a typical rifle is fired, there are two acoustic phenomena that can be observed. The first is the muzzle blast that is created by the explosion inside the barrel as the bullet exits the muzzle. This sound propagates from the muzzle spherically at the speed of sound. The second is a miniature sonic boom, the ballistic shockwave, that is generated by the supersonic projectile. This is a conical wavefront with its axis being the bullet trajectory and it propagates at the speed of sound also. Both the muzzle blast and the shockwave can be detected by regular microphones. A typical acoustic shooter localization system relies on one or a couple of wired microphone arrays with precisely known microphone separation. This makes it possible to estimate the Angle of Arrival (AOA) of both events by measuring the Time of Arrival (TOA) at every microphone and using the known geometry. Then a simple analytical formula containing the AOAs of the two acoustic events and the Time Difference of Arrival (TDOA) between the muzzle blast and the shockwave provides the shooter location [1]. The first sensor network-based countersniper

system was introduced in 2003 by [2], [3]. The system is based on potentially hundreds of inexpensive sensor nodes deployed in the area of interest forming an ad hoc multihop network. The acoustic sensors measure the Time of Arrival (ToA) of muzzle blasts and ballistic shockwaves, pressure waves induced by the supersonic projectile, send the data to a base station where a sensor fusion algorithm determines the origin of the shot. The obvious disadvantage of such a system is its static nature. Nowadays some western armies use mobile sensors equipped with three or four microphones with a FPGA programmed to detect the TOAs (Time of Arrivals) of the two acoustic phenomena named above. With these informations, each sensor is able to estimate the shooter position. The US Army has recently deployed a large number of personal wearable shooter location systems by QinetiQ [4]. The British military has been using these for a number of years now. The shockwave and muzzle blast discrimination task is very challenging because of the very fast rise times of the associated acoustic signals and because they can be easily masked by reverberations and noise. Consequently, the sensors currently deployed adopt a time domain analysis based detection, which is very precise but requires a huge sampling frequency, in the order of MS/s [1]. This thesis proposes the design of a distributed low cost static network of single channel static sensors. The signal analysis technique is a Joint Time Frequency (JTF) technique. The choice of a JTF technique arises from the necessity of measuring TOAs with low cost sensors and ADCs (Analog to Digital Converters) which cannot afford a sampling frequency higher than 250kS/s. Various JTF techniques for this application have been already proposed in literature, this kind of approach is suggested by the fact that the energy produced by these two sources have differing durations, rise times and arrival times, hence the separation can be done by joint time-frequency analysis techniques. Typically the energy of the muzzle blast signature is in the 300÷1000 Hz range, while the energy of the shockwave is mainly in the 3÷7 kHz range. The three most studied JTF techniques for this application are: the Smoothed Pseudo Wigner-Ville Distribution (SPWVD), the Discrete Wavelet Transform (DWT) and the Short Time Fourier Transform (STFT). While the DWT appears to be the best tradeoff between discrimination precision and computational complexity, the STFT is the less computationally expensive one. Hence it was chosen, being the sensors cheapness and low computational load two of the most important design specifications for the work of this thesis. The wireless protocol choice was based on the following observations: the network is distributed on a not large area (a few tens of meters), the protocol should be as power saving as possible. The protocol that was found the most satisfying for this network design is ZigBee (IEEE 802.15.4 standard), in particular a star topology was considered. In the idea of the network, the sensors deliver TOA informations to a centralized gateway which performs the shooter position estimation with proper algorithms. A ZedBoard was thought as the centralized gateway, because it is supplied by a Zynq SoC, which appears to be one of the most promising architectures

in the class of embedded systems. Real data acquisition and signal analysis are provided with an Ultramic microphone, then a shooter-microphone distance is computed with a single-channel single sensor approach, presented for the first time in[5]. The good experimental results encourage further investigations: the error is confined between 0.5 and 3.1 meters.

1.2 Target Application

Before thinking on the specific design of the network, the target application must be understood, because it is the most important guideline that allows to determine precisely the project specifications. Despite of some current trends in this research field [6], we did not think on soldier-worn or vehicle mounted multichannel mobile sensors. This is because the aim of the thesis was not to develop a system that supports a group of soldiers, a fire team or a patrol unit, during its movements on the battlefield. We thought, instead, on a static and distributed sensor network for monitoring an area around a base, for example a *FOB* (*Forward Operating Base*) in Afghanistan, and localizing an enemy sniper after a shot detection. The idea was originated from the observation that in current warfare, our soldiers are deployed in insidious and wide territories but deeply known by the enemy, and they are often target of well hidden snipers who hit men also inside the base sometimes. Not rarely these snipers are equipped with old but excellent soviet rifles, as the illustrious *Dragunov* in figure 1.1. Modelling such a sensor network, some issues characterizing the other kind of sensor network previously exposed were avoided:

- the management of sensors mobility;
- changing multipath features of the environment.

The first issue derives from the fact that, for a soldier-worn or vehicle mounted sensor network, sensors continuously move in the battlefield. While the second is due to the fact that sensors migration changes network configuration and also network geographic location, thus the multipath characteristics of the environment are constantly modified. Dealing with a static network for FOB surveillance, instead, other issues must be considered:

- sensors must be robust to weathering and their power consumption must be as low as possible, in order to reduce battery changes;
- the sensors must be small and hideable.

The first item comes from the tactical requirement of limiting to the minimum the circumstances in which soldiers or technical operators exit from the base and work in open field in the proximity of the hostile territory. This network concept can be somehow contextualized in the so called *Sensor Rocks* area of interest. This topic concerns with sensor networks that are made of motes camouflaged as rocks and disseminated in a vast area for revealing anomalies. Probably the best example of sensor rocks network is *SPAN (Self Powered Ad hoc Networks)* [7], now tested by US Army, precisely it uses sensors for movement detection. In fact the aim of this thesis is to give the start to the project of a Shooter Localization Sensor Rocks Network for Italian Armed Forces. The target application was not the only guideline that influenced the network specifications. Being this thesis a sort of feasibility study for an hypothetical future Italian Armed Forces project, the present european economic crisis could not be ignored, thus another fundamental parameter for this work was the cheapness. Briefly resuming all the specifications, we can say that the sensors must be:

- as robust as possible;
- small and hideable;
- as less power consuming as possible;
- low cost.



FIGURE 1.1: A Russian soldier taking aim with his Dragunov

1.3 Network General Features

In the imagined network there are at least three peripheral acoustic sensors and one centralized gateway. The characteristics of the acoustic peripherals of the sensors are described in chapters 2 and 4. The gateway is situated into the base or onboard a vehicle, hence for it the requirements in terms of robustness to weathering, power efficiency and small dimensions are extremely relaxed. The sensors should be deployed in a possibly square area with sides about 20÷30m long. The size constraints are the result of a tradeoff research between having a network widely distributed and making the Far Field approximation likely assumable. This assumption is necessary for the processing that will be presented in section 3.3. Obviously for FOB surveillance applications more than one network would be necessary, deployed along the base perimeter. After the gunshot acoustic signal detection, they perform a shockwave and muzzle blast discrimination (see section 2.3) and transmit the extracted information to the gateway using a wireless communication protocol: ZigBee (IEEE 802.15.4).

1.3.1 Wireless Protocol Selection

The protocol choice was based on the following two considerations:

1. the network is distributed on a not large area (a few tens of meters);
2. the protocol should be as power saving as possible.

A good comparison between the wireless protocols inspected for this choice is [8]. Bluetooth (over IEEE 802.15.1), ultra-wideband (UWB, over IEEE 802.15.3), ZigBee (over IEEE 802.15.4), and Wi-Fi (over IEEE 802.11) are four protocol standards for short range wireless communications with low power consumption. From an application point of view, Bluetooth is intended for a cordless mouse, keyboard, and hands-free headset, UWB is oriented to high-bandwidth multimedia links, ZigBee is designed for reliable wirelessly networked monitoring and control networks, while Wi-Fi is directed at computer- to computer connections as an extension or substitution of cabled networks. UWB and Wi-Fi provide a higher data rate, while Bluetooth and ZigBee give a lower one. In general, the Bluetooth, UWB, and ZigBee are intended for WPAN communication (about 10m), while Wi-Fi is oriented to WLAN (about 100m). However, ZigBee can also reach 100m in some applications. In figure 1.2, a graph represents data rate vs range for different protocols. FCC power spectral density emission limit for UWB emitters operating in the UWB band is -41.3 dBm/MHz. The nominal transmission power is 0 dBm for both Bluetooth and ZigBee, and 20 dBm for Wi-Fi. Bluetooth and ZigBee

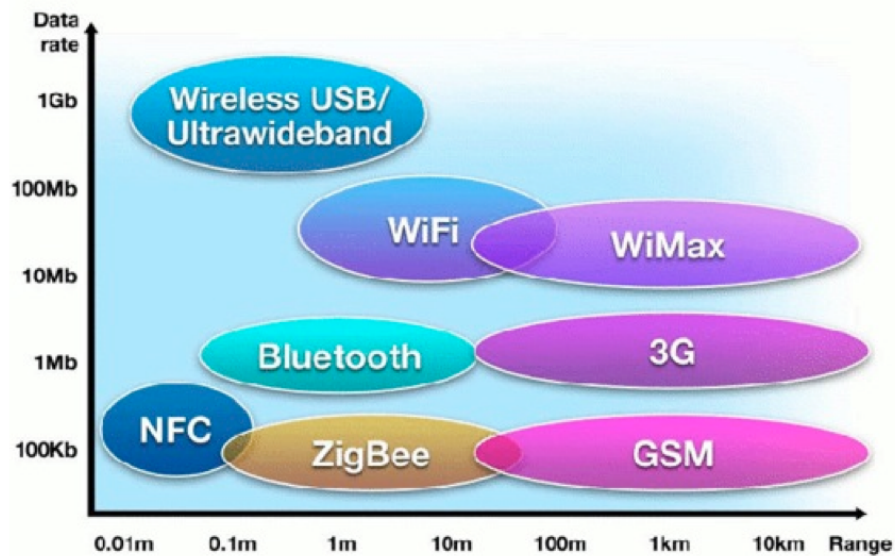


FIGURE 1.2: Data rate vs Range for different wireless protocols

are intended for portable products, short ranges, and limited battery power. Consequently, it offers very low power consumption and, in some cases, will not measurably affect battery life. UWB is proposed for short range and high data rate applications. On the other hand, Wi-Fi is designed for a longer connection and supports devices with a substantial power supply. Obviously, the Bluetooth and ZigBee protocols consume less power as compared with UWB and Wi-Fi. Based on the bit rate, a comparison of normalized energy consumption is provided in 1.3. From the mJ/Mb unit point of view, the UWB and Wi-Fi have better efficiency in energy consumption. In summary, Bluetooth and ZigBee are suitable for low data rate applications with limited battery power (such as mobile devices and battery-operated sensor networks), due to their low power consumption leading to a long lifetime. On the other hand, for high data rate implementations (such as audio/video surveillance systems), UWB and Wi-Fi would be better solutions because of their low normalized energy consumption. From previous dissertations, it is clear that the protocol that offers the best performance in terms of power efficiency (nominal transmission power is 0 dBm) for a network with sensors 20÷30 meters distant is the ZigBee (IEEE 802.15.4). In addition to being well fitted for networks of geographic dimensions comparable to the ones of this project and being power efficient, this protocol takes other advantages in terms of:

- signal security;
- network size.

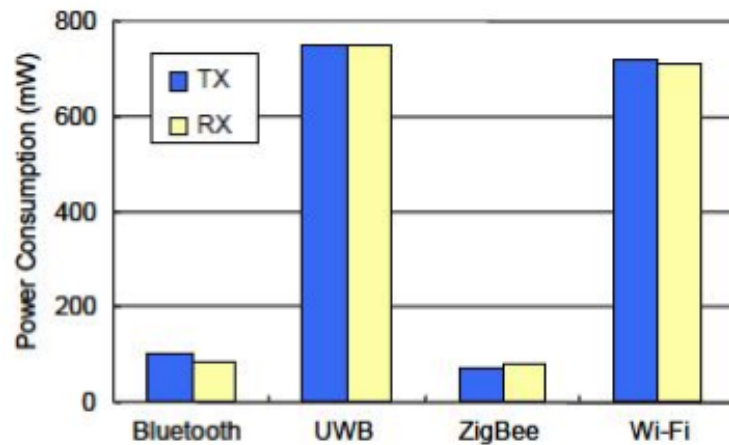


FIGURE 1.3: Comparison of the normalized energy consumption for each protocol

For what concerns the second item, we report here that a ZigBee star network can reach a number of nodes equal to 6500, but this is not in the interest of this application. In fact for shooter localization a number of nodes between 3 and 5 should be enough.

On the other hand, signal security is always a relevant item. In the following a brief comparison between ZigBee and the other wifi protocols previously recalled from the security point of view is exposed. All the four protocols have the encryption and authentication mechanisms. Bluetooth uses the E0 stream cipher and shared secret with 16-bit cyclic redundancy check (CRC), while UWB and ZigBee adopt the advanced encryption standard (AES) block cipher with counter mode (CTR) and cipher block chaining message authentication code (CBC-MAC), also known as CTR with CBC-MAC (CCM), with 32-bit and 16-bit CRC, respectively. In 802.11, Wi-Fi uses the RC4 stream cipher for encryption and the CRC-32 checksum for integrity. However, several serious weaknesses were identified by cryptanalysts, any wired equivalent privacy (WEP) key can be cracked with readily available software in two minutes or less, and thus WEP was superseded by Wi-Fi protected access 2 (WPA2), i.e. IEEE 802.11i standard, of which the AES block cipher and CCM are also employed.

Anyhow choosing ZigBee comports also some drawbacks:

- low data rate;
- long transmission time.

As specified in [9], The standard IEEE 802.15.4 offers two PHY (physical layer) options based on the frequency band. Both are based on direct sequence spread spectrum (DSSS) with 16 channels and 0.3/0.6 MHz bandwidth in the first frequency band, 2 MHz bandwidth in the second. The data rate is 250 kb/s at 2.4 GHz, 40kbps at 915

MHz and 20 kb/s at 868 MHz. The higher data rate at 2.4GHz is attributed to a higher-order modulation scheme. Lower frequency provides longer range due to lower propagation losses. Low rate can be translated into better sensitivity and larger coverage area. Higher rate means higher throughput, lower latency or lower duty cycle. So the highest achievable data rate with ZigBee is 250 kb/s when working in the 2.4 GHz band, which is the lowest if compared to 1 Mb/s of Bluetooth, 54 Mb/s of Wi-Fi and 110 Mb/s of UWB (reachable only with the unapproved IEEE 802.15.3a standard).

The transmission time depends on the data rate, the message size, and the distance between two nodes. Obviously, the required transmission time is proportional to the data payload size and disproportional to the maximum data rate. Thus, the transmission time for the ZigBee is longer than the others because of the lower data rate (250 Kbit/s).

1.3.2 ZigBee Topologies

ZigBee over IEEE 802.15.4, defines specifications for low rate WPAN (LR-WPAN) for supporting simple devices that consume minimal power and typically operate in the personal operating space (POS) of 10m. ZigBee provides self organized, multi-hop, and reliable mesh networking with long battery lifetime. The IEEE 802.15.4 standard supports multiple network topologies. In the standard, three general types are discussed:

1. star networks;
2. peer to peer networks;
3. cluster-tree networks.

For the network designed in this paper the star topology was preferred, but the reasons will be detailed later in section [1.3.4](#).

Before exposing the characteristics of the two different topologies, it is better to give the definitions of some key ZigBee terms:

- **PAN coordinator:** the PAN coordinator is the node (strictly speaking, the coordinator node) that initiates the network and is the primary controller of the network. The PAN coordinator may transmit beacons and can communicate directly with any device in range. Depending on the network design, it may have memory sufficient to store information on all devices in the network, and must have memory sufficient to store routing information as required by the algorithm employed by the network.

- **Coordinator:** the coordinator may transmit beacons and can communicate directly with any device in range. A coordinator may become a PAN coordinator, should it start a new network.
- **Device:** a network device does not beacon and can directly communicate only with a coordinator or PAN coordinator.
- **Full function device (FFD):** an FFD can operate in any of the three network roles (PAN coordinator, coordinator, or device). It must have memory sufficient to store routing information as required by the algorithm employed by the network. The complete protocol set is implemented in an FFD.
- **Reduced function device (RFD):** an RFD is a very low cost device, with minimal memory requirements. It can only function as a network device. Its role is limited to star topology or an end device in peer-to-peer network. It cannot become a PAN coordinator. It does not have the need to send large amounts of data and may only associate with a single FFD at a time. Consequently, the RFD can be implemented using minimal resources and memory capacity.

In the star network, the master device is the PAN coordinator (an FFD), and the other network nodes may either be FFDs or RFDs. In the peer-to-peer network, FFDs are used, one of which is the PAN coordinator. RFDs may be used in a peer-to-peer network, but they can only communicate with a single FFD belonging to the network, and so do not save true "peer-to-peer" communication. A ZigBee system consists of several components. The most basic one is the device. A device can be a full-function device (FFD) or reduced-function device (RFD). A network shall include at least one FFD, operating as the PAN coordinator. The FFD can operate in three modes: a personal area network (PAN) coordinator, a router, or a device. A RFD is intended for simple applications that do not need to send large amounts of data. A FFD can talk to RFDs or FFDs while a RFD can only talk to a FFD.

1.3.2.1 Star Topology

In the star topology, figure 1.4, the communication is established between devices and a single central controller, called the PAN coordinator. The PAN coordinator may be AC powered while other devices will most likely be battery powered. After a FFD is activated for the first time, it may establish its own network and become the PAN coordinator. Each star topology network chooses a PAN identifier, which is not currently used by any other network within the communication range. This allows each star network to operate independently. Beacon may be used to synchronize every node with PAN coordinator.

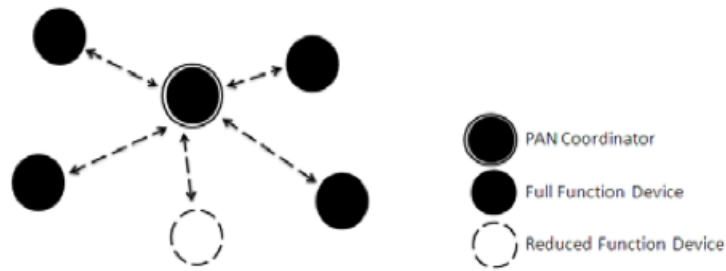


FIGURE 1.4: ZigBee star topology

1.3.2.2 Peer-to-Peer (Mesh) Topology

In peer-to-peer (mesh) topology, figure 1.5, there is also one PAN coordinator. In contrast to star topology, any device can communicate with any other device as long as they are in range of one another. A peer-to-peer network can be ad hoc, self organizing and self-healing. Applications such as industrial control and monitoring, wireless sensor networks, asset and inventory tracking would benefit from such topology. It also allows multiple hops to route messages from any device to any other device in the network. It can provide reliability by multipath routing. Beacon is not used for peer-to-peer topology. This reduces control and increases collisions as compared to the beacon enabled network.

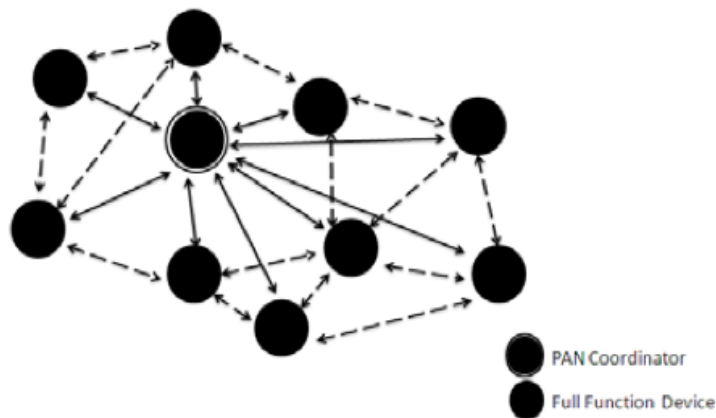


FIGURE 1.5: ZigBee peer to peer topology

1.3.2.3 Cluster-Tree Topology

Cluster-tree network, figure 1.6, is a special case of a peer-to-peer network in which most devices are FFDs and a RFD may connect to a cluster-tree network as a leaf node at the

end of a branch. Any of the FFD can act as a router and provide synchronization services to other devices and routers. Only one of these routers is the PAN coordinator. The PAN coordinator forms the first cluster by establishing itself as the cluster head (CLH) with a cluster identifier (CID) of zero, choosing an unused PAN identifier, and broadcasting beacon frames to neighbouring devices. A candidate device receiving a beacon frame may request the CLH to join the network. If the PAN coordinator permits the device to join, it will add this new device as a child device in its neighbour list. The newly joined device will add the CLH as its parent in its neighbour list and begins transmitting periodic beacons such that other candidate devices may then join the network at that device. Once application or network requirements are met, the PAN coordinator may instruct a device to become the CLH of a new cluster adjacent to the first one. The advantage of this multi-cluster, hierarchical structure is the increased coverage area at the cost of increased message latency.

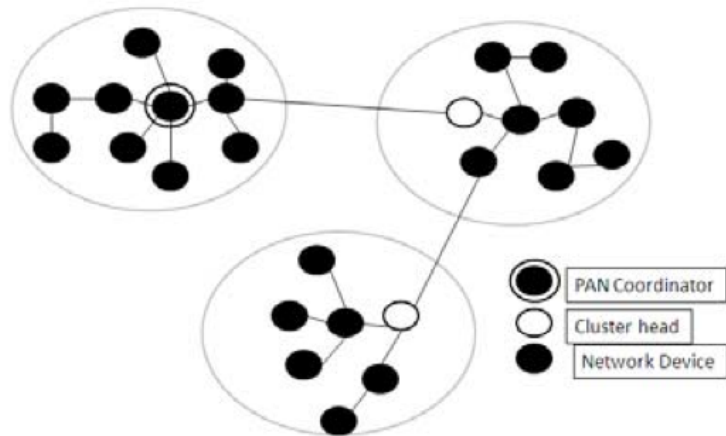


FIGURE 1.6: ZigBee cluster-tree topology

1.3.3 ZigBee Channel Access Policy

The network uses two types of channel access mechanism, one based on slotted CSMA-CA (*Carrier Sensing Multiple Access -Collision Avoidance*) in which the slots are aligned with the beacon frames sent periodically by the PAN coordinator, and another based on unslotted CSMA-CA [8]. The star-shaped hierarchical topology and beacon-enabled slotted CSMA-CA regime appear better suited for a sensor network implementation than their peer-to-peer and unslotted CSMA-CA counterparts, respectively. In the former case, the PAN coordinator can also act as the network sink that collects the data from individual sensor nodes; furthermore, the CSMA-CA mechanism simplifies synchronization and forwarding of data for further processing. In beacon-enabled networks,

channel time is divided into super-frames that are bounded by beacon transmissions from the coordinator. The basic time units of the MAC protocol are backoff periods to which all transmissions must be synchronized; at the 250 kb/s data rate, the duration of one backoff period is $t_{boff} = 0.32$ ms. In the uplink direction individual nodes access the channel using the CSMA-CA algorithm, and the channel must be idle for two successive backoff periods before transmission can start. If the channel is found busy, the random backoff countdown is repeated, possibly with a larger starting value.

1.3.4 The Topology chosen for this Network Design

As anticipated at the beginning of the chapter, in our idea, the network consists of three or four acoustic sensors which implement gunshot signal analysis and muzzle blast and shockwave TOAs estimation and then deliver the informations to a centralized gateway, that performs the shooter position estimation with the proper algorithm. Each acoustic sensor does not need to communicate with other sensors, but just with the centralized controller. For this reason, the star topology is evidently the best ZigBee topology for this network. Furthermore, it can be noticed that each sensor does not transmit huge amount of data: just two TOA values and, at most, the two coordinates of its position. This feature is also connected to the specification of low cost and power saving sensor nodes (see section 1.2), thus all the motes can be simple RFDs while the centralized gateway is an FFD acting as PAN coordinator, hence it can provide motes synchronization services for channel access with a beacon as explained in section 1.3.3. It is very important not to confuse "synchronization" regarding channel multiple access and synchronization regarding shockwave and muzzle blast TOAs estimation. These are two different problems and while the first is solved by adopting a beacon enabled ZigBee star network, the solution of the second problem, which resorts to GPS (*Global Positioning System*) will be explained later in section 1.4.1.

As already specified in section 1.3, for an application like FOB surveillance, more than one network should be deployed along the FOB perimeter in order to cover all the possible directions of enemy attacks. The shooter position estimations, available at the interested gateway (or gateways), must be delivered to a central command and control station. This station could be distant many tens of meters to the gateways, hence a ZigBee protocol could not be enough. Consequently, the gateway is linked via ZigBee (in particular it is the PAN coordinator) to its network sensors, and via Wi-Fi or WiMax to the command and control station. Notice that adding Wi-Fi or WiMax capabilities to the centralized gateway means increasing its power consumption, but, as formerly mentioned, this is not a problem since the gateway is positioned in the

proximity or inside the base, thus it can be AC powered (also a typical characteristic of PAN coordinators) and it can be bigger than the sensor nodes.

1.4 Synchronization Issues

As just explained in previous section 1.3.4, it is necessary to a distinction between:

- synchronization for wireless channel multiple access;
- synchronization to make consistent the estimated TOA values between various sensors.

The first item has been already discussed in section 1.3.4, and, for this network designed, solved by adopting a beacon enabled ZigBee star topology (which uses CSMA-CA method). The second item is going to be examined in this section and concerns with the gunshot acoustic signal analysis and parameters extraction. It has been previously anticipated that this analysis essentially consists in estimating the instants of arrival of two different acoustic phenomena: bullet shockwave and muzzle blast. Instants of arrival can be equivalently defined as the onset instants or the peak instants. If the sensors of a network are not synchronized in terms of signal acquisition, i.e. the nodes do not have a common time reference, then they would extract TOA values not compatible within different sensors. In this case, the only approach available is the asynchronous algorithm presented in [10] and briefly resumed in section 3.4, which relies just on the difference between shockwave and muzzle blast TOAs (TDOA) at each sensor. At the moment, for this network design, a synchronous approach is favorite, thus sensor synchronization must be furnished, trying to maintain the hardware cheapness, simplicity and low computational load of the sensor nodes.

Several synchronization methods have been proposed in literature: from *FTSP* (*Flooding Time Synchronization Protocol*) [11], to the post-facto approach [1], [2].

These methods are quite complex and can increase nodes computational load. We preferred a GPS based synchronization [12], adopted also by [13]. This choice adds a new specification to the sensor nodes, i.e. the RFDs of the ZigBee star topology: the need of a GPS antenna module.

1.4.1 GPS Clock Synchronization

Even fairly accurate computer clocks are likely to vary due to manufacturing defects, changes in temperature, electric and magnetic interference, the age of the quartz crystal,

or even system load. Additionally, even the smallest errors in keeping time can significantly add up over a long period. Consider two clocks that are synchronized at the beginning of the year, but one consistently takes an extra 0.04 milliseconds to increment itself by a second. By the end of a year, the two clocks will differ in time by more than 20 minutes. If a clock is off by just 10 parts per million, it will gain or lose almost a second a day. Clocks locked to atomic standards are much more stable timekeepers. Rubidium, Cesium and Hydrogen Maser clocks can be much more accurate. Of the three, Rubidium clocks often provide the best combination of cost, size and overall performance and are often a requirement for high reliability master clock systems. However, atomic clocks themselves do not guarantee traceability and synchronization with other clocks. That where GPS comes in. GPS satellites (and now other global navigation systems) include three or four atomic clocks that are monitored and controlled to be highly synchronized and traceable to national and international standards (known as UTC). So for time synchronization, the GPS signal is received, processed by a local master clock, time server, or primary reference, and passed on to "slaves" and other devices, systems, or networks so their "local clocks" are likewise synchronized to UTC. Typical accuracies range from better than 500 nanoseconds to a few milliseconds depending on the synchronization protocol. It is the process of synchronization to GPS that can provide atomic clock accuracy without the need for a local atomic clock. Still, local atomic clocks are sometimes desired as a long-term back-up solution to loss-of-GPS, either in the case of a weather-related outage, GPS interference, or other scenarios. In any case, GPS clock synchronization eliminates the need for manual clock setting (an error-prone process) to establish traceability to national and international standards so various events can be correlated even when they are time-stamped by different clocks. The benefits are numerous and include: legally validated time stamps, regulatory compliance, secure networking, and operational efficiency.

1.5 Hardware description

1.5.1 Sensor Nodes

Resuming here the specifications on the motes, i.e. the RFDs in the ZigBee star topology, they must:

- have ZigBee connectivity;
- have a GPS module;
- be low power consuming;

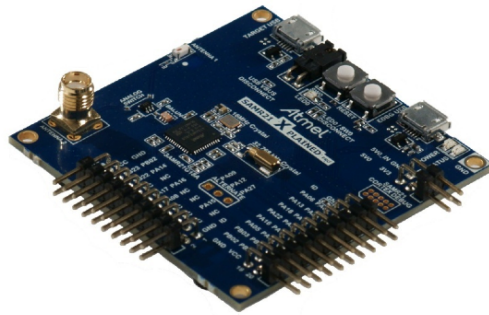


FIGURE 1.7: SAM R21 photo

- be small.

As sensor node board, the **Atmel SAM R21** was chosen [14], it is represented in figures 1.7 and 1.8. The Atmel SAM R21 series of low power microcontrollers combines the 32-bit ARM Cortex M0+ processor and an integrated ultra-low power 2.4 GHz ISM band Zig-Bee certified transceiver. Operating with ZigBee in the 2.4 GHz band, a maximum data rate of 250 kb/s can be assured (see section 1.3.1). The most energy efficient ARM processor yet, the Cortex-M0+ builds on the Cortex-M0 processor, while further reducing energy consumption and increasing performance. The SAM R21 ARM Cortex-M0+ based MCUs operate at 48MHz and feature a two-stage pipeline, single-cycle I/O access, single-cycle 32x32 multiplier, event system, and a fast and flexible interrupt controller. They are also highly efficient, reaching 2.14 CoreMark/MHz - 0.93 DMIPS/MHz.

For what concerns the requirement of small dimensions, the single-chip series is available in extremely small 5x5mm 32-pin and 7x7mm 48-pin package.

The SAM R21 implements a wide range of features to drive down power consumption, including low-power oscillators, clock gating and pre-scaling, Atmel SleepWalking™ technology and a proprietary low-power process. All this enables $70\mu\text{A}/\text{MHz}$ in active mode and less than $3.5\mu\text{A}$ with full RAM retention and RTC running in sleep mode.

For adding the acoustic sensor, a MEMS can be connected to the analog peripherals, the analog received signal can be quantized by a 12-bit 300kbps ADC.

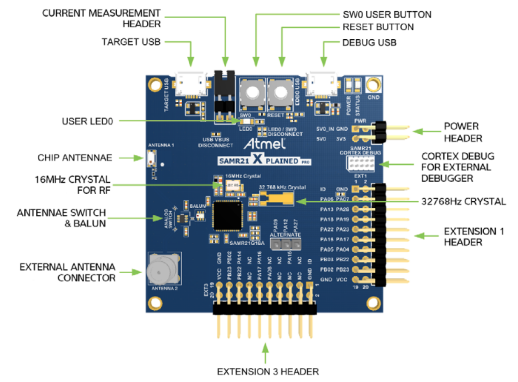


FIGURE 1.8: SAM R21 peripherals

To integrate a GPS antenna module, it was considered the **UC530 Fastrax GPS Antenna Module** [15] in figure 1.11. This device more important characteristics are:

- it is extremely small: 9.6 x 14.0 x 1.95 mm;
- it has ultra-low power consumption: 45 mW;
- it is self-assistant for 3 days;
- it guarantees a position accuracy of 2.5 m;
- it has few anti-jamming capabilities, in particular active Continuous Wave detection and removal.

Notice that a position precision of 2.5 m could result in significant sensor localization error. In [2] and [13] studies on the sensor positioning error on the final shooter localization are presented. In both case, the shooter localization involves the implementation of a sensor fusion algorithm which has been not considered for this network project yet. Anyhow these studies indicate that a 2.5 m positioning error can result in a positioning error of $5 \div 6$ m. Although the actual error for the network presented in this thesis should be investigated with several experiments, before considering such GPS module not enough precise, three important observation must be highlighted:

1. in this thesis we considered a network which motes are positioned not so far from a FOB, thus their deployment is not random and their precise position can be known a priori, without resorting to GPS;
2. the GPS module is mostly necessary for giving sensors a common time reference;
3. an hypothetical shooter localization error of $5 \div 6$ m, may be more that sufficient for some applications, if, for example, the shooter localization position estimation is delivered to an attack helicopter or to a mortar battery.

These observations lead to consider UC530 Fastrax GPS Antenna Module possibly suitable for this network sensor nodes. Linked to the condition of motes low power consumption, is the demand of long battery life. A possibility that should be deeply investigated in this direction is using solar energy, with proper solar panels, as power supply for the Atmel boards.

1.5.2 The Centralized Gateway

Recalling in few words the previous discussions about the centralized gateway:



FIGURE 1.9: UC530 Fastrax GPS Antenna Module

- it is an FFD and a PAN coordinator in the ZigBee star topology;
- it has ZigBee and WiFi connectivities;
- it is AC powered;
- it can be bigger than the sensor nodes;
- it has more computational capabilities than the other motes.

As centralized gateway **ZedBoard** (*Zynq Evaluation Development Board*) [16] was chosen. ZedBoard is intended to be a community development platform evaluation and development board based on the Xilinx Zynq-7000 All Programmable System on Chip. Combining a dual Cortex-A9 Processing System with an 85000 7-Series Programmable Logic cells, the board contains interfaces and supporting functions to enable a wide range of applications. In figure 1.10 the ZedBoard block diagram is showed.

1.5.2.1 Why Zynq SoC

One of the most important issues that embedded designers have to take care is the overburden of the system. The system bottlenecks imply the increment of the data latency, delay interrupt handling and lower data throughput among others. Parallel Processing is efficient for critical system performance but a central controller and memory management is needed; one possible solution for parallel processing can be achieved within a FPGA. One traditional solution is building a discrete hybrid system, a microcontroller put together with a FPGA, that unites the best of both worlds but with some exceptions like: the bandwidth between the two ICs is limited; increased power consumption; increased PCB size and layout complexity; performing limit of the interface that connects the microprocessor and the FPGA among others. The demands of today's technology

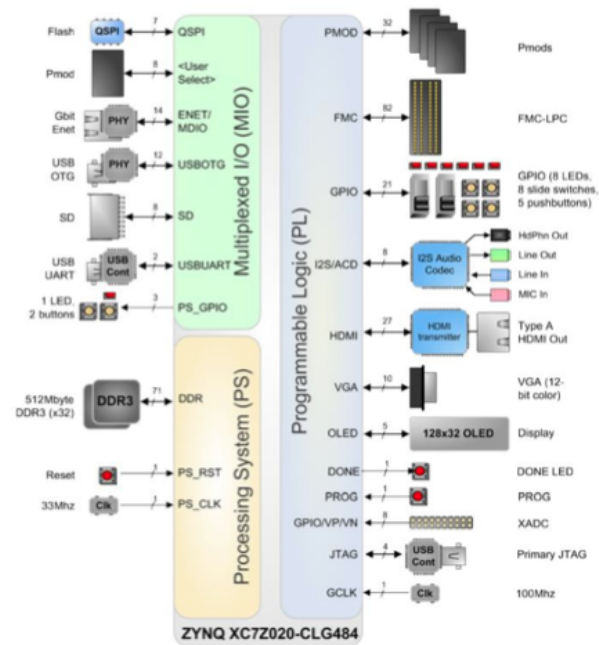


FIGURE 1.10: ZedBoard block diagram

results in the fusion of a processing system and a programmable logic in a single device, that is where the Zynq-7000 All Programmable System on Chip appears. Zynq-7000 is not an FPGA with an embedded PowerPC2, it is a 28 nm programmable-logic fabric 7 Series family FPGA coupled with a dual ARM Cortex-A9 MP Core processor in a single chip with a wide range of hard-core interface functions like high performance I/Os, Gigabit Transceivers, high throughput AXI (Advanced eXtensively Interface) up to three thousand PS to PL connections. This two-chip combo All Programmable SoC will reduce the cost, size, complexity and power consumption of the system; at the same time increasing the system performance.

The Zynq-7000 family is based on the Xilinx All Programmable SoC (AP SoC) architecture. These products integrate a feature-rich dual-core ARM Cortex-A9 MPCore based processing system (PS) and Xilinx programmable logic (PL) in a single device. The ARM Cortex-A9 MPCore CPUs are the heart of the PS which also includes on-chip memory, external memory interfaces, and a rich set of I/O peripherals. The range of devices in the Zynq-7000 AP SoC family enables designers to target cost-sensitive as well as high-performance applications from a single platform using industry-standard tools. While each device in the Zynq-7000 family contains the same PS, the PL and I/O resources vary between the devices. Figure ?? illustrates the functional blocks of the Zynq-7000 AP SoC. The PS and the PL are on separate power domains, enabling the user of these devices to power down the PL for management if required. The processors

in the PS always boot first. The PL can be configured as part of the boot process or configured at some point in the future. Additionally, the PL can be completely reconfigured or used with partial, dynamic reconfiguration (PR). PR allows configuration of a portion of the PL.

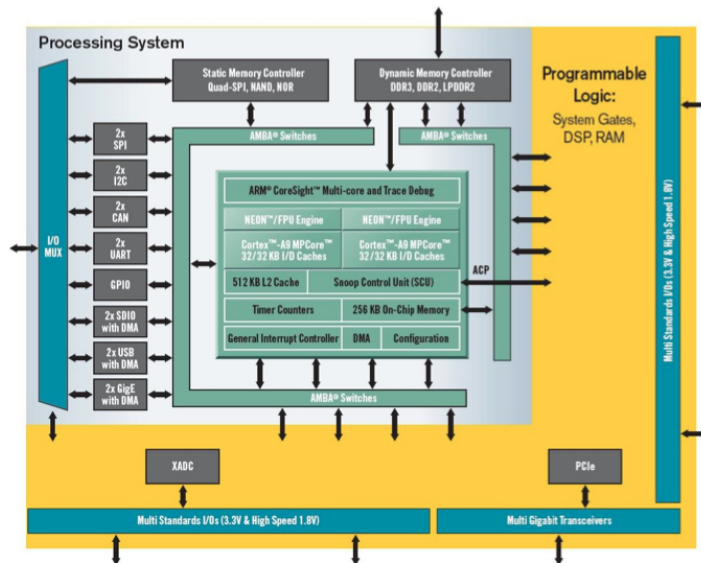


FIGURE 1.11: Zynq-7000 AP SoC Overview

1.5.2.2 Programming ZedBoard with System Generator for DSP: a failed attempt

Shooter localization algorithms may be complex. Thus an implementation on ZedBoard using Xilinx Vivado software, hence programming in C and Verilog or VHDL languages can be tricky and complicated. Consequently the possibility of programming ZedBoard with very high level synthesis is very attractive, because despite a worse hardware optimization, such techniques could make the programming much simpler and help to save time. In particular it was attempt to implement some localization algorithm programming ZedBoard with **System Generator for DSP**, a software tool belonging to the Mathworks Package for Xilinx FPGAs. As the title of this section suggests, after some trouble, this approach was abandoned, hence only a few words are going to be spent here. Anyhow we consider important to expose the method and the reasons of its failure to avoid future developers of the thesis useless losses of time. The purpose of System Generator is being a system level modelling tool that facilitates FPGA hardware design. It extends Simulink in many ways to provide a modelling environment that is well suited to hardware design. Its aim is providing high-level abstractions that are automatically compiled into an FPGA at the push of a button and also access to underlying FPGA

resources through low-level abstractions, allowing the construction of highly efficient FPGA designs. It was found that this compilation into FPGAs is so far from being automatic, except for some typical application and the pre-prepared tutorials published in the manuals. Above all, the System Generator blockset, in figure 1.13, is very limited for our application: we succeeded only to create a system generator model for the single sensor approach for shooter ranging (in figure 1.12). The blockset is not enough for other localization algorithms that will be exposed in Chapter 3. Anyhow it was tried

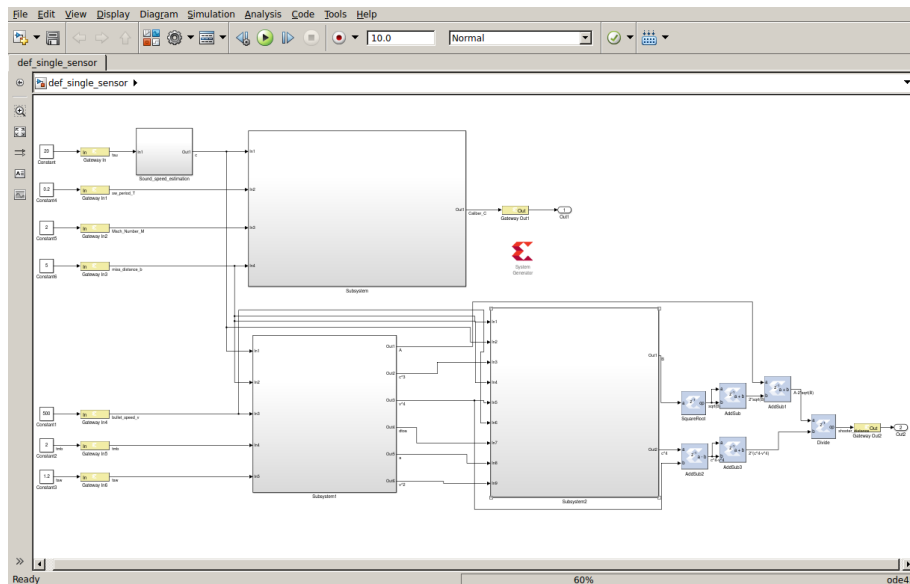


FIGURE 1.12: System Generator Model for Single Sensor approach

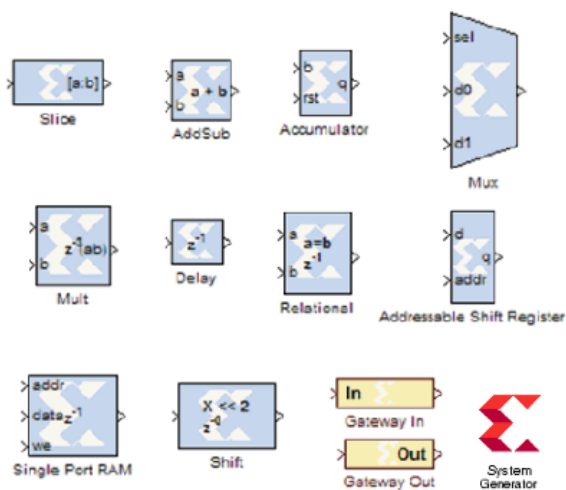


FIGURE 1.13: Xilinx System Generator for DSP Blockset

to program ZedBoard with this model. The synthesis was correct but when trying to implement the model on the specific hardware, various errors were discovered. These

errors mainly concerned with I/O overutilization: it was then clear that many low level clock settings had to be changed. At this point, it was established that there were no advantages in adopting such approach to program ZedBoard.

Chapter 2

Gunshot Acoustic Parameters Estimation

2.1 Overview on gunshot acoustic signals

A typical gun shot audio signal recording is the one represented in figure 2.1. This image clearly puts in evidence the two main acoustic effects of a gunshot: the Bullet Shockwave and the Muzzle Blast. From this image it can also be noticed that, for enough large distances between the microphone recording the signal and the shooter (as in an actual battlefield they can be), the amplitude dynamic of the Muzzle Blast is much lower than the Shockwave one, this is one of the gunshot acoustic features that make Muzzle Blast and Shockwave discrimination a difficult and challenging task. The temporal

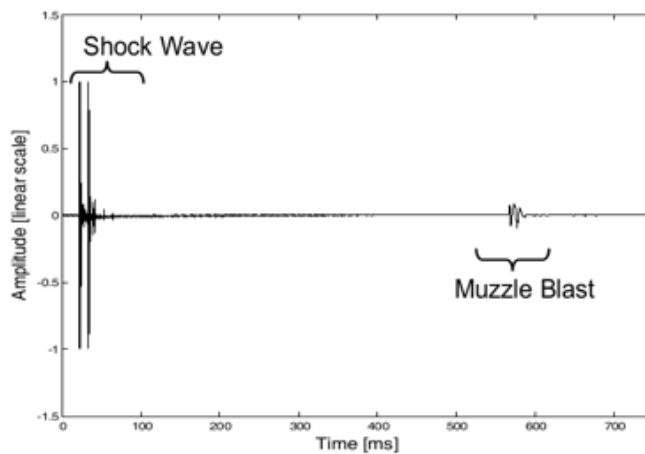


FIGURE 2.1: A typical gunshot acoustic signal

distance between these two events is related with the distance from the shooter to the

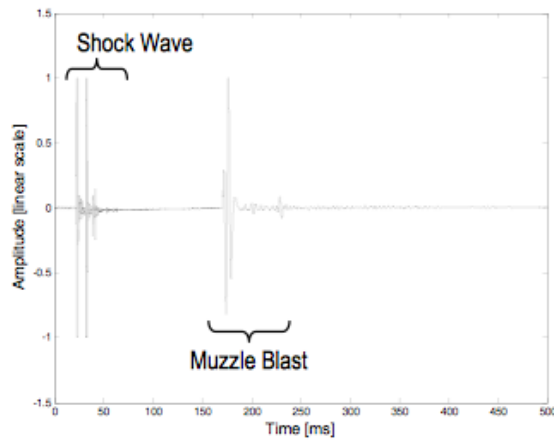


FIGURE 2.2: Gunshot recording 91 meters downrange

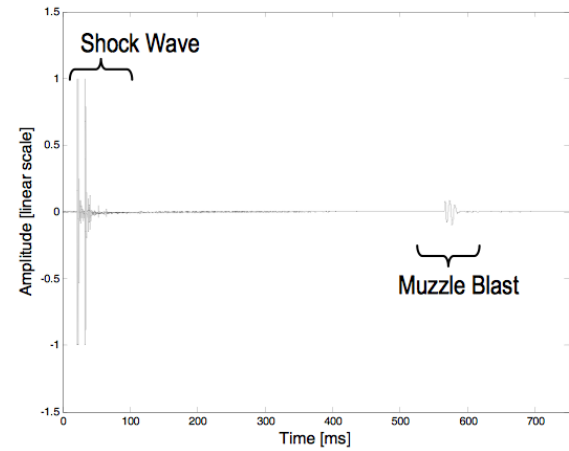


FIGURE 2.3: Gunshot recording 352 meters downrange

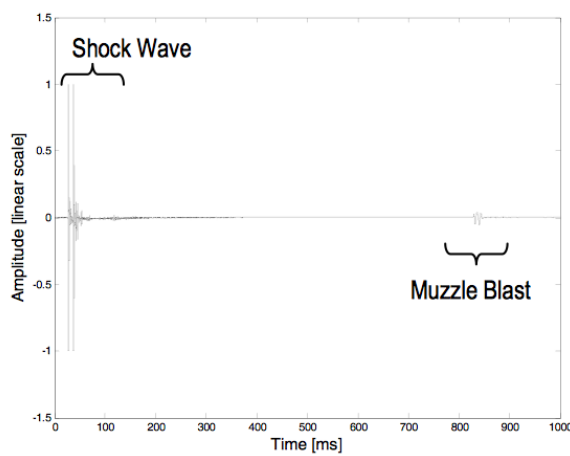


FIGURE 2.4: Gunshot recording 549 meters downrange

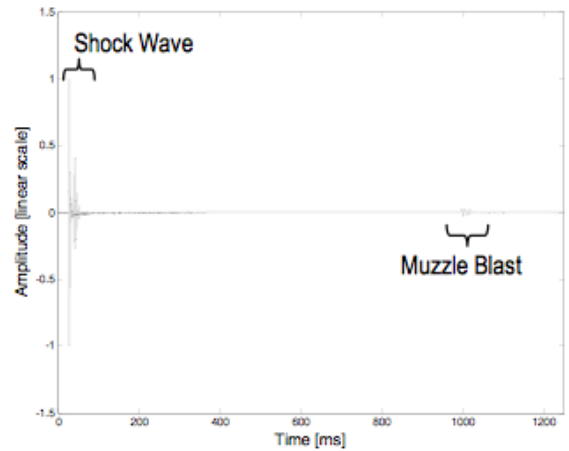


FIGURE 2.5: Gunshot recording 732 meters downrange

microphone that is recording the signal: the higher the distance shooter-microphone, the higher the separation in time between Shockwave and Muzzle Blast. The distance shooter-microphone affects not only the temporal separation between Muzzle Blast and Shock Wave, but also the amplitude dynamic of them. Such effect is underlined by figures 2.2, 2.3, 2.4, 2.5, obtained from[17]. This is particularly evident on the Muzzle Blast amplitude. The reason of this will be clearer in the paragraphs 2.1.1 and 2.1.2, after a digression on the physical causes of these two acoustics phenomena.

But the acoustic signal features of a gunshot hardly depends also on a lot of other parameters: weapon characteristics, kind of environment, climatic conditions and so on. As deeply investigated in this thesis, the identification of parameters such as the difference of the instants of arrival of Shockwave and Muzzle Blast, the duration of the Shockwave and others can provide information about the gun location with respect to

the microphone (or the array of microphones) and the speed and trajectory of the bullet. The principal difficulty when interpreting such recordings arises from reverberation (overlapping acoustic signal reflections) due to the gun shot sound reflecting off and diffracting around nearby surfaces. Further details will be provided in paragraph 2.2.1. This problem is particularly critical in environments strongly affected by multi path, such as the urban one, where the tactical danger of snipers (and so their localisation) is mainly present. Another problem is that if the microphone performs the recording at distances very far from the bullet's trajectory, the shock wave will have expanded sufficiently by spatial spreading that it may no longer be detectable compared to ambient noise.

2.1.1 Muzzle Blast

A conventional firearm uses a confined explosive charge to propel the bullet out of the gun barrel. The hot, rapidly expanding gases cause an acoustic blast to emerge from the barrel. The acoustic disturbance lasts few milliseconds (around 3 or 5 milliseconds) and propagates through the air at the speed of sound (c). The sound level of the muzzle blast is often highly directional: it is strongest in the direction the barrel is pointing and decreases as the off-axis angle increases. The peak sound pressure level associated with the muzzle blast can exceed 150 dB in the vicinity of the firearm. Once the gunpowder combustion is complete, the firearm itself may produce much more subtle mechanical sounds, such as post-shot motion of the trigger and cocking mechanism, ejection of the spent cartridge, and positioning of new ammunition. These characteristic sounds may be of interest for forensic study if the microphone is located sufficiently close to the firearm to pick up the tell-tale sonic information. A microphone located in the vicinity of the gunshot will detect the muzzle blast signal once the sound propagation travels at the speed of sound from the gun to the microphone position. However, the muzzle blast signal will also reflect off the ground and off other nearby surfaces, resulting in a complicated received signal consisting of multiple overlapping reflections (see figure 2.6). Anyway, unfortunately, muzzle blast is not necessarily a reliable acoustic source of analysis. In addition to being quite highly directional, as just explained, it may also be obscured by barriers and other obstacles blocking the direct path between the firearm and the microphone location (see also paragraph 2.2.1). Furthermore, some firearms can be equipped with an acoustical suppressor ("silencer") to alter or reduce the muzzle blast sound level. For all these reasons, even if this is not the case of this thesis, it is remarkable that someone has already presented shooter localisation algorithms with the purpose of relying only on the bullet's shock wave information, not using the muzzle blast one [18], [19]. It is important to underline that such algorithms require synchronization of the

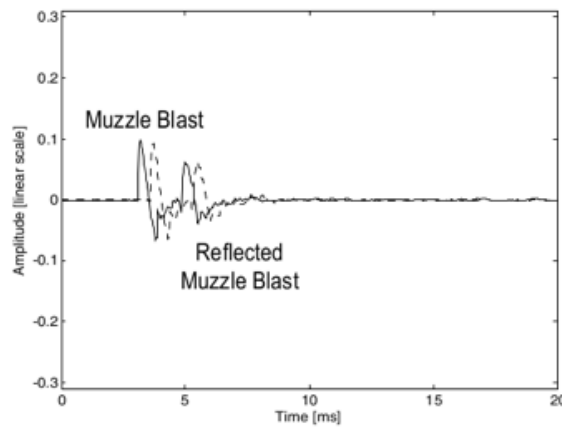


FIGURE 2.6: A Muzzle Blast signal affected by multi path in a reverberant environment

sensor composing the network, and this situation could not be simulated in this work, having only one microphone at disposal. At the end, another muzzle blast feature is now exposed: the muzzle speed dependence on bullet speed and deceleration. This relation can be relevant since, under the hypothesis of known (or estimated) bullet coefficient, it leads to a possible weapon classification (as reported in [13]), being the combination of muzzle speed and bullet coefficient often uniquely identifying the weapon used. An approximated form for this equation is (2.1):

$$v_{muzzle} = \sqrt{v_{bullet}^2 - 2ar} \quad (2.1)$$

Where:

- v_{bullet} is the average bullet speed;
- a is the bullet deceleration, a negative number which can be computed from the ballistic coefficient of the bullet;
- r is the range to the shooter position.

2.1.2 Bullet Shock Wave

Depending on the size of the charge, the mass of the bullet, and other factors, the bullet may be traveling at supersonic speed. A supersonic bullet causes a characteristic shock wave pattern as it moves through the air. The shock wave expands as a cone behind the bullet with a wave front propagating outward at the speed of sound. Its geometry depends upon the ratio between the bullet's speed v_{bullet} and the sound speed c . The

ratio (2.2) is known as the Mach Number of the moving object.

$$M = \frac{v_{bullet}}{c} \quad (2.2)$$

Thus a projectile traveling faster than the speed of sound has $M > 1$, while a subsonic projectile has a fractional Mach Number ($0 < M < 1$). Specifically for supersonic projectiles, the angle between the bullet path and the resulting shock wave is given by:

$$\theta_M = \arcsin\left(\frac{1}{M}\right) \quad (2.3)$$

where θ_M is referred to as a Mach Angle. The Mach Angle ranges from nearly 90° for barely supersonic bullets to 30° or less for high-velocity projectiles (see figure 2.7). Notice that, utilizing definition (2.2), equation (2.1) can be expressed as a function of Mach Number like this (2.4):

$$v_{muzzle} = \sqrt{(Mc)^2 - 2ar} \quad (2.4)$$

The speed of sound in dry air increases with the temperature and can be calculated as:

$$c = c_0 \times \sqrt{1 + \frac{\tau}{273.15} \frac{m}{s}} \quad (2.5)$$

where τ is the temperature in degrees Celsius and c_0 (which is equal to 331.3 m/s) is the speed of sound at 0°C . Notice that (2.5) is true only under the assumption of dry air, humidity changes may require corrections to this formula (see paragraph 2.2.3). It is now clear that the Mach Number depends on the temperature, thus temperature must be considered in order to compute the correct speed of sound, and consequently the correct Mach Number. For example, a projectile traveling at 800 m/s has Mach Number 2.42 at 0°C , or 2.33 (3.7% lower) at room temperature (20°C). The bullet traveling faster than the speed of sound outpaces the propagating shock wave and the muzzle blast wave fronts which move at the speed of sound. This means that a microphone located down range from the shooting position will typically receive the shock wave arrival considerably before the arrival of the muzzle blast. For supersonic projectiles, a set of microphones placed at known locations within the path of the shock wave can provide an estimate the shock's propagation direction. Note, however, that determining the bullet's trajectory from the shock propagation vector requires knowledge of the bullet velocity, v_{bullet} , or sufficient spatial information to deduce the Mach angle. Two bullets following the same path but at different speeds may create substantially differing shock wave propagation directions, as was shown in figure 2.7. In other words, if v_{bullet} is not known, then the shock wave cone angle is also not known, and the bullet's trajectory cannot be determined exactly without additional spatial information. This physical

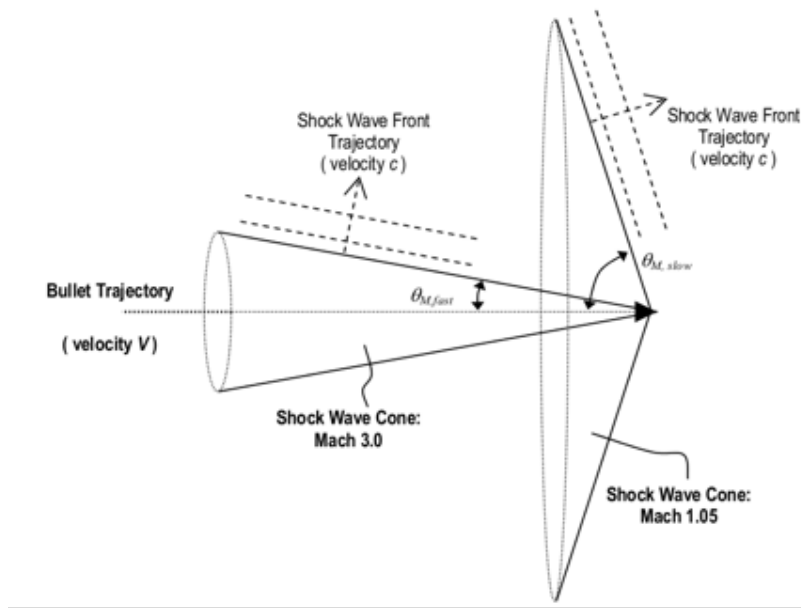


FIGURE 2.7: The Shock Wave cone

reality is important to consider when examining gunshot acoustic evidence. The acoustic shockwave from the bullet has a very rapid rise to a positive overpressure maximum (few microseconds), followed by a corresponding under pressure minimum. As the shockwave propagates, the nonlinear behaviour of the air causes the pressure disturbance to form an "N" shape with a rapid onset, a ramp to the minimum pressure, and then an abrupt offset. This "N" shape is the distinguishing feature of the bullet shock wave. The time interval of the "N" wave between the over- and under-pressure is proportional to the size of the projectile, this relation is expressed by Whitham equation (2.6) (riferimento al tuo articolo)

$$T = \frac{1.82Mb^{\frac{1}{4}}}{c(M^2 - 1)^{\frac{3}{8}}} \times \frac{d}{l^{\frac{1}{4}}} \approx \frac{1.82d}{c} \times \left(\frac{Mb}{l}\right)^{\frac{1}{4}} \quad (2.6)$$

where:

- M is the bullet's Mach Number;
- b is the **miss distance**, and is defined as the distance between the bullet's trajectory and the microphone at the loin of closest approach, i.e. the perpendicular distance between the bullet's path and the microphone;
- c is the speed of sound;
- d is the maximum bullet's diameter;
- l is the bullet length.

A typical bullet a few centimetres long has an inter shock interval of less than $200 \mu\text{sec}$, see figure 2.8. From equation (2.6), the dependence of the shockwave duration from the

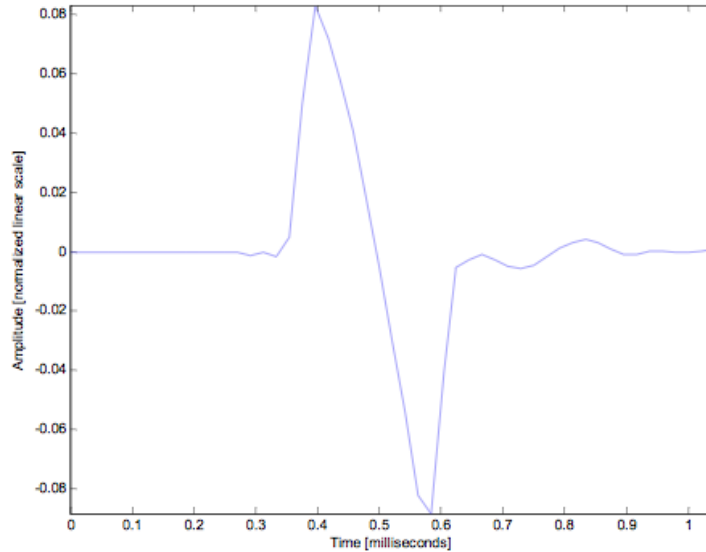


FIGURE 2.8: A typical N shaped shock wave of a few centimetres long bullet, with a period of about $200 \mu\text{sec}$

bullet dimension, d and l , is clear. These two parameters identify the weapon caliber, or equivalently the so called **Bullet Coefficient**, defined as:

$$C = \frac{d^4}{l} \quad (2.7)$$

Thus, substituting definition (2.7) in equation (2.6), the relation between the shockwave period T and the Bullet Coefficient C can be revealed:

$$C = \frac{T^4 c^4}{1.82^4 M b} \quad (2.8)$$

So, if the shockwave period T can be estimated from the received signal, the bullet Mach Number M and the miss distance b are known, then also the bullet coefficient, hence the weapon caliber, can be computed.

2.2 Environmental Effects on Gunshot Acoustical Features

Because of several environmental causes, the received acoustic signal from a gunshot may be affected by different disturbances which can provoke errors in muzzle blast and shockwave parameters measurement [17]. Outdoor sound propagation may vary greatly at distances of hundreds of meters from the source due to spatially varying atmospheric

conditions, diffraction around obstructing objects, and reflections from the ground and other surfaces. Furthermore, acoustical propagation may be affected by wind, temperature gradients, and frequency-dependent atmospheric absorption. As the specific environmental conditions will vary greatly from one location to another, it is not possible to generate a single, comprehensive mathematical model applicable in general. Thus, empirical models are generally required to account for the environmental behaviour. Ordinary audible sounds fall within a pressure range that is well modelled with linear differential wave equations, but the shock waves caused by supersonic projectiles give rise to nonlinear behaviour in the air. Although it is possible to predict the acoustical properties of a particular environment using standard mathematical techniques, the absorption, attenuation, and reflection properties of objects encountered by the acoustical disturbance must be known (or accurately estimated) for both the linear and nonlinear propagation regimes.

2.2.1 Reverberant Environment: Sound Reflection and Multi-Path

Gunshot sounds propagating over ground will encounter attenuation by acoustic energy losses due to scattering. Smooth and hard ground will generally produce less absorption than rough surfaces such as vegetation. For what concerns the shock wave, if solid surfaces are present nearby, the passing shock wave cone will be partially absorbed and partially reflected by the surface. Thus, a microphone in the vicinity will pick up both the original shock wave and the reflected shock wave with a delay corresponding to the path length difference. Higher frequencies (shorter wavelengths) are almost always attenuated

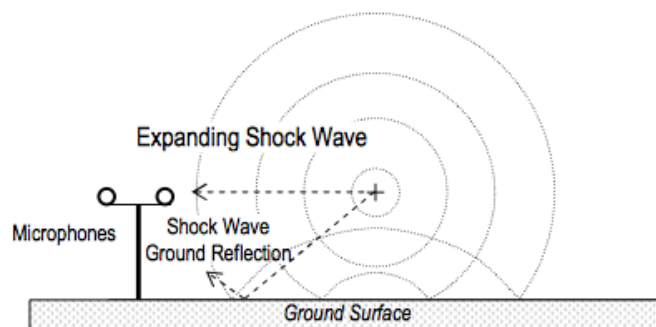


FIGURE 2.9: Simplified geometry of shock wave ground reflection on solid surfaces

more than lower frequencies. Measurements in forested areas show that absorption and scattering can achieve significant attenuation. There can also be significant attenuation by acoustic shadowing when a solid obstacle obscures the "Line of Sight" (*LoS*) between the source and the acoustic sensor, this particularly affects the muzzle blast rather than

the shock wave, as already anticipated in 2.1. In other words, the very short duration of the muzzle blast and the acoustic shock waves act like acoustic impulses, so gunshot recordings obtained in complicated surroundings will consist of the convolution of the gun's report and the acoustic impulse response of the local reverberant environment, resulting in substantial temporal smearing. In fact, reverberant recordings will typically contain more information about the acoustical surroundings than about the gun or the projectile. Deconvolution of the gunshot from the reverberant background can be attempted, but no completely reliable means to accomplish this task for gunshots has been published.

2.2.2 Effect of Wind

If the air itself is moving due to wind, the sound propagating through the air will be carried in the moving air mass. If the gun is stationary, the muzzle blast sound waves will emanate essentially spherically. The spatial motion of the muzzle blast wave front will therefore consist of the vector sum of the spherically expanding sound vector and the wind vector. Similarly, the shock wave cone formed by the supersonic projectile will be altered by the wind. The wind effect can be viewed as a shift in the origin of the sound propagation. In other words, the wave front launched at the source is carried by the wind as it propagates toward the receiver. The propagating wave front is systematically shifted due to the wind so that by the time it arrives at the receiver position the apparent location of the sound source (or the bullet's trajectory) has been shifted as well. Wind motion is generally accompanied by a wind gradient, with the wind speed typically faster at altitude and slower near the ground. The result is that sound waves propagating upwind are "bent" upwards and those propagating downwind are "bent" downwards. Although wind speeds are a small fraction of the speed of sound, the wind alteration causes a direction-dependent change in sound speed, and a corresponding Doppler-like frequency shift.

2.2.3 Temperature and Humidity Effects on the Speed of Sound

Even if in 2.1.2 a formula for the speed of sound depending on the temperature in degrees Celsius has already been given, it has also been specified that equation (2.5) is valid only if dry air is assumable. This paragraph will go through details about this clarification. The theoretical expression for the speed of sound c in an ideal gas is

$$c = \sqrt{\frac{P\gamma}{\rho}} \quad (2.9)$$

where:

- P is the ambient pressure;
- ρ is the gas density;
- γ is the ratio of the specific heat of gas at constant pressure to the one at constant volume.

The term γ depends on the number of degrees of freedom of the gaseous molecule. The number of degrees of freedom itself depends upon the complexity of the molecule:

- $\gamma = 1.67$ for monatomic molecules;
- $\gamma = 1.40$ for diatomic molecules;
- $\gamma = 1.33$ for triatomic molecules.

Since air is composed primarily of diatomic molecules, the speed of sound in air is approximately:

$$c = \sqrt{\frac{1.4P}{\rho}} \quad (2.10)$$

The speed of sound c in dry air has the following experimentally verified constrained values:

$$c = 331.45 \pm 0.05 \frac{m}{s} \quad (2.11)$$

for audio frequencies, at 0°C, at 1atm (760 mm Hg) with 0.03 mol-% of carbon dioxide.

2.2.3.1 Temperature Dependence

Notoriously the equation of state of air of an ideal gas is:

$$PV = RT \quad (2.12)$$

where:

- $R=8.314472 \frac{J}{molK}$ is the Universal Gas Constant;
- M is the mean molecular weight of the gas at sea level;
- T is the absolute temperature in degrees Kelvin.

Knowing then the definition of density ρ , i.e. mass per unit volume, equation (2.10) can be written in an alternative way, in order to emphasise the dependence upon temperature of the speed of sound:

$$c = \sqrt{\frac{1.4RT}{M}} \frac{m}{s} \quad (2.13)$$

Equation (2.13) reveals the temperature dependence and pressure independence of the speed of sound. An increase in pressure results in an equal increase in density. Therefore there is no change in velocity due to a change in pressure. But this is true only if the temperature remains constant. Temperature changes cause density changes which do not affect pressure. Thus density is not a two-way street. Changes in pressure affect density but not vice versa. Since R and M are constants, the speed of sound may be shown to have a first-order dependence on temperature as follows:

$$c = c_0 \sqrt{\frac{T}{273.15}} \quad (2.14)$$

where T is the temperature in kelvins and c_0 equals the reference speed of sound under defined conditions, as anticipated in paragraph 2.1.2. The speed of sound is seen to increase as the square root of the absolute temperature (see figure 2.10). Substituting centigrade conversion factors and the reference speed of sound gives equation (2.5).

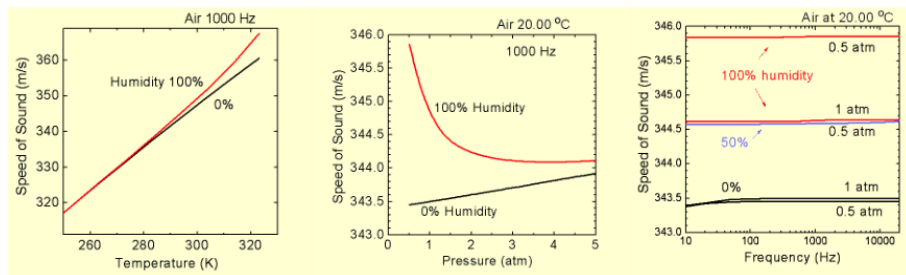


FIGURE 2.10: Speed of Sound variations with Temperature, Humidity and Frequency

2.2.3.2 Effect of Variable Temperature

The air temperature in the atmosphere is generally not uniform, and as indicated by equation (2.5), there will be spatial variations in the sound speed (higher speed in warmer air, lower speed in cooler air). In daytime, particularly in summer months, the ground surface is often warmer than the upper air. In this situation sound propagation tends to be bent upwards slightly due to the temperature gradient: the wave front in the warm air near the surface propagates faster than the wave front in cooler air higher above the ground. Conversely, in winter or at night, when the temperature near the ground is likely to be lower than that of the upper air, sound waves tend to be bent

downwards. The combined effects of wind and temperature gradients can cause sound levels measured at some distance from the source to be very different from predictions based on geometrical spreading and atmospheric absorption considerations alone. These differences may be 20 dB or more over distances of a few hundred meters.

2.2.3.3 Humidity Effects

All previous discussion assumed dry air. Attention turns now to the effects of moisture on the speed of sound. Moisture affects the density of air and hence, from equation (2.9), the speed of sound in air. Moist air is less dense than dry air (not particularly obvious), so ρ in equation (2.9) gets smaller. This causes an increase in the speed of sound. Moisture also causes the specific-heat ratio γ to decrease, which would cause the speed of sound to decrease. However, the decrease in density dominates, so the speed of sound increases with increasing moisture (see figure 2.10).

2.2.3.4 Frequency Dependent Sound Absorption due to Humidity

The relative humidity of the air causes frequency-dependent sound absorption due to molecular thermal relaxation. The attenuation is found to increase monotonically with increasing frequency (see figure 2.11), and is the greatest for relative humidity in the 10-30% range. Below 4 kHz, the worst-case humidity attenuation corresponds to 0.1 dB/m.

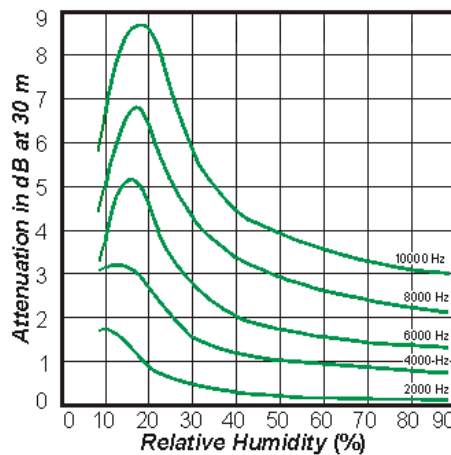


FIGURE 2.11: Frequency Dependent Sound Absorption versus Humidity variations

2.3 Shock Wave and Muzzle Blast Discrimination

2.3.1 Discrimination Issues

Determining the location of prospective snipers requires a precise estimation of the trajectory of the bullet. This goal is achievable by measuring the arrival times of the acoustic energy at several locations in space. It has been just discussed that for a supersonic projectile fired from a gun, both the acoustic shock wave and the muzzle blast may be observed. For acoustic sensor systems attempting to determine a bullet trajectory, the challenge is to first correctly classify the transient signal as either a shock wave or a muzzle blast and then calculate the direction of arrival via appropriate arrival times across a sensor array. An incorrect classification will result in large estimation errors of the projectile's trajectory. Thus proper discrimination between the arrival energies associated to shock wave and muzzle blast must be achieved, this stems from the propagation pattern of the two different energies. Briefly resuming the most important results of section 2.1, here it is only remembered that the muzzle blast energy will appear as a far field plane wave originating from the gun, while the shock wave will propagate in the form of a cone trailing the bullet with angle θ_M which is given by equation (2.3) and is called **Mach Angle**. The worst case estimation is when an acoustic system incorrectly classifies an arrival of a shock wave as a muzzle blast, consequently the estimate of shot origin will be perpendicular to the shock cone, determining the trajectory of the projectile with an error of 90° . The discrimination process is generally trivial in the case of ideal shock wave and muzzle blast arrival energies as the shock wave duration time is in the microsecond range (paragraph 2.1.2) while the muzzle blast duration time is on the order of few milliseconds (paragraph 2.1.1). This can be extremely challenging when the shockwave has lost substantial high frequency content. The change in spectral characteristics can stem from propagation over a long distance, propagation over snow covered terrain or arriving from a non perfect reflector. Also, many practical systems may have insufficient bandwidth to preserve rise time characteristics of the shock wave, in fact at least 100 kHz of bandwidth are necessary and this particular requirement was fundamental in the research of the microphone chosen for this thesis, which has a bandwidth of about 125 kHz. Muzzle blast signatures can also be affected by multi-path situations producing faster rise times than expected (paragraph 2.2.1).

2.3.2 Approaches presented in Literature

Because of the rapidly increasing interest on the shooter localization problem (due to the already mentioned changes in modern warfare) and being the shockwave and muzzle

blast discrimination probably the most challenging task in such application, this is a very explored topic in scientific literature. There are several characteristics of acoustic shockwaves and muzzle blasts that distinguish their detection and signal processing algorithms from regular audio applications. Both events are transient by their nature and present very intense stimuli to the microphones. This is increasingly problematic with low cost electret microphones, designed for picking up regular speech or music. Although mechanical damping of the microphone membranes can mitigate the problem, this approach is not without side effects. The detection algorithms have to be robust enough to handle severe nonlinear distortion and transitory oscillations. The more robust the detection algorithm is, the less performant (thus less expensive) the microphone can be, but very robust algorithms require high computational capability on the sensor boards. Otherwise, less robust detection algorithms allow to use less performant (thus cheaper) sensor boards, but professional and expensive microphones become necessary, with better characteristics in terms of bandwidth (100 kHz at least) and handleable sound pressure peak (in excess of 160 dB *re* μP). In what follows, the most relevant approaches are reported and examined, these can be grouped in two main classes:

- Time Domain analysis;
- Joint Time-Frequency and Wavelet analysis.

Essentially the first kind of discrimination approaches can guarantee much better performance despite of very strict requirements in terms of sampling frequency, while the second kind can reach lower performances but with more reasonable sampling frequencies. The choice done for this work belongs to the second class, as it will be detailed later 2.3.2.3, thus in this paragraph 2.3.2 only one algorithm belonging to the first class is going to be exposed (the one that is thought to be the best), while for the second class various approaches are going to be explained, including the one used in this thesis (the Spectrogram approach).

2.3.2.1 A Time Domain Analysis: a State Machine fed by Zero Crossing encoding

This detection algorithm is the one used by Vanderbilt University Research Group, in particular it is mentioned in [6]. Here it has been decided to expose this algorithm as an example for the time-domain ones, because it was applied in experiments using an actual sensor network with real shots obtaining satisfying results. Going through details, with this approach two different detectors are applied in order to find the most important characteristics of the shock wave and the muzzle blast in the time-domain using simple

state machine logic. In the case of [6], the detectors are implemented as independent IP cores within a FPGA. Recalling what is detailed in section 2.1.2, the most conspicuous characteristics of an acoustic shock wave are the steep rising edges at the beginning and end of the signal, particularly at the beginning the edge is very short, around 2-3 μs . This feature requires an high sampling frequency to detect the shock wave rising edge if working in the time-domain: to detect an edge that lasts about 2-3 μs , a sampling period of 1 μs or smaller is needed, i.e. a sampling rate of at least 1 MS/s, which is a very critical specific. Also, the length of the N-wave is fairly predictable and is relatively short (200-300 μs). The shockwave detection core is continuously looking for two rising edges within a given interval. The state machine of the algorithm is shown in figure 2.12. The input parameters are the minimum steepness of the edges (D, E), and the bounds on the length of the wave (L_{min}, L_{max}). The only feature calculated by the core is the length of the observed shockwave signal. In contrast to shockwaves, the muzzle blast

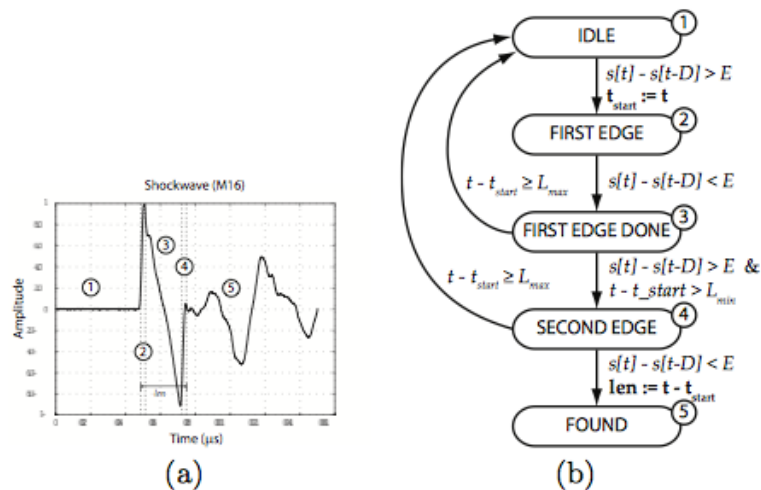


FIGURE 2.12: Shockwave signal generated by a 5.56 x 45 mm NATO projectile (a) and the state machine of the shockwave detection algorithm (b)

signatures are characterized by a long initial period (1-5 ms) where the first half period is significantly shorter than the second half. The state machine (figure ??) does not work on the raw samples directly, but is fed by a zero crossing (ZC) encoder. After the initial triggering, the detector attempts to collect those ZC segments which belong to the first period (positive amplitude) while discarding too short segments (called garbage in image 2.13), effectively implementing a rudimentary low-pass filter in the ZC domain. After it encounters a sufficiently long negative segment, it runs the same collection logic for the second half period. The initial triggering mechanism is based on two amplitude thresholds: one static amplitude level and a dynamically computed one. The latter one is essential to adapt the sensor to different ambient noise environments and to temporarily suspend the muzzle blast detector after a shock wave event (oscillations

in the analog section or reverberations in the sensor enclosure might otherwise trigger false muzzle blast detections). As demonstrated in [6], this time-domain shock wave and

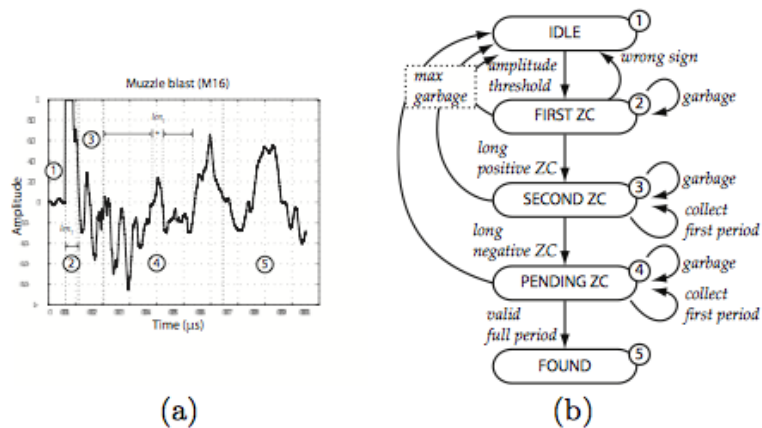


FIGURE 2.13: Muzzle blast signature produced by an M16 assault rifle (a) and the corresponding state machine algorithm detection logic (b)

muzzle blast discrimination approach results in very high shooter localization accuracy when deployed in a synchronized microphone array, on the other hand such application imposes to have ADCs which perform a huge sample frequency, in the order of MS/s.

2.3.2.2 Joint Time-Frequency and Wavelet Analysis

As anticipated before, less computationally heavy and less requiring, in terms of sampling rate, techniques to classify signals as either shockwaves or muzzle blasts use joint time-frequency spectrograms to analyze the typical separation of the shockwave and muzzle blast transients both in time and frequency. Such approach is justified by the fact that the energy produced by these two sources have differing durations, rise times and arrival times, hence the separation can be done by joint time-frequency analysis techniques. Typically the energy of the muzzle blast signature is in the $300 \div 1000$ Hz range, while the energy of the shockwave is mainly in the $3 \div 7$ kHz range. The JTF techniques for this application mostly proposed in literature are:

1. Short Time Fourier Transform (STFT);
2. Smoothed Pseudo Wigner-Ville Distribution (SPWVD);
3. Discrete Wavelet Transformation (DWT).

1. The STFT is perhaps the simplest and most commonly used JTF technique employed in general signal processing. The STFT results are easy to interpret, and

generally characterize a signal frequency content over time. The largest drawback to this approach is that resolution obtainable in both the frequency and time domain are related and drive in opposite directions, i.e. better time resolution will produce poorer frequency resolution and vice-versa. The results of the STFT are generally squared to produce a Spectrogram which represents the signals power distribution over time and frequency. The equation defining the spectrogram for a given signal is:

$$Spectrogram(t, w) = (STFT(t, w))^2 = \left(\int s(\beta)w(\beta - t)e^{-j\omega t} d\beta \right)^2 \quad (2.15)$$

Further the experimental results reported in [20] prove that while this technique can easily separate shockwave and muzzle blast signals, it greatly distorts shockwave features in time. The same observation was done after the experiments carried out for this thesis, as it will be exposed in next chapter 4. For this reason, a STFT analysis for muzzle blast and shockwave discrimination and times of arrival extraction, cannot perform:

- an accurate shockwave duration estimation;
- a detailed shockwave signal shape characterization.

These considerations carry to other two conclusions:

- because of the first item, a system using the spectrogram cannot implement a caliber estimation using equation (2.8);
- because of the second item, a system using the spectrogram cannot actuate a precise bullet speed (Mach Number) extraction from the shockwave shape, as proposed in [21].

2. The DWT technique is attractive for two main reasons:

- The DWT can be implemented as a successive bank of digital filters, which is a highly efficient process to implement.
- A key feature of the DWT is that the time and frequency resolutions vary over the JTF space. At higher frequencies it guarantees better time resolution with reduced frequency resolution while at low frequencies a better frequency resolution with reduced time resolution.

This varying time resolution is important for the shockwave and muzzle blast data which both have sharp initial rise times but radically different durations. The DWT preserves the time location of the initial singularity allowing correct measurements of events onsets for both classes of signals [22].

3. The final JTF technique applied is the SWVD. A key reason for using the SWVD is that it clearly shows a signals frequency changes over time as compared with STFT. The application of this technique to muzzle blast and shockwave was investigated also by (ESTIMATION OF SHOCKWAVE PARAMETERS OF PROJECTILES VIA WIGNER-TYPE TIME-FREQUENCY SIGNAL ANALYSIS). The SWVD is however very compute intensive for long duration signals. While the SWVD offers better frequency resolution performance than the STFT, it does suffer from cross-term interference patterns which can make the spectrum difficult to interpret. Even if the smoothing function reduces cross-term effect, excessive smoothing reduces the SWVD to a STFT under specific conditions, negating any benefits. The equation defining the SWVD is (2.16):

$$\widetilde{W}_f(x, t) = 2 \int_{-\infty}^{+\infty} f(\tau + \tau') f(\tau - \tau') w_f(\tau') w_f^*(-\tau') e^{-2ix\tau'} d\tau' \quad (2.16)$$

Where:

- $w(t)$ represents the smoothing function;
- the star symbol $*$ is the conjugation operator.

An interesting work of comparison between these three techniques has already been done by Brian t. Mays in [20], validated by both simulated and experimental data. The most important result of this comparison is that no single method appears to be optimum:

- The STFT method is the most classic and well understood technique and is the less computationally expensive algorithm but has very poor ability to accurately resolve transient event onset times while simultaneously resolving frequency. The DFT performs well with respect to resolving onset times and isolating frequency content but still suffers when similar signals are interference sources.
- The SWVD technique shows promises but is costly to implement in terms of required processing power. This may limit its application to many realisable systems. Beyond the calculation complexity, the cross-term interference patterns which plague the SWVD must be fully investigated for known interference sources.
- For systems which must balance performance with implementation complexity, the DWT appears to be the best of the three candidate JTF technique described above.

These final conclusions were decisive for the shockwave-muzzle blast discrimination technique choice done for this thesis.

2.3.2.3 Final Considerations and Signal Analysis Technique Choice

At first, the research was limited to the class of JTF approaches because the hardware at disposal could not afford a sampling frequency high enough to deploy a time-domain signal analysis. As explained in chapter 1, some of the most important design specifications were cheapness, low power consumption and low compute expensiveness of the motes. Thus it is clear that the best choice in order to satisfy these specifications is the STFT, i.e. the Spectrogram approach. Recalling the considerations previously exposed about the STFT, in particular the fact that STFT greatly distorts shockwave time characteristics, choosing such technique means automatically renouncing to perform weapon caliber, bullet mach number and miss distance estimation.

Chapter 3

Shooter Localization Problem

3.1 The Geometry of the problem and derivation of Shooter Equations

In this section general notions about the geometry of the bullet trajectory and shockwave cone will be given, and a set of three parameters, which are necessary for next mathematical calculations, will be computed, particularly:

- Shockwave Time of Arrival, denoted as t_k , where k indicates the k -th sensor;
- Muzzle Blast Time of Arrival, denoted as T_k
- Time Difference of Arrivals between shockwave and muzzle blast (TDOA)

The notation and also figure 3.1 have been imported from [10]. In figure 3.1:

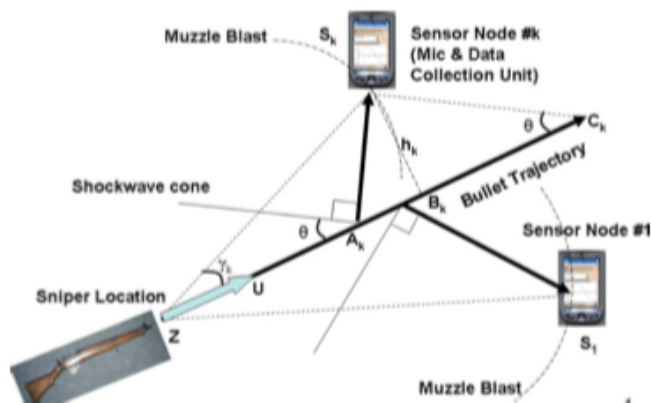


FIGURE 3.1: Geometry of the bullet's trajectory and shockwave cone

- Z denotes the position of the shooter;
- S_k denotes the position of the shooter;
- U is the unit vector in the direction of the bullet trajectory;
- γ_k is the miss angle, i.e. the angle between the trajectory of the bullet and the line joining the sniper location and the sensor location;
- θ is the Mach Angle as already specified in equation (2.3).

The shock wave propagates perpendicular to the cone surface and reaches the sensor, S_k . The point where the shock wave radiates towards the sensor is denoted by A_k . By the time the shock wave reaches the sensor S_k , the bullet has traveled from A_k to C_k and the miss distance is given by $h_k = \|S_k - B_k\|$, where $\|B\|$ denotes the norm of the vector B . Consequently, the expressions for T_k and t_k are respectively relations (3.1) and (3.2):

$$T_k = \frac{\|S_k - Z\|}{c} \quad (3.1)$$

$$t_k = \frac{\|A_k - Z\|}{cM} + \frac{\|C_k - A_k\|}{c} \quad (3.2)$$

Where:

- M is the Mach Number;
- c is the Speed of Sound.

The first part of expression (3.2) for t_k corresponds to the time it takes the bullet to travel the distance $\|A_k - Z\|$, and the second part corresponds to the time taken by the shock wave to propagate from A_k to the sensor S_k . In the time the shockwave propagates from A_k to S_k , the bullet travels from A_k to C_k , thus the bullet travels from Z to C_k . Then, t_k can be rewritten as equation (3.3):

$$t_k = \frac{\|C_k - Z\|}{cM} = \frac{\|B_k - Z\|}{cM} + \frac{\|C_k - B_k\|}{cM} = \frac{1}{cM} [(S_k - Z)^T U + h_k \cot \theta] \quad (3.3)$$

where ".^T" represents the transpose operator. The derivation in equation (3.3) uses the fact that $\|B_k - Z\|$ is the projection of the vector $S_k - Z$ onto the trajectory of the bullet U . Since the following relation (3.4) is true

$$\cos \gamma_k = \frac{(S_k - Z)^T U}{\|S_k - Z\|} \quad (3.4)$$

and also the following system (3.5)

$$\begin{cases} \sin \theta = \frac{1}{M} \\ \|B_k - Z\| = \|S_k - Z\| \cos \gamma_k \\ h_k = \|S_k - Z\| \sin \gamma_k \end{cases} \quad (3.5)$$

then the shockwave time of arrival t_k can be written as (3.6):

$$t_k = \frac{\|S_k - Z\|}{c} (\sin \theta \cos \gamma_k + \cos \theta \sin \gamma_k) = \frac{\|S_k - Z\|}{c} \sin(\theta + \gamma_k) \quad (3.6)$$

Therefore the TDOA τ_k is given by (3.7):

$$\tau_k \triangleq T_k - tk = \frac{\|S_k - Z\|}{c} [1 - \sin(\theta + \gamma_k)] \quad (3.7)$$

TDOA information τ_k is very important for this thesis, not only because it is fundamental for dissertation about asynchronous sensors network in section 3.4, but especially because this thesis experimental validation is based on a single sensor approach (see section 4.3) and such an approach, under certain hypothesis, uses just this single input data.

3.2 Introduction to different algorithm typologies

Different kinds of sensor networks imply different localization algorithms. These sensor networks can be roughly grouped in two main typologies:

1. Multi-Channel Acoustic Sensor Networks
2. Single-Channel Acoustic Sensor Networks

3.2.1 Multi-Channel Acoustic Sensor Networks

Nowadays scientific research on this topic is principally focused on sensor fusion algorithms in networks with multichannel acoustic sensor nodes, which means that each mote is capable of computing its own estimation of shooter position, or, at least, his bearing, using the informations of different Time of Arrivals (ToAs) of shockwave and muzzle blast at each microphone, [1][18], [23]. Hence the sensor fusion algorithm, that runs on a centralized controller, performs a statistical optimisation of the shooter position calculation: collecting all the estimations computed by the peripheral motes and assigning a reliability value to each of them, knowing also the sensors positions, the

central algorithm returns a probabilistic optimised shooter position estimation. Statistical optimum criteria commonly applied for this application are the typical Maximum Likelihood Ratio (ML) or Minimum Mean Square Error (MMSE). But recently, with research progresses, these approaches are becoming more and more sophisticated and the algorithms are no more similar to simple ML or MMSE estimators, even if these two criteria continue to be the below estimation philosophy. The increasing interest in the direction of this kind of acoustic sensor networks is probably due to the fact that in actual shooter detection systems currently deployed in the battlefield by some armies, as far as it is known by US, British and Canadian Armies (for example, SWATS by QinteQ North America, Boomerang Warrior-X by BBN Technologies, PinPoint by BioMimetic Systems), each sensor operates separately to the others and these systems are not design to exploit the sensor network layout of all the soldier to help increasing accuracy. This lack of networking is likely deriving from the difficulty in such sensor networks to manage multi path, which is often very considerable in urban warfare, and mobility (obviously soldiers continuously and rapidly move during a firefight).

3.2.2 Single-Channel Acoustic Sensor Networks

The other typology, instead, is referred to sensor networks which employ nodes with a single microphone [24] , [13], consequently at each node only a pair of ToA information (one for shockwave and one for muzzle blast) is available, hence the ToAs from all the sensors are needed for computing the estimation of the shooter position. This calculation is performed by a central controller. An implied hypothesis for the correct working of this sensor networks is clearly the possibility of correct communication between the sensors, or better, between each sensor and the controller. So a reliable network setup is necessary and, anyhow, the environment in which the sensors will be deployed must be deeply studied, in order to avoid impossibility of sensor-controller communications because of, for instance, an obstacle in the Line of Sight (*LoS*) or too strong multi path affection eccetera.

3.2.3 Final Observations

Two important observations that can be easily done when looking for advantages, disadvantages and differences between the two kinds of networks illustrated above, are:

- Multi-Channel Acoustic Sensor Networks are more robust to network configuration changes, environmental obstacles on the *LoS* or, in the worst case, if the network is down every sensor should be able to continue computing its own shooter position estimation

- Using microphones situated very close to each other, Multi-Channel Acoustic Sensor Networks require the utilization of high performance microphones (in order to be able to detect very small differences in the ToAs) and motes with a quite good computational capability hardware embedded

From these observations, and considering the network design specification of this thesis (detailed in chapter 1), it is immediate to conclude that for this work the typology of interest was the second one: Single-Channel Acoustic Sensor Networks, in order to have low cost sensors with not much performant hardware onboard. Thus in what follows the multi-channel acoustic sensor networks will not be considered any more, but not for a lack of importance, just because they lie outside of the main topic of the thesis. Another relevant differentiation that must be underlined when dealing with sensor networks for shooter localization is the one between:

- Synchronous Sensor Networks
- Asynchronous Sensor Networks

There is no need of further exposing what are the difference between these two categories, since their names are quite self-explaining. Both of them presents advantages and disadvantages, essentially synchronous sensor networks guarantee better performances despite hard issues with synchronization implementation, while asynchronous sensor networks are easier to deploy but result in worse performances (performances highly dependent on the reliability of the statistical model of signal and noise). For the sensor network project presented here, both of this approaches were examined because it was not found a reason to consider one of them depreciabile at all ore one definitely optimum with respect to the other. Hence in next sections, synchronous and asynchronous shooter localization algorithms for single-channel sensor network are going to be investigated.

3.3 Synchronous sensors network shooter localization Algorithm

Even if the article [1] deals with a multi-channel sensor network, some of the algorithms there proposed can be extended to the single-channel sensor case, with the appropriate observations. The fundamental difference that must be understood before proceeding is the following:

- in [1] the microphones detecting shockwaves and muzzle blasts are very close to each other (being located on the same sensor board), thus their spacing is order of

magnitude smaller than the distance to the sound source, this allows to approximate the approaching sound wave front with a plane (**Far Field Assumption**);

- on the contrary, in the case of single-channel sensor networks, depending on the particular nodes distribution within the sensor field related to the specific network, the Far Field Approximation could not be so thoughtlessly assumable, being the microphones spacing, in general larger than in the case described above.

For the purpose of this paper, thinking to a sensor field not much large with respect to the hypothetical distance of the shooter, the Far Field condition will be considered assumable anyway, but it is important that the proposed processing returns estimations that will be more precise with increasing shooter distance at constant microphone spacing. Now that this clarifications are defined, algorithm details can be analyzed.

3.3.1 Direction of Arrival

The first step of the sensor fusion is to calculate the muzzle blast and shockwave Direction of Arrival (DoA) also called Angle of Arrival (AoA). Since in this thesis purpose the designed network needs to be as simple and as cheap as possible, the minimum sensors number is going to be always considered. Consequently now the problem is formalized for 3 microphones, which is the minimum number in order to compute shockwave and/or muzzle blast DoA. To compute shooter position may be necessary two groups of three microphones instead, hence 6 microphones, in the case each group is not able to detect both shockwave and muzzle blast but only one of them (it will be clear in section 3.3.2). Let P_1 , P_2 and P_3 be the positions of the microphones ordered by time of arrival $t_1 \leq t_2 \leq t_3$. First a simple geometry validation step is applied. The measured time difference between two microphones cannot be larger than the sound propagation time between the two microphones:

$$|t_i - t_j| \leq \left| \frac{P_i - P_j}{c} \right| + \epsilon \quad (3.8)$$

Where:

- c is again the speed of sound;
- ϵ is the maximum measurement error.

If this condition does not hold, the corresponding detections are discarded. Now let us define some new variables:

- $v(x, y, z)$ is the normal vector of the unknown direction of arrival;

- $r_1(x_1, y_1, z_1)$ is the vector from P_1 to P_2 ;
- $r_2(x_2, y_2, z_2)$ is the vector from P_1 to P_3

Now consider the projection of the direction of the motion of the wave front (v) to r_1 divided by the speed of sound (c). Thus equation (3.9) returns how long it takes the wave front to propagate from P_1 to P_2 :

$$vr_1 = c(t_2 - t_1) \quad (3.9)$$

The same relationship (equation (3.10)) holds for r_2 and v :

$$vr_2 = c(t_3 - t_1) \quad (3.10)$$

Further, v is a normal vector, hence (3.11):

$$vv = 1 \quad (3.11)$$

Moving from vectors to coordinates using the dot product definition leads to a quadratic system (3.12):

$$\begin{cases} xx_1 + yy_1 + zz_1 = c(t_2 - t_1) \\ xx_2 + yy_2 + zz_2 = c(t_3 - t_1) \\ x^2 + y^2 + z^2 = 1 \end{cases} \quad (3.12)$$

The solution steps are here omitted since they are straightforward, but long.

3.3.2 Worst Case shooter position estimation

Here a special case is considered, a kind of worst case. The sensor field consists of 2 groups composed by 3 microphones each one. The worst case assumption consists in the fact that each group succeeds in detecting either the shockwave ToA and AoA (with the method explained in the previous section 3.3.1) or the muzzle blast ToA and AoA. Otherwise, if the acoustic sensors are capable of detecting both shockwave and muzzle blast, a network of three microphones is sufficient. Let us define:

- P_1 and P_2 are the centroids of the two different groups of microphones;
- P is the shooter position;
- t is the instant when the shot was fired.

Please notice that P_1 and P_2 have a different meaning with respect to the same symbols when adopted in previous section 3.3.1, in order to avoid confusion. Obviously both P and t are unknown. The muzzle blast detection is at position P_1 with time t_1 and AoA u . The shockwave detection is at P_2 with time t_2 and AoA v . Both u and v are normal vectors. It is shown below that these measurements are sufficient to compute the position of the shooter (P). A possible geometrical representation is shown in figure 3.2: Let P'_2 be the point on the extended shockwave cone surface where PP'_2 is perpendicular

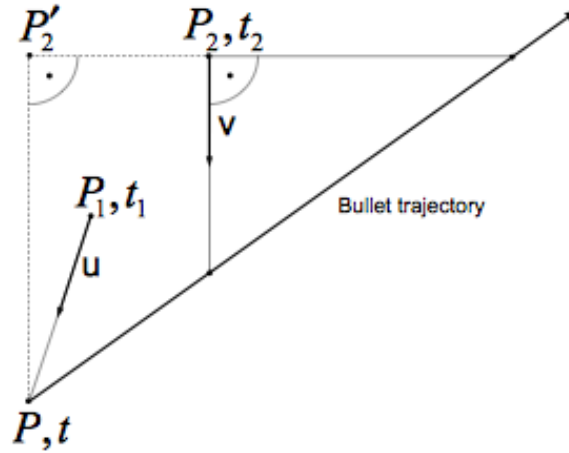


FIGURE 3.2: Section plane of a shot (at P) and two groups of three microphones each one at P_1 and P_2

to the surface. Note that PP'_2 is parallel with v . Since P'_2 is on the cone surface which hits P_2 , a sensor at P'_2 would detect the same shockwave time of arrival (t_2). The cone surface travels at the speed of sound (c), so we can express P using P'_2 in equation (3.13):

$$P = P'_2 + cv(t_2 - t) \quad (3.13)$$

P can also be expressed from P_1 with equation (3.14):

$$P = P_1 + cu(t_1 - t) \quad (3.14)$$

yielding relation (3.15):

$$P_1 + cu(t_1 - t) = P'_2 + cv(t_2 - t) \quad (3.15)$$

$P_2P'_2$ is perpendicular to v , this implies equation (3.16)

$$0 = (P'_2 - P_2)v \quad (3.16)$$

yielding relation (3.17), which contains only one unknown t :

$$0 = [P_1 + cu(t_1 - t) - cv(t_2 - t) - P_2]v \quad (3.17)$$

Expliciting t , relation (3.18) is easily obtained:

$$t = \frac{(P_1 - P_2)v}{c} + \frac{vut_1 - t_2}{uv - 1} \quad (3.18)$$

Substituting the value of t obtained by equation (3.18) in relation (3.14), P is immediate to compute.

Now it is very simple to consider the case in which 3 microphones are enough, i.e. when they succeed in detecting both shockwave and muzzle blast. This situation can be seen as a particular case of the one that has been just described but with $P_1 = P_2$. In this case t is given by equation (3.19)

$$t = \frac{vut_1 - t_2}{uv - 1} \quad (3.19)$$

Again, substituting the value of t obtained by equation (3.19) in (3.14), P is returned. Here it was assumed that the shockwave is a cone which is only true for constant projectile speeds, which is an approximation assumed for the whole thesis. In reality, the angle of the cone slowly grows. The decelerating bullet results in a smaller time difference between the shockwave and the muzzle blast detections because the shockwave generation slows down with the bullet. A smaller time difference results in a smaller range, so the above formula underestimates the true range. Anyhow, it can still be used with a proper deceleration correction function. A list of all approximations assumed for this work and a summary of their effects on measurements and estimation will be presented in chapter 4.

3.4 Asynchronous sensors network shooter localization Algorithm

As showed in the previous section, algorithms for sniper localization in synchronous sensors network use TDOA of either muzzle blast or shockwave signals at various sensors. In a network in which sensors are not able to get synchronized together, such algorithms are clearly not available. In [10] the sensors used are single microphone sensors capable of recording both muzzle blast and the shock wave. The algorithm for sniper localization uses the TDOAs between muzzle blast and shock wave at each sensor. This requires that the events, namely, muzzle blast and shock wave, are detected at each sensor.

From these detections, the time of arrival (TOA) of muzzle blast and shock wave are estimated. The difference in TOAs of muzzle blast and the shock wave provides the TDOA between them (see also section 3.1 and relation (3.7)). Also in this case, bullet speed is assumed to be constant. The algorithm description is long and quite tricky and, being this a design for a synchronous network, beyond the interest of this paper. The interested reader is redirected to [10], here it is just remarked that this algorithm relies on the assumption of gaussian TDOA measurement errors.

3.5 A special case: Single (Single-Channel) Sensor

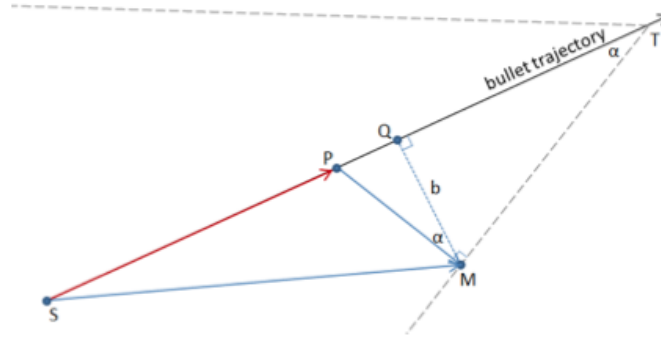
As highlighted in some papers [13], [5], the main disadvantages of a networked approach is its reliance on the network itself: it would be interesting to investigate if a single-channel single sensor node is able to somehow localize the shooter. It was found in [5] that under certain conditions, a sensor capable of detecting both shockwave and muzzle blast can compute the range to the shooter, even with a single shot. These conditions are:

- known weapon caliber;
- known bullet average speed, or, equivalently, its mach number.

This section is very important, being this approach the only one validated with experiments in this work. Anyhow there is a little difference between what is described in [5] and [13] and what was implemented with this thesis: in [5] and [13] it is assumed that the sensor is able to perform a shockwave period measurement, thus a miss-distance estimation is possible too inverting Whitham equation (2.6). Here such processing is not possible, in fact it has already been explained in sections 2.3.2.2 and 2.3.2.3, that the technique used for shockwave and muzzle blast discrimination and estimation is the Spectrogram and this precludes the possibility of an accurate shockwave duration estimation. Thus here another assumption must be added:

- miss distance is given

Now that all the hypothesis have been disclosed, details of the ranging algorithm are going to be declared. Considering figure 3.3, points S and M represent the locations of the shooter and the microphone detecting the acoustic events, respectively. Let us denote the range, i.e. the Euclidean distance between points S and M , with $d_{M,S}$.

FIGURE 3.3: Geometry of a shot from S and the acoustic events observed at M

Assuming line of sight (LOS) conditions, the detection time of the muzzle blast can be written as (3.20):

$$t_{mb} = t_{shot} + \frac{d_{M,S}}{c} \quad (3.20)$$

where t_{shot} is the time of shot. That is, the muzzle blast travels at the speed of sound from S to M , and is, therefore, detected $\frac{d_{M,S}}{c}$ seconds after the weapon is fired. Since the shockwave does not originate from the muzzle but from the bullet traveling along its trajectory, we need to consider the time traveled by the bullet from the muzzle to a point P on the trajectory, as well as the time traveled by the wavefront of the shockwave from point P to the microphone. P is defined such that the vector \vec{PM} is a normal vector of the shockwave front. For simplicity, let us assume for now that the bullet travels at a known constant speed v_{bullet} . Notice that if the mach number is given, v_{bullet} can be derived by equation (2.2). The shockwave detection time can be written as:

$$t_{sw} = t_{shot} + \frac{d_{P,M}}{c} + \frac{d_{S,P}}{v_{bullet}} \quad (3.21)$$

The measured shockwave-muzzle blast TDOA can be expressed as (3.22):

$$t_{mb} - t_{sw} = \frac{d_{M,S}}{c} - \frac{d_{S,P}}{v_{bullet}} - \frac{d_{P,M}}{c} \quad (3.22)$$

Since we assume a constant bullet speed v_{bullet} , the shockwave front has a conical shape, such that the angle between the trajectory and the conical surface is (3.23)

$$\alpha = \arcsin\left(\frac{c}{v_{bullet}}\right) \quad (3.23)$$

Knowing α and the miss distance $d_{Q,M}$, $d_{S,P}$ and $d_{P,M}$ can be expressed with relations respectively (3.24) and (3.25):

$$d_{S,P} = d_{S,Q} - d_{P,Q} \quad (3.24)$$

$$d_{P,M} = \frac{d_{Q,M}}{\cos(\alpha)} \quad (3.25)$$

where, using the Pythagorean Theorem, $d_{S,Q}$ is (3.26):

$$d_{S,Q} = \sqrt{d_{S,M}^2 - d_{Q,M}^2} \quad (3.26)$$

and $d_{P,Q}$ is equal to (3.27):

$$d_{P,Q} = d_{Q,M} \tan(\alpha) \quad (3.27)$$

Therefore in equation (3.22) the only unknown variable is the range to the shooter $d_{M,S}$. Its solution results in a closed-form formula (3.28) that is extremely fast to evaluate on the target hardware (in our case using a Zynq 7020, in particular on a ZedBoard):

$$d_{M,S} = \frac{1}{2(c^4 - v_{bullet}^4)} (P - 2\sqrt{Q}) \quad (3.28)$$

where P and Q are given respectively by relations (3.29) and (3.30):

$$\begin{aligned} P = & -2v_{bullet}^3 d_{Q,M} \sqrt{v_{bullet}^2 + c^2} - 2(t_{mb} - t_{sw}) c^3 v_{bullet}^2 \\ & + 2c^2 d_{Q,M} v_{bullet} \sqrt{v_{bullet}^2 + c^2} - 2(t_{mb} - t_{sw}) c v_{bullet}^4 \end{aligned} \quad (3.29)$$

$$\begin{aligned} Q = & -2c^4 v_{bullet}^4 d_{Q,M}^2 + 2(t_{mb} - t_{sw})^2 c^6 v_{bullet}^4 + \\ & (t_{mb} - t_{sw})^2 c^4 v_{bullet}^6 - 2c^7 d_{Q,M} (t_{mb} - t_{sw}) v_{bullet} \sqrt{v_{bullet}^2 + c^2} \\ & + c^8 (t_{mb} - t_{sw})^2 v_{bullet}^2 + 2c^8 d_{Q,M}^2 + 2v_{bullet}^5 d_{Q,M} \\ & \sqrt{v_{bullet}^2 + c^2} (t_{mb} - t_{sw}) c^3 \end{aligned} \quad (3.30)$$

While relaxing the assumption of constant bullet speed and incorporating a weapon-specific deceleration constant in the equations would result in more precise results, finding the solution for $d_{M,S}$ would require numerical methods, making the algorithm more computationally expensive.

Chapter 4

Experimental Validation

4.1 Experimental Setup

As previously anticipated, for this work an experimental campaign was conducted. Only one microphone was at disposal, thus a single-channel single sensor case was simulated, in particular the microphone used was an **Ultramic250K** by Dodotronic [25], developed with the support of *CIBRA* (*Centro Interdisciplinare di Bioacustica e Ricerche Ambientali*) of University of Pavia (cibra@unipv.it). The chosen signal analysis technique is a JTF one, specifically the **STFT**, i.e. the **Spectrogram**, detailed in section 2.3.2.2. The purposes of the experiments were:

1. investigate the suitability of the specific microphone for such application;
2. obtain a real signal in a known geographic and climatic situation for studying the environmental effects on the signal's distortion;
3. execute a shooter ranging with single sensor algorithm exposed in section 3.5 in order to validate not the algorithm itself (which reliability and resolution have already been proven in [5], but the specific shockwave and muzzle blast detection technique adopted, i.e. the STFT).

Referring to the third purpose, that represents probably the most important, we can say that as far as we know, the Spectrogram for gunshot acoustic signal analysis has always been studied comparing its results, in terms of shockwave and muzzle blasts time of arrivals, with the ones obtained using more performant techniques or looking at the time domain signal. Until now no one appeared to have already verified which are the effects of applying the STFT for gunshot signal analysis on the final result of a shooter localization: the shooter position estimation, or, in this case, the shooter distance. The

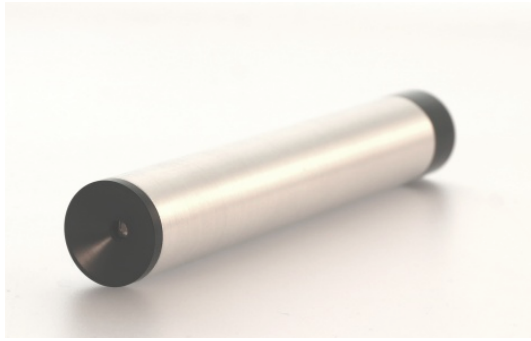


FIGURE 4.1: Ultramic microphone

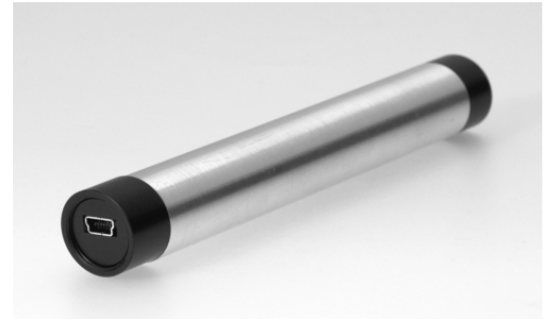


FIGURE 4.2: Ultramic microphone with the mini-USB port visible

conclusions of this novel investigation (section 4.3) encourage further studies in this direction.

4.1.1 The Microphone

4.1.1.1 Specifications and Hardware Description

The specifications for the microphone that were considered before its choice were:

- a Bandwidth of at least 100kHz;
- a sampling frequency possibly around $150 \div 200$ kS/s;
- capability of handling sound pressure peaks that can be in excess of 160dB *re* μ Pa.

Ultramic250K by Dodotronic (see figures 4.1, 4.2) was chosen, it satisfies the required specifications having the following characteristics:

- a Bandwidth of 125kHz;
- a sampling frequency of 250kS/s;
- it is able to handle sound pressure peaks that can be in excess of 160dB *re* μ Pa.

Ultramic is a MEMS (Micro Electro-Mechanical Systems) microphone, thus it is very sensitive with a good signal/noise ratio and small form factor. An integrated low pass 8th order filter is provided in order to reduce aliasing artifacts. A typical frequency response of MEMS sensor is showed in figure 4.3 It is an ultrasound microphone with integrated digital to analog converter with the sampling rate given above (250 kHz for the version we used). The samples are quantized with a 16 bit resolution. A 32 bit 80 Mhz microcontroller is integrated (see figure 4.4). It is provided of a USB device

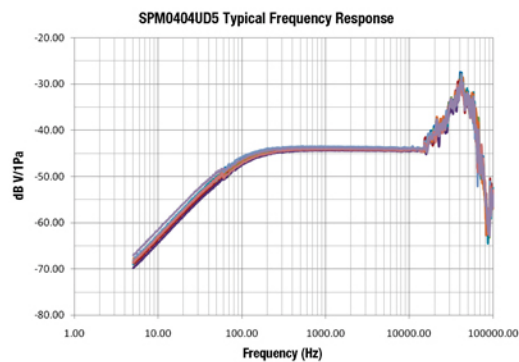


FIGURE 4.3: A typical frequency response of MEMS sensors.



FIGURE 4.4: Ultramic internal electronics

full speed port 2.0 with a mini B USB connector (see figure 4.2). The USB port allows an easy connection to any PC or Mac computer, the device is recognized as an *HID* (human interface device) microphone so no driver installation is required. It appears as a single channel audio input device, however if recording in stereo the two channels will appear identical. Ultramic can be easily opened, via a screw on the back. There are two switches, near the front of the microphone (see figure 4.5), that can be turned on and off with a fingernail, in order to change amplification between the three hardware levels (table 4.1). In details, the amplification is provided by two different amplifiers: the first has a fixed gain of 32 dB, while the second has a variable gain which can be set using the two switches as described above.

Even if Ultramic and its software (see section 4.1.1.2) were developed thinking on a Windows host PC, the Ultramic series was tested also with other operating systems running on PCs but also on some tablets. In the table below (table 4.2) a resume of compatibilities for Ultramic 250K is reported:

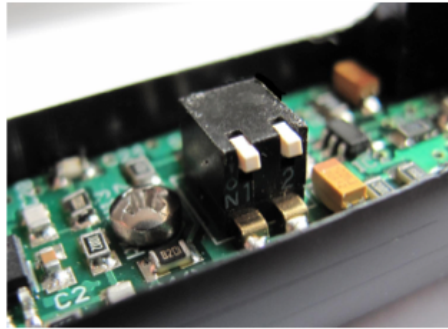


FIGURE 4.5: Ultrasonic gain configuration switches

Level	Configuration	Total Gain
High	both up	72 dB
Medium	left down, right up	58 dB
Low	both down	35.5 dB

TABLE 4.1: Ultrasonic gain configuration settings

OS	Compatibility
Windows Vista	YES
Windows XP	NO
Windows 7	YES
Windows 8 and 8.1	NO
Linux Ubuntu	YES
Linux Debian	YES
Linux Android	Not all tablet tested
Mac Os	YES

TABLE 4.2: Ultrasonic os compatibilities

4.1.1.2 Ultrasonic's Software Description

The proper software to work with Ultrasonic microphones series is **SEA**. It was developed to have real-time sound analysis capabilities on a Windows PC and it's well suited to be used in conjunction with Ultrasonic200k and Ultrasonic250K. Mainly developed for bioacoustic studies, this software can be used for a wide range of applications requiring real-time display of sounds and vibrations. It allows to display in real-time the spectrographic features of sounds acquired by any sound device compatible with Windows, including Windows 7 64bit. It also allows to record, play and analyze standard 16 bit *.wav* files, either stereo or mono. SEA is available in two versions: **SeaPro**, commercial version for professional use, and **SeaWave**, free. For this thesis, SeaWave was used. SeaWave is a light version of the SeaPro package. This free version has some limitations, however it can satisfy most user needs. Recording functions are limited to single file

recording with 680MB max file size. Other limitations are about the advanced settings panel for fine tuning the aspect and behaviour of the program and for saving/restoring the user configuration. For Linux and Mac users: SeaWave and the UltraMics work well in virtual windows under both Linux and Mac OSX (VMWARE and PARALLELS have been positively tested). Another simpler option is to use the CrossOver utility to "encapsulate" SeaWave and make it appearing as a Mac App. SeaWave can display and record real time spectrogram and sampled signal in the time domain. It is very deeply configurable by the user who can set: the kind of windowing (Hamming, Hanning etcetera), the number of points of the FFT, the file format in which the record must be saved and many other parameters. In figure 4.6 is represented a bat recording with SeaWave, the image is obtained from [26].

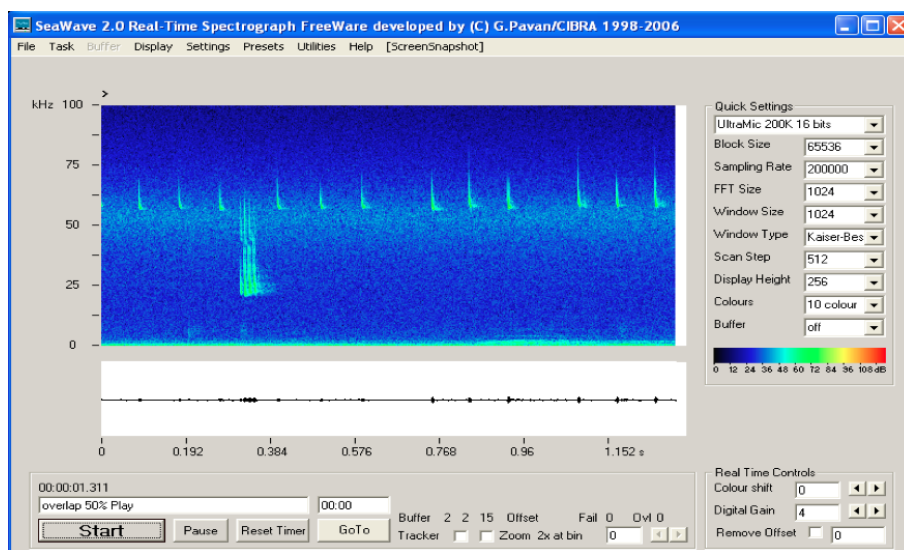


FIGURE 4.6: bat recording with SeaWave

4.1.2 Sensor Configuration

For the experiments, the microphone was positioned over a tripod about 1m high (in figure 4.7). It was then connected to a PC running Linux Ubuntu 14.04 LTS and SeaWave ran on a Windows 7 VMWare virtual machine, as showed in figure 4.8. In order to minimize microphone saturation problems, the amplification level was set as "Low" (gain of 35.5 dB), so with both the switches down (see section 4.1.1.1 and table 4.1).

4.1.3 The Weapons

Two kinds of weapons were used, with two different calibers:



FIGURE 4.7: Sensor Configuration: the microphone is positioned on a tripod and connected to a PC via USB

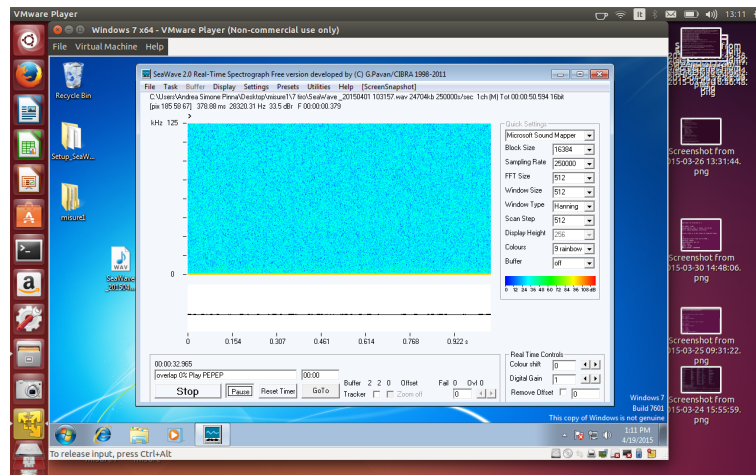


FIGURE 4.8: SeaWave running on Windows 7 virtual machine on a Linux PC

- 5.56mm NATO, in figure 4.9;
- .308 Winchester, in figure 4.10.

In figure 4.11 a comparison between the two complete rounds with bullet in case can be found. In table 4.3 some of the most relevant ballistic specifications of the weapons used are reported. Notice that:

- l is the bullet's length;
- d is the bullet's diameter;
- C stands for *Bullet Coefficient*: a parameter already described in section 2.1.2 and specified by equation (2.7);
- v_{bullet} is the bullet's initial speed (also referred to as *muzzle velocity*), for the next calculations the bullet's average speed is considered equal to v_{bullet} , doing a strong approximation.

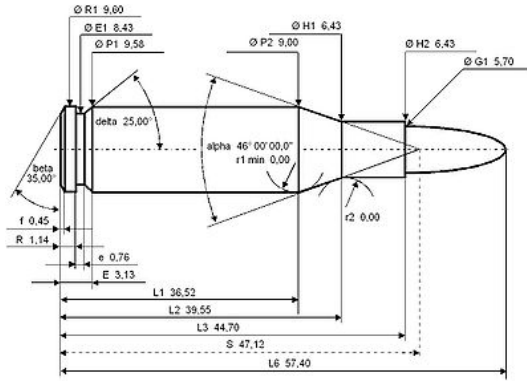


FIGURE 4.9: 5.565mm NATO Cartridge dimensions.

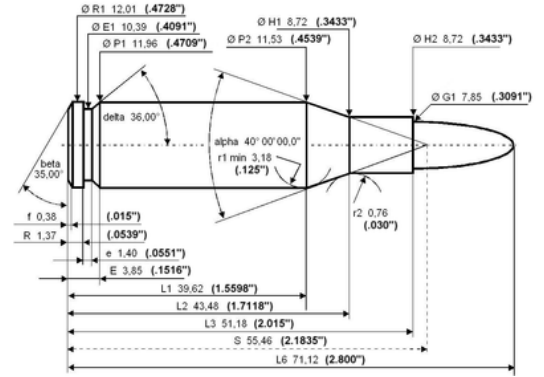


FIGURE 4.10: .308 Winchester Cartridge dimensions.



FIGURE 4.11: A comparison between a 5.56mm NATO bullet (left) and a .308 Winchester bullet (right)

Caliber	d	l	v_{bullet}	C
5.56mm NATO	5.56 mm	45 mm	850 m/s	21.2367
.308 Winchester	7.8 mm	51 mm	\simeq 810 m/s	72,5785

TABLE 4.3: Weapons Characteristics

The precise models of the weapons cannot be disclosed here because they represent confidential information. For the same reason, information about the location where the experimental campaign took place will not be conveyed.

4.1.4 The Measurements

The experimental campaign consisted of 7 shots made with the two weapons described above at different distances from the target and locating the sensor at different positions.

In figure 4.12, a simple representation of the geometry of the experiments has been drawn, where:

- the blue points are the sensor positions;
- the red points are the positions of the shooter;
- the yellow square is the target;
- the green background represents vegetation (particularly a forest);
- the light grey background represents tarmac;
- the dark grey background is a building;
- the brown arrow is the trajectory of the bullets;
- the light blue arrows indicate the miss distances.

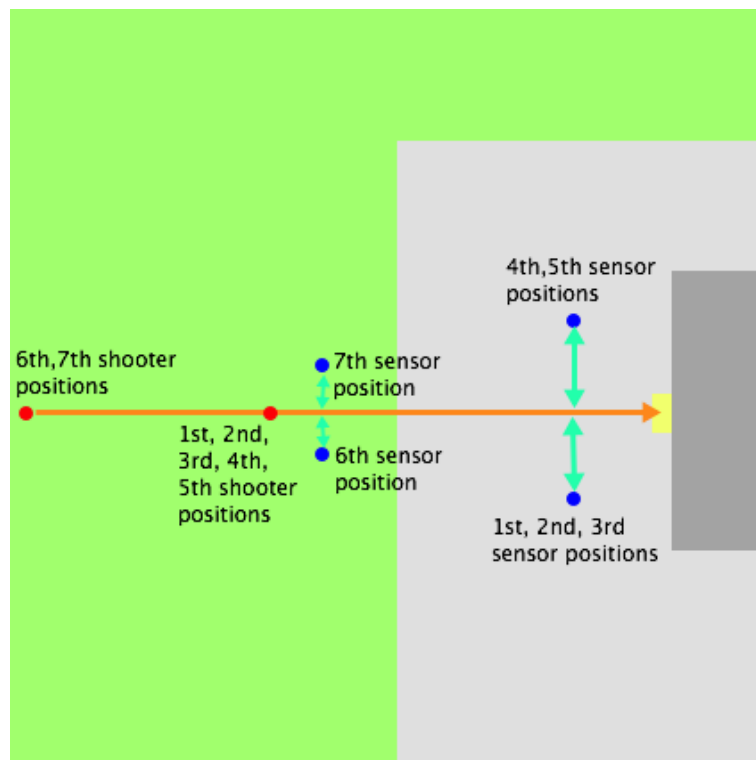


FIGURE 4.12: Geometry of the experimental campaign

In table 4.4 are summarized all the parameters concerning with the experiments, which are needed to perform the consecutive processing and to interpret the results. Where:

- under the voice *Weapon* is reported just the caliber, but the muzzle speed is intended to be the one reported in table 4.3 for each caliber;

Shot	Weapon	d	b
1 st	5.56 mm NATO	49 m	7.5 m
2 nd	5.56mm NATO	49 m	7.5 m
3 rd	.308 Winchester	49 m	7.5 m
4 th	.308 Winchester	49 m	7.5 m
5 th	5.56mm NATO	49 m	7.5 m
6 th	.308 Winchester	107 m	4 m
7 th	.308 Winchester	107 m	4 m

TABLE 4.4: Shots Acquisition Settings

- d is the distance shooter - sensor;
- b stands for miss distance.

4.2 Signals Analysis and Features Extraction

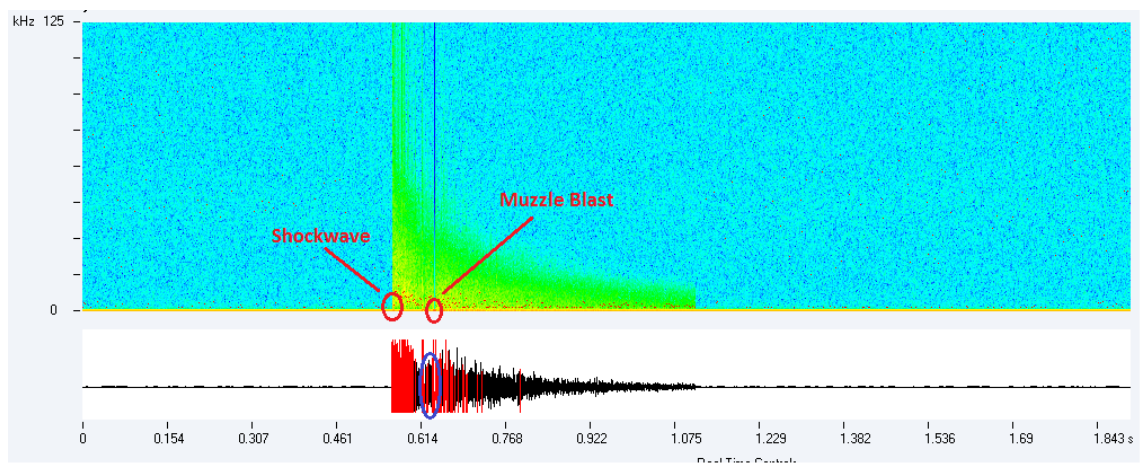
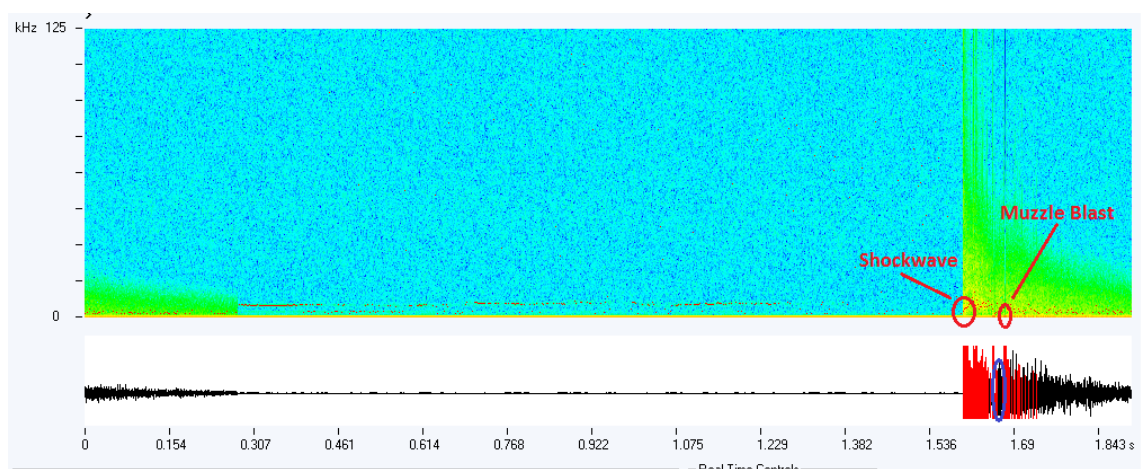
As previously anticipated (in section 4.1.2), the signals acquired by Ultramic microphone were then analyzed using SeaWave software running on a Windows 7 VMWare virtual machine on a PC with Linux Ubuntu 14.04 LTS operating system. SeaWave reports both a coloured spectrogram and a time-domain amplitude representation of the sampled signal. Further details on the Digital Signal Processing implemented for these experiments with SeaWave are:

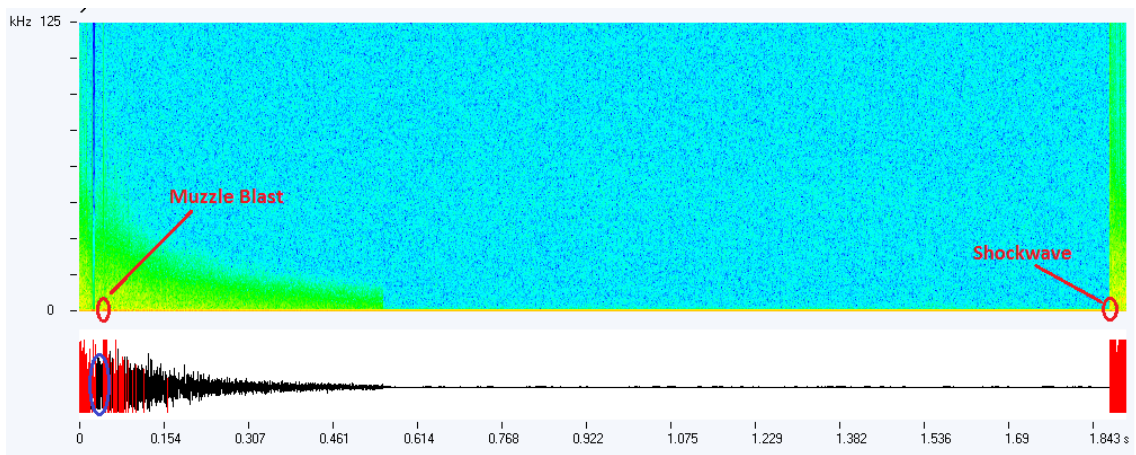
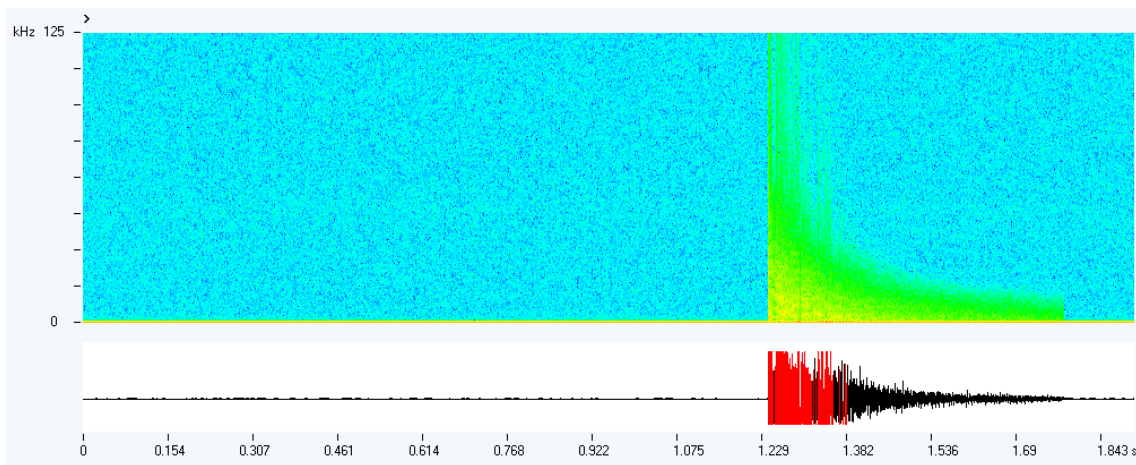
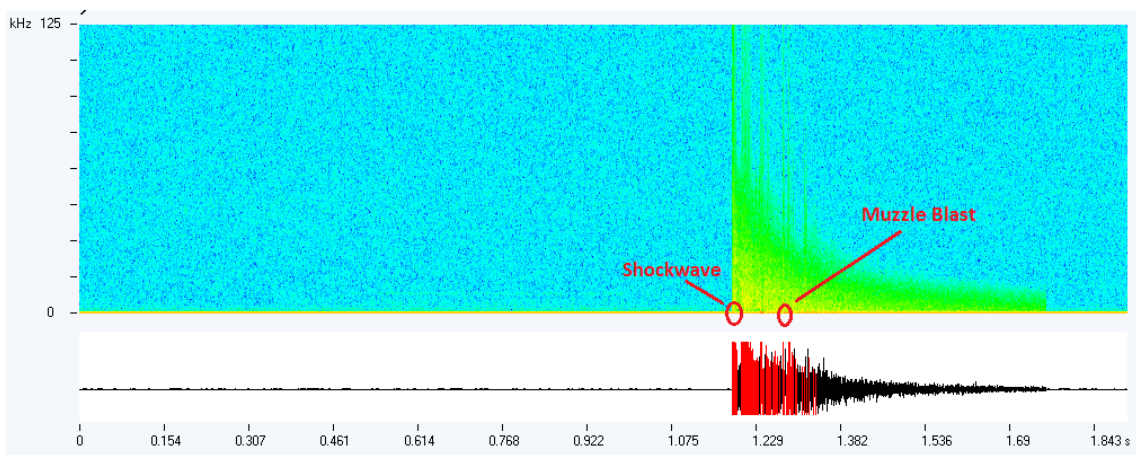
- *FFT* size: 512 samples;
- window type: *Hanning*;
- window size: 512 samples;
- scan step: 512 samples;

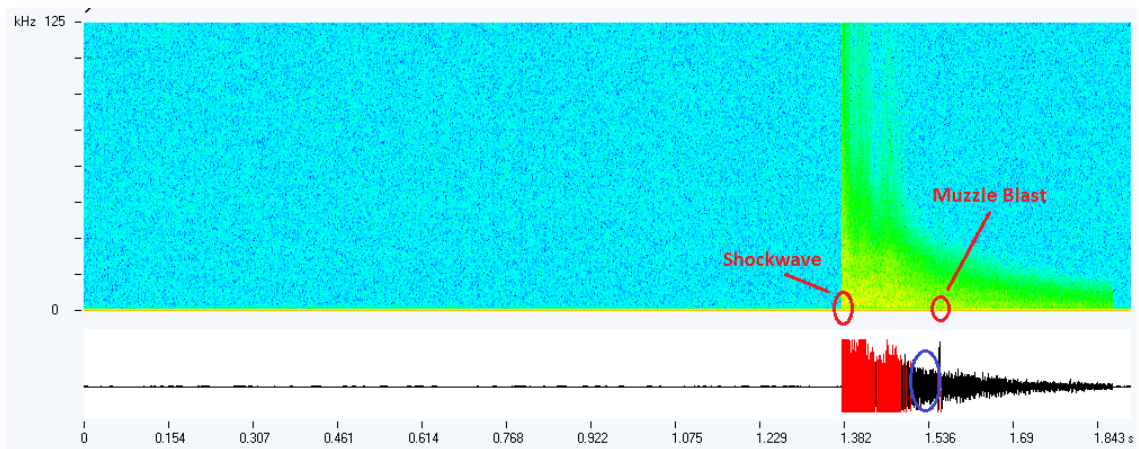
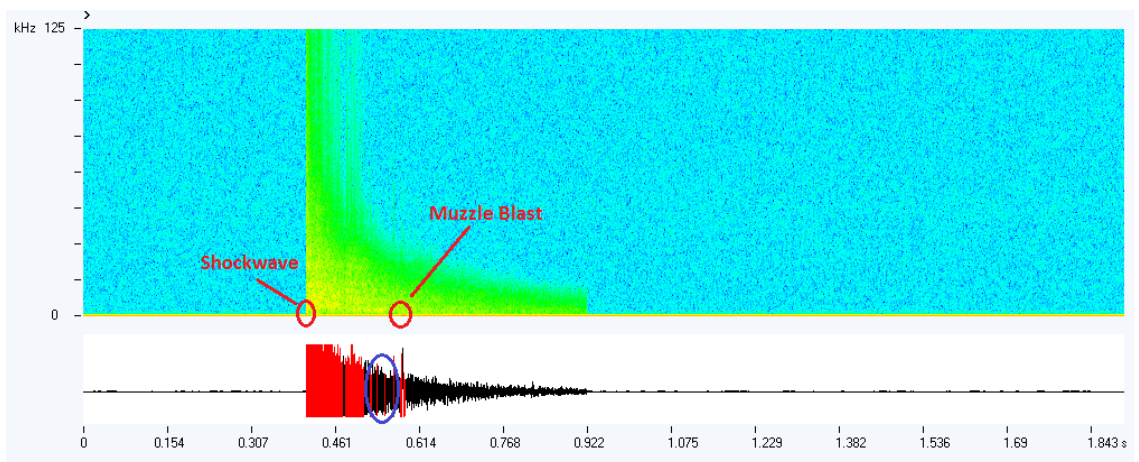
Notice that being the window size and the scan step equal to each other, means that there is no window overlapping. In figures 4.13, 4.14, 4.15, 4.16, 4.17, 4.18, 4.19:

- the upper graph is the spectrogram and the lower one is the sampled signal in the time domain;
- the x-axis is the time (in samples) for both the graphs, while the y-axis is the spectrum (in kHz) for the spectrogram, and the amplitude for the time-domain signal;

- in the time-domain signals are somewhere present red stripes, these indicate that, at the corresponding samples, the microphone was in saturation;
- the spectrogram amplitude (in dB) is suggested by the different colours: briefly, the amplitude is increasing if going from dark blue to red, passing through green and yellow;
- the red circles added in almost all the spectrograms highlight shockwave and muzzle blast detections;
- the blue circles added in almost all the time-domain signals highlight the part of the signal between the last repetition of the shockwave (due to reverberations) and the muzzle blast.

FIGURE 4.13: 1st Shot SpectrogramFIGURE 4.14: 2nd Shot Spectrogram

FIGURE 4.15: 3rd Shot SpectrogramFIGURE 4.16: 4th Shot SpectrogramFIGURE 4.17: 5th Shot Spectrogram

FIGURE 4.18: 6th Shot SpectrogramFIGURE 4.19: 7th Shot Spectrogram

4.2.1 Considerations and TDOA calculation

- Looking at the graphs, a very important consideration can be immediately made: on seven measurements, only one resulted in no shockwave and muzzle blast detection, in particular the measurement of the fourth shot (figure 4.16). Another remarkable characteristic of these graphs is the conspicuous affection by multi path. The reader can have a feeling of it by observing that: the shockwave duration is known to last about $200 \div 300 \mu\text{s}$, depending on various factors (see section 2.1.2), thus on the graphs reported here it should appear almost as a single stripe, hence every wide stripe (often red because of the saturation) before the low signal period (blue circle) is a shockwave reflection. Because of this strong reverberation contribution to the received signal, in the case of fourth and fifth shots, in figures 4.16 and 4.17 respectively, shockwave and muzzle blast peaks both in time and frequency domains are very hard to find. For the fourth shot, no detection was performed at all. Trying to correlate these results with the relative measurement

situations, the first information which can be extracted is that, while the weapons relative to these two measurements are different, the sensor location is exactly the same (refer to image 4.12 and look for 4th, 5th sensor position). So the reason of the problem is probably linked to the environment, not to the kind of weapon. The main noticeable difference between this position and the one of first, second and third measurements (for which the signal analysis did not present too many problems) is that in the fourth and fifth case the forest is situated not only in the front of the microphone, but also on one side. This consideration does not demonstrate anything, but can be a good start point for a deeper acoustic fading characterization of the experiments zone (which is not the aim of this work).

- A further investigation on multi path requires to make a difference between the measurements effectuated with the sensor on tarmac (1st, 2nd, 3rd, 4th, 5th), which graphs are showed in figures 4.13, 4.14, 4.15, 4.16, 4.17, and the ones effectuated with the sensor on the topsoil (6th, 7th), which graphs are showed in figures 4.18 and 4.19. The reason of this classification is the evident different signal distortion because of reverberations in the two cases. In the case of tarmac, multi path is clearly stronger than in the case of topsoil. Someone can think this was obvious, but it was not, since in this case the vegetation was not simple short grass, but a forest, thus a lot of big trees (potential sources of reflections) were present and very close both to the shooter and the sensor. Even if a large number of vegetation reverberation sources was situated within the sensor field, the shapes of the received signals lead us to conclude that tarmac is an extremely more problematic terrain with respect to topsoil in terms of acoustic multi path.
- Maintaining this classifications between the first five measurements and the last two, a last observation can be exposed, but no concerning with multi path any more, dealing instead with shooter-microphone distance. In the first five experiments, when the shooter was about 50 meters downrange, the muzzle blast peak is closer to the shockwave one, while in the last two, when the shooter was about 107 meters downrange, the two peaks time separation is bigger. This feature was expected from conclusions of chapter 2. But the remarkable consideration that can be made from these experiments is that this feature can compromise a correct muzzle blast and shockwave discrimination, especially in a reverberant environment, as it happened for the fourth shot and, in some measure, also for the fifth, which feature extraction is very hard (and will result in bad ranging resolution, see section 4.3).

In table 4.5 all the *TDOAs* computed after the experiments are reported. *TDOA* mathematical expression is given by equation (3.7).

Number of Shot	TDOA
1	71.68 ms
2	73.728 ms
3	63.48 ms
4	NO DETECTION
5	63.488 ms
6	169.9840 ms
7	172.032 ms

TABLE 4.5: computed TDOAs

4.3 Final Shooter Ranging

The last part of the experimental validation consisted of using the computed TDOAs in order to perform a ranging of the shooter. As already declared, having only one microphone at disposal, this operation was carried out by adopting the single channel single sensor approach described in section 3.5 and implementing the relative algorithm with a simple C script which was made running on a ZedBoard using Linaro OS (see Appendix A for details on how to boot Linaro on ZedBoard). The C script is reported in Appendix B. The input data of the script, and of the algorithm in general, are:

1. temperature;
2. miss distance;
3. bullet speed;
4. TDOA.

The temperature information is used for estimating the speed of sound from relation (2.5). Then, the calculated speed of sound along with all the other input data are elaborated to determine the shooter distance estimation. In table 4.6 the results are summarized .

4.3.1 Results Interpretation

Even if the number of shots is not enough to obtain any statistics, these results are anyhow worthy of consideration and encourage further investigations in the direction followed by this thesis.

Apart of the fourth shot case, in which no detection was possible, the error is confined between 0.05742 meters and 3.1083 meters. The mean absolute range estimation error

Shot	Weapon	b	TDOA	d	estimated range	error
1 st	5.56 mm NATO	7.5 m	71.68 ms	49 m	50.6472 m	1.6472 m
2 st	5.56 mm NATO	7.5 m	73.728 ms	49 m	51.8349 m	2.8349 m
3 rd	.308 Winchester	7.5 m	63.48 ms	49 m	49.6015 m	0.6015 m
4 th	.308 Winchester	7.5 m	X	49 m	X	X
5 th	5.56mm NATO	7.5 m	63.488 ms	49 m	45.8917 m	3.1083 m
6 th	.308 Winchester	4 m	169.9840 ms	107 m	106.4258 m	0.5742 m
7 th	.308 Winchester	4 m	172.032 ms	107 m	107.6462 m	0.6462m m

TABLE 4.6: Estimated Shooter Distances

is then 1,568717 m, but we recall the fact that this is not a much reliable data from a statistical point of view. Anyway, this is better than what was expected. In fact the resolution appears to be smaller than, or, at least, comparable with the one reached in [13], where is reported a mean absolute range estimation error of 3.5 m. The important outcome that must be put in evidence is that the resolution of our experiment is better than the one of [13], even though for this thesis a less complex TDOA estimation approach was adopted (the STFT) and, above all, a significantly lower sampling rate: 250 kS/s instead of 1MS/s. But it is remarkable that in [13] several shots were measured, so the error there declared is a statistically more accurate value than the one obtained from our experiments.

Notice also that the previously observed TDOA extraction complexity decreasing with increasing shooter distance, appears to put into effect a better ranging resolution. This is just an hypothesis not having sufficient informations, but the results suggest to deeper examine the possible resolution dependence on the shooter distance.

4.4 Error Sources

Errors in the estimated range to the shooter can be grouped in two main categories:

- errors that affects the TDOA estimation;
- errors that affects the ranging algorithm. The sources of these errors are various and contribute in different measures to the final performance. The error sources are going to be listed in what follows divided in two classes, depending on what of the two classes of errors described above they are related to.

4.4.1 TDOA estimation Error Sources

Fundamentally three factors belong to this kind of error sources:

- microphone performances;
- sampling rate;
- shockwave and muzzle blast discrimination signal analysis technique.

4.4.1.1 Microphone Performances

In our case, microphone performances and sampling rate are somehow related to each other, being Ultramic a MEMS, thus the ADC is integrated in the microphone itself. Anyhow, microphone performances that can deal with this application are *Sensitivity*, *Signal to Noise Ratio* and *Factor Form*. For Ultramic these are recognized to be very good, hence it is unlikely that they can have significantly affected TDOA estimations.

4.4.1.2 Sampling Rate

On the contrary the sampling rate is a very determinant factor, its importance is due to the gunshot acoustical signal features, in particular, the very fast rise times of shockwave and muzzle blast. As already explained, the sampling rate must be very high (in the MS/s order) if time domain signal analysis are adopted. Other signal analysis techniques allow lower sample rates, despite lower resolution and, above all, the impossibility (or at least difficulty) of correctly measuring some useful parameters, like the shockwave duration. In our case the sampling rate was 250 kHz and the signal was analyzed through a STFT. For systems using JTF techniques such sampling rate is quite satisfying. Anyhow, a growth of its value certainly assures better performances in terms of shockwave and muzzle blast peaks instants revelation, but it's not sure that, if continuing using a JTF technique, the ranging resolution improvement due to a sampling rate increase will counterbalance the hardware cost increase.

4.4.1.3 Shockwave and Muzzle Blast discrimination Signal Analysis Technique

For what concerns the signal analysis technique, i.e. the STFT (or spectrogram) for this thesis, it is necessary to remind that this technique is the less computationally expensive but also the less performing of the JTF techniques proposed in literature until now (see section 2.3.2.2). In details, the STFT appears to be the most vulnerable to multi path and noise (shockwave reflections and noise may sometimes be detected as muzzle blast or completely mask it). Further, with respect to DWT and SWVD, the STFT shows poorer ability to accurately resolve transient event onset times while

simultaneously resolving frequency content. Regardless of all these disadvantages, the ranging results obtained during this experiments are surprisingly encouraging. Anyway, the main drawback of this technique is the shockwave time features degradation and the consequent impossibility of estimating the shockwave period, thus the miss distance knowing the caliber or vice versa, and of accurately determining the bullet Mach Number (or the bullet's speed) from the shockwave as described in [21].

4.4.2 Ranging Algorithm Error Sources

The ranging algorithm error sources are essentially all the assumption made on various environmental and ballistic factors. Summarising them:

- air was assumed to be dry, thus humidity effects were not considered;
- temperature was assumed to be constant at 20°;
- wind was not contemplated;
- the bullet speed was assumed to be constant over the sensor field and equal to its initial speed.

The first three items affect the computation of the speed of sound c , as already deeply disclosed in sections 2.2.3, and 2.2.2, the interested reader is redirected to those sections for details. The speed of sound, then, is used to compute the microphone-shooter distance, so temperature, humidity and wind approximations indirectly influence ranging precision.

On the other hand, ranging algorithm precision is directly affected by the approximation of constant bullet velocity. This approximation actually consists of two different approximations:

1. the bullet speed is considered constant over the sensor field;
2. the average bullet speed is considered equal to the muzzle velocity, i.e. bullet's initial speed.

A way to avoid both of this approximation is estimating the instantaneous bullet speed at the CPA (Closest Point of Approach), i.e. when its distance to the microphone is equal to the miss distance, from the shape of the shockwave, as described in [21]. It has been already declared that this technique cannot be applied if the signal analysis method distorts the shockwave time features. This is the case of the spectrogram method and, in general, of almost all JTF techniques used in systems that cannot afford sampling frequencies in the MS/s order. Thus in our case this approach cannot be followed.

Anyhow a manner to reduce (not to delete) the effects of these assumptions exists

and can be applied also when the shockwave shape cannot be examined. Knowing the caliber of the weapon, hence the projectile, its deceleration a can be determined from the *Ballistic Coefficient* (BC), typically specified by the manufacturer. Thus, once the range to the shooter has been computed the first time, knowing the deceleration a and the muzzle velocity v_{muzzle} , equation (2.1) can be inverted in order to find v_{bullet} , the average bullet speed over the sensor field. Then, the single sensor algorithm can be performed again with this new value for the bullet speed, in order to refine the range estimation. This process can be executed more and more times. This iterative approach to relax the constant bullet speed assumption leads to more precise results, but may require numerical methods to perform the single sensor ranging algorithm, making the algorithm more computationally expensive.

4.5 Conclusions

In this paper the idea for the design of a not expensive, static and distributed sensor network for shooter localization directed to Italian Armed Forces was illustrated and validated with experiments. This design was implemented taking in consideration: the target application (i.e. FOB surveillance) and cheapness. At first, a study on the most appropriate wireless protocol for this network led us to prefer ZigBee (IEEE 802.15.4 standard), being it very power efficient and suitable for sensor networks with a not wide geographic distribution. Details on the desired star topology were then exposed: the sensor nodes are simple RFDs that perform shockwave and muzzle blast TOAs estimation and then deliver these informations to a centralized gateway, which is a FFD, in particular a PAN coordinator. Sensors synchronization can be achieved with GPS. Also particulars on the hardware were specified. It is remarkable that the centralized gateway in our idea is a ZedBoard, the development board for the Zynq SoC, which is furnished of a dual core ARM Cortex A9 processor (the Processing System) and an Artix 7 FPGA (the Programmable Logic). This architecture was chosen, because currently it is probably the most powerful in the class of embedded systems. After a dissertation on gunshot acoustic signal features, the most common in literature shockwave and muzzle blast discrimination techniques were presented. Our attention focused on the JTF (Joint Time-Frequency) signal analysis methods, because several studies show that, with respect to Time Domain signal analysis methods, they can allow to significantly reduce the sampling rate, and the low is the sampling rate, the cheaper and more power saving are the sensor boards. It was decided to try a STFT (Short Time Fourier Transform), i.e. a spectrogram, approach, being this the least computational expensive despite a certain performance degradation. The novel contribution of this paper is the experimental validation of the STFT for this application, using it in a complete shooter localization

problem with real data acquisition. In fact, until now the STFT reliability had been tested just looking at the signal analyzed in different ways and comparing the results in terms of TOAs, while here the STFT effect is checked directly on the final shooter localization. More in details, the STFT effect is tested in a shooter ranging problem with experimental data: just one microphone was at disposal, thus a single-channel single sensor approach was implemented. The ranging algorithm is performed by Zed-Board with a simple C script running on it. The microphone used for the experimental measurements was an Ultramic 250K with an ADC providing a 250KS/s sampling rate and the spectrograms were furnished by the SeaWave software. Ranging experimental results are strongly encouraging: even though the number of analyzed shots is too small to make any statistical consideration, the obtained errors (confined between 0.5 m and 3.1 m) are comparable or smaller than the ones reported in other works for which a Time Domain analysis and a much higher sampling rate (in the order of 1 MS/s) were employed. Experiments with more microphones to try a shooter position estimation (not only the range) and algorithms implementation on the boards proposed here are left to future works.

Appendix A

Getting Started Guide to ZedBoard and Linaro Ubuntu OS for Linux users

A.1 Purpose of the Paper

The aim of this appendix is to be a quick start guide to zedboard and embedded linux for completely newbies. For this paper a Mac Book Pro running Ubuntu 14.04 LTS was used, so what follows works properly for this combination of hardware and software but it is supposed to be valid for any pc running a linux distribution (with the proper adaptation). Once zedboard is purchased from [16], it can be noticed that a 4 GB SD card is present in the box. This SD card contains only a demo linux image: if linux is booted on zedboard from this sd card only a default image will be displayed on the output monitor (two linux penguins in the upper left side). Even if this action can appear useless, it may be a good way to check if everything is ok in the hardware setup (see [27]) and if the pc from which linux is booted is able to communicate correctly with zedboard via serial port. Linux on zynq can be booted in two different ways:

1. from a pc connected to ZedBoard through the JTAG port;
2. from an SD card.

For this work, the SD boot mode was chosen because it is the most appropriate for stand-alone applications. So a topic to debate is formatting the SD card and boot linux on zedboard in the SD mode. A useful, complete and official guide to such a work is [28], but following this some problems were encountered and some points seemed to be

too tricky. The solutions were found at [29]. Anyhow here a complete guide, from the beginning to the end of the process, will be given. Different file system can run on zynq, for this case Linaro was chosen for its user-friendly and Ubuntu-like GUI. The SD card was a 8GB Kingston (micro SD plus SD adapter). Four items needs to be placed onto the SD card: a Linux file system (Linaro in this case), a Linux kernel image, a BOOT.BIN file, a compiled device tree.

A.2 Boot Linaro Ubuntu on ZedBoard

A.2.1 Formatting the SD Card

The first thing to do is formatting the SD card with the correct two partitions: a FAT file system (at least 1GB) and an ext4 file system (at least 3 GB). This second partition is only necessary when using Linaro. Here is the first trouble a linux user (or at least an ubuntu user) could face with, at it was experienced for this work. Ubuntu (and maybe other linux distributions) have serious problem to read SD card readers. When inserting SD card in our Mac SD card reader, nothing happened and the device was not detect. Thinking on a hardware problem (SD card reader not working), the same thing was tried on Mac OS X, i.e. the SD card was insert in the SD reader of the pc while using the partition running Mac OS X and the device was immediately and correctly recognized. This carried us to the conclusion that probably Ubuntu has some problems in recognizing SD card readers. This hypothesis found heavy confirms on Ubuntu forums. The problem was overrun using a SD-USB adapter (Ubuntu has no problems with USB ports) and it worked. The reader is strongly recommended to adopt a similar solution if using an Ubuntu distribution. The first step is identifying the SD card device node running `lsblk` command on a bash terminal. Before inserting the storage device, this command should return back something like this on the shell A.1. After inserting the SD card you should

```
pinnone@pinnone-MacBookPro:~$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda         8:0     0 233,8G  0 disk
├─sda1      8:1     0   200M  0 part /boot/efi
├─sda2      8:2     0   88,5G  0 part
├─sda3      8:3     0   619,9M  0 part
├─sda4      8:4     0  136,6G  0 part /
└─sda5      8:5     0    7,9G  0 part [SWAP]
sr0         11:0    1  1024M  0 rom
```

FIGURE A.1: output of `lsblk` command.

instead see something like the following image. Notice A.2 that a new line containing

```

pinnone@pinnone-MacBookPro:~$ lsblk
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda         8:0     0 233,8G  0 disk
├─sda1     8:1     0   200M  0 part /boot/efi
├─sda2     8:2     0   88,5G  0 part
├─sda3     8:3     0   619,9M  0 part
├─sda4     8:4     0  136,6G  0 part /
└─sda5     8:5     0    7,9G  0 part [SWAP]
sdb         8:16    1    7,5G  0 disk
└─sdb1     8:17    1    7,5G  0 part /media/pinnone/9016-4EF8
sr0        11:0    1   1024M  0 rom

```

FIGURE A.2: output of lsblk command with sd device.

the SD card device node appears, in the example above it is `/dev/sdb` and there is only one partition called `sdb1`. Some linux distributions may not automatically display any eventual SD card partition so, to be sure to see any existent partition, the command `df` should be run from the terminal [A.3](#). The next steps are the following:

```

pinnone@pinnone-MacBookPro:~$ df
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/sda4      140832640 80491844  53163836  61% /
none              4           0           4   0% /sys/fs/cgroup
udev           4029568      12    4029556   1% /dev
tmpfs           808076      1528    806548   1% /run
none             5120         0         5120   0% /run/lock
none           4040364      156    4040208   1% /run/shm
none           102400        68     102332   1% /run/user
/dev/sda1       201633      20695    180938   11% /boot/efi
/dev/sdb1       7830528       32    7830496   1% /media/pinnone/9016-4EF8

```

FIGURE A.3: output of df command with sd device.

- unmount any existing partition on the SD card
- delete any existing partition
- create 2 primary partitions
- format partition 1 to FAT and partition 2 to EXT4

This procedure for properly formatting the SD card can be done with two different approaches: with bash commands from a terminal or with a GUI using *GParted*. Anyhow both of them are going to be illustrated here.

A.2.1.1 Command Line Approach

Call **umount** for all of the partitions mounted on the SD card to remove them [A.4](#). Once all of the partitions are unmounted, use **fdisk** tool for repartitioning. Type **fdisk**

```
pinnone@pinnone-MacBookPro:~$ sudo umount /media/pinnone/9016-4EF8
[sudo] password for pinnone:
pinnone@pinnone-MacBookPro:~$ df
Filesystem            1K-blocks      Used Available Use% Mounted on
/dev/sda4             140832640    80492136   53163544   61% /
none                   4              0           4         0% /sys/fs/cgroup
udev                  4029568        12       4029556    1% /dev
tmpfs                  808076        1528       806548    1% /run
none                   5120           0          5120     0% /run/lock
none                  4040364        156       4040208    1% /run/shm
none                  102400         72        102328    1% /run/user
/dev/sda1              201633        20695       180938   11% /boot/efi
```

FIGURE A.4: unmounting the partitions.

on the terminal to open the device and issue command **p** to print the current SD card partition table. Then **d** command will delete any existing partitions [A.5](#). After deleting the previous partitions, create the new partitions with **n** command. The 2 primary partitions must have the following features:

- Partition 1: primary partition starting from the first cylinder with a size of 1 GB;
- Partition 2: primary partition starting from the next available cylinder that ideally takes up the remainder of the available space on the SD card.

The command sequence needed to do this process is displayed in the next image [A.6](#). Once these changes are made, command **w** will write them to the SD card and exit **fdisk** [A.7](#). Run **lsblk** once again to check if the partition is active. A prompt like [A.8](#) should be seen. The final step is creating the file systems. Format partition 1 to FAT with label ZED_BOOT and partition 2 to EXT4 with label ROOT_FS. Utility **mkfs** can be used to format the partitions [A.9](#).

A.2.1.2 GParted GUI Approach

GParted is a free partition editor for graphically managing our disk partitions. It is used for creating, deleting, resizing, moving, checking and copying partitions, and the file systems on them. This is useful for creating space for new operating systems, re-organizing disk usage, copying data residing on hard disks and mirroring one partition with another (disk imaging). To install gparted open a terminal and run:

```
pinnone@pinnone-MacBookPro:~$ sudo fdisk /dev/sdb

Command (m for help): p

Disk /dev/sdb: 8026 MB, 8026849280 bytes
224 heads, 16 sectors/track, 4374 cylinders, total 15677440 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1             8192     15677439     7834624    b   W95 FAT32

Command (m for help): d
Selected partition 1

Command (m for help): p

Disk /dev/sdb: 8026 MB, 8026849280 bytes
224 heads, 16 sectors/track, 4374 cylinders, total 15677440 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
```

FIGURE A.5: deleting the partitions.

```
$ sudo apt-get install gparted
```

If installation aborts returning a warning such as the one in [A.10](#), try to run

```
$ sudo apt-get update
```

In the case of this paper this action was enough. Once installation has finished, start GParted running:

```
$ sudo gparted
```

When GParted partitioning window is displayed, select *GParted Devices - /dev/sdb* (obviously */dev/sdb* is valid in this case and it must be substituted with the current device node) [A.11](#). Depending on the presence of any existing partition, one of the following windows should be seen [A.12](#),[A.13](#). If any existing partition is detected (as

```
Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-15677439, default 2048): 2048
Last sector, +sectors or +size{K,M,G} (2048-15677439, default 15677439): +1G

Command (m for help): p

Disk /dev/sdb: 8026 MB, 8026849280 bytes
64 heads, 16 sectors/track, 15310 cylinders, total 15677440 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1            2048     2099199      1048576   83   Linux

Command (m for help): n
Partition type:
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
Select (default p): p
Partition number (1-4, default 2): 2
First sector (2099200-15677439, default 2099200):
Using default value 2099200
Last sector, +sectors or +size{K,M,G} (2099200-15677439, default 15677439):
Using default value 15677439
```

FIGURE A.6: creating new partitions (first part)

[A.13](#) highlights), it must be unmounted and deleted before doing any further partition: select the partition, right click and choose **Unmount**. Then select the partition again, right click and choose **Delete**. Be aware of the fact that to make any action executive on GParted, after the declaration of the action, the green "v" on the upper side of the window must be clicked (see figure [A.14](#)) Now the SD card is completely unallocated (as in the first figure) and two new partitions can be added. The first partition will be made of about 52MB and formatted in FAT32, it is recommendable to leave 4 MB of free space preceding the partition. To start partitioning, click the button on the upper left side of the window, showed in figure [A.16](#), then follow the example in figure [A.15](#). The second partition will be 8 GB and formatted in ext4 (figure [A.17](#)).


```
Command (m for help): p
Disk /dev/sdb: 8026 MB, 8026849280 bytes
64 heads, 16 sectors/track, 15310 cylinders, total 15677440 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1             2048        2099199     1048576    83  Linux
/dev/sdb2          2099200     15677439     6789120    83  Linux

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

FIGURE A.7: creating new partitions (second part)

```
pinnone@pinnone-MacBookPro:~$ lsblk
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda         8:0    0 233,8G  0 disk
├─sda1      8:1    0   200M  0 part /boot/efi
├─sda2      8:2    0   88,5G  0 part
├─sda3      8:3    0   619,9M  0 part
├─sda4      8:4    0  136,6G  0 part /
└─sda5      8:5    0    7,9G  0 part [SWAP]
sdb         8:16   1    7,5G  0 disk
├─sdb1      8:17   1     1G  0 part
└─sdb2      8:18   1    6,5G  0 part
sr0        11:0    1  1024M  0 rom
```

FIGURE A.8: checking new partitions with `lsblk` command

A.2.2 Linux File System

ZedBoard supports different Linux file systems, for this paper the **Linaro Ubuntu** distribution was used. It is a complete Linux distribution based on Ubuntu. It includes a graphical desktop that displays via the onboard HDMI port. Linaro executes from a separate partition on the SD card and all changes made are written to memory. The utility of Linaro is that, unlike other Linux distributions for ZedBoard as **BusyBox**, it will save files even after the device is power down and rebooted. Linaro provides several types of build that can be roughly classified in two families: distributions with no desktop (very lightweight) and distributions with desktop. Linaro file systems are

```
root@pinnone-MacBookPro:/home/pinnone# sudo mkfs -t vfat -n BOOT /dev/sdb1
mkfs.fat 3.0.26 (2014-03-07)
root@pinnone-MacBookPro:/home/pinnone# sudo mkfs -t ext4 -L rootfs /dev/sdb2
mke2fs 1.42.9 (4-Feb-2014)
Filesystem label=rootfs
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
486720 inodes, 1945344 blocks
97267 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=1992294400
60 block groups
32768 blocks per group, 32768 fragments per group
8112 inodes per group
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
```

FIGURE A.9: formatting the two partitions with **mkfs** utility

```
E: Failed to fetch http://it.archive.ubuntu.com/ubuntu/pool/main/g/gtkmm2.4/libgtkmm-2.4-1c2a_2.24.4-1ubuntu1_amd64.deb Size mismatch
E: Failed to fetch http://it.archive.ubuntu.com/ubuntu/pool/main/g/gparted/gparted_0.18.0-1_amd64.deb Size mismatch
E: Unable to fetch some archives, maybe run apt-get update or try with --fix-missing?
```

FIGURE A.10: **error** reported after the first attempt to install GParted

available at [30], it is suggested to select the 12.09 release, because it has been tested with ZedBoard. Then "precise images" must be selected and then the choice is between various distribution of the 12.09 release. For this work, at first, a distribution with a graphical desktop was tested on ZedBoard ([31], suggested in [28]). Anyhow this distribution is quite heavy and working with a separate monitor, keyboard and mouse may be not much comfortable. The lightest version of Linaro is the one called "**nano**", but it can have some problems with the ZedBoard ethernet device. In our opinion, the best tradeoff between lightweight and high performances is the Linaro alip version, which works well with openssh server, x11, tight vnc server and lxde, allowing the users to remotely control ZedBoard from their pc (without attaching monitor, keyboard, mouse etc) in a more comfortable way (see next chapters for details). Once the tarball

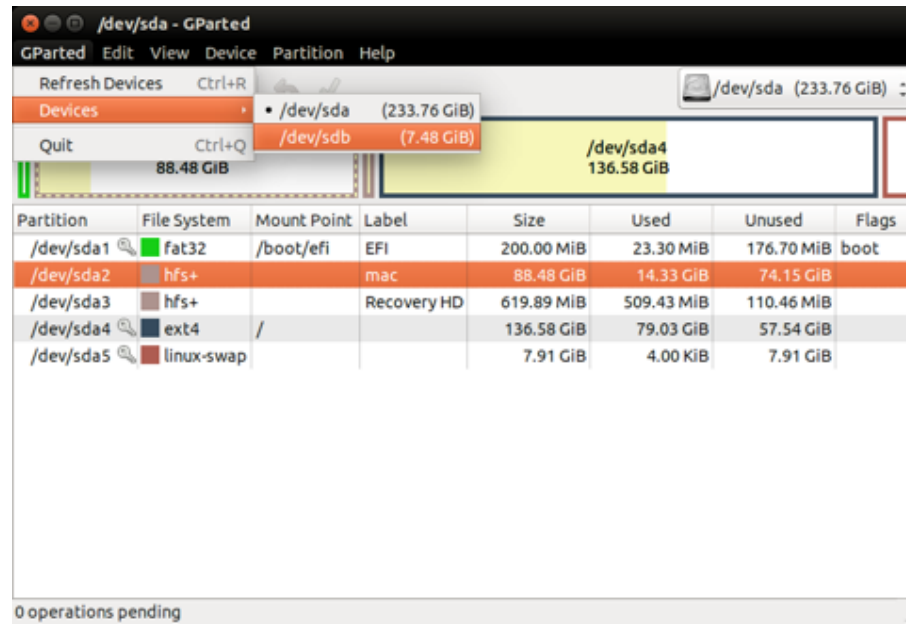


FIGURE A.11: starting GParted and selecting the SD card device node

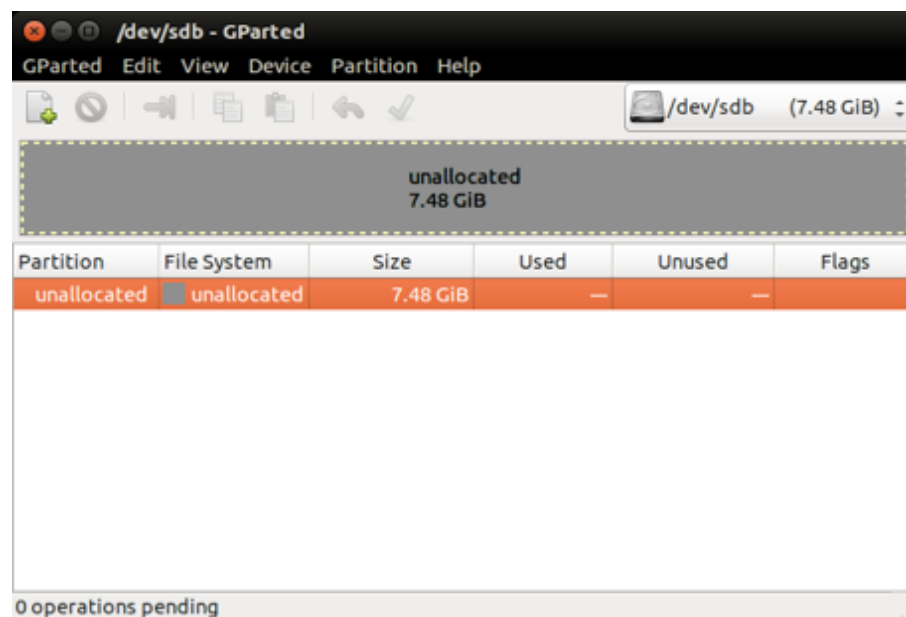


FIGURE A.12: No partition onto the SD card

is downloaded, to be ready to boot Linaro Ubuntu on Zedboard, these steps have to be accomplished:

- Copy the Linaro File System to the ext4 partition of the SD Card
- Build the Linux Kernel
- Fix the BOOT.BIN File
- Compile the Device Tree

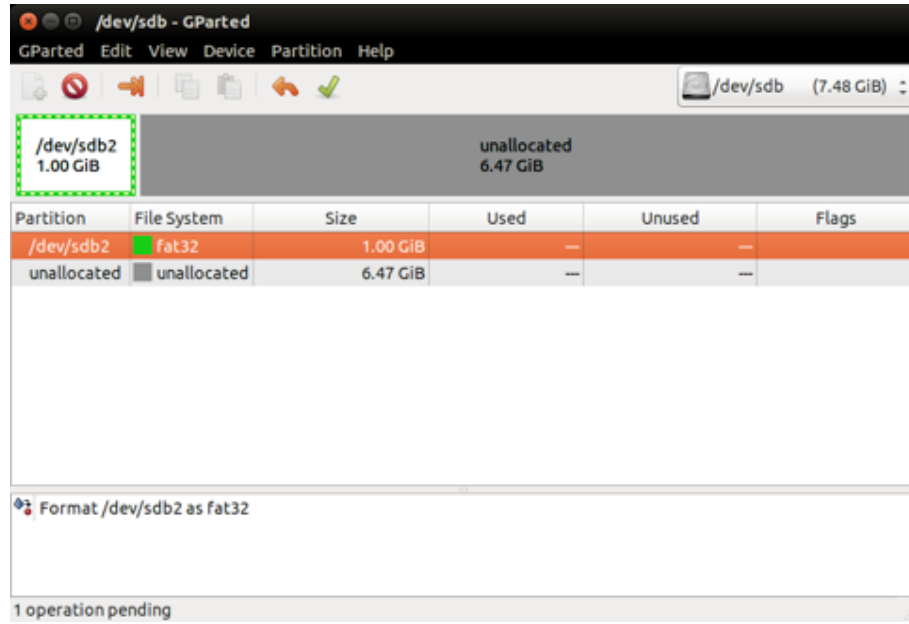


FIGURE A.13: One existing partition onto the SD card



FIGURE A.14: symbol to click for executing actions on GParted

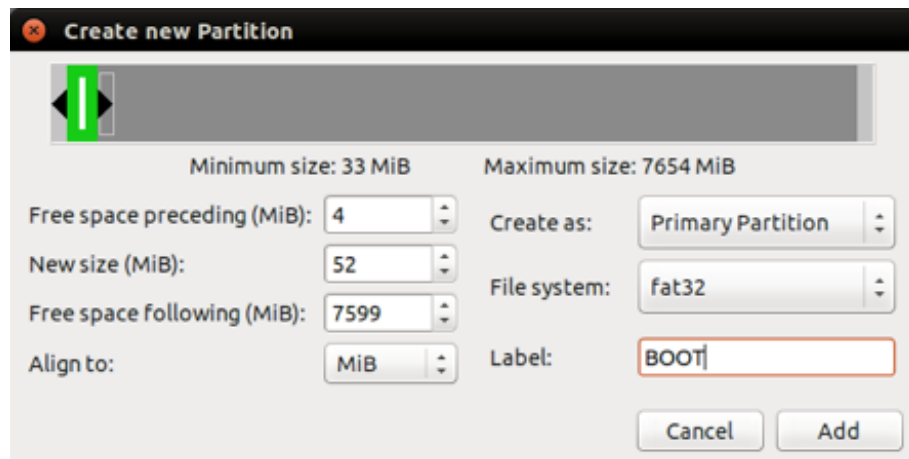


FIGURE A.15: editing the fat32 partition labeled BOOT



FIGURE A.16: symbol to click for editing partitions on GParted

- Boot the SD Card on the ZedBoard

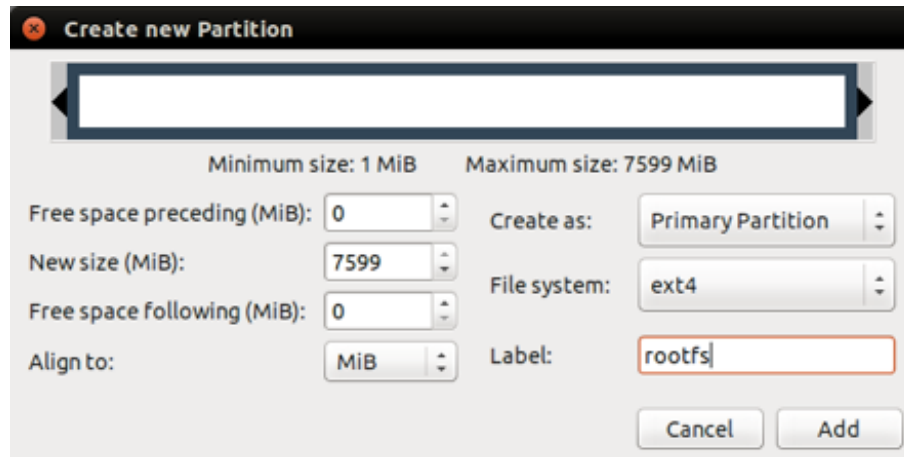


FIGURE A.17: editing the ext4 partition labeled rootfs

A.2.2.1 Copy the Linaro File System to the ext4 partition of the SD Card

A temporary folder named Linaro must be created and the zipped Linaro image must be copied there and successively unpacked, type:

```
$ mkdir -p /tmp/linaro
```

Then cd into the directory where the zipped Linaro image is and type (see figure A.18):

```
$ sudo cp image-name /tmp/linaro/fs.tar.gz
```

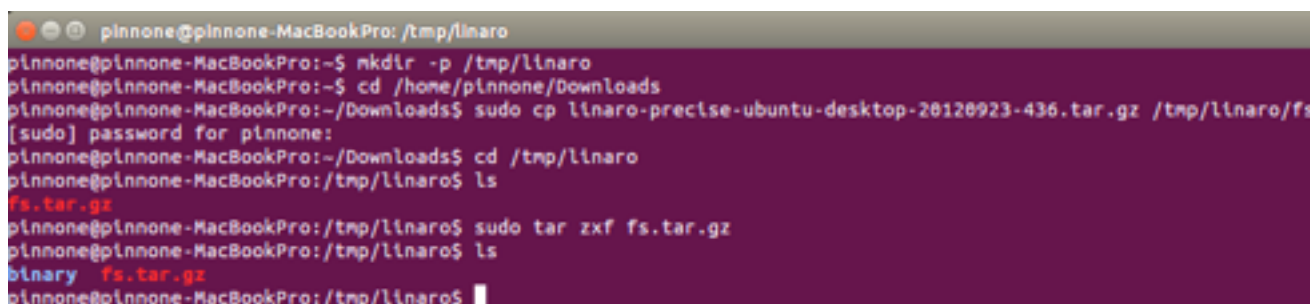


FIGURE A.18: Unzip the Linaro image in a temporary folder named "linaro"

Now insert the SDcard and mount it to `/tmp/sdext4`, obviously the device node of the ext4 partition of your SD card could be different from that one displayed below and you have to replace it (figure A.19). Now, for copying the root file system onto the SD card, it is strongly recommended to use the command `rsync` in order to preserve those attributes of the special files contained in the root file system that should be unchanged.

```
pinnone@pinnone-MacBookPro: ~
pinnone@pinnone-MacBookPro:~$ mkdir -p /tmp/sd_ext4
pinnone@pinnone-MacBookPro:~$ sudo mount /dev/sdb2 /tmp/sd_ext4
[sudo] password for pinnone:
pinnone@pinnone-MacBookPro:~$ █
```

FIGURE A.19: Mount the SD card to `/tmp/sd_ext4`

Once this process has completed, unmount before removing the SD card to make sure all the files have been synchronized to it.

```
pinnone@pinnone-MacBookPro: /tmp/linaro/binary/boot/filesystem.dir
pinnone@pinnone-MacBookPro:~$ cd /tmp/linaro
pinnone@pinnone-MacBookPro:/tmp/linaro$ cd binary/boot/filesystem.dir
pinnone@pinnone-MacBookPro:/tmp/linaro/binary/boot/filesystem.dir$ pwd
/tmp/linaro/binary/boot/filesystem.dir
pinnone@pinnone-MacBookPro:/tmp/linaro/binary/boot/filesystem.dir$ sudo rsync -a ./ /tmp/
[sudo] password for pinnone:
pinnone@pinnone-MacBookPro:/tmp/linaro/binary/boot/filesystem.dir$ sudo umount /tmp/sd_ext4
pinnone@pinnone-MacBookPro:/tmp/linaro/binary/boot/filesystem.dir$ █
```

FIGURE A.20: Umount the SD card

A.2.2.2 Build the Linux Kernel

Install ARM GNU Toolchain Before building the kernel, the first thing a user should do is making sure of having installed the ARM GNU tools from xilinx ([32]). With the latest versions of Xilinx ISE and Xilinx Vivado, this tools are already include. In our case so it was, in particular Xilinx ISE 14.7 and Xilinx Vivado 2014.4 were installed. As far as we know, these versions are probably not the first versions with ARM GNU tools included, anyhow we can assert that if a user install these versions (or also only one of them, either ISE or Vivado) he can be sure that the tools are included. Once the correct installation of these tools is assured, it is necessary to set the tool chain environment variables. For the oldest version of ISE and Vivado, the toolchain is contained in a folder named "CodeSourcery". For users who are in this situation (enough old version of Xilinx software), the instruction given by [28] are correct and reported here for completeness. open the `bashrc` file in your home folder (with a text editor like `gedit` or `emacs`) and add the lines below:

```
PATH=~ /CodeSourcery/Sourcery_CodeBench_Lite_for_Xilinx_GNU_Linux/bin:$PATH
```

```
export CROSS_COMPILE=arm-xilinx-linux-gnueabi-
```

You could need to add something to the path highlighted in red, depending on where your **CodeSourcery** folder is located. For users who installed later versions of Xilinx software (as we were) this instruction is no more valid. It was discovered that the file to be included in the path for the newest versions is the *settings64.sh* or *settings32.sh*, depending on the kind of hardware platform (64 or 32 bit CPU), this file is present both in the ISE_DS folder and in the Vivado folder. To avoid modifying the bashrc, we preferred to use the following bash commands, suggested at [33] and it worked well (figure A.2.2.2):

```
$ export CROSS_COMPILE=arm-xilinx-linux-gnueabi-
$ source /opt/Xilinx/14.7/ISE_DS/settings64.sh
```

Or it should be equivalent to run:

```
$ export CROSS_COMPILE=arm-xilinx-linux-gnueabi-
$ source /opt/Xilinx_Vivado/Vivado/2014.4/settings64.sh
```

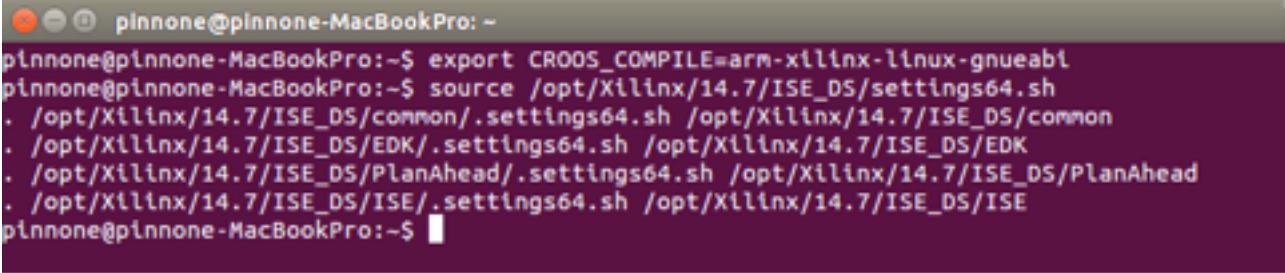
A terminal window screenshot on a MacBook Pro. The prompt is 'pinnone@pinnone-MacBookPro: ~'. The user enters the command 'export CROSS_COMPILE=arm-xilinx-linux-gnueabi-' and then 'source /opt/Xilinx/14.7/ISE_DS/settings64.sh'. The terminal output shows the execution of several sub-commands: './opt/Xilinx/14.7/ISE_DS/common/.settings64.sh /opt/Xilinx/14.7/ISE_DS/common', './opt/Xilinx/14.7/ISE_DS/EDK/.settings64.sh /opt/Xilinx/14.7/ISE_DS/EDK', './opt/Xilinx/14.7/ISE_DS/PlanAhead/.settings64.sh /opt/Xilinx/14.7/ISE_DS/PlanAhead', and './opt/Xilinx/14.7/ISE_DS/ISE/.settings64.sh /opt/Xilinx/14.7/ISE_DS/ISE'. The prompt returns to '\$'.

FIGURE A.21: Set ARM GNU Toolchain path

Remember that you could need to adapt the paths to your particular installation folders. In [33] it can be also read that Ubuntu 12.04 LTS *x86_64* users may run into issues related to missing dependencies when installing the Xilinx tools. This release of Ubuntu lacks some needed 32-bit libraries which need to be installed. Even if we used Ubuntu 14.04 LTS we faced the same problem, which is simply fixable by installing the 32 bit libraries with the bash command reported below:

```
$ sudo apt-get install ia32-libs
```

Obtain git repository Install git with

```
$ sudo apt-get install git
```

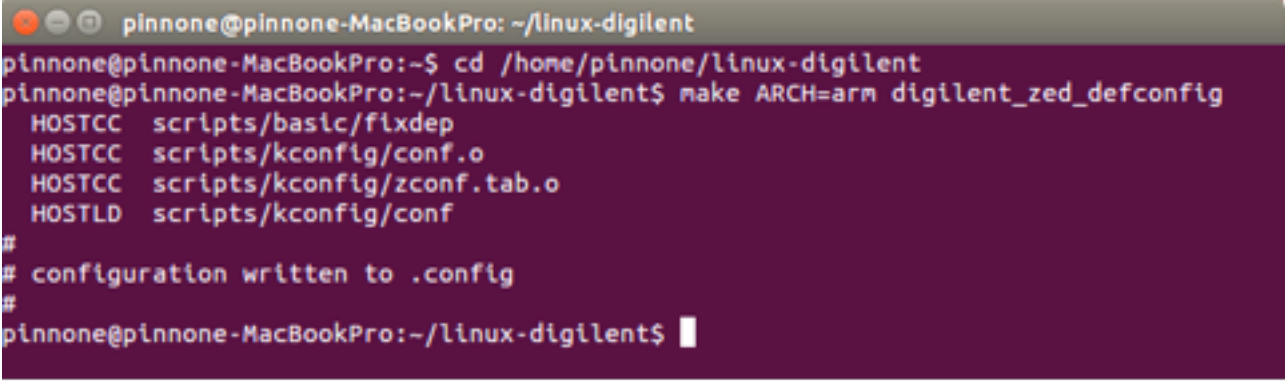
Once git is installed, open a terminal and change to the directory where you would like to place the Linux Kernel source code, then run:

```
$ git clone https://github.com/Digilent/linux-digilent.git
```

A.2.2.3 Configure the Kernel

After the download has completed, change to the **linux digilent** directory and run the command below to configure the kernel for the ZedBoard (figure A.2.2.3):

```
$ make ARCH=arm digilent_zed_defconfig
```



```
pinnone@pinnone-MacBookPro: ~/linux-digilent
pinnone@pinnone-MacBookPro:~$ cd /home/pinnone/linux-digilent
pinnone@pinnone-MacBookPro:~/linux-digilent$ make ARCH=arm digilent_zed_defconfig
HOSTCC  scripts/basic/flxdep
HOSTCC  scripts/kconfig/conf.o
HOSTCC  scripts/kconfig/zconf.tab.o
HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
pinnone@pinnone-MacBookPro:~/linux-digilent$
```

FIGURE A.22: Configure Linux Kernel for ZedBoard

To view or alter the default configuration of the Linux Kernel for the ZedBoard, run the command below.

```
$ make ARCH=arm menuconfig
```

A window like the one displayed in figure A.2.2.3 will pop up. From this configuration interface a user can select drivers built into the Kernel and those built as loadable modules. It is important to modify the default kernel configuration only if it is necessary and not to add useless driver: the adding of not needed drivers may cause an incorrect working of Linaro on ZedBoard or a performance degradation.

A.2.2.4 Build the Kernel

After properly configuring the kernel, proceed to build it by running.

```
$ make ARCH=arm
```

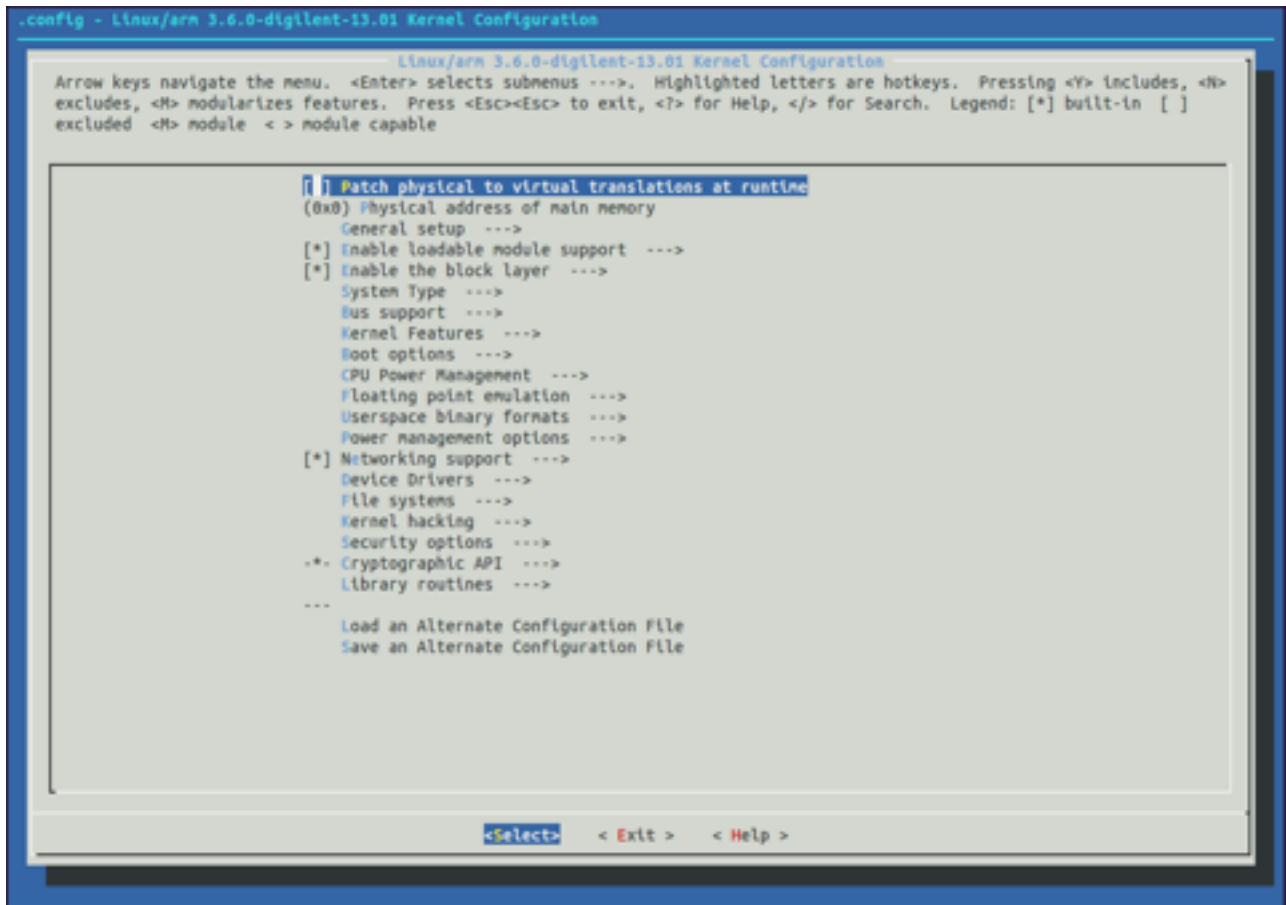


FIGURE A.23: Kernel configuration Menu

If the build completes without errors, you will find the built kernel image (a single binary file named **zImage**) at *linux-digilent/arch/arm/boot/zImage* see figure A.24. Copy the file to the FAT partition of the SD card (which is labeled BOOT). If an error like the one

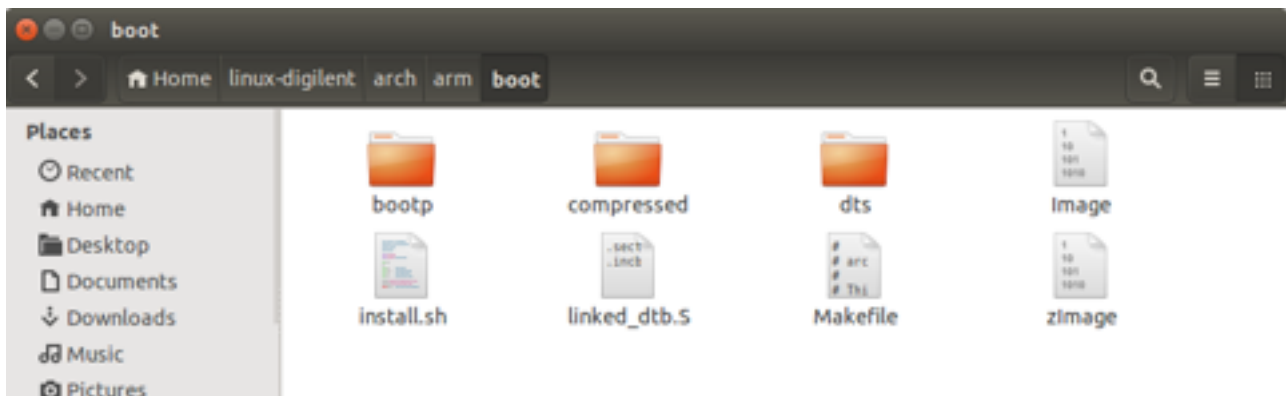


FIGURE A.24: The zImage present in the folder *linux-digilent/arch/arm/boot/zImage*

showed below occurs (figure ??), this signify that the ARM GNU toolchain environment

variables are not set (the tools are not in your path). In this case check the correct executions of the steps exposed in [A.2.2.2](#).

```
pinnone@pinnone-MacBookPro:~/linux-digilent$ make ARCH=arm
make: arm-xilinx-linux-gnueabi-gcc: Command not found
scripts/kconfig/conf --silentoldconfig Kconfig
make: arm-xilinx-linux-gnueabi-gcc: Command not found
  CHK      include/linux/version.h
  CHK      include/generated/utsrelease.h
make[1]: `include/generated/mach-types.h' is up to date.
  CC      kernel/bounds.s
/bin/sh: 1: arm-xilinx-linux-gnueabi-gcc: not found
make[1]: *** [kernel/bounds.s] Error 127
make: *** [prepare0] Error 2
```

FIGURE A.25: This error occurs when the ARM GNU Toolchain is not in the path

A.2.2.5 Obtain the BOOT.BIN File

The BOOT.BIN file is a container for the various Xilinx specific files that initially configure the two sections of the Zynq AP SoC, i.e. the programmable logic (the FPGA) and the processing system (the two ARM Cortex A9). This container also holds uboot, which is a second-stage bootloader responsible for loading Linux. A prebuilt BOOT.BIN file is present in the sd image folder contained in the 4GB SD card provided with ZedBoard. The BOOT.BIN must be copied to the FAT partition of your SD card. If a user is interested in building his own BOOT.BIN file, we suggest to see [ZedBoard Linux Hardware Design](#) available at [\[34\]](#). Here is a source code for a Xilinx Embedded Design Kit (EDK) project, distributed by Digilent, that will configure the Zynq part on the Zedboard in a manner that allows Linux to communicate properly with the onboard hardware. Further useful and very good explanation on what BOOT.BIN contains and how to create your own BOOT.BIN can be found at [\[35\],\[36\],\[37\]](#).

A.2.2.6 Compile the Device Tree

The device tree is a data structure that describes the hardware present in your system to the Linux Kernel. The Digilent Linux repository contains a default device tree for the ZedBoard that corresponds with the Linux Hardware Design. It may be found at [linux-digilent/arch/arm/boot/digilent-zed.dts](#). The device tree is also where the kernel is told what file system to load. This means that it is necessary to modify the **digilent-zed.dts** file to indicate which file system is in use, consequently two different actions must be done depending on how you are going to boot the ZedBoard, i.e. if you are

using a Linaro file system or a ramdisk image(as the one provided with the 4GB SD card in the ZedBoard box). Open **digilent-zed.dts** in a text editor. The lines containing the "bootargs" definitions will probably look like figure A.26: If you are using a Linaro

```
chosen {
    bootargs = "console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlyprintk rootfstype=ext4 rootwait devtmpfs.mount=1";
    /* bootargs = "console=ttyPS0,115200 root=/dev/ram rw initrd=0x800000,8M init=/init earlyprintk rootwait
devtmpfs.mount=1"; */
    linux,stdout-path = "/axi@0/serial@e0001000";
};
```

FIGURE A.26: Line containing the "bootargs" in the **digilent-zed.dts** file at the beginning

file system (as it probably is if you are reading this paper), substitute the 1 at the end of the first line with a 0, obtaining what is showed in figure A.27: Now the compiled device

```
chosen {
    bootargs = "console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlyprintk rootfstype=ext4 rootwait devtmpfs.mount=0";
    /* bootargs = "console=ttyPS0,115200 root=/dev/ram rw initrd=0x800000,8M init=/init earlyprintk rootwait
devtmpfs.mount=1"; */
    linux,stdout-path = "/axi@0/serial@e0001000";
};
```

FIGURE A.27: Line containing the "bootargs" in the **digilent-zed.dts** file after the modifications in order to boot ZedBoard with a Linaro file system

tree should be at linux-digilent/devicetree.dtb, copy this file to the FAT partition of the SD card and it should be finally ready to be plugged into the ZedBoard for beginning the boot process. Now onto the SD card you should have: in the fat partition the compiled device tree, the zImage you obtained building the kernel and the BOOT.BIN copied from the 4GB SD card provided with zedboard, while in the ext4 partition you should have the Linaro file system. As it can be noticed, compiling the device tree appears to be quite tricky and it may lead to do errors. All this steps can be skipped by directly downloading a pre-compiled devicetree from [38] in the form of a devicetree-linaro.dtb file (in figure A.28 the link for download is showed). Anyhow this was the procedure followed for this work and it worked well, so we copied the BOOT.BIN provided with ZedBoard, we downloaded the pre-compiled device tree from [38] and we booted the ZedBoard.

A.2.2.7 Boot the SD card onto the ZedBoard

1. Insert the SD card into the ZedBoard
2. Set the jumpers of the ZedBoard in the SD-boot mode as follows (see also [39] and [40]):

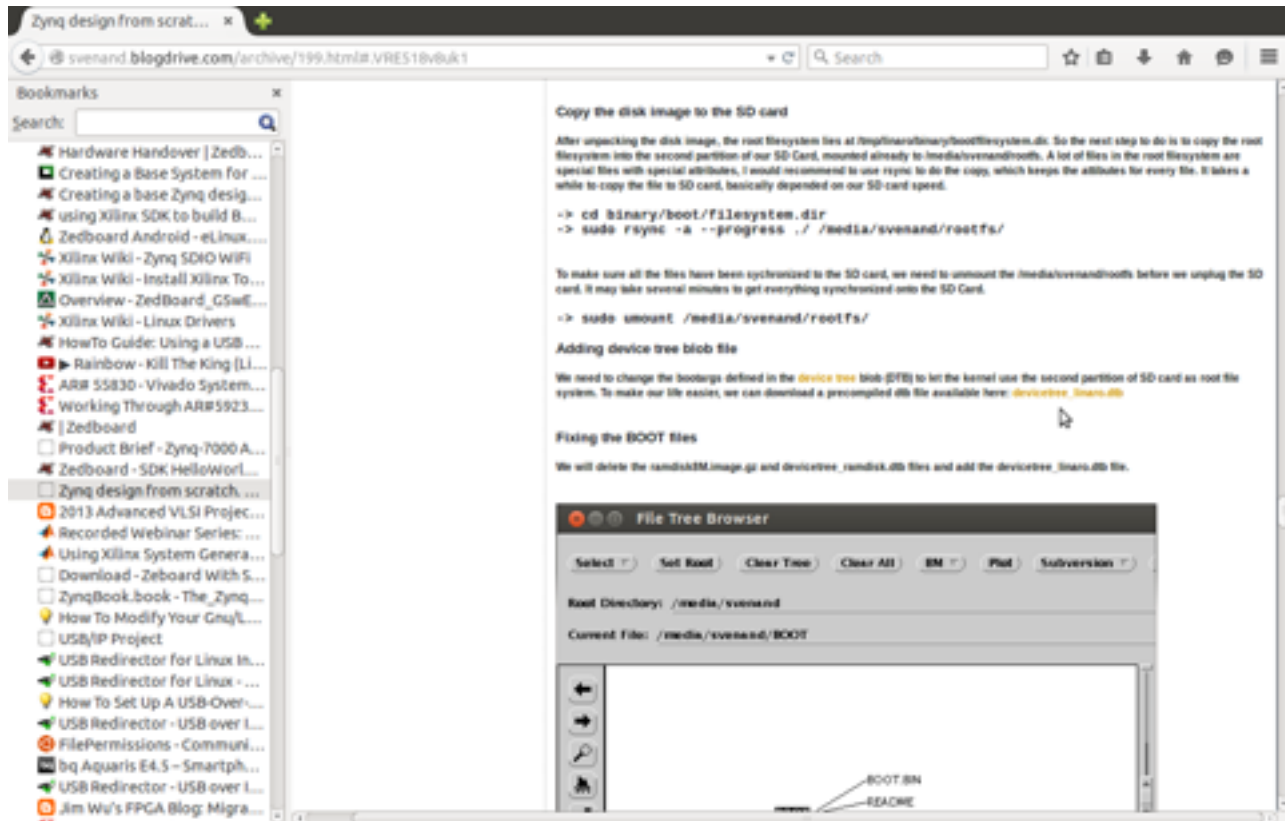


FIGURE A.28

- MIO 6: set to GND
 - MIO 5: set to 3V3
 - MIO 4: set to 3V3
 - MIO 3: set to GND
 - MIO 2: set to GND
 - VADJ Select: set to 1V8
 - JP6: shorted
 - JP2: shorted
 - All other jumpers should be left unshorted
3. Attach a computer running a terminal emulator (see next chapters for details) to the UART port with a Micro-USB cable. Configure the terminal emulator with the following settings:
- Baud Rate: 115200
 - 8 data bits
 - 1 stop bit
 - no parity
 - no flow control

4. connect any peripherals you want to use in Linux. If you use a Linaro file system, it would be useful to connect a monitor to the HDMI port and a USB hub to the USBOTG port, in order to attach a mouse and a keyboard to the USB hub.
5. Attach a 12V power supply to the ZedBoard and power it on.
6. Use the terminal emulator to boot the ZedBoard following the instruction given in next chapters.

A.2.3 Boot Linaro using Minicom

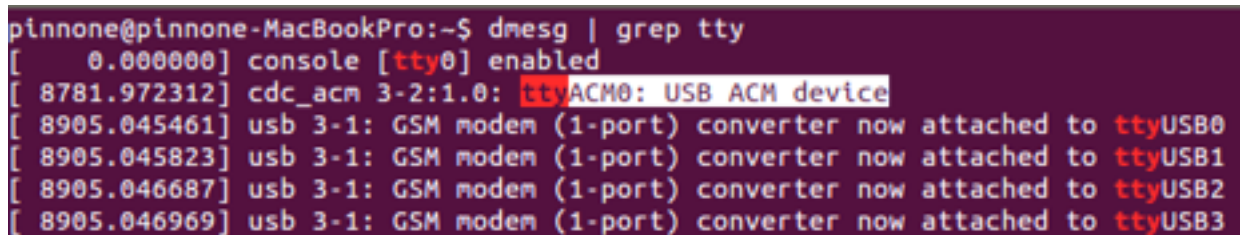
In Linux, a terminal emulator which can be used to boot up the Zedboard is **minicom**, it can be obtained by running:

```
$ sudo apt-get install minicom
```

Once minicom is installed and ZedBoard is connected via UART port to your pc and powered on, check the serial device name of ZedBoard from a bash shell with:

```
$ dmesg | grep tty
```

the name will probably be *ttyACM0* as in the case showed in the figure [A.29](#) Now run

A terminal window screenshot showing the output of the command 'dmesg | grep tty'. The output lists several USB ACM devices. The first line is '[0.000000] console [tty0] enabled'. The second line is '[8781.972312] cdc_acm 3-2:1.0: ttyACM0: USB ACM device', where 'ttyACM0: USB ACM device' is highlighted with a red box. The following three lines show USB ACM devices attached to ttyUSB0, ttyUSB1, ttyUSB2, and ttyUSB3, each with a timestamp around 8905.045461 to 8905.046969.

```
pinnone@pinnone-MacBookPro:~$ dmesg | grep tty
[ 0.000000] console [tty0] enabled
[ 8781.972312] cdc_acm 3-2:1.0: ttyACM0: USB ACM device
[ 8905.045461] usb 3-1: GSM modem (1-port) converter now attached to ttyUSB0
[ 8905.045823] usb 3-1: GSM modem (1-port) converter now attached to ttyUSB1
[ 8905.046687] usb 3-1: GSM modem (1-port) converter now attached to ttyUSB2
[ 8905.046969] usb 3-1: GSM modem (1-port) converter now attached to ttyUSB3
```

FIGURE A.29: Check ZedBoard device name with **dmesg** — **grep tty**, it will probably be *ttyACM0*

the following command as Super User (this is very important), see figure [A.30](#):

```
$ minicom -s
```

This action will result in the opening of the prompt reported in figure [A.31](#). Go to **Serial Port Setup** and press Enter. The window in figure [A.32](#) will pop up. Pressing the letters displayed on the left side, you can change the corresponding serial port settings. In the case of ZedBoard, you should change the serial port settings as:

- A: /dev/ttyACM0 (probably)

```
root@pinnone-MacBookPro: /home/pinnone
pinnone@pinnone-MacBookPro:~$ sudo su
[sudo] password for pinnone:
root@pinnone-MacBookPro:/home/pinnone# minicom -s
```

FIGURE A.30: Start minicom

```
+-----[configuration]-----+
| Filenames and paths          |
| File transfer protocols     |
| Serial port setup           |
| Modem and dialing           |
| Screen and keyboard         |
| Save setup as dfl           |
| Save setup as..             |
| Exit                         |
| Exit from Minicom           |
+-----+
```

FIGURE A.31: Minicom settings window

- B: leave as default
- C: leave as default
- D: leave as default
- E: 115200 8N1
- F: NO
- G: NO

Once all these features are set, exit from minicom and you should see something like figure A.33: If such a window is displayed, power off and power on the ZedBoard again, then you should see feedback arriving to minicom, until another window with a count-down to autoboot is displayed (figure A.34), at that time press any key to stop the autoboot process. Now you are on the board, run the following command to start to boot Linaro:

```
$ run sdboot_linaro
```

```
+-----+
| A -   Serial Device       : /dev/tty8
| B - Lockfile Location    : /var/lock
| C -   Callin Program     :
| D -   Callout Program    :
| E -   Bps/Par/Bits       : 115200 8N1
| F - Hardware Flow Control : Yes
| G - Software Flow Control : No
|
| Change which setting? █
+-----+
| Screen and keyboard
| Save setup as dfl
| Save setup as..
| Exit
| Exit from Minicom
+-----+
```

FIGURE A.32: Minicom serial port setup

```
Welcome to minicom 2.7

OPTIONS: I18n
Compiled on Jan  1 2014, 17:13:19.
Port /dev/ttyACM0, 14:47:36

Press CTRL-A Z for help on special keys

@
Copying Linux from SD to RAM...
Device: SDHCI
Manufacturer ID: 41
OEM: 3432
Name: SD16G
█ran Speed: 25000000
```

FIGURE A.33: Window displayed when closing minicom after serial port set up

The boot process will start, it may be not so fast. At the end you should see the Linaro Ubuntu desktop on your HDMI monitor (if you attached one) and the following on your terminal (figure A.35): This signifies that you are on the root of Linaro and you can either operate from this UART terminal, or disconnect your pc from the UART port and use the keyboard and the mouse attached to the USB OTG port to control the

```
root@plnnone-MacBookPro: /home/plnnone
Press CTRL-A Z for help on special keys

U-Boot 2011.03-dirty (Jul 11 2012 - 16:07:00)

DRAM:  512 MiB
MMC:   SDHCI: 0
Using default environment

In:    serial

U-Boot 2011.03-dirty (Jul 11 2012 - 16:07:00)

DRAM:  512 MiB
MMC:   SDHCI: 0
Using default environment

In:    serial
Out:   serial
Err:   serial
Net:   zynq_gem
Hit any key to stop autoboot:  0
zed-boot>
```

FIGURE A.34: Stop ZedBoard auto boot process

```
root@plnnone-MacBookPro: /home/plnnone/linux-digilent
[ 1.570000] NET: Registered protocol family 17
[ 1.580000] VFP support v0.3: implementor 41 architecture 3 part 30 variant 9
rev 4
[ 1.580000] Registering SWP/SWPB emulation handler
[ 1.590000] registered taskstats version 1
[ 1.590000] adv7511-hdmi-snd adv7511_hdmi_snd.2: CODEC adv7511.2-0039 not reg
istered
[ 1.600000] drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
[ 1.610000] platform adv7511_hdmi_snd.2: Driver adv7511-hdmi-snd requests pro
be deferral
[ 1.610000] ALSA device list:
[ 1.620000]   No soundcards found.
[ 2.600000] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. 0
pts: (null)
[ 2.610000] VFS: Mounted root (ext4 filesystem) on device 179:2.
[ 2.610000] Freeing init memory: 148K
[ 3.480000] init: ureadahead main process (638) terminated with status 5
Last login: Thu Jan  1 00:00:12 UTC 1970 on tty1
cat: /var/lib/update-notifier/fsck-at-reboot: No such file or directory
run-parts: /etc/update-motd.d/98-fsck-at-reboot exited with return code 1
Welcome to Linaro 12.09 (GNU/Linux 3.6.0-digilent-13.01-00002-g06b3889 armv7l)

* Documentation:  https://wiki.linaro.org/
root@linaro-ubuntu-desktop:~#
```

FIGURE A.35: Linaro boot process finished

complete linux system running on the ZedBoard (using obviously the desktop displayed on the monitor).

A.2.4 Booting Linaro using GNU Screen

Exactly the same passages can be done using GNU screen, which is a faster way to connect to serial port. If you don't have this program installed, you can obtain with:

```
$ sudo apt-get install screen
```

Once it is installed and ZedBoard is connected via UART to your pc, power on the board and, always as super user, run:

```
$ screen /dev/ttyACM0 115200
```

You should see figure [A.34](#) after a few seconds. It is important not to wait too much time, after powering on the ZedBoard, to run the command above (you should run it at least before the blue led on the board starts blinking). From the prompt in figure [A.35](#), run also in this case:

```
$ run sdboot_linaro
```

and the boot process will begin. Once the boot process ends, whatever method you used (minicom or gnu screen), you should see a (Linaro) Ubuntu desktop on the monitor ZedBoard has been previously connected to, as in figure [A.36](#).



FIGURE A.36: Linaro Ubuntu desktop appears on the monitor

A.3 How to share a PC Internet Connection with Zed-Board

Giving internet connection to ZedBoard with Linaro running is fundamental, in order to be able to make updates, upgrades and install new software. A simple way to give internet connection to ZedBoard is sharing the one of your PC, using it as a DHCP server. Supposing a user has already booted ZedBoard from a pc as explained above, to share its internet connection the following steps have to be followed:

1. click on the wireless symbol in the upper right corner of your ubuntu desktop and then click on **Edit connections** (figure A.37)

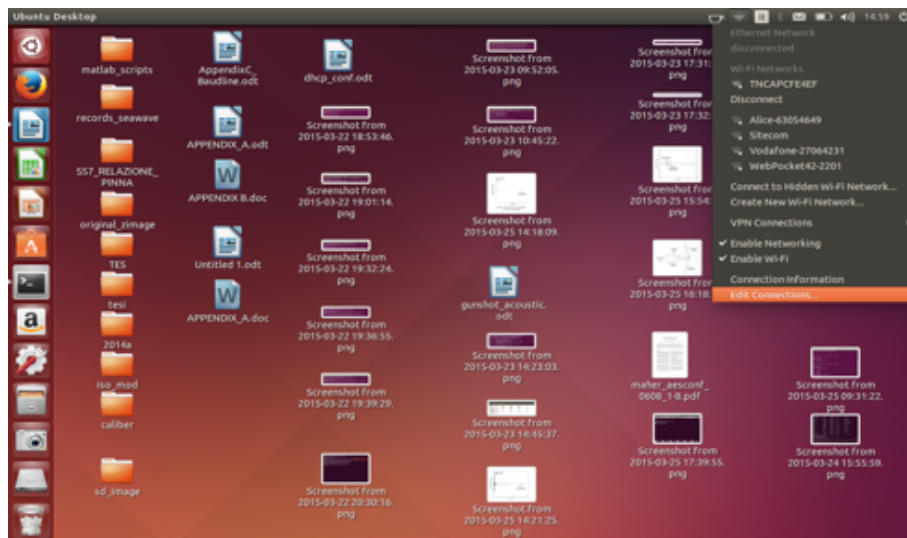


FIGURE A.37: Edit a new connection

2. In the window that will pop up, click on **Add** (figure A.38)
3. Choose an ethernet connection type (figure A.39)
4. In the editing connection window, the only option that must be set is: **IPv4Settings / Shared to other computers**, as in figure A.40
5. Save and close, connect your ZedBoard to the pc with an ethernet cable and you should see an advice appearing in the upper right corner of the desktop indicating that the shared connection is available, thus ZedBoard is connected to the internet.

A.4 ZedBoard Remote Control

In what follows, it is supposed that a **Linaro-ALIP** version is installed, because, as already written, it is the best version in order to control ZedBoard with Ssh and VNC.

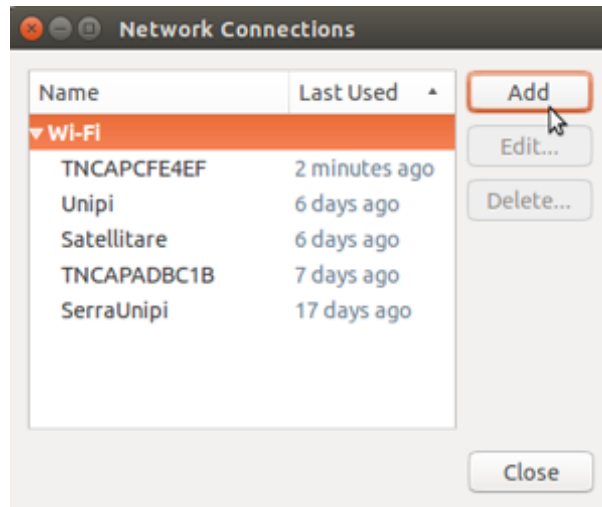


FIGURE A.38: Add a new connection



FIGURE A.39: Choose an ethernet connection

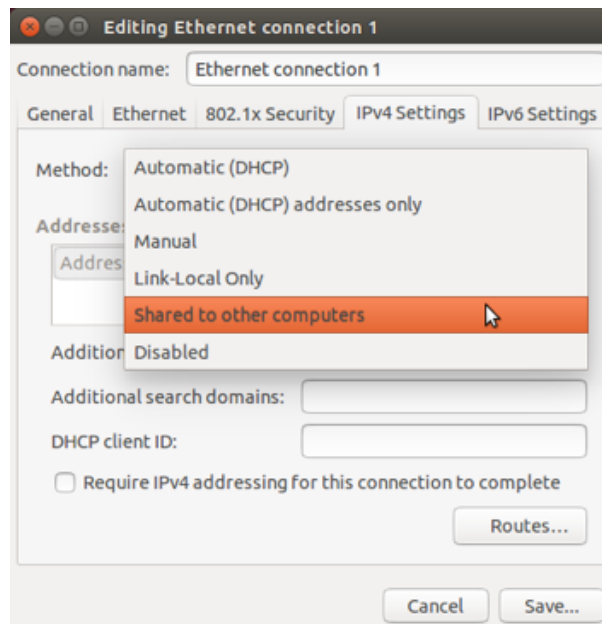


FIGURE A.40: Share the connection to other computers

A.4.1 Connecting to ZedBoard with SSH

For connecting a host pc to ZedBoard with Ssh, two different hardware configurations are possible:

1. Zedboard is connected to a network through the host pc, being connected to it with an ethernet cable (figure [A.41](#));
2. Zedboard is connected directly to a network (for example to a modem with an ethernet cable) and the host PC is connected to the same network (figure [A.42](#)).

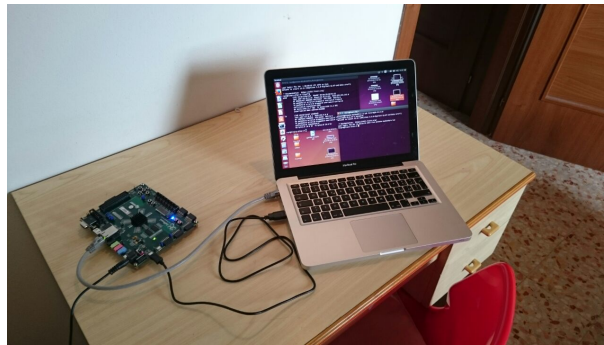


FIGURE A.41: First ssh ZedBoard configuration

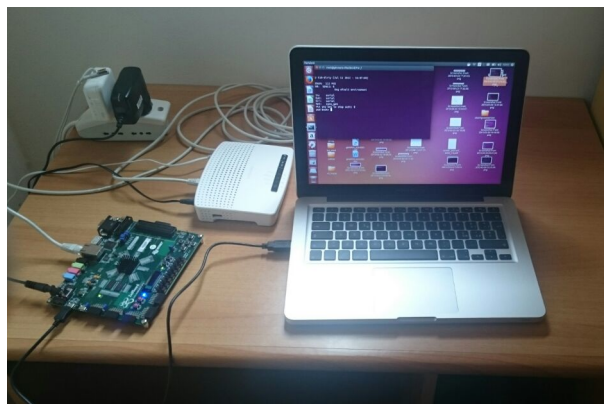


FIGURE A.42: Second ssh ZedBoard configuration

Obviously in these two configuration descriptions it has been omitted that (at least at the beginning), for booting the ZedBoard, the host pc must be connected to the UART port of the ZedBoard with an USB-miniUSB cable. Once one of these configurations is set up, boot Linaro on the ZedBoard, and then look for the device IP address running on the UART terminal (figure [A.43](#)):

```
$ ifconfig
```

Notice that in image [A.43](#) the IP address of the ZedBoard is quite unusual (10.42.0.35), this happens in the case of configuration 1 (probably because the host pc is acting as

```
root@linaro-alip:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0a:35:00:01:22
          inet addr:10.42.0.35  Bcast:10.42.0.255  Mask:255.255.255.0
```

FIGURE A.43: Finding ZedBoard IP address

a DHCP server). Otherwise the IP address will be similar to the one of the host pc (). Supposing that at the first time you boot linaro-alip on the ZedBoard **open ssh** is not installed, run this sequence of bash commands in order to install **open ssh server** on Linaro:

```
$ apt-get update
$ apt-get upgrade
$ apt-get install openssh-server
$ sync
```

Now that the IP address is known, two other informations are needed in order to set up a SSH connection with ZedBoard: the device name and the password. Usually the host name can be found with the command:

```
$ hostname
```

Depending on the specific Linaro distribution installed on the SD card, this command will return different names, for what was experienced for this work they are:

- for the desktop distribution: *linaro-ubuntu-desktop*;
- for the nano distribution: *linaro-nano*;
- for the alip distribution: *linaro-alip*;

But when trying to create an SSH connection using these device names, no one of them worked. Finally it was discovered that, even though hostname command returns these names, the default name, independently on which Linaro distribution is installed, is always linaro and the same is the default password. So now that all the necessary informations are at disposal, run:

```
$ ssh linaro@device ip address
```

Remember that linaro is the password too (figure A.44). Now the host pc is controlling zedboard via SSH.

```
pinnone@pinnone-MacBookPro:~$ ssh linaro@10.42.0.35
linaro@10.42.0.35's password:
Welcome to Linaro 12.11 (GNU/Linux 3.3.0-digilent-12.07-zed-beta armv7l)

* Documentation: https://wiki.linaro.org/
Last login: Thu Mar 26 12:21:53 2015 from pinnone-macbookpro.lan
linaro@linaro-alip:~$
```

FIGURE A.44: Creating the SSH connection with ZedBoard

A.4.1.1 SSH with X11: a GUI for applications

It could be desirable to have a GUI access to the applications in Linaro, hence it would be necessary to create a Ssh connection using also x11. To do so, it is necessary, obviously, to have x11 installed on Linaro. With linaro-alip it should be installed as default, for this work it was. If it is not, in a linaro terminal try to run:

```
$ apt-get install x11-xserver-utils xserver-xorg xserver-xorg-video-fbdev xserver-xorg
```

When x11 is installed on Linaro, open a terminal on the host pc and run (see figure A.45):

```
$ssh -X linaro@device ip address
```

```
pinnone@pinnone-MacBookPro:~$ ssh -X linaro@10.42.0.35
linaro@10.42.0.35's password:
Welcome to Linaro 12.11 (GNU/Linux 3.3.0-digilent-12.07-zed-beta armv7l)

* Documentation: https://wiki.linaro.org/
Last login: Mon Mar 30 12:18:04 2015 from 10.42.0.1
linaro@linaro-alip:~$ ls
```

FIGURE A.45: Using x11 with SSH

Now you can open applications with a GUI support; for example you can run (see figure A.46):

```
$ gedit
```

A.4.2 Controlling ZedBoard with (tight) VNC

The first thing to know is that the only configuration that allows to control zedboard with VNC is configuration 2. The second thing to know is that a normal VNC server version would run quite slowly on the ZedBoard (because it is too heavy), so it is suggested to download the tight version: open a terminal on linaro (for example from the UART terminal used for the boot process) and run:

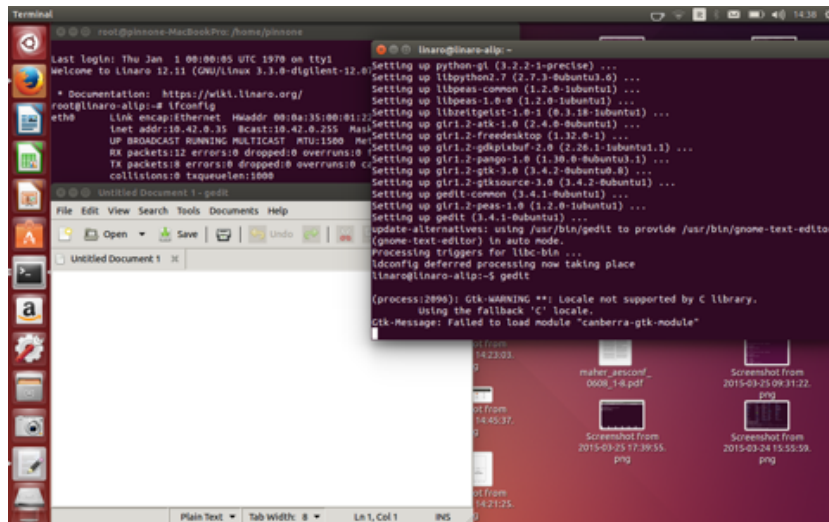


FIGURE A.46: a x11 GUI for gedit

```
$ sudo apt-get install lxde
```

Now on the host computer a *VNCviewer* or *Remmina* must be installed. In our case, *VNCviewer* was downloaded from [41]. The downloaded file is a compressed one, after it is unpacked in the desired folder it must be made executable, to do so `cd` into the location folder and use the command:

```
$ sudo sudo chmod +x VNC-Viewer-5.2.3-Linux-x64
```

Now start a vnc server session on the ZedBoard, from a Linaro terminal (see figure A.47):

```
$ vncserver
```

Choosing a password will be required. Before ending the process, the terminal will return a desktop name for the vncserver session, in the case of the figure below it is **linaro-alip:1** (see figure A.47). Now in a terminal on the host pc start *VNCviewer* (`cd`

```
root@linaro-alip:~# vncserver
New 'X' desktop is linaro-alip:1
Starting applications specified in /root/.vnc/xstartup
Log file is /root/.vnc/linaro-alip:1.log
root@linaro-alip:~#
```

FIGURE A.47: Starting a VNC server session on the ZedBoard

into the directory where the executable is located and `./vncviewer` as in figure A.48).

```
pinnone@pinnone-MacBookPro:~$ cd /home/pinnone
pinnone@pinnone-MacBookPro:~$ ./VNC-Viewer-5.2.3-Linux-x64
```

FIGURE A.48: Starting *VNCviewer* on the host PC

FIGURE A.49: VNCviewer GUI

The window in figure A.49 will pop up. Insert the vnc server desktop name and click on connect. At the window in figure A.50 insert the password previously chosen and click **ok**, then the zedboard linaro desktop will be displayed (figure A.51). Clicking on the

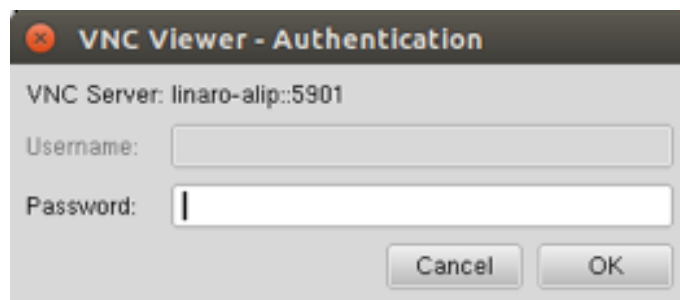


FIGURE A.50: VNCviewer GUI authentication



FIGURE A.51: VNC lxde remote Linaro desktop

button in the lower left corner, look for *Xterm* and open it: it is a complete terminal for Linaro on ZedBoard (figure A.52). If you want to close a vnc server session, but you

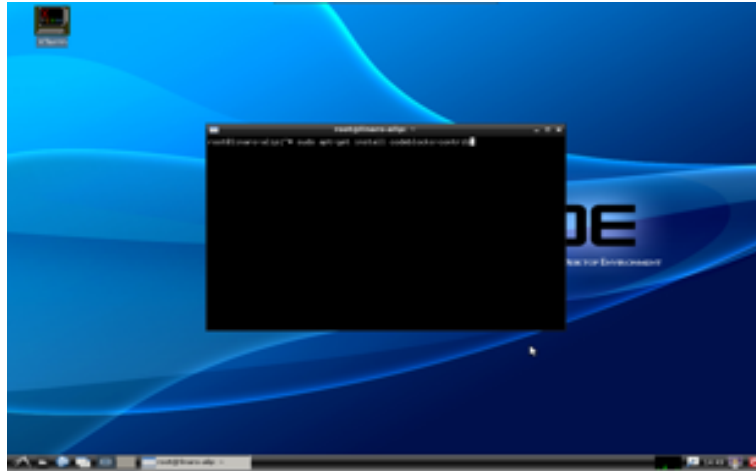


FIGURE A.52: Xterm running on a VNC lxde remote linaro desktop

do not want to poweroff the Zedboard, on the UART terminal (hoping it has not been disconnected) run:

```
$ vncserver -kill :1
```

If you want to poweroff the ZedBoard, on the *Xterm* terminal running on the VNC lxde desktop simply type:

```
$ sudo poweroff
```

This is a very important advice: independently on which way you are controlling the ZedBoard (with a linaro desktop on a monitor, with a UART terminal, with SSH or with VNC) when you want to power it off always run the previous command before switching off the board, this will assure that the correct shutting down sequence is followed. If this advice is not respected, some data can be lost or the operating system onto the SD card may get damaged.

A.5 Bug Fixed

During the production of this paper, a bug affecting the correct shut down was discovered. Informations given by Ubuntu forums made us know that this bug has been already quite well experienced by ubuntu users. Consequently it is probably related to Ubuntu distributions in general, not especially to the Linaro one or to ZedBoard. Anyhow it is reported here. The problem consists in a anomalous slow shut down, also, after the command **poweroff** has been made running, the terminal returns the message in figure A.53. This seems to be caused by network manager still running during the power

```
root@linaro-ubuntu-desktop:~# poweroff
Broadcast message from root@linaro-ubuntu-desktop
(/dev/ttyPS0) at 0:01 ...

The system is going down for power off NOW!
root@linaro-ubuntu-desktop:~# Checking for running unattended-upgrades:
modem-manager[1347]: <info> Caught signal 15, shutting down..
```

FIGURE A.53: warning message identifying the bug

off process. The solution is manually stopping network manager before shut down. To do so, it is sufficient to run:

```
$ sudo service network-manager stop
```

Or, equivalently:

```
$ sudo stop network-manager
```

Appendix B

Script in C for Single Channel Single Sensor shooter ranging

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
float sound_speed(double tau);
float single_sensor(float c, double b, double dtoa, double v);
int main(){
double b = // input data ;
double tau = // input data;
double dtoa = // input data;
double v = // input data;
float c = sound_speed(tau);
float a = sqrt((pow(v,2))+pow(c,2));
float A = 2*((b*v*a*pow(c,2))-(b*a*pow(v,3))-(dtoa*pow(c,3)*pow(v,2))-(dtoa*c*pow(v,4)));
float B = -2*(pow(b,2)*pow(v,4)*pow(c,4))+2*(pow(dtoa,2)*pow(c,6)*pow(v,4))-2*(pow(c,7)*b*dtoa*
+ pow(dtoa,2)*pow(c,4)*pow(v,6) + pow(c,8)*pow(dtoa,2)*pow(v,2);
float distance = (A - 2*sqrt(B))/(2*(pow(c,4)-pow(v,4)));
printf("%f ",distance);
return 0; }

float sound_speed(double tau){

float c; float temp; float result;

c = 1 + (tau/273.15); temp = sqrt(c); result = 331.3*temp;

return result;
```


}

Bibliography

- [1] Peter Volgyesi, Gyorgy Balogh, Andras Nadas, Christopher B Nash, and Akos Ledeczi. Shooter localization and weapon classification with soldier-wearable networked sensors. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 113–126. ACM, 2007.
- [2] Gyula Simon, Miklós Maróti, Ákos Lédeczi, György Balogh, Branislav Kusy, András Nádas, Gábor Pap, János Sallai, and Ken Frampton. Sensor network-based countersniper system. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 1–12. ACM, 2004.
- [3] Ákos Lédeczi, András Nádas, Péter Völgyesi, György Balogh, Branislav Kusy, János Sallai, Gábor Pap, Sebestyén Dóra, Károly Molnár, Miklós Maróti, et al. Countersniper system for urban warfare. *ACM Transactions on Sensor Networks (TOSN)*, 1(2):153–177, 2005.
- [4] Qinetiq ears gunshot localization system website. URL <http://www.qinetiq-na.com/products-security-ears.htm>.
- [5] Janos Sallai, Peter Volgyesi, Ken Pence, and Akos Ledeczi. Fusing distributed muzzle blast and shockwave detections. In *Information Fusion (FUSION), 2011 Proceedings of the 14th International Conference on*, pages 1–8. IEEE, 2011.
- [6] Jemin George and Lance M Kaplan. Shooter localization using soldier-worn gunfire detection systems. In *Information Fusion (FUSION), 2011 Proceedings of the 14th International Conference on*, pages 1–8. IEEE, 2011.
- [7] URL <http://www.lockheedmartin.com/us/products/span.html>.
- [8] Karunakar Pothuganti and Anusha Chitneni. A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi.
- [9] Manoj Kumar. Zigbee: The low data rate wireless technology for ad-hoc and sensor networks. In *NCCI 2010-National Conference on Computational Instrumentation CSIO*, 2010.

-
- [10] Thyagaraju Damarla, Lance M Kaplan, and Gene T Whipps. Sniper localization using acoustic asynchronous sensors. *Sensors Journal, IEEE*, 10(9):1469–1478, 2010.
- [11] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49. ACM, 2004.
- [12] . URL <http://www.spectracomcorp.com/ProductsServices/ByApplication/GPSClockSynchronization/tabid/100/Default.aspx>.
- [13] János Sallai, Ákos Lédeczi, and Péter Völgyesi. Acoustic shooter localization with a minimal number of single-channel wireless sensor nodes. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, pages 96–107. ACM, 2011.
- [14] Atmel sam r21 brochure. URL http://www.atmel.com/Images/45067A-SAM-R21_Brochure_E_A4_021214_Web.pdf.
- [15] Uc530 fastrax gps antenna module - product summary, . URL http://u-blox.com/images/downloads/Product_Docs/UC530_ProductSummary_%28FTX-HW-12004%29.pdf.
- [16] . URL <http://zedboard.org>.
- [17] Rob Maher. Acoustical characterization of gunshots. In *Signal Processing Applications for Public Security and Forensics, 2007. SAFE'07. IEEE Workshop on*, pages 1–5. IET, 2007.
- [18] János Sallai, Péter Völgyesi, Ákos Lédeczi, Ken Pence, Ted Bapty, Sandeep Neema, and James R Davis. Acoustic shockwave-based bearing estimation. In *Proceedings of the 12th international conference on Information processing in sensor networks*, pages 217–228. ACM, 2013.
- [19] T Makinen, P Pertila, and Pasi Auranen. Supersonic bullet state estimation using particle filtering. In *Signal and Image Processing Applications (ICSIPA), 2009 IEEE International Conference on*, pages 150–155. IEEE, 2009.
- [20] Brian T Mays. Shockwave and muzzle blast classification via joint time frequency and wavelet analysis. Technical report, DTIC Document, 2001.
- [21] Gregory L Duckworth, Douglas C Gilbert, and James E Barger. Acoustic counter-sniper system. In *Enabling Technologies for Law Enforcement and Security*, pages 262–275. International Society for Optics and Photonics, 1997.

- [22] Shie Qian and Dapang Chen. Joint time-frequency analysis. *Signal Processing Magazine, IEEE*, 16(2):52–67, 1999.
- [23] Jemin George, Lance M Kaplan, Socrates Deligeorges, and George Cakiades. Multi-shooter localization using finite point process. In *Information Fusion (FUSION), 2014 17th International Conference on*, pages 1–7. IEEE, 2014.
- [24] Lindgren David, Wilsson Olof, Gustafsson Fredrik, and Habberstad Hans. Shooter localization in wireless microphone networks. *Eurasip journal on advances in signal processing*, 2010, 2010.
- [25] . URL <http://www.dodotronic.com/acoustic-devices/ultramics>.
- [26] Ultramic user guide, . URL http://www.dodotronic.com/zoologia/files/Ultramic_User_Guide.pdf.
- [27] . URL <http://zedboard.org/content/zedboard-bring>.
- [28] Getting started with embedded linux – zedboard, . URL https://www.digilentinc.com/Data/Products/EMBEDDED-LINUX/ZedBoard_GSwEL_Guide.pdf.
- [29] URL <http://svenand.blogdrive.com/archive/199.html#.VPR6N8v8uk0>.
- [30] . URL <http://releases.linaro.org>.
- [31] . URL <http://releases.linaro.org/12.09/ubuntu/precise-images/ubuntu-desktop/linaro-precise-ubuntu-desktop-20120923>.
- [32] URL <http://wiki.xilinx.com/zynq-tools>.
- [33] URL <http://www.wiki.xilinx.com/Install+Xilinx+Tools>.
- [34] . URL <http://www.digilent.com/zedboard>.
- [35] . URL <http://www.wiki.xilinx.com/Prepare+boot+image>.
- [36] . URL <http://www.xilinx.com/training/zynq/how-to-create-zynq-boot-image-using-xilinx-sdk.html>.
- [37] Zynq-7000 all programmable soc software developers guide, . URL http://www.xilinx.com/support/documentation/user_guides/ug821-zynq-7000-swdev.pdf.
- [38] URL <http://svenand.blogdrive.com/archive/199.html#.VPR6N8v8uk0>.
- [39] Zedboard - getting started guide, . URL <http://zedboard.org/sites/default/files/documentations/GS-AES-Z7EV-7Z020-G-V7.pdf>.

- [40] Zedboard - hardware user's guide, . URL http://zedboard.org/sites/default/files/ZedBoard_HW_UG_v1_1.pdf.
- [41] URL <http://www.realvnc.com/download/viewer/>.